Loop freedom of the (untimed) AODV routing protocol
Timothy Bourke, Peter Höfner

To cite this version:
Timothy Bourke, Peter Höfner. Loop freedom of the (untimed) AODV routing protocol. 2014, pp.496. hal-01104033

HAL Id: hal-01104033
https://inria.hal.science/hal-01104033
Submitted on 15 Jan 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Loop freedom of the (untimed) AODV routing protocol

Timothy Bourke\textsuperscript{1} \quad Peter Höfner\textsuperscript{2}

November 3, 2014

\textsuperscript{1}Inria, École normale supérieure, and NICTA
\textsuperscript{2}NICTA and Computer Science and Engineering, UNSW
Abstract

The Ad hoc On-demand Distance Vector (AODV) routing protocol [6] allows the nodes in a Mobile Ad hoc Network (MANET) or a Wireless Mesh Network (WMN) to know where to forward data packets. Such a protocol is ‘loop free’ if it never leads to routing decisions that forward packets in circles.

This development mechanises an existing pen-and-paper proof of loop freedom of AODV [4]. The protocol is modelled in the Algebra of Wireless Networks (AWN), which is the subject of an earlier paper [3] and mechanization [1]. The proof relies on a novel compositional approach for lifting invariants to networks of nodes.

We exploit the mechanization to analyse several variants of AODV and show that Isabelle/HOL can re-establish most proof obligations automatically and identify exactly the steps that are no longer valid. Each of the variants is essentially a modified copy of the main development.

Further documentation is available in [2].
2 Variant B: Forwarding the Route Reply 170
  2.1 Predicates and functions used in the AODV model 170
    2.1.1 Sequence Numbers 170
    2.1.2 Modelling Routes 170
    2.1.3 Routing Tables 171
    2.1.4 Updating Routing Tables 174
    2.1.5 Route Requests 184
    2.1.6 Queued Packets 184
    2.1.7 Route Requests 185
  2.2 AODV protocol messages 185
  2.3 The AODV protocol 186
    2.3.1 Data state 186
    2.3.2 Auxiliary message handling definitions 188
    2.3.3 The protocol process 189
  2.4 Invariant assumptions and properties 195
  2.5 Quality relations between routes 197
    2.5.1 Net sequence numbers 197
    2.5.2 Comparing routes 199
    2.5.3 Comparing routing tables 200
    2.5.4 Strictly comparing routing tables 205
  2.6 Invariant proofs on individual processes 209
  2.7 The quality increases predicate 219
  2.8 The ‘open’ AODV model 226
  2.9 Global invariant proofs over sequential processes 227
  2.10 Routing graphs and loop freedom 245
    2.11 Lift and transfer invariants to show loop freedom 247
      2.11.1 Lift to parallel processes with queues 247
      2.11.2 Lift to nodes 248
      2.11.3 Lift to partial networks 249
      2.11.4 Lift to closed networks 250
      2.11.5 Transfer into the standard model 250
      2.11.6 Loop freedom of AODV 252

3 Variant C: From Groupcast to Broadcast 253
  3.1 Predicates and functions used in the AODV model 253
    3.1.1 Sequence Numbers 253
    3.1.2 Modelling Routes 253
    3.1.3 Routing Tables 254
    3.1.4 Updating Routing Tables 257
3.1.5 Route Requests ................................................................. 265
3.1.6 Queued Packets .............................................................. 265
3.1.7 Comparison with the original technical report ..................... 266
3.2 AODV protocol messages ................................................... 266
3.3 The AODV protocol ............................................................. 267
3.3.1 Data state ................................................................. 267
3.3.2 Auxiliary message handling definitions ......................... 268
3.3.3 The protocol process .................................................... 270
3.4 Invariant assumptions and properties .................................... 275
3.5 Quality relations between routes .......................................... 277
3.5.1 Net sequence numbers ................................................ 277
3.5.2 Comparing routes ........................................................ 279
3.5.3 Comparing routing tables ............................................. 280
3.5.4 Strictly comparing routing tables .................................... 285
3.6 Invariant proofs on individual processes ................................ 289
3.7 The quality increases predicate ........................................... 298
3.8 The ‘open’ AODV model ...................................................... 305
3.9 Global invariant proofs over sequential processes .................... 306
3.10 Routing graphs and loop freedom ........................................ 323
3.11 Lift and transfer invariants to show loop freedom .................. 325
3.11.1 Lift to parallel processes with queues ............................. 325
3.11.2 Lift to nodes ............................................................. 327
3.11.3 Lift to partial networks .............................................. 328
3.11.4 Lift to closed networks .............................................. 329
3.11.5 Transfer into the standard model .................................... 329
3.11.6 Loop freedom of AODV .............................................. 331

4 Variant D: Forwarding the Route Request 332
4.1 Predicates and functions used in the AODV model .................. 332
4.1.1 Sequence Numbers ..................................................... 332
4.1.2 Modelling Routes ....................................................... 332
4.1.3 Routing Tables .......................................................... 333
4.1.4 Updating Routing Tables ............................................. 336
4.1.5 Route Requests ........................................................ 346
4.1.6 Queued Packets ........................................................ 346
4.1.7 Comparison with the original technical report ................... 347
4.2 AODV protocol messages .................................................. 347
4.3 The AODV protocol .......................................................... 348
4.3.1 Data state ............................................................... 348
4.3.2 Auxiliary message handling definitions .......................... 350
4.3.3 The protocol process .................................................. 351
4.4 Invariant assumptions and properties ................................... 357
4.5 Quality relations between routes ........................................ 359
4.5.1 Net sequence numbers .............................................. 359
4.5.2 Comparing routes ..................................................... 361
4.5.3 Comparing routing tables ........................................... 362
4.5.4 Strictly comparing routing tables .................................. 367
4.6 Invariant proofs on individual processes ................................ 371
4.7 The quality increases predicate ......................................... 381
4.8 The ‘open’ AODV model .................................................... 388
4.9 Global invariant proofs over sequential processes .................... 389
4.10 Routing graphs and loop freedom ....................................... 407
4.11 Lift and transfer invariants to show loop freedom .................. 409
4.11.1 Lift to parallel processes with queues ............................. 409
4.11.2 Lift to nodes ............................................................. 410
4.11.3 Lift to partial networks .............................................. 411
4.11.4 Lift to closed networks .............................................. 412
5 Variants A–D: All proposed modifications

5.1 Predicates and functions used in the AODV model
  5.1.1 Sequence Numbers
  5.1.2 Modelling Routes
  5.1.3 Routing Tables
  5.1.4 Updating Routing Tables
  5.1.5 Route Requests
  5.1.6 Queued Packets
  5.1.7 Comparison with the original technical report

5.2 AODV protocol messages

5.3 The AODV protocol
  5.3.1 Data state
  5.3.2 Auxiliary message handling definitions
  5.3.3 The protocol process

5.4 Invariant assumptions and properties

5.5 Quality relations between routes
  5.5.1 Net sequence numbers
  5.5.2 Comparing routes
  5.5.3 Comparing routing tables
  5.5.4 Strictly comparing routing tables

5.6 Invariant proofs on individual processes

5.7 The quality increases predicate

5.8 The ‘open’ AODV model

5.9 Global invariant proofs over sequential processes

5.10 Routing graphs and loop freedom

5.11 Lift and transfer invariants to show loop freedom
  5.11.1 Lift to parallel processes with queues
  5.11.2 Lift to nodes
  5.11.3 Lift to partial networks
  5.11.4 Lift to closed networks
  5.11.5 Transfer into the standard model
  5.11.6 Loop freedom of AODV
0.1 Basic data types and constants

theory Aodv_Basic
imports Main "../AWN/AWN_SOS"
begin

These definitions are shared with all variants.

type_synonym rreqid = nat
type_synonym sqn = nat
datatype k = Known | Unknown
abbreviation kno where "kno ≡ Known"
abbreviation unk where "unk ≡ Unknown"
datatype p = NoRequestRequired | RequestRequired
abbreviation noreq where "noreq ≡ NoRequestRequired"
abbreviation req where "req ≡ RequestRequired"
datatype f = Valid | Invalid
abbreviation val where "val ≡ Valid"
abbreviation inv where "inv ≡ Invalid"

lemma not_ks [simp]:
"(x ≠ kno) = (x = unk)"
"(x ≠ unk) = (x = kno)"
by (cases x,clarsimp+)+

lemma not_ps [simp]:
"(x ≠ noreq) = (x = req)"
"(x ≠ req) = (x = noreq)"
by (cases x,clarsimp+)+

lemma not_ffs [simp]:
"(x ≠ val) = (x = inv)"
"(x ≠ inv) = (x = val)"
by (cases x,clarsimp+)+

end

0.2 Predicates and functions used in the AODV model

theory Aodv_Data
imports Aodv_Basic
begin

0.2.1 Sequence Numbers

Sequence numbers approximate the relative freshness of routing information.

definition inc :: "sqn ⇒ sqn"
  where "inc sn ≡ if sn = 0 then sn else sn + 1"

lemma less_than_inc [simp]: "x ≤ inc x"
  unfolding inc_def by simp

lemma inc_minus_suc_0 [simp]:
  "inc x - Suc 0 = x"
  unfolding inc_def by simp

lemma inc_never_one' [simp, intro]: "inc x ≠ Suc 0"
  unfolding inc_def by simp

lemma inc_never_one [simp, intro]: "inc x ≠ 1"
  by simp
0.2.2 Modelling Routes

A route is a 6-tuple, \((dsn, dsk, flag, hops, nhip, pre)\) where \(dsn\) is the ‘destination sequence number’, \(dsk\) is the ‘destination-sequence-number status’, \(flag\) is the route status, \(hops\) is the number of hops to the destination, \(nhip\) is the next hop toward the destination, and \(pre\) is the set of ‘precursor nodes’ those interested in hearing about changes to the route.

\[
\text{type synonym } r = \text{"sqn} \times k \times f \times \text{nat} \times \text{ip} \times \text{ip set}\]

\[
definition \text{proj2 : } \text{"r} \Rightarrow \text{sqn" } (\"\pi_2\")
\text{where } \pi_2 \equiv \lambda(dsn, _, _, _, _, _). dsn
\]

\[
definition \text{proj3 : } \text{"r} \Rightarrow k" (\"\pi_3\")
\text{where } \pi_3 \equiv \lambda(_, dsk, _, _, _, _). dsk
\]

\[
definition \text{proj4 : } \text{"r} \Rightarrow f" (\"\pi_4\")
\text{where } \pi_4 \equiv \lambda(_, _, flag, _, _, _). flag
\]

\[
definition \text{proj5 : } \text{"r} \Rightarrow \text{nat}\" (\"\pi_5\")
\text{where } \pi_5 \equiv \lambda(_, _, hops, _, _). hops
\]

\[
definition \text{proj6 : } \text{"r} \Rightarrow \text{ip}\" (\"\pi_6\")
\text{where } \pi_6 \equiv \lambda(_, _, _, nhip, _). nhip
\]

\[
definition \text{proj7 : } \text{"r} \Rightarrow \text{ip set}\" (\"\pi_7\")
\text{where } \pi_7 \equiv \lambda(_, _, _, _, _, pre). pre
\]

\[
\text{lemma projs [simp]:}
\pi_2(dsn, dsk, flag, hops, nhip, pre) = dsn
\pi_3(dsn, dsk, flag, hops, nhip, pre) = dsk
\pi_4(dsn, dsk, flag, hops, nhip, pre) = flag
\pi_5(dsn, dsk, flag, hops, nhip, pre) = hops
\pi_6(dsn, dsk, flag, hops, nhip, pre) = nhip
\pi_7(dsn, dsk, flag, hops, nhip, pre) = pre
\text{by (clarsimp simp: proj2_def proj3_def proj4_def proj5_def proj6_def proj7_def)+}
\]

\[
\text{lemma proj3_pred [intro]: } [P \text{ kno}; P \text{ unk}] \implies P (\pi_3 x)
\text{ by (rule k.induct)}
\]

\[
\text{lemma proj4_pred [intro]: } [P \text{ val}; P \text{ inv}] \implies P (\pi_4 x)
\text{ by (rule f.induct)}
\]

\[
\text{lemma proj6_pair_snd [simp]:}
\text{fixes dsn' r}
\text{shows } \pi_6 (dsn', \text{snd}(r)) = \pi_6(r)
\text{ by (cases r) simp}
\]

0.2.3 Routing Tables

Routing tables map ip addresses to route entries.

\[
\text{type synonym } rt = \text{"ip} \Rightarrow r"
\]

\[
\text{syntax}
\"_Sigma_route" :: \text{"rt} \Rightarrow \text{ip} \Rightarrow r\" (\"\sigma_{route}(\_, \_')\")
\]

\[
\text{translations}
\"\sigma_{route}(rt, dip)\" \Rightarrow \text{"rt dip"}
\]

\[
definition \text{sqn : } \text{"rt} \Rightarrow \text{ip} \Rightarrow \text{sqn}\" 
\text{where } \text{"sqn rt dip \equiv case } \sigma_{route}(\text{rt, dip}) \text{ of Some } r \Rightarrow \pi_2(r) \mid \text{None } \Rightarrow 0"
\]

\[
definition \text{sqnf : } \text{"rt} \Rightarrow \text{ip} \Rightarrow k"
\text{where } \text{"sqnf rt dip \equiv case } \sigma_{route}(\text{rt, dip}) \text{ of Some } r \Rightarrow \pi_3(r) \mid \text{None } \Rightarrow \text{unk}"
\]
abbreviation flag :: "rt ⇒ ip ⇒ f"
  where "flag rt dip ≡ map_option π₄ (σ_route(rt, dip))"

abbreviation dhops :: "rt ⇒ ip ⇒ nat"
  where "dhops rt dip ≡ map_option π₅ (σ_route(rt, dip))"

abbreviation nhop :: "rt ⇒ ip ⇒ ip"
  where "nhop rt dip ≡ map_option π₆ (σ_route(rt, dip))"

abbreviation precs :: "rt ⇒ ip ⇒ ip set"
  where "precs rt dip ≡ map_option π₇ (σ_route(rt, dip))"

definition vD :: "rt ⇒ ip set"
  where "vD rt ≡ {dip. flag rt dip = Some val}"

definition iD :: "rt ⇒ ip set"
  where "iD rt ≡ {dip. flag rt dip = Some inv}"

definition kD :: "rt ⇒ ip set"
  where "kD rt ≡ {dip. rt dip ≠ None}"

lemma kD_is_vD_and_iD: "kD rt = vD rt ∪ iD rt"
  unfolding kD_def vD_def iD_def by auto

lemma vD_iD_gives_kD [simp]:
  "∀ ip rt. ip ∈ vD rt ⇒ ip ∈ kD rt"
  "∀ ip rt. ip ∈ iD rt ⇒ ip ∈ kD rt"
  unfolding kD_is_vD_and_iD by simp_all

lemma kD_Some [dest]:
  fixes dip rt
  assumes "dip ∈ kD rt"
  shows "∃ dsn dsk flag hops nhip pre.
      σ_route(rt, dip) = Some (dsn, dsk, flag, hops, nhip, pre)"
  using assms unfolding kD_def by simp

lemma kD_None [dest]:
  fixes dip rt
  assumes "dip ∉ kD rt"
  shows "σ_route(rt, dip) = None"
  using assms unfolding kD_def
  by (metis (mono_tags) mem_Collect_eq)

lemma vD_Some [dest]:
  fixes dip rt
  assumes "dip ∈ vD rt"
  shows "∃ dsn dsk hops nhip pre.
      σ_route(rt, dip) = Some (dsn, dsk, val, hops, nhip, pre)"
  using assms unfolding vD_def by simp

lemma vD_empty [simp]: "vD Map.empty = {}"
  unfolding vD_def by simp

lemma iD_Some [dest]:
  fixes dip rt
  assumes "dip ∈ iD rt"
  shows "∃ dsn dsk hops nhip pre.
      σ_route(rt, dip) = Some (dsn, dsk, inv, hops, nhip, pre)"
  using assms unfolding iD_def by simp

lemma val_is_vD [elim]:
  fixes ip rt
  assumes "ip ∈ kD(rt)"
and "the (flag rt ip) = val"
shows "ip\in\upsilon D(rt)"
using assms unfolding \upsilon D_def by auto

lemma inv_is_iD [elim]:
  fixes ip rt
  assumes "ip\in kD(rt)"
  and "the (flag rt ip) = inv"
  shows "ip\in iD(rt)"
  using assms unfolding iD_def by auto

lemma iD_flag_is_inv [elim, simp]:
  fixes ip rt
  assumes "ip\in iD(rt)"
  shows "the (flag rt ip) = inv"
proof -
  from \langle ip\in iD(rt) \rangle have "ip\in kD(rt)" by auto
  with assms show \?thesis unfolding iD_def by auto
  qed

lemma kD_but_not_vD_is_iD [elim]:
  fixes ip rt
  assumes "ip\in kD(rt)"
  and "ip/\in vD(rt)"
  shows "ip\in iD(rt)"
proof -
  from \langle ip\in kD(rt) \rangle obtain dsn dsk f hops nhop pre
    where rtip: "rt ip = Some (dsn, dsk, f, hops, nhop, pre)"
      by (metis kD_Some)
  from \langle ip/\in vD(rt) \rangle have "f \neq val"
    proof (rule contrapos_nn)
      assume "f = val"
      with rtip have "the (flag rt ip) = val" by simp
      with \langle ip\in kD(rt) \rangle show "ip\in vD(rt)" ..
      qed
  with rtip have "the (flag rt ip)= inv" by simp
  with \langle ip\in kD(rt) \rangle show "ip\in iD(rt)" ..
  qed

lemma vD_or_iD [elim]:
  fixes ip rt
  assumes "ip\in kD(rt)"
  and "ip\in vD(rt) = \Rightarrow P rt ip"
  and "ip\in iD(rt) = \Rightarrow P rt ip"
  shows "P rt ip"
proof -
  from \langle ip\in kD(rt) \rangle have "ip\in vD(rt) \cup iD(rt)"
    by (simp add: kD_is_vD_and_iD)
  thus \?thesis by (auto elim: assms(2-3))
  qed

lemma proj5_eq_dhops: "\lambda dip rt. dip\in kD(rt) = \Rightarrow \pi_5 (the (rt dip)) = the (dhops rt dip)"
unfolding sqn_def by (drule kD_Some) clarsimp

lemma proj4_eq_flag: "\lambda dip rt. dip\in kD(rt) = \Rightarrow \pi_4 (the (rt dip)) = the (flag rt dip)"
unfolding sqn_def by (drule kD_Some) clarsimp

lemma proj2_eq_sqn: "\lambda dip rt. dip\in kD(rt) = \Rightarrow \pi_2 (the (rt dip)) = sqn rt dip"
unfolding sqn_def by (drule kD_Some) clarsimp

lemma kD_sqnf_is_proj3 [simp]:
  "\lambda ip rt. ip\in kD(rt) = \Rightarrow sqnf rt ip = \pi_3 (the (rt ip))"
unfolding sqnf_def by auto
lemma vD_flag_val [simp]: "\(\forall dip rt. dip \in vD (rt) \implies \text{the (flag rt dip)} = \text{val}\)"
unfolding vD_def by clarsimp

lemma kD_update [simp]: "\(\forall nip v. kD (rt(nip \mapsto v)) = \text{insert nip (kD rt)}\)"
unfolding kD_def by auto

lemma kD_empty [simp]: "kD Map.empty = {}"
unfolding kD_def by simp

lemma ip_equal_or_known [elim]:
  fixes rt ip ip'
  assumes "ip = ip' \lor ip \in kD(rt)"
  and "ip = ip' \implies P rt ip ip'"
  and "[ [ ip \neq ip'; ip \in kD(rt) ] ] \implies P rt ip ip'"
  shows "P rt ip ip'"
  using assms by auto

0.2.4 Updating Routing Tables
Routing table entries are modified through explicit functions. The properties of these functions are important in invariant proofs.

Updating Precursor Lists

definition addpre :: "r \Rightarrow ip set \Rightarrow r"
  where "addpre r npre \equiv \text{let (dsn, dsk, flag, hops, nhip, pre) = r in}
  (dsn, dsk, flag, hops, nhip, pre \cup npre)"

lemma proj2_addpre:
  fixes v pre
  shows "\(\pi_2\) (addpre v pre) = \(\pi_2\) (v)"
  unfolding addpre_def by (cases v) simp

lemma proj3_addpre:
  fixes v pre
  shows "\(\pi_3\) (addpre v pre) = \(\pi_3\) (v)"
  unfolding addpre_def by (cases v) simp

lemma proj4_addpre:
  fixes v pre
  shows "\(\pi_4\) (addpre v pre) = \(\pi_4\) (v)"
  unfolding addpre_def by (cases v) simp

lemma proj5_addpre:
  fixes v pre
  shows "\(\pi_5\) (addpre v pre) = \(\pi_5\) (v)"
  unfolding addpre_def by (cases v) simp

lemma proj6_addpre:
  fixes dsn dsk flag hops nhip pre npre
  shows "\(\pi_6\) (addpre v npre) = \(\pi_6\) (v)"
  unfolding addpre_def by (cases v) simp

lemma proj7_addpre:
  fixes dsn dsk flag hops nhip pre npre
  shows "\(\pi_7\) (addpre v npre) = \(\pi_7\) (v) \cup npre"
  unfolding addpre_def by (cases v) simp

lemma addpre_empty: "addpre r {} = r"
  unfolding addpre_def by simp

lemma addpre_r:
"addpre (dsn, dsk, fl, hops, nhip, pre) npre = (dsn, dsk, fl, hops, nhip, pre ∪ npre)"
unfolding addpre_def by simp

lemmas addpre_simps [simp] = proj2_addpre proj3_addpre proj4_addpre proj5_addpre proj6_addpre proj7_addpre addpre_empty addpre_r

definition addpreRT :: "rt ⇒ ip ⇒ ip set ⇀ rt"
where "addpreRT rt dip npre ≡
map_option (λs. rt (dip ↦→ addpre s npre)) (σroute(rt, dip))"

lemma snd_addpre [simp]:
"⋀dsn dsn' v pre. (dsn, snd(addpre (dsn', v) pre)) = addpre (dsn, v) pre"
unfolding addpre_def by clarsimp

lemma proj2_addpreRT [simp]:
fixes ip rt ip' npre
assumes "ip∈kD rt"
and "ip'∈kD rt"
shows "π2(the (the (addpreRT rt ip' npre) ip)) = π2(the (rt ip))"
using assms [THEN kD_Some] unfolding addpreRT_def by clarsimp

lemma proj3_addpreRT [simp]:
fixes ip rt ip' npre
assumes "ip∈kD rt"
and "ip'∈kD rt"
shows "π3(the (the (addpreRT rt ip' npre) ip)) = π3(the (rt ip))"
using assms [THEN kD_Some] unfolding addpreRT_def by clarsimp

lemma proj5_addpreRT [simp]:
"⋀rt dip ip npre. dip∈kD(rt) =⇒ π5(the (the (addpreRT rt dip npre) ip)) = π5(the (rt ip))"
unfolding addpreRT_def by auto

lemma flag_addpreRT [simp]:
fixes rt pre ip dip
assumes "dip ∈ kD rt"
shows "flag (the (addpreRT rt dip pre)) ip = flag rt ip"
unfolding addpreRT_def
using assms [THEN kD_Some] by (clarsimp)

lemma kD_addpreRT [simp]:
fixes rt dip npre
assumes "dip ∈ kD rt"
shows "kD (the (addpreRT rt dip npre)) = kD rt"
unfolding kD_def addpreRT_def
using assms [THEN kD_Some] by clarsimp blast

lemma vD_addpreRT [simp]:
fixes rt dip npre
assumes "dip ∈ kD rt"
shows "vD (the (addpreRT rt dip npre)) = vD rt"
unfolding vD_def addpreRT_def
using assms [THEN kD_Some] by clarsimp auto

lemma iD_addpreRT [simp]:
fixes rt dip npre
assumes "dip ∈ kD rt"
shows "iD (the (addpreRT rt dip npre)) = iD rt"
unfolding iD_def addpreRT_def
using assms [THEN kD_Some] by clarsimp auto

lemma nhop_addpreRT [simp]:
fixes rt pre ip dip
assumes "dip ∈ kD rt"
shows "nhop (the (addpreRT rt dip pre)) ip = nhop rt ip"
unfolding sqn_def addpreRT_def
using assms [THEN kD_Some] by (clarsimp)

lemma sqn_addpreRT [simp]:
  fixes rt pre ip dip
  assumes "dip ∈ kD rt"
  shows "sqn (the (addpreRT rt dip pre)) ip = sqn rt ip"
unfolding sqn_def addpreRT_def
using assms [THEN kD_Some] by (clarsimp)

lemma dhops_addpreRT [simp]:
  fixes rt pre ip dip
  assumes "dip ∈ kD rt"
  shows "dhops (the (addpreRT rt dip pre)) ip = dhops rt ip"
unfolding addpreRT_def
using assms [THEN kD_Some] by (clarsimp)

lemma sqnf_addpreRT [simp]:
  "⋀ip dip. ip ∈ kD(rt ξ) =⇒ sqnf (the (addpreRT (rt ξ) ip npre)) dip = sqnf (rt ξ) dip"
unfolding sqnf_def addpreRT_def by auto

Updating route entries

lemma in_kD_case [simp]:
  fixes dip rt
  assumes "dip ∈ kD(rt)"
  shows "(case rt dip of None ⇒ en | Some r ⇒ es r) = es (the (rt dip))"
using assms [THEN kD_Some] by auto

lemma not_in_kD_case [simp]:
  fixes dip rt
  assumes "dip /∈ kD(rt)"
  shows "(case rt dip of None ⇒ en | Some r ⇒ es r) = en"
using assms [THEN kD_None] by auto

lemma rt_Some_sqn [dest]:
  fixes rt and ip dsn dsk flag hops nhip pre
  assumes "rt ip = Some (dsn, dsk, flag, hops, nhip, pre)"
  shows "sqn rt ip = dsn"
unfolding sqn_def using assms by simp

lemma not_kD_sqn [simp]:
  fixes dip rt
  assumes "dip /∈ kD(rt)"
  shows "sqn rt dip = 0"
using assms unfolding sqn_def by simp

definition update_arg_wf :: "r ⇒ bool"
where "update_arg_wf r ≡ π_4 r = val ∧
  (π_2 r = 0) = (π_3 r = unk) ∧
  (π_3 r = unk → π_5 r = 1)"

lemma update_arg_wf_gives_cases:
  "⋀r. update_arg_wf r =⇒ (π_2 r = 0) = (π_3 r = unk)"
unfolding update_arg_wf_def by simp

lemma update_arg_wf_tuples [simp]:
  "⋀nhip pre. update_arg_wf (0, unk, val, Suc 0, nhip, pre)"
  "⋀n hops nhip pre. update_arg_wf (Suc n, kno, val, hops, nhip, pre)"
unfolding update_arg_wf_def by auto

lemma update_arg_wf_tuples' [elim]:

lemma wf_r_cases [intro]:
  fixes P r
  assumes "update_arg_wf r"
  and c1: "n hops nhip pre. P (0, unk, val, Suc 0, nhip, pre)"
  and c2: "dsn hops nhip pre. dsn > 0 ⇒ P (dsn, kno, val, hops, nhip, pre)"
  shows "P r"
proof -
  obtain dsn dsk flag hops nhip pre
    where *: "r = (dsn, dsk, flag, hops, nhip, pre)"
    by (cases r)
  with ⟨⟨ update_arg_wf r ⟩⟩
    have wf1: "flag = val"
    and wf2: "dsn = 0 → dsk = unk"
    and wf3: "dsn > 0 → (hops = Suc 0)"
    unfolding update_arg_wf_def
    by (auto)
  have "P (dsn, dsk, flag, hops, nhip, pre)"
    proof (cases dsk)
      assume "dsk = unk"
      moreover with wf2 wf3
      have "dsn = 0" and "hops = Suc 0"
        by (auto)
      ultimately show "P (dsn, dsk, flag, hops, nhip, pre)"
        using ⟨⟨ flag = val ⟩⟩
        by (simp (rule c1))
    next
      assume "dsk = kno"
      moreover with wf2
      have "dsn > 0"
        by (simp)
      ultimately show "P (dsn, dsk, flag, hops, nhip, pre)"
        by (simp (rule c2))
    qed
  with * show "P r" by simp
qed

definition update :: "rt ⇒ ip ⇒ r ⇒ rt"
  where "update rt ip r ≡
    case σroute(rt, ip) of
      None ⇒ rt (ip ↦ r)
    | Some s ⇒
      if π2(s) < π2(r) then rt (ip ↦ addpre r (π7(s)))
      else if π2(s) = π2(r) ∧ (π5(s) > π5(r) ∨ π4(s) = inv)
        then rt (ip ↦ addpre r (π7(s)))
      else if π3(r) = unk
        then rt (ip ↦ (π2(s), snd (addpre r (π7(s)))))
      else rt (ip ↦ addpre s (π7(r)))"

lemma update_simps [simp]:
  fixes r s nrt nr nr' ns rt ip
  defines "s ≡ the σroute(rt, ip)"
  and "nr ≡ addpre r (π7(s))"
  and "nr' ≡ (π3(s), π3(nr), π4(nr), π5(nr), π6(nr), π7(nr))"
  and "ns ≡ addpre s (π7(r))"
  shows
    "[ip /∈ kD(rt)]" ⇒ update rt ip r = rt (ip ↦ r)"
    "[ip ∈ kD(rt); sqn rt ip < π2(r)]" ⇒ update rt ip r = rt (ip ↦ nr)"
    "[ip ∈ kD(rt); sqn rt ip = π2(r);
      the (dhops rt ip) > π5(r)]" ⇒ update rt ip r = rt (ip ↦ nr)"
    "[ip ∈ kD(rt); sqn rt ip = π2(r);
      flag rt ip = Some inv]" ⇒ update rt ip r = rt (ip ↦ nr)"
    "[ip ∈ kD(rt); π3(r) = unk; (π2(r) = 0) = (π3(r) = unk)]" ⇒ update rt ip r = rt (ip ↦ nr')"
    "[ip ∈ kD(rt); sqn rt ip ≥ π2(r); π3(r) = kno;
      sqn rt ip = π2(r) ⇒ the (dhops rt ip) ≤ π5(r) ∧ the (flag rt ip) = val ]
      ⇒ update rt ip r = rt (ip ↦ ns)
    "
proof -
  assume "ip /∈ kD(rt)"
  hence "σroute(rt, ip) = None" ..
  thus "update rt ip r = rt (ip ↦ r)"
    unfolding update_def by simp
next

assume "ip ∈ kD(rt)"
  and "sqn rt ip < π_2(r)"
from this obtain dsn dsk fl hops nhip pre
  where "rt ip = Some (dsn, dsk, fl, hops, nhip, pre)"
  by (metis kD_Some)

with (sqn rt ip < π_2(r)) show "update rt ip r = rt (ip ↦→ nr)"
  unfolding update_def nr_def s_def by auto

defined

next

assume "ip ∈ kD(rt)"
  and "sqn rt ip = π_2(r)"
from this obtain dsn dsk fl hops nhip pre
  where "rt ip = Some (dsn, dsk, fl, hops, nhip, pre)"
  by (metis kD_Some)

with (sqn rt ip = π_2(r)) show "update rt ip r = rt (ip ↦→ nr)"
  unfolding update_def nr_def s_def by auto

defined

next

assume "ip ∈ kD(rt)"
  and "sqn rt ip = π_2(r)"
from this obtain dsn dsk fl hops nhip pre
  where "rt ip = Some (dsn, dsk, fl, hops, nhip, pre)"
  by (metis kD_Some)

with (sqn rt ip = π_2(r)) show "update rt ip r = rt (ip ↦→ nr)"
  unfolding update_def nr_def s_def by auto

defined

next

assume "ip ∈ kD(rt)"
  and "(π_3(r) = 0) = (π_3(r) = unk)"
from this obtain dsn dsk fl hops nhip pre
  where "rt ip = Some (dsn, dsk, fl, hops, nhip, pre)"
  by (metis kD_Some)

with ((π_2(r) = 0) = (π_3(r) = unk)) show "update rt ip r = rt (ip ↦→ nr)"
  unfolding update_def nr_def s_def by auto

defined

next

assume "ip ∈ kD(rt)"
  and otherassms: "sqn rt ip ≥ π_2(r)"
  "π_3(r) = kno"
  "sqn rt ip = π_2(r)"  the (dhops rt ip) ≤ π_5(r) ∧ the (flag rt ip) = val"
from this obtain dsn dsk fl hops nhip pre
  where "rt ip = Some (dsn, dsk, fl, hops, nhip, pre)"
  by (metis kD_Some)

with otherassms show "update rt ip r = rt (ip ↦→ ns)"
  unfolding update_def ns_def s_def by auto

defined

lemma update_cases [elim]:
  assumes "(π_2(r) = 0) = (π_3(r) = unk)"
  and c1: "[| ip ∉ kD(rt) |] =⇒ P (rt (ip ↦→ r))"
  and c2: "[| ip ∈ kD(rt); sqn rt ip < π_2(r) |] =⇒ P (rt (ip ↦→ addpre r (π_7(the σroute(rt, ip))))))"
  and c3: "[| ip ∈ kD(rt); sqn rt ip = π_2(r); the (dhops rt ip) > π_5(r) |] =⇒ P (rt (ip ↦→ addpre r (π_7(the σroute(rt, ip))))))"
  and c4: "[| ip ∈ kD(rt); sqn rt ip = π_2(r); the (flag rt ip) = inv |] =⇒ P (rt (ip ↦→ addpre r (π_7(the σroute(rt, ip))))))"
  and c5: "[| ip ∈ kD(rt); π_3(r) = unk |] =⇒ P (rt (ip ↦→ (π_7(the σroute(rt, ip)), π_3(r), π_4(r), π_5(r), π_6(r), π_7(addpre r (π_7(the σroute(rt, ip)))))))}"

qed
and c6: "[ip ∈ kD(rt); sqn rt ip ≥ π_2(r); π_3(r) = kno;
sqn rt ip = π_3(r) \implies \text{the (dhops rt ip) ≤ π}_5(r) \land \text{the (flag rt ip) = val]}
⇒ P (rt (ip ↦ addpre (the σ_route(rt, ip)) (π_7(r))))"
shows "(P (update rt ip r))"
proof (cases "ip ∈ kD(rt)")
assume "ip ∉ kD(rt)"
with c1 show ?thesis
by simp
next
assume "ip ∈ kD(rt)"
moreover then obtain dsn dsk fl hops nhip pre
where rteq: "rt ip = Some (dsn, dsk, fl, hops, nhip, pre)"
by (metis kD_Some)
moreover obtain dsn' dsk' fl' hops' nhip' pre'
where req: "r = (dsn’, dsk’, fl’, hops’, nhip’, pre’)"
by (cases r) metis
ultimately show ?thesis
using ⟨(π_2(r) = 0) = (π_3(r) = unk)⟩
c2 [OF ⟨ip ∈ kD(rt)⟩]
c3 [OF ⟨ip ∈ kD(rt)⟩]
c4 [OF ⟨ip ∈ kD(rt)⟩]
c5 [OF ⟨ip ∈ kD(rt)⟩]
c6 [OF ⟨ip ∈ kD(rt)⟩]
unfolding update_def sqn_def by auto
qed

lemma update_cases_kD:
assumes "(π_2(r) = 0) = (π_3(r) = unk)"
and "ip ∈ kD(rt)"
and c2: "sqn rt ip < π_2(r) \implies P (rt (ip ↦ addpre r (π_7(the σ_route(rt, ip)))))"
and c3: "[sqn rt ip = π_2(r); the (dhops rt ip) > π_5(r)]
⇒ P (rt (ip ↦ addpre r (π_7(the σ_route(rt, ip)))))
and c4: "[sqn rt ip = π_2(r); the (flag rt ip) = inv]
⇒ P (rt (ip ↦ addpre r (π_7(the σ_route(rt, ip)))))
and c5: "π_3(r) = unk \implies P (rt (ip ↦ (π_2(the σ_route(rt, ip)), π_3(r), π_4(r), π_5(r), π_6(r),
π_7(addpre r (π_7(the σ_route(rt, ip)))))))"
and c6: "[sqn rt ip ≥ π_2(r); π_3(r) = kno;
sqn rt ip = π_3(r) \implies \text{the (dhops rt ip) ≤ π}_5(r) \land \text{the (flag rt ip) = val]}
⇒ P (rt (ip ↦ addpre (the σ_route(rt, ip)) (π_7(r))))"
shows "(P (update rt ip r))"
using assms(1) proof (rule update_cases)
assume "sqn rt ip < π_2(r)"
thus "P (rt(ip ↦ addpre r (π_7(the (rt ip)))))" by (rule c2)
next
assume "sqn rt ip = π_2(r)"
and "the (dhops rt ip) > π_5(r)"
thus "P (rt(ip ↦ addpre r (π_7(the (rt ip)))))" by (rule c3)
next
assume "sqn rt ip = π_3(r)"
and "the (flag rt ip) = inv"
thus "P (rt(ip ↦ addpre r (π_7(the (rt ip)))))" by (rule c4)
next
assume "π_3(r) = unk"
thus "P (rt (ip ↦ (π_2(the σ_route(rt, ip)), π_3(r), π_4(r), π_5(r), π_6(r),
π_7(addpre r (π_7(the (rt ip)))))))"
by (rule c5)
next
assume "sqn rt ip ≥ π_2(r)"
and "π_3(r) = kno"
and "sqn rt ip = π_3(r) \implies \text{the (dhops rt ip) ≤ π}_5(r) \land \text{the (flag rt ip) = val]}
thus "P (rt (ip ↦ addpre (the (rt ip)) (π_7(r))))"
by (rule c6)
qed (simp add: ip ∈ kD(rt))

lemma in_kD_after_update [simp]:
  fixes rt nip dsn dsk flag hops nhip pre
  shows "kD (update rt nip (dsn, dsk, flag, hops, nhip, pre)) = insert nip (kD rt)"
  unfolding update_def
  by (cases "rt nip") auto

lemma nhop_of_update [simp]:
  fixes rt dip dsn dsk flag hops nhip
  assumes "rt ≠ update rt dip (dsn, dsk, flag, hops, nhip, {})"
  shows "the (nhop (update rt dip (dsn, dsk, flag, hops, nhip, {})) dip) = nhip"
  proof -
    from assms have update_neq: "∀ v. rt dip = Some v ⇒ update rt dip (dsn, dsk, flag, hops, nhip, {}) ≠ rt(dip ↦→ addpre (the (rt dip)) (π7 (dsn, dsk, flag, hops, nhip, {})))" by auto
    show ?thesis proof (cases "rt dip = None")
      assume "rt dip = None"
      thus ?thesis unfolding update_def by clarsimp
    next
      assume "rt dip ≠ None"
      then obtain v where "rt dip = Some v" by (metis not_None_eq)
      with update_neq [OF this] unfolding update_def by auto
    qed
  qed

lemma sqn_if_updated:
  fixes rip v rt ip
  shows "sqn (λ x. if x = rip then Some v else rt x) ip = (if ip = rip then π2(v) else sqn rt ip)"
  unfolding sqn_def by simp

lemma update_sqn [simp]:
  fixes rt dip rip dsn dsk hops nhip pre
  assumes "(dsn = 0) = (dsk = unk)"
  shows "sqn rt dip ≤ sqn (update rt rip (dsn, dsk, val, hops, nhip, pre)) dip"
  proof (rule update_cases)
    show "(π2 (dsn, dsk, val, hops, nhip, pre) = 0) = (π3 (dsn, dsk, val, hops, nhip, pre) = unk)" by simp (rule assms)
  qed (clarsimp simp: sqn_if_updated sqn_def)+

lemma sqn_update_bigger [simp]:
  fixes rt ip ip' dsn dsk flag hops nhip pre
  assumes "1 ≤ hops"
  shows "sqn rt ip ≤ sqn (update rt ip' (dsn, dsk, flag, hops, nhip, pre)) ip"
  using assms unfolding update_def sqn_def
  by (clarsimp split: option.split) auto

lemma dhops_update [intro]:
  fixes rt dsn dsk flag hops ip rip nhip pre
  assumes "1 ≤ hops"
  shows "∀ ip∈kD rt. the (dhops rt ip) ≥ 1"
  using ip proof
    assume "ip = rip ∧ Suc 0 ≤ hops" thus ?thesis
    unfolding update_def using ex
    by (cases "rip ∈ kD rt") (drule(1) bspec, auto)
  next
    assume "ip ≠ rip ∧ ip∈kD rt" thus ?thesis
lemma update_another [simp]:
  fixes dip ip rt dsn dsk flag hops nhip pre
  assumes "ip ≠ dip"
  shows "(update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = rt ip"
  using assms unfolding update_def
  by (clarsimp split: option.split)

lemma nhop_update_another [simp]:
  fixes dip ip rt dsn dsk flag hops nhip pre
  assumes "ip ≠ dip"
  shows "nhop (update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = nhop rt ip"
  using assms unfolding update_def
  by (clarsimp split: option.split)

lemma dhops_update_another [simp]:
  fixes dip ip rt dsn dsk flag hops nhip pre
  assumes "ip ≠ dip"
  shows "dhops (update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = dhops rt ip"
  using assms unfolding update_def
  by (clarsimp split: option.split)

lemma sqn_update_same [simp]:
  "∀ rt ip dsn dsk flag hops nhip pre. sqn (rt(ip ↦ v)) ip = π₂(v)"
  unfolding sqn_def
  by simp

lemma dhops_update_changed [simp]:
  fixes rt dip osn hops nhip
  assumes "rt ≠ update rt dip (osn, kno, val, hops, nhip, {})"
  shows "the (dhops (update rt dip (osn, kno, val, hops, nhip, {})) dip) = hops"
  using assms unfolding update_def
  by (clarsimp split: option.split_asm option.split split_if_asm) auto

lemma nhop_update_unk_val [simp]:
  "∀ rt dip ip dsn hops npre. the (nhop (update rt dip (dsn, unk, val, hops, ip, npre)) dip) = ip"
  unfolding update_def
  by (clarsimp split: option.split)

lemma nhop_update_changed [simp]:
  fixes rt dip dsn dsk flg hops sip
  assumes "update rt dip (dsn, dsk, flg, hops, sip, {}) ≠ rt"
  shows "the (nhop (update rt dip (dsn, dsk, flg, hops, sip, {})) dip) = sip"
  using assms unfolding update_def
  by (clarsimp split: option.splits split_if_asm) auto

lemma update_rt_split_asm:
  "∀ rt ip dsn dsk flag hops sip. P (update rt ip (dsn, dsk, flag, hops, sip, {}))
  = (¬ (rt = update rt ip (dsn, dsk, flag, hops, sip, {})) ∧ ¬ P rt
  ∨ rt ≠ update rt ip (dsn, dsk, flag, hops, sip, {})
  ∧ ¬ P (update rt ip (dsn, dsk, flag, hops, sip, {}))))"
  by auto

lemma sqn_update [simp]: "∀ rt dip dsn dsk flag hops sip.
  rt ≠ update rt dip (dsn, kno, flg, hops, sip, {})
  ⇒ sqn (update rt dip (dsn, kno, flg, hops, sip, {})) dip = dsn"
  unfolding update_def
  by (clarsimp split: option.split split_if_asm) auto

lemma sqnf_update [simp]: "∀ rt dip dsn dsk flag hops sip.
  rt ≠ update rt dip (dsn, dsk, flg, hops, sip, {})"
\[ \Rightarrow \text{sqnf (update rt dip (dsn, dsk, flg, hops, sip, {})) dip = dsk} \]

unfolding update_def sqnf_def
by (clarsimp split: option.splits split_if_asm) auto

lemma update_kno_dsn_greater_zero:
"\forall rt dip ip dsn hops npre. 1 \leq dsn \Rightarrow 1 \leq \text{(sqn (update rt dip (dsn, kno, val, hops, ip, npre)) dip)}"
unfolding update_def
by (clarsimp split: option.splits)

lemma proj3_update [simp]: "\forall rt dip dsn dsk flg hops sip.
rt \neq \text{update rt dip (dsn, dsk, flg, hops, sip, {})}
\Rightarrow \pi_3(\text{(update rt dip (dsn, dsk, flg, hops, sip, {})) dip}) = dsk"
unfolding update_def sqnf_def
by (clarsimp split: option.splits split_if_asm) auto

lemma nhop_update_changed_kno_val [simp]: "\forall rt ip dsn dsk hops nhip.
rt \neq \text{update rt ip (dsn, kno, val, hops, nhip, {})}
\Rightarrow \text{the(nhop (update rt ip (dsn, kno, val, hops, nhip, {})) ip}) = nhip"
using assms unfolding update_def
by (clarsimp split: option.split_asm option.split split_if_asm) auto

lemma flag_update [simp]: "\forall rt dip dsn dsk flg hops sip.
rt \neq \text{update rt dip (dsn, kno, flg, hops, sip, {})}
\Rightarrow \text{the(flag (update rt dip (dsn, kno, flg, hops, sip, {})) dip}) = flg"
unfolding update_def
by (clarsimp split: option.split split_if_asm) auto

lemma the_flag_Some [dest!]:
fixes ip rt
assumes "the (flag rt ip) = x"
and "ip \in kD rt"
shows "flag rt ip = Some x"
using assms by auto

lemma kD_update_unchanged [dest]:
fixes rt dip dsn dsk flag hops nhip pre
assumes "rt = \text{update rt dip (dsn, dsk, flag, hops, nhip, pre)}"
shows "dip \in kD(rt)"
proof -
have "dip \in kD(update rt dip (dsn, dsk, flag, hops, nhip, pre))" by simp
with assms show ?thesis by simp
qed

lemma nhop_update [simp]: "\forall rt dip dsn dsk flg hops sip.
rt \neq \text{update rt dip (dsn, dsk, flg, hops, sip, {})}
\Rightarrow \text{the(nhop (update rt dip (dsn, dsk, flg, hops, sip, {})) dip}) = sip"
unfolding update_def sqnf_def
by (clarsimp split: option.splits split_if_asm) auto

lemma sqn_update_another [simp]:
fixes dip ip rt dsn dsk flag hops nhip pre
assumes "ip \neq dip"
shows "sqn (update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = sqn rt ip"
using assms unfolding update_def sqn_def
by (clarsimp split: option.splits) auto

lemma sqnf_update_another [simp]:
fixes dip ip rt dsn dsk flag hops nhip pre
assumes "ip \neq dip"
shows "sqnf (update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = sqnf rt ip"
using assms unfolding update_def sqnf_def
by (clarsimp split: option.splits) auto

lemma vD_update_val [dest]:
"\( \text{dip} \in vD(\text{update} \ \text{dip}' \ (\text{dsn}, \text{dsk}, \text{val}, \text{hops}, \text{nhip}, \text{pre})) \implies (\text{dip} \in vD(\text{rt}) \lor \text{dip} = \text{dip}') \)"

unfolding \text{update}\_def \ vD\_def by (clarsimp split: option.split\_asm split\_if\_asm)

Invalidating route entries

definition invalidate :: "\( \text{rt} \implies (ip \mapsto \text{sqn}) \implies \text{rt} \)"

where "\( \text{invalidate} \ \text{rt} \ \text{dests} \equiv \lambda \text{ip}. \ \text{case} \ (\text{rt} \ \text{ip}, \ \text{dests} \ \text{ip}) \ \text{of} \)
\( \text{None}, \ _) \ \Rightarrow \ \text{None} \)
\( \text{Some} \ \text{s}, \ \text{None} \ \Rightarrow \ \text{Some} \ \text{s} \)
\( \text{Some} \ (\_, \ \text{dsk}, \ _, \ \text{hops}, \ \text{nhip}, \ \text{pre}), \ \text{Some} \ \text{rsn} \ \Rightarrow \)
\( \text{Some} \ (\text{rsn}, \ \text{dsk}, \ \text{inv}, \ \text{hops}, \ \text{nhip}, \ \text{pre}) \)"

lemma proj3_invalidate [simp]:
"\( \lambda \text{dip}. \ \pi_3(\text{the} \ (\text{invalidate} \ \text{rt} \ \text{dests} \ \text{dip})) = \pi_3(\text{the} \ (\text{rt} \ \text{dip})) \)"

unfolding invalidate\_def by (clarsimp split: option.split)

lemma proj5_invalidate [simp]:
"\( \lambda \text{dip}. \ \pi_5(\text{the} \ (\text{invalidate} \ \text{rt} \ \text{dests} \ \text{dip})) = \pi_5(\text{the} \ (\text{rt} \ \text{dip})) \)"

unfolding invalidate\_def by (clarsimp split: option.split)

lemma proj6_invalidate [simp]:
"\( \lambda \text{dip}. \ \pi_6(\text{the} \ (\text{invalidate} \ \text{rt} \ \text{dests} \ \text{dip})) = \pi_6(\text{the} \ (\text{rt} \ \text{dip})) \)"

unfolding invalidate\_def by (clarsimp split: option.split)

lemma proj7_invalidate [simp]:
"\( \lambda \text{dip}. \ \pi_7(\text{the} \ (\text{invalidate} \ \text{rt} \ \text{dests} \ \text{dip})) = \pi_7(\text{the} \ (\text{rt} \ \text{dip})) \)"

unfolding invalidate\_def by (clarsimp split: option.split)

lemma invalidate\_kD\_inv [simp]:
"\( \\text{inject} \ (\text{invalidate} \ \text{rt} \ \text{dests}) = \text{kD} \ \text{rt} \)"

unfolding invalidate\_def \ kD\_def by (simp split: option.split)

lemma invalidate\_sqn:
fixes \text{rt} \ \text{dip} \ \text{dests}

assumes "\( \forall \text{rsn}. \ \text{dests} \ \text{dip} = \text{Some} \ \text{rsn} \implies \text{sqn} \ \text{rt} \ \text{dip} \leq \text{rsn} \)"

shows "\( \text{sqn} \ \text{rt} \ \text{dip} \leq \text{sqn} \ (\text{invalidate} \ \text{rt} \ \text{dests}) \ \text{dip} \)"

proof (cases "\( \text{dip} \notin \text{kD}(\text{rt}) \)"
assume "\( \neg \text{dip} \notin \text{kD}(\text{rt}) \)"

hence "\( \text{dip} \in \text{kD}(\text{rt}) \)" by simp

then obtain \text{dsn} \ \text{dsk} \ \text{flag} \ \text{hops} \ \text{nhip} \ \text{pre} \ where \ "\( \text{rt} \ \text{dip} = \text{Some} \ (\text{dsn}, \ \text{dsk}, \ \text{flag}, \ \text{hops}, \ \text{nhip}, \ \text{pre}) \)"

by (metis \text{kD}\_Some)

with assms show "\( \forall \text{rsn}. \ \text{dests} \ \text{dip} = \text{Some} \ \text{rsn} \implies \text{sqn} \ (\text{invalidate} \ \text{rt} \ \text{dests}) \ \text{dip} \leq \text{rsn} \)"

by (cases "\text{dests} \ \text{dip}\)" (auto simp add: invalidate\_def \ sqn\_def)

qed simp

lemma sqn\_invalidate\_in\_dests [simp]:
fixes \text{dests} \ \text{ipa} \ \text{rsn} \ \text{rt}

assumes "\( \forall \text{rsn}. \ \text{dests} \ \text{ipa} = \text{Some} \ \text{rsn} \implies \text{sqn} \ \text{rt} \ \text{ipa} = \text{rsn} \)"

and "\( \text{ipa} \notin \text{kD}(\text{rt}) \)"

shows "\( \forall \text{rsn}. \ \text{dests} \ \text{ipa} = \text{Some} \ \text{rsn} \implies \text{sqn} \ (\text{invalidate} \ \text{rt} \ \text{dests}) \ \text{ipa} = \text{rsn} \)"

unfolding invalidate\_def \ sqn\_def

using assms(1) \ \text{assms}(2) [THEN \text{kD}\_Some]

by clarsimp

lemma dhops\_invalidate [simp]:
"\( \lambda \text{dip}. \ \text{dhops} (\text{invalidate} \ \text{rt}\) \ \text{dests} \ \text{dip}) = \text{the} (\text{dhops} \ \text{rt} \ \text{dip})\)"

unfolding invalidate\_def by (clarsimp split: option.split)

lemma sqn\_f\_invalidate [simp]:
"\( \lambda \text{dip}. \ \text{sqn} (\text{invalidate} \ (\text{rt} \ \xi) \ (\text{dests} \ \xi)) \ \text{dip} = \text{sqn} (\text{rt} \ \xi) \ \text{dip} \)"

unfolding sqn\_f\_def invalidate\_def by (clarsimp split: option.split)
lemma nhop_invalidate [simp]:
  "\forall dip. the (nhop (invalidate (rt ξ) (dests ξ)) dip) = the (nhop (rt ξ) dip)"
unfolding invalidate_def by (clarsimp split: option.split)

lemma invalidate_other [simp]:
  fixes rt dests dip
  assumes "dip ∉ dom(dests)"
  shows "invalidate rt dests dip = rt dip"
  using assms unfolding invalidate_def by (clarsimp split: option.split_asm)

lemma invalidate_none [simp]:
  fixes rt dests dip
  assumes "dip ∉ kD(rt)"
  shows "invalidate rt dests dip = None"
  using assms unfolding invalidate_def by clarsimp

lemma vD_invalidate_vD_not_dests:
  "\forall dip rt dests. dip ∈ vD(invalidate rt dests) =⇒ dip ∈ vD(rt) ∧ dests dip = None"
unfolding invalidate_def vD_def by (clarsimp split: option.split_asm)

lemma sqn_invalidate_not_in_dests [simp]:
  fixes dests dip rt
  assumes "dip ∉ dom(dests)"
  shows "sqn (invalidate rt dests dip) = sqn rt dip"
  using assms unfolding sqn_def by simp

lemma invalidate_changes:
  fixes rt dests dip dsn dsk flag hops nhip pre
  assumes "invalidate rt dests dip = Some (dsn, dsk, flag, hops, nhip, pre)"
  shows "dsn = (case dests dip of None ⇒ π₂(the (rt dip)) | Some rsn ⇒ rsn)
   ∧ dsk = π₃(the (rt dip))
   ∧ flag = (if dests dip = None then π₄(the (rt dip)) else inv)
   ∧ hops = π₅(the (rt dip))
   ∧ nhip = π₆(the (rt dip))
   ∧ pre = π₇(the (rt dip))"
  using assms unfolding invalidate_def
  by (cases "rt dip", clarsimp, cases "dests dip") auto

lemma proj3_inv: "\forall dip rt dests. dip ∈ kD (rt) =⇒ π₃(the (invalidate rt dests dip)) = π₃(the (rt dip))"
  by (clarsimp simp: invalidate_def kD_def split: option.split)

lemma dests_iD_invalidate [simp]:
  assumes "dests ip = Some rsn"
  and "ip ∈ kD(rt)"
  shows "ip ∈ iD(invalidate rt dests)"
  using assms(1) assms(2) [THEN kD_Some] unfolding invalidate_def iD_def
  by (clarsimp split: option.split_asm)

0.2.5 Route Requests

Generate a fresh route request identifier.

definition nrreqid :: "(ip × rreqid) set ⇒ ip ⇒ rreqid"
  where "nrreqid rreqs ip ≡ Max ({n. (ip, n) ∈ rreqs} ∪ {0}) + 1"

0.2.6 Queued Packets

Functions for sending data packets.

type_synonym store = "ip ⇒ (p × data list)"
definition \( \sigma_{\text{queue}} :: \text{store} \Rightarrow \text{ip} \Rightarrow \text{data list} \) (\( \sigma_{\text{queue}}'(\_, \_') \))
where \( \sigma_{\text{queue}}(\text{store}, \text{dip}) \equiv \text{case store dip of None } \Rightarrow [ ] \mid \text{Some } (p, q) \Rightarrow q \)

definition \( qD :: \text{store} \Rightarrow \text{ip set} \)
where \( qD \equiv \text{dom} \)

definition \( \text{add} :: \text{data} \Rightarrow \text{ip} \Rightarrow \text{store} \Rightarrow \text{store} \)
where \( \text{add }d \text{ dip store } \equiv \text{case store dip of} \)
\( \text{None } \Rightarrow \text{store (dip } \mapsto\text{ (req, } [d]\text{))} \)
\( \text{Some } (p, q) \Rightarrow \text{store (dip } \mapsto\text{ (p, q } @\text{ [d]\text{))}} \)

lemma \( qD_{\text{add}} \text{ [simp]}: \)
fixes \( d \) dip store
shows \( "qD(\text{add }d \text{ dip store}) = \text{insert dip (qD store)}" \)
unfolding \( \text{add_def Let_def qD_def} \)
by \( (\text{clarsimp split: option.split}) \)

definition \( \text{drop} :: \text{ip } \Rightarrow \text{store } \rightarrow \text{store} \)
where \( \text{drop dip store } \equiv \)
\( \text{map_option (λ(p, q). if tl q = [] then store (dip := None) else store (dip } \mapsto\text{ (p, tl q))) (store dip}) \)

definition \( \sigma_{\text{p-flag}} :: \text{store} \Rightarrow \text{ip } \Rightarrow \text{p} \) (\( \sigma_{\text{p-flag}}'(\_, \_') \))
where \( \sigma_{\text{p-flag}}(\text{store}, \text{dip}) \equiv \text{map_option fst (store dip)} \)

definition \( \text{unsetRRF} :: \text{store} \Rightarrow \text{ip } \Rightarrow \text{store} \)
where \( \text{unsetRRF store dip } \equiv \text{case store dip of} \)
\( \text{None } \Rightarrow \text{store} \)
\( \text{Some } (p, q) \Rightarrow \text{store (dip } \mapsto\text{ (noreq, q)}) \)

definition \( \text{setRRF} :: \text{store} \Rightarrow \text{(ip } \Rightarrow \text{sqn)} \Rightarrow \text{store} \)
where \( \text{setRRF store dests } \equiv \lambda \text{dip}. \text{if dests dip } \Rightarrow \text{None then store dip} \)
\( \text{else map_option (λ(_, q). (req, q)) (store dip}) \)

0.2.7 Comparison with the original technical report

The major differences with the AODV technical report of Fehnker et al are:

1. \( \text{nhop} \) is partial, thus a ‘the’ is needed, similarly for \( \text{dhops} \) and \( \text{addpreRT} \).
2. \( \text{precs} \) is partial.
3. \( \sigma_{\text{p-flag}}(\text{store}, \text{dip}) \) is partial.
4. The routing table (\( \text{rt} \)) is modelled as a map (\( \text{ip } \Rightarrow \text{r option} \)) rather than a set of 7-tuples, likewise, the \( \text{r} \)
is a 6-tuple rather than a 7-tuple, i.e., the destination ip-address (\( \text{dip} \)) is taken from the argument to the function, rather than a part of the result. Well-definedness then follows from the structure of the type and more related facts are available automatically, rather than having to be acquired through tedious proofs.
5. Similar remarks hold for the dests mapping passed to \( \text{invalidate} \), and \( \text{store} \).

end

0.3 AODV protocol messages

theory Aodv_Message
imports Aodv_Basic
begin
datatype \( \text{msg} = \)
\( \text{Rreq nat rreqid ip sqn k ip sqn ip} \)
\( \mid \text{Rrep nat ip sqn ip} \)
instantiation msg :: msg begin

definition newpkt_def [simp]: "newpkt ≡ λ(d, dip). Newpkt d dip"

definition eq_newpkt_def: "eq_newpkt m ≡ case m of Newpkt d dip ⇒ True | _ ⇒ False"

instance by intro_classes (simp add: eq_newpkt_def)
end

The msg type models the different messages used within AODV. The instantiation as a msg is a technicality due to the special treatment of newpkt messages in the AWN SOS rules. This use of classes allows a clean separation of the AWN-specific definitions and these AODV-specific definitions.

definition rreq :: "nat × rreqid × ip × sqn × k × ip × sqn × ip ⇒ msg"
  where "rreq ≡ λ(hops, rreqid, dip, dsn, dsk, oip, osn, sip). Rreq hops rreqid dip dsn dsk oip osn sip"

lemma rreq_simp [simp]:
  "rreq(hops, rreqid, dip, dsn, dsk, oip, osn, sip) = Rreq hops rreqid dip dsn dsk oip osn sip"
  unfolding rreq_def by simp

definition rrep :: "nat × ip × sqn × ip × ip ⇒ msg"
  where "rrep ≡ λ(hops, dip, dsn, oip, sip). Rrep hops dip dsn oip sip"

lemma rrep_simp [simp]:
  "rrep(hops, dip, dsn, oip, sip) = Rrep hops dip dsn oip sip"
  unfolding rrep_def by simp

definition rerr :: "(ip ⇒ sqn) × ip ⇒ msg"
  where "rerr ≡ λ(dests, sip). Rerr dests sip"

lemma rerr_simp [simp]:
  "rerr(dests, sip) = Rerr dests sip"
  unfolding rerr_def by simp

lemma not_eq_newpkt_rreq [simp]: "¬eq_newpkt (Rreq hops rreqid dip dsn dsk oip osn sip)"
  unfolding eq_newpkt_def by simp

lemma not_eq_newpkt_rrep [simp]: "¬eq_newpkt (Rrep hops dip dsn oip sip)"
  unfolding eq_newpkt_def by simp

lemma not_eq_newpkt_rerr [simp]: "¬eq_newpkt (Rerr dests sip)"
  unfolding eq_newpkt_def by simp

lemma not_eq_newpkt_pkt [simp]: "¬eq_newpkt (Pkt d dip sip)"
  unfolding eq_newpkt_def by simp

definition pkt :: "data × ip × ip ⇒ msg"
  where "pkt ≡ λ(d, dip, sip). Pkt d dip sip"

lemma pkt_simp [simp]:
  "pkt(d, dip, sip) = Pkt d dip sip"
  unfolding pkt_def by simp

end

0.4 The AODV protocol

theory Aodv imports Aodv_Data Aodv_Message "../AWN/AWN_SOS_Labels" "../AWN/AWN_Invariants"
0.4.1 Data state

record state =  
ip :: "ip"  
sn :: "sqn"  
rt :: "rt"  
rreqs :: "(ip × rreqid) set"  
store :: "store"  
msg :: "msg"  
data :: "data"  
dests :: "ip ↦ sqn"  
pre :: "ip set"  
rreqid :: "rreqid"  
dip :: "ip"  
oip :: "ip"  
hops :: "nat"  
dsn :: "sqn"  
dsk :: "k"  
osn :: "sqn"  
sip :: "ip"

abbreviation aodv_init :: "ip ⇒ state" where  "aodv_init i ≡ (| ip = i, sn = 1, rt = empty, rreqs = {}, store = empty,  
msg = (SOME x. True), data = (SOME x. True), dests = (SOME x. True), pre = (SOME x. True), rreqid = (SOME x. True),  
dip = (SOME x. True), oip = (SOME x. True), hops = (SOME x. True), dsn = (SOME x. True), dsk = (SOME x. True),  
osn = (SOME x. True), sip = (SOME x. x ≠ i) |)"

lemma some_neq_not_eq [simp]: "¬((SOME x :: nat. x ≠ i) = i)"  
  by (subst some_eq_ex) (metis zero_neq_numeral)

definition clear_locals :: "state ⇒ state"  
where "clear_locals ξ = ξ (|  
msg := (SOME x. True), data := (SOME x. True), dests := (SOME x. True), pre := (SOME x. True), rreqid := (SOME x. True),  
dip := (SOME x. True), oip := (SOME x. True), hops := (SOME x. True), dsn := (SOME x. True), dsk := (SOME x. True),  
osn := (SOME x. True), sip := (SOME x. x ≠ ip ξ) |)"
lemma clear_locals_sip_not_ip [simp]: \(\neg (\text{sip (clear_locals } \xi) = \text{ip } \xi)\)
unfolding clear_locals_def by simp

lemma clear_locals_but_not_globals [simp]:
"\text{ip (clear_locals } \xi) = \text{ip } \xi"
"\text{sn (clear_locals } \xi) = \text{sn } \xi"
"\text{rt (clear_locals } \xi) = \text{rt } \xi"
"\text{rreqs (clear_locals } \xi) = \text{rreqs } \xi"
"\text{store (clear_locals } \xi) = \text{store } \xi"
unfolding clear_locals_def by auto

0.4.2 Auxiliary message handling definitions

definition is_newpkt
where "is_newpkt \xi \equiv \text{case msg } \xi \text{ of}
Newpkt \text{ data' dip'} \Rightarrow \{ \xi[\text{data := data', dip := dip'}] \}
| _ \Rightarrow \{\}"

definition is_pkt
where "is_pkt \xi \equiv \text{case msg } \xi \text{ of}
Pkt \text{ data' dip' oip'} \Rightarrow \{ \xi[\text{data := data', dip := dip', oip := oip'}] \}
| _ \Rightarrow \{\}"

definition is_rreq
where "is_rreq \xi \equiv \text{case msg } \xi \text{ of}
Rreq \text{ hops' rreqid' dip' dsn' dsk' oip' osn' sip'} \Rightarrow
\{ \xi[\text{hops := hops', rreqid := rreqid', dip := dip', dsn := dsn',
    dsk := dsk', oip := oip', osn := osn', sip := sip'}] \}
| _ \Rightarrow \{\}"

lemma is_rreq_asm [dest!]:
assumes "\xi' \in \text{is_rreq } \xi"
shows "(\exists \text{hops' rreqid' dip' dsn' dsk' oip' osn' sip'}.
\text{msg } \xi = \text{Rreq hops' rreqid' dip' dsn' dsk' oip' osn' sip'} \wedge
\xi' = \xi[\text{hops := hops', rreqid := rreqid', dip := dip', dsn := dsn',
    dsk := dsk', oip := oip', osn := osn', sip := sip'}]"
using assms unfolding is_rreq_def
by (cases "msg \xi") simp_all

definition is_rrep
where "is_rrep \xi \equiv \text{case msg } \xi \text{ of}
Rrep \text{ hops' dip' dsn' oip' sip'} \Rightarrow
\{ \xi[\text{hops := hops', dip := dip', dsn := dsn', oip := oip', sip := sip'}] \}
| _ \Rightarrow \{\}"

lemma is_rrep_asm [dest!]:
assumes "\xi' \in \text{is_rrep } \xi"
shows "(\exists \text{hops' dip' dsn' oip' sip'}.
\text{msg } \xi = \text{Rrep hops' dip' dsn' oip' sip'} \wedge
\xi' = \xi[\text{hops := hops', dip := dip', dsn := dsn', oip := oip', sip := sip'}]"
using assms unfolding is_rrep_def
by (cases "msg \xi") simp_all

definition is_rerr
where "is_rerr \xi \equiv \text{case msg } \xi \text{ of}
Rerr \text{ dests' sip'} \Rightarrow \{ \xi[\text{dests := dests', sip := sip'}] \}
| _ \Rightarrow \{\}"

lemma is_rerr_asm [dest!]:
assumes "\xi' \in \text{is_rerr } \xi"
shows "(\exists \text{dests' sip'}.\n\text{msg } \xi = \text{Rerr dests' sip'} \wedge
\xi' = \xi[\text{dests := dests', sip := sip'}]]"
using assms unfolding is_rerr_def
by (cases "msg ξ") simp_all

lemmas is_msg_defs =
is_rerr_def is_rrep_def is_rreq_def is_pkt_def is_newpkt_def

lemma is_msg_inv_ip [simp]:
"ξ' ∈ is_rerr ξ ⟹ ip ξ' = ip ξ"
"ξ' ∈ is_rrep ξ ⟹ ip ξ' = ip ξ"
"ξ' ∈ is_rreq ξ ⟹ ip ξ' = ip ξ"
"ξ' ∈ is_pkt ξ ⟹ ip ξ' = ip ξ"
"ξ' ∈ is_newpkt ξ ⟹ ip ξ' = ip ξ"
unfolding is_msg_defs
by (cases "msg ξ",clarsimp+)+

lemma is_msg_inv_sn [simp]:
"ξ' ∈ is_rerr ξ ⟹ sn ξ' = sn ξ"
"ξ' ∈ is_rrep ξ ⟹ sn ξ' = sn ξ"
"ξ' ∈ is_rreq ξ ⟹ sn ξ' = sn ξ"
"ξ' ∈ is_pkt ξ ⟹ sn ξ' = sn ξ"
"ξ' ∈ is_newpkt ξ ⟹ sn ξ' = sn ξ"
unfolding is_msg_defs
by (cases "msg ξ",clarsimp+)+

lemma is_msg_inv_rt [simp]:
"ξ' ∈ is_rerr ξ ⟹ rt ξ' = rt ξ"
"ξ' ∈ is_rrep ξ ⟹ rt ξ' = rt ξ"
"ξ' ∈ is_rreq ξ ⟹ rt ξ' = rt ξ"
"ξ' ∈ is_pkt ξ ⟹ rt ξ' = rt ξ"
"ξ' ∈ is_newpkt ξ ⟹ rt ξ' = rt ξ"
unfolding is_msg_defs
by (cases "msg ξ",clarsimp+)+

lemma is_msg_inv_rreqs [simp]:
"ξ' ∈ is_rerr ξ ⟹ rreqs ξ' = rreqs ξ"
"ξ' ∈ is_rrep ξ ⟹ rreqs ξ' = rreqs ξ"
"ξ' ∈ is_rreq ξ ⟹ rreqs ξ' = rreqs ξ"
"ξ' ∈ is_pkt ξ ⟹ rreqs ξ' = rreqs ξ"
"ξ' ∈ is_newpkt ξ ⟹ rreqs ξ' = rreqs ξ"
unfolding is_msg_defs
by (cases "msg ξ",clarsimp+)+

lemma is_msg_inv_store [simp]:
"ξ' ∈ is_rerr ξ ⟹ store ξ' = store ξ"
"ξ' ∈ is_rrep ξ ⟹ store ξ' = store ξ"
"ξ' ∈ is_rreq ξ ⟹ store ξ' = store ξ"
"ξ' ∈ is_pkt ξ ⟹ store ξ' = store ξ"
"ξ' ∈ is_newpkt ξ ⟹ store ξ' = store ξ"
unfolding is_msg_defs
by (cases "msg ξ",clarsimp+)+

lemma is_msg_inv_sip [simp]:
"ξ' ∈ is_pkt ξ ⟹ sip ξ' = sip ξ"
"ξ' ∈ is_newpkt ξ ⟹ sip ξ' = sip ξ"
unfolding is_msg_defs
by (cases "msg ξ",clarsimp+)+

0.4.3 The protocol process

datatype pseqp =
| PAodv
| PNewPkt
| PPkt
| PRreq
fun nat_of_seqp :: "pseqp ⇒ nat"
where
  "nat_of_seqp PAodv = 1"
| "nat_of_seqp PPkt = 2"
| "nat_of_seqp PNewPkt = 3"
| "nat_of_seqp PRreq = 4"
| "nat_of_seqp PRrep = 5"
| "nat_of_seqp PRerr = 6"

instantiation "pseqp" :: ord begin
  definition less_eq_seqp [iff]: "l1 ≤ l2 = (nat_of_seqp l1 ≤ nat_of_seqp l2)"
  definition less_seqp [iff]: "l1 < l2 = (nat_of_seqp l1 < nat_of_seqp l2)"
instance ..
end

abbreviation AODV where
  "AODV ≡ λ_. [clear_locals] call(PAodv)"

abbreviation PKT where
  "PKT args ≡
  [ξ. let (data, dip, oip) = args ξ in
   (clear_locals ξ) (| data := data, dip := dip, oip := oip |)]
call(PPkt)"

abbreviation NEWPKT where
  "NEWPKT args ≡
  [ξ. let (data, dip) = args ξ in
   (clear_locals ξ) (| data := data, dip := dip |)]
call(PNewPkt)"

abbreviation RREQ where
  "RREQ args ≡
  [ξ. let (hops, rreqid, dip, dsn, dsk, oip, osn, sip) = args ξ in
   (clear_locals ξ) (| hops := hops, rreqid := rreqid, dip := dip,
   dsn := dsn, dsk := dsk, oip := oip,
   osn := osn, sip := sip |)]
call(PRreq)"

abbreviation RREP where
  "RREP args ≡
  [ξ. let (hops, dip, dsn, oip, sip) = args ξ in
   (clear_locals ξ) (| hops := hops, dip := dip, dsn := dsn,
   oip := oip, sip := sip |)]
call(PRrep)"

abbreviation RERR where
  "RERR args ≡
  [ξ. let (dests, sip) = args ξ in
   (clear_locals ξ) (| dests := dests, sip := sip |)]
call(PRerr)"

fun Γ_AODV :: "(state, msg, pseqp, pseqp label) seqp_env"
where
  "Γ_AODV PAodv = labelled PAodv (}
receive(\lambda msg'. \xi ( | msg := msg' |)).

(is_newpkt) NEWPKT(\lambda x. (data \xi, ip \xi))

(is_pkt) PKT(\lambda x. (data \xi, dip \xi, oip \xi))

(is_rreq)
\[
(x, \xi (rt := update (rt \xi) (sip \xi) (0, unk, val, 1, sip \xi, \}) |)
RREQ(\lambda x. (hops \xi, rreqid \xi, dip \xi, dsn \xi, oip \xi, osn \xi, sip \xi))
\]

(is_rrrep)
\[
(x, \xi (rt := update (rt \xi) (sip \xi) (0, unk, val, 1, sip \xi, \}) |)
RREPRE(\lambda x. (hops \xi, dip \xi, dsn \xi, oip \xi, sip \xi))
\]

(is_rerr)
\[
(x, \xi (rt := update (rt \xi) (sip \xi) (0, unk, val, 1, sip \xi, \}) |)
RERR(\lambda x. (dests \xi, sip \xi))
\]

\{ x, \xi ( dip := dip |) \} dip \in qD(store \xi) \cap vD(rt \xi)
\[
\{ x, \xi (data := \text{hd}(\sigma_queue(store \xi, dip \xi)) |) \}
unicast(\lambda x. the (nhop (rt \xi) (dip \xi)), \lambda x. pkt(data \xi, dip \xi, ip \xi)).
\{ x, \xi (store := the (drop (dip \xi) (store \xi)) |) \}
ADDV()
\]

\{ x, \xi ( dests := (\lambda rip. if (rip \in vD (rt \xi) \land nhop (rt \xi) rip = nhop (rt \xi) (dip \xi))
then Some (inc (sqn (rt \xi) rip)) else None) |) \}
\[
\{ x, \xi (rt := invalidate (rt \xi) (dests \xi) |) \}
\{ x, \xi (store := setRRF (store \xi) (dests \xi)) |) \}
\[
\{ x, \xi (pre := \bigcup \{ \text{the (prec \xi (rt \xi) rip) | rip \in dom (dests \xi) \} |) \}
\}
\{ x, \xi (dests := (\lambda rip. if ((dests \xi) rip \neq None \land \text{the (prec \xi (rt \xi) rip) \neq \})
then (dests \xi) rip else None) |) \}
\}
groupcast(\lambda x. pre \xi, \lambda x. rerr(dests \xi, ip \xi)). ADDV()
\]

\{ x, \xi ( dip := dip |) \} dip \in qD(store \xi) - vD(rt \xi) \land the (\sigma_p-flag(store \xi, dip \xi) = req |
\[
\{ x, \xi (store := unsetRRF (store \xi) (dip \xi) |) \}
\{ x, \xi (sn := inc (sn \xi) |) \}
\{ x, \xi (rreqid := nrreqid (rreqs \xi) (ip \xi) |) \}
\{ x, \xi (rreqs := rreqs \xi \cup \{ (ip \xi, rreqid \xi) \} |)
\}
broadcast(\lambda x. rreq(0, rreqid \xi, dip \xi, sqn (rt \xi) (dip \xi), sqnf (rt \xi) (dip \xi),
ip \xi, sn \xi, ip \xi)). ADDV()"

/ "\text{ADDV} PNewPkt = labelled PNewPkt (
\{ x, \xi ( dip \xi = ip \xi)
\}
\{ x, \xi ( data \xi). ADDV() \}
\{ x, \xi ( dip \xi \neq ip \xi)
\{ x, \xi ( store := add (data \xi) (dip \xi) (store \xi) |) \}
\}
ADDV())"

/ "\text{ADDV} PPkt = labelled PPkt (
\{ x, \xi ( dip \xi = ip \xi)
\}
\{ x, \xi ( data \xi). ADDV() \}
\{ x, \xi ( dip \xi \neq ip \xi)
\{ x, \xi ( dip \xi \in vD (rt \xi))
\}
unicast(\lambda x. the (nhop (rt \xi) (dip \xi)), \lambda x. pkt(data \xi, dip \xi, oip \xi)). ADDV()
\}
\[
\{ x, \xi ( rt := invalidate (rt \xi) (dests \xi) |) \}
\{ x, \xi ( store := setRRF (store \xi) (dests \xi)) |) \}
\[
\{ x, \xi ( pre := \bigcup \{ \text{the (prec \xi (rt \xi) rip) | rip \in dom (dests \xi) \} |) \}
\}
\{ x, \xi ( dests := (\lambda rip. if ((dests \xi) rip \neq None \land \text{the (prec \xi (rt \xi) rip) \neq \})
then (dests \xi) rip else None) |) \}
\}
groupcast(\lambda x. pre \xi, \lambda x. rerr(dests \xi, ip \xi)). ADDV()
\]
\{ x, \xi ( dip \xi \notin vD (rt \xi))
\}
groupcast(\lambda x. the (prec \xi (rt \xi) (dip \xi)),
\lambda x. rerr([\text{dip} \xi \mapsto \text{sqn} (rt \xi) (dip \xi)], ip \xi)). ADDV()"
\]
\} x, \xi ( dip \xi \notin vD (rt \xi))
\}
groupcast(\lambda x. the (prec \xi (rt \xi) (dip \xi)),
\lambda x. rerr([\text{dip} \xi \mapsto \text{sqn} (rt \xi) (dip \xi)], ip \xi)). ADDV()"
\]
AODV()
)
)
)*

"\(\Gamma_{\text{AODV}}\) \ PRreq = labelled PRreq ( \\
\langle \xi. (\text{oip} \ \xi, \text{rreqid} \ \xi) \in \text{rreqs} \ \xi \rangle \\
AODV() \\
\oplus \langle \xi. (\text{oip} \ \xi, \text{rreqid} \ \xi) \notin \text{rreqs} \ \xi \rangle \\
[\xi. \ \xi () \ \text{rt} := \text{update} (\ \text{rt} \ \xi) \ (\text{oip} \ \xi) \ (\text{osn} \ \xi, \ \text{kno}, \ \text{val}, \ \text{hops} \ \xi + 1, \ \text{sip} \ \xi, \ \{\}) \ |] \\
[\xi. \ \xi () \ \text{rreqs} := \text{rreqs} \ \xi \cup \ \{\text{oip} \ \xi, \ \text{rreqid} \ \xi\} \ |] \\
( \\
(\xi. \ \xi () \ \text{dip} \ \xi = \text{ip} \ \xi) \\
(\xi. \ \xi () \ \text{sn} := \text{max} (\text{sn} \ \xi) \ (\text{dsn} \ \xi) \ |] \\
\text{unicast}(\lambda \xi. \ \text{the} (\text{nhop} (\ \text{rt} \ \xi) \ (\text{oip} \ \xi)), \ \lambda \xi. \ \text{rrep}(0, \ \text{dip} \ \xi, \ \text{sn} \ \xi, \ \text{oip} \ \xi, \ \text{ip} \ \xi)).AODV() \\
\oplus \langle \xi. \ \text{dip} \ \xi \notin \ \text{ip} \ \xi \rangle \\
( \\
(\xi. \ \xi () \ \text{dip} \ \xi \in \text{vD} (\ \text{rt} \ \xi) \ \wedge \ \text{dsn} \ \xi \leq \text{sqn} (\ \text{rt} \ \xi) \ (\text{dip} \ \xi) \ \wedge \ \text{sqnf} (\ \text{rt} \ \xi) \ (\text{dip} \ \xi) = \text{kno} \\
(\xi. \ \xi () \ \text{rt} := \text{the} (\text{addpreRT} \ (\ \text{rt} \ \xi) \ (\text{dip} \ \xi) \ \{\text{sip} \ \xi\} \ |] \\
(\xi. \ \xi () \ \text{store} := \text{setRFF} (\ \text{store} \ \xi) \ (\text{dests} \ \xi) \ |] \\
(\xi. \ \xi () \ \text{pre} := \bigcup \ \{ \text{the} (\text{precs} (\ \text{rt} \ \xi) \ \text{rip}) \ | \ \text{rip} \ \text{rip} \in \text{dom} (\ \text{dests} \ \xi) \ \} \ |] \\
(\xi. \ \xi () \ \text{dests} := (\lambda \text{rip}. \ \text{if} \ ((\text{dests} \ \xi) \ \text{rip} \neq \text{None} \ \wedge \ \text{the} (\text{precs} (\ \text{rt} \ \xi) \ \text{rip}) \neq \{\}) \\
\text{then} (\text{dests} \ \xi) \ \text{rip} \ \text{else} \ \text{None}) \ |] \\
\text{groupcast}(\lambda \xi. \ \text{pre} \ \xi, \ \lambda \xi. \ \text{rrr}(\text{dests} \ \xi, \ \text{ip} \ \xi)).AODV() \\
\oplus \langle \xi. \ \text{dip} \ \xi \notin \ \text{vD} (\ \text{rt} \ \xi) \ \vee \ \text{sqn} (\ \text{rt} \ \xi) \ (\text{dip} \ \xi) < \text{dsn} \ \xi \ \vee \ \text{sqnf} (\ \text{rt} \ \xi) \ (\text{dip} \ \xi) = \text{unk} \\
\text{broadcast}(\lambda \xi. \ \text{rreq}(\text{hops} \ \xi + 1, \ \text{rreqid} \ \xi, \ \text{dip} \ \xi, \ \text{max} (\text{sqn} (\ \text{rt} \ \xi) \ (\text{dip} \ \xi)) \ (\text{dsn} \ \xi), \ \text{dsk} \ \xi, \ \text{oip} \ \xi, \ \text{osn} \ \xi, \ \text{ip} \ \xi)). \\
AODV() \\
) \\
)"
\[
\text{then Some (inc (sqn (rt \xi) rip)) else None)}\\
\text{groupcast(}λξ. \text{pre} ξ, \lambdaξ. \text{rerr}(\text{dests} ξ, ip ξ))\text{.AODV()}\\
\oplus ⟨ξ. oip ξ \notin vD (rt ξ)⟩\text{AODV()}\\
\oplus ⟨ξ. rt ξ = update (rt ξ) (dip ξ) (dsn ξ, kno, val, hops ξ + 1, sip ξ, \{\})⟩\text{AODV()}\\
\]

\[
\text{declare } \Gamma_{AODV}.\text{sims [simp del, code del]}\\
\text{lemmas } \Gamma_{AODV}.\text{sims [simp, code]} = \Gamma_{AODV}.\text{simps [simplified]}\\
\]

\[
\text{fun } \Gamma_{AODV}.\text{skeleton}\\
\text{where}\\
\text{"} \Gamma_{AODV}.\text{skeleton PAodv = seqp_skeleton (}\Gamma_{AODV} \text{ PAodv)"}\\
\text{"} \Gamma_{AODV}.\text{skeleton PNewPkt = seqp_skeleton (}\Gamma_{AODV} \text{ PNewPkt)"}\\
\text{"} \Gamma_{AODV}.\text{skeleton PPkt = seqp_skeleton (}\Gamma_{AODV} \text{ PPkt)"}\\
\text{"} \Gamma_{AODV}.\text{skeleton PRreq = seqp_skeleton (}\Gamma_{AODV} \text{ PRreq)"}\\
\text{"} \Gamma_{AODV}.\text{skeleton PRrep = seqp_skeleton (}\Gamma_{AODV} \text{ PRrep)"}\\
\text{"} \Gamma_{AODV}.\text{skeleton PRerr = seqp_skeleton (}\Gamma_{AODV} \text{ PRerr)"}\\
\]

\[
\text{lemma } \Gamma_{AODV}.\text{skeleton_wf [simp]:}\\
\text{"} \text{wellformed } \Gamma_{AODV}.\text{skeleton}"\\
\text{proof (rule, intro allI)}\\
\text{fix pn pn'}\\
\text{show "call(pn') } \notin \text{ stermsl (}\Gamma_{AODV}.\text{skeleton pn)"}\\
\text{by (cases pn) simp_all}\\
\text{qed}\\
\]

\[
\text{declare } \Gamma_{AODV}.\text{skeleton.simps [simp del, code del]}\\
\text{lemmas } \Gamma_{AODV}.\text{skeleton.simps [simp, code]} = \Gamma_{AODV}.\text{skeleton.simps [simplified }\Gamma_{AODV}.\text{simps seqp_skeleton.simps]}}\\
\]

\[
\text{lemma aodv_proc_cases [dest]:}\\
\text{fixes p pn}\\
\text{shows "} p \in \text{ ctermsl (}\Gamma_{AODV} \text{ pn) } \Rightarrow (p \in \text{ ctermsl (}\Gamma_{AODV} \text{ PAodv) } \lor (p \in \text{ ctermsl (}\Gamma_{AODV} \text{ PNewPkt) } \lor (p \in \text{ ctermsl (}\Gamma_{AODV} \text{ PPkt) } \lor (p \in \text{ ctermsl (}\Gamma_{AODV} \text{ PRreq) } \lor (p \in \text{ ctermsl (}\Gamma_{AODV} \text{ PRrep) } \lor (p \in \text{ ctermsl (}\Gamma_{AODV} \text{ PRerr))))"}\\
\text{using assms}\\
\text{by (cases pn) simp_all}\\
\]

\[
\text{definition } \sigma_{AODV} :: \text{"} ip \Rightarrow (\text{state } \times (\text{state, msg, pseqp, pseqp label) seqp) set}"\]

28
where \[ \sigma_{AODV} i \equiv \{(aodv_init i, \Gamma_{AODV} PAodv)\} \]

abbreviation paodv :: "\( \text{ip} \Rightarrow (\text{state} \times (\text{state, msg, pseqp, pseqp label}) \text{ seqp, msg seq_action}) \text{ automaton} \)"

where
\[ \text{paodv i} \equiv (\text{init} = \sigma_{AODV} i, \text{trans} = \text{seqp}\_sos \Gamma_{AODV}) \]

lemma aodv_trans: "\( \text{trans (paodv i)} = \text{seqp}\_sos \Gamma_{AODV} \)"
  by simp

lemma aodv_control_within [simp]: "\( \text{control}_w\_\text{ithin} \Gamma_{AODV} \text{ (init (paodv i))} \)"
  unfolding \( \sigma_{AODV} \)_def by (rule control_withinI) (auto simp del: \( \Gamma_{AODV} \)_simps)

lemma aodv_wf [simp]: "\( \text{wellformed} \Gamma_{AODV} \)"
proof (rule, intro allI)
  fix pn pn'
  show "\( \text{call(pn')} \notin \text{stermsl (} \Gamma_{AODV} \text{ pn)} \)"
    by (cases pn) simp_all
qed

lemmas aodv_labels_not_empty [simp] = labels_not_empty [OF aodv_wf]

lemma aodv_ex_label [intro]: "\( \exists l. l \in \text{labels} \Gamma_{AODV} p \)"
by (metis aodv_labels_not_empty all_not_in_conv)

lemma aodv_ex_labelE [elim]:
  assumes "\( \forall l \in \text{labels} \Gamma_{AODV} p. P \ l \ p \)"
  and "\( \exists p \ l. P \ l \ p \implies Q \)"
  shows "Q"
  using assms by (metis aodv_ex_label)

lemma aodv_simple_labels [simp]: "\( \text{simple_labels} \Gamma_{AODV} \)"
proof
  fix p p'
  assume "\( p \in \text{subterms} (\Gamma_{AODV} \text{ pn}) \)"
  thus "\( \exists \ l. \text{ labels} \Gamma_{AODV} p = \{l\} \)"
    by (cases pn) (simp_all cong: seqp_congs | elim disjE)+
qed

lemma \( \sigma_{AODV} \)_labels [simp]: "\( (\xi, p) \in \sigma_{AODV} i \implies \text{labels} \Gamma_{AODV} p = \{PAodv-:0\} \)"
unfolding \( \sigma_{AODV} \)_def by simp

lemma aodv_init_kD_empty [simp]: "\( (\xi, p) \in \sigma_{AODV} i \implies kD (\text{rt} \ \xi) = \{\} \)"
unfolding \( \sigma_{AODV} \)_def kD_def by simp

lemma aodv_init_sip_not_ip [simp]: "\( \neg (\text{sip} (\text{aodv}_\text{init} i) = i) \)" by simp

lemma aodv_init_sip_not_ip' [simp]:
  assumes "\( (\xi, p) \in \sigma_{AODV} i \)"
  shows "\( \text{sip} \ \xi \neq \text{ip} \ \xi \)"
  using assms unfolding \( \sigma_{AODV} \)_def by simp

lemma aodv_init_sip_not_i [simp]:
  assumes "\( (\xi, p) \in \sigma_{AODV} i \)"
  shows "\( \text{sip} \ \xi \neq i \)"
  using assms unfolding \( \sigma_{AODV} \)_def by simp

lemma clear_locals_sip_not_ip':
  assumes "\( \text{ip} \ \xi = i \)"
  shows "\( \neg (\text{sip (clear_locals} \ \xi) = i) \)"
  using assms by auto
Stop the simplifier from descending into process terms.
declare seqp_congs [cong]

Configure the main invariant tactic for AODV.
declare
Gamma [cterms_env]
adx_proc_cases [ctermsl_cases]
seq_invariant_ctermsI [OF adox_wf adox_control_within adox_simple_labels adox_trans, cterms_intro]
seq_step_invariant_ctermsI [OF adox_wf adox_control_within adox_simple_labels adox_trans, cterms_intro]
end

0.5 Invariant assumptions and properties

theory Aodv_Predicates
imports Aodv
begin
Definitions for expression assumptions on incoming messages and properties of outgoing messages.

abbreviation not_Pkt :: "msg ⇒ bool"
where "not_Pkt m ≡ case m of Pkt _ _ _ ⇒ False | _ ⇒ True"
definition msg_sender :: "msg ⇒ ip"
where "msg_sender m ≡ case m of Rreq _ _ _ _ _ _ _ ipc ⇒ ipc
| Rrep _ _ _ _ ipc ⇒ ipc
| Rerr _ ipc ⇒ ipc
| Pkt _ _ ipc ⇒ ipc"

lemma msg_sender_simps [simp]:
"⋀hops rreqid dip dsn dsk oip osn sip. msg_sender (Rreq hops rreqid dip dsn dsk oip osn sip) = sip"
"⋀hops dip dsn oip sip. msg_sender (Rrep hops dip dsn oip sip) = sip"
"⋀dests sip. msg_sender (Rerr dests sip) = sip"
"⋀d dip sip. msg_sender (Pkt d dip sip) = sip"
unfolding msg_sender_def by simp_all
definition msg_zhops :: "msg ⇒ bool"
where "msg_zhops m ≡ case m of Rreq hopsc _ dipc _ _ _ oipc _ sipc ⇒ hopsc = 0 −→ oipc = sipc
| Rrep hopsc dipc _ _ sipc ⇒ hopsc = 0 −→ dipc = sipc
| _ ⇒ True"

lemma msg_zhops_simps [simp]:
"⋀hops rreqid dip dsn dsk oip osn sip.
msg_zhops (Rreq hops rreqid dip dsn dsk oip osn sip) = (hops = 0 −→ oip = sip)"
"⋀hops dip dsn oip sip. msg_zhops (Rrep hops dip dsn oip sip) = (hops = 0 −→ dip = sip)"
"⋀dests sip. msg_zhops (Rerr dests sip) = True"
"⋀d dip. msg_zhops (Newpkt d dip) = True"
"⋀d dip sip. msg_zhops (Pkt d dip sip) = True"
unfolding msg_zhops_def by simp_all
definition rreq_rrep_sn :: "msg ⇒ bool"
where "rreq_rrep_sn m ≡ case m of Rreq _ _ _ _ _ _ _ osnc _ ⇒ osnc ≥ 1
| Rrep _ _ dsnc _ _ ⇒ dsnc ≥ 1
| _ ⇒ True"

lemma rreq_rrep_sn_simps [simp]:
"⋀hops rreqid dip dsn dsk oip osn sip.
rreq_rrep_sn (Rreq hops rreqid dip dsn dsk oip osn sip) = (osn ≥ 1)"
"⋀hops dip dsn oip sip. rreq_rrep_sn (Rrep hops dip dsn oip sip) = (dsn ≥ 1)"
"⋀dests sip. rreq_rrep_sn (Rerr dests sip) = True"
definition rreq_rrep_fresh :: "rt ⇒ msg ⇒ bool"
where "rreq_rrep_fresh crt m ≡ case m of Rreq hopsc _ _ _ _ oipc osnc ipcc ⇒ (ipcc ≠ oipc →
    oipc ∈ kD(crt) ∧ (sqn crt oipc > osnc
    ∨ (sqn crt oipc = osnc
    ∧ the (dhops crt oipc) ≤ hopsc
    ∧ the (flag crt oipc) = val)))
| Rrep hopsc dipc dsnc _ ipcc ⇒ (ipcc ≠ dipc →
    dipc ∈ kD(crt)
    ∧ sqn crt dipc = dsnc
    ∧ the (dhops crt dipc) = hopsc
    ∧ the (flag crt dipc) = val)
| _ ⇒ True"

lemma rreq_rrep_fresh [simp]:
"∀hops rreqid dip dsn dsk oip osn sip.
  rreq_rrep_fresh crt (Rreq hops rreqid dip dsn dsk oip osn sip) =
  (sip ≠ oip → dip ∈ kD(crt)
    ∧ sqn crt dip = dsn
    ∧ the (dhops crt dip) = hops
    ∧ the (flag crt dip) = val)"

"∀hops dip dsn oip sip. rreq_rrep_fresh crt (Rrep hops dip dsn oip sip) =
  (sip ≠ dip → dip ∈ kD(crt)
    ∧ sqn crt dip = dsn
    ∧ the (dhops crt dip) = hops
    ∧ the (flag crt dip) = val)"

"∀dests sip. rreq_rrep_fresh crt (Rerr dests sip) = True"
"∀d dip. rreq_rrep_fresh crt (Newpkt d dip) = True"
"∀d dip sip. rreq_rrep_fresh crt (Pkt d dip sip) = True"

definition rerr_invalid :: "rt ⇒ msg ⇒ bool"
where "rerr_invalid crt m ≡ case m of Rerr destsc _ ⇒ (∀ripc ∈ dom(destsc).
    (ripc ∈ iD(crt) ∧ the (destsc ripc) = sqn crt ripc))
| _ ⇒ True"

lemma rerr_invalid [simp]:
"∀hops rreqid dip dsn dsk oip osn sip.
  rerr_invalid crt (Rreq hops rreqid dip dsn dsk oip osn sip) = True"

"∀hops dip dsn oip sip. rerr_invalid crt (Rrep hops dip dsn oip sip) = True"

"∀dests sip. rerr_invalid crt (Rerr dests sip) = (∀ripc ∈ iD(destsc).
    ripc ∈ iD(crt) ∧ the (destsc ripc) = sqn crt ripc)"

"∀d dip. rerr_invalid crt (Newpkt d dip) = True"
"∀d dip sip. rerr_invalid crt (Pkt d dip sip) = True"

definition initmissing :: "(nat ⇒ state option) × 'a ⇒ (nat ⇒ state) × 'a"
where "initmissing σ = (λi. case (fst σ) i of None ⇒ aodv_init i | Some s ⇒ s, snd σ)"

lemma not_in_net_ips_fst_init_missing [simp]:
assumes "i /∈ net_ips σ"
shows "fst (initmissing (netgmap fst σ)) i = aodv_init i"
using assms unfolding initmissing_def by simp

lemma fst_initmissing_netgmap_pair_fst [simp]:
"fst (initmissing (netgmap (λ(p, q). (fst (id p), snd (id p), q)) s))
  = fst (initmissing (netgmap fst s))"
unfolding initmissing_def by auto
We introduce a streamlined alternative to \texttt{initmissing} with \texttt{netgmap} to simplify invariant statements and thus facilitate their comprehension and presentation.

```plaintext
lemma \texttt{fst_initmissing_netgmap_default_aodv_init_netlift}:
"fst (initmissing (netgmap \texttt{fst} s)) = default aodv_init (netlift \texttt{fst} s)"

unfolding \texttt{initmissing_def default_def}
by (simp add: \texttt{fst_netgmap_netlift del: One_nat_def})
```

definition
\texttt{netglobal} :: "((\texttt{nat} \Rightarrow \texttt{state}) \Rightarrow \texttt{bool}) \Rightarrow ((\texttt{state} \times 'b) \times 'c) net_state \Rightarrow \texttt{bool}"
where
"\texttt{netglobal} P \equiv (\lambda s. P (default aodv_init (netlift \texttt{fst} s)))"

end

0.6 Quality relations between routes

theory \textit{Fresher}
import \textit{Aodv_Data}
begin

0.6.1 Net sequence numbers

On individual routes

definition
\texttt{nsqn} :: "r \Rightarrow \texttt{sqn}"
where
"\texttt{nsqn} r \equiv \texttt{if } \pi_4(r) = \texttt{val} \lor \pi_2(r) = 0 \texttt{then } \pi_2(r) \texttt{else } (\pi_2(r) - 1)"

lemma \texttt{nsqnr_def'}:
"\texttt{nsqn} r = (\texttt{if } \pi_4(r) = \texttt{inv} \texttt{then } \pi_2(r) - 1 \texttt{else } \pi_2(r))"

unfolding \texttt{nsqn_r_def} by simp

lemma \texttt{nsqn_r_zero [simp]}:
"\forall dsn dsk flag hops nhip pre. \texttt{nsqn} (0, dsk, flag, hops, nhip, pre) = 0"

unfolding \texttt{nsqn_r_def} by clarsimp

lemma \texttt{nsqn_r_val [simp]}:
"\forall dsn dsk hops nhip pre. \texttt{nsqn} (dsn, dsk, val, hops, nhip, pre) = dsn"

unfolding \texttt{nsqn_r_def} by clarsimp

lemma \texttt{nsqn_r_inv [simp]}:
"\forall dsn dsk hops nhip pre. \texttt{nsqn} (dsn, dsk, inv, hops, nhip, pre) = dsn - 1"

unfolding \texttt{nsqn_r_def} by clarsimp

lemma \texttt{nsqn_r_lte_dsn [simp]}:
"\forall dsn dsk flag hops nhip pre. \texttt{nsqn} (dsn, dsk, flag, hops, nhip, pre) \leq dsn"

unfolding \texttt{nsqn_r_def} by clarsimp

On routes in routing tables

definition
\texttt{nsqn} :: "rt \Rightarrow \texttt{ip} \Rightarrow \texttt{sqn}"
where
"\texttt{nsqn} r \equiv \lambda rt dip. \text{\texttt{case } \sigma}_{route}(rt, dip) \texttt{of } \texttt{None } \Rightarrow 0 | \texttt{Some } r \Rightarrow \texttt{nsqn}(r)"

lemma \texttt{nsqn_sqn_def}:
"\forall rt dip. \texttt{nsqn} rt dip = (\texttt{if } flag rt dip = \texttt{Some } \texttt{val} \lor \texttt{sqn} rt dip = 0
\texttt{then } \texttt{sqn} rt dip \texttt{else } \texttt{sqn} rt dip - 1)"

unfolding \texttt{nsqn_def sqn_def} by (clarsimp split: option.split aut)

lemma \texttt{not_in_kD_nsqn [simp]}:
asumes "dip \notin kD(rt)"
shows "\texttt{nsqn} rt dip = 0"
```
using assms unfolding nsqn_def by simp

lemma kD_nsqn:
  assumes "dip ∈ kD(rt)"
  shows "nsqn rt dip = nsqn rt (the (σ_route(rt, dip)))"
  using assms [THEN kD_Some] unfolding nsqn_def by clarsimp

lemma nsqr_r_flag_pred [simp, intro]:
  fixes dsn dsk flag hops nhip pre
  assumes "P (nsqn rt (dsn, dsk, val, hops, nhip, pre))"
  and "P (nsqn rt (dsn, dsk, inv, hops, nhip, pre))"
  shows "P (nsqn rt (dsn, dsk, flag, hops, nhip, pre))"
  using assms by (cases flag) auto

lemma nsqn_r_addpreRT_inv [simp]:
  "∀ rt dip npre dip'. dip ∈ kD(rt) ⇒ nsqn rt (the (the (addpreRT rt dip npre) dip')) = nsqn rt (the (rt dip'))"
  unfolding addpreRT_def nsqn_r_def by (frule kD_Some) (clarsimp split: option.split)

lemma sqn_nsqn:
  "∀ rt dip. sqn rt dip - 1 ≤ nsqn rt dip"
  unfolding sqn_def nsqn_def by (clarsimp split: option.split)

lemma nsqn_sqn: "nsqn rt dip ≤ sqn rt dip"
  unfolding sqn_def nsqn_def by (cases "rt dip") auto

lemma val_nsqn_sqn [elim, simp]:
  assumes "ip ∈ kD(rt)"
  and "the (flag rt ip) = val"
  shows "nsqn rt ip = sqn rt ip"
  using assms unfolding nsqn_sqn_def by auto

lemma vD_nsqn_sqn [elim, simp]:
  assumes "ip ∈ vD(rt)"
  shows "nsqn rt ip = sqn rt ip - 1"
  proof -
    from ⟨ip ∈ vD(rt)⟩ have "ip ∈ kD(rt)"
    and "the (flag rt ip) = val" by auto
    thus ?thesis ..
  qed

lemma inv_nsqn_sqn [elim, simp]:
  assumes "ip ∈ kD(rt)"
  and "the (flag rt ip) = inv"
  shows "nsqn rt ip = sqn rt ip - 1"
  using assms unfolding nsqn_sqn_def by auto

lemma iD_nsqn_sqn [elim, simp]:
  assumes "ip ∈ iD(rt)"
  shows "nsqn rt ip = sqn rt ip - 1"
  proof -
    from ⟨ip ∈ iD(rt)⟩ have "ip ∈ kD(rt)"
    and "the (flag rt ip) = inv" by auto
    thus ?thesis ..
  qed

lemma nsqn_update_changed_kno_val [simp]: "∀ rt ip dsn dsk hops nhip. rt ≠ update rt ip (dsn, kno, val, hops, nhip, {}) 
  ⇒ nsqn(update rt ip (dsn, kno, val, hops, nhip, {})) ip = dsn"
  using assms unfolding nsqn_r_def update_def
  by (clarsimp simp: kD_nsqn split: option.split_asm option.split split_if_asm)

lemma nsqn_addpreRT_inv [simp]:
  "∀ rt dip npre dip’. dip ∈ kD(rt) ⇒
  nsqn (the (addpreRT rt dip npre)) dip’ = nsqn rt dip’"

unfolding addpreRT_def nsqn_def nsqn_.def
by (frule kD_Some) (clarsimp split: option.split)

lemma nsqn_update_other [simp]:
  fixes dsn dsk flag hops dip nhip pre rt ip
  assumes "dip ≠ ip"
  shows "nsqn (update rt ip (dsn, dsk, flag, hops, nhip, pre)) dip = nsqn rt dip"
  using assms unfolding nsqn_def
  by (clarsimp split: option.split)

lemma nsqn_invalidate_eq:
  assumes "dip ∈ kD(rt)"
  and "dests dip = Some rsn"
  shows "nsqn (invalidate rt dests) dip = rsn - 1"
  using assms proof
  - from assms obtain dsk hops nhip pre
    where "invalidate rt dests dip = Some (rsn, dsk, inv, hops, nhip, pre)"
    unfolding invalidate_def by auto
  moreover from ⟨dip ∈ kD(rt)⟩ have "dip ∈ kD(invalidate rt dests)" by simp
  ultimately show ?thesis
    using ⟨dests dip = Some rsn⟩ by simp
  qed

lemma nsqn_invalidate_other [simp]:
  assumes "dip ∈ kD(rt)"
  and "dip/∈ dom dests"
  shows "nsqn (invalidate rt dests) dip = nsqn rt dip"
  using assms by (clarsimp simp add: kD_nsqn)

0.6.2 Comparing routes

definition
  fresher :: "r ⇒ r ⇒ bool" ("(_/ ⊑ _)" [51, 51] 50)
  where
    "fresher r r’ ≡ ((nsqn, r < nsqn, r’) ∨ (nsqn, r = nsqn, r’ ∧ π₅(r) ≥ π₅(r’)))"

lemma fresherI1 [intro]:
  assumes "nsqn, r < nsqn, r’"
  shows "r ⊑ r’"
  unfolding fresher_def using assms by simp

lemma fresherI2 [intro]:
  assumes "nsqn, r = nsqn, r’" 
  and "π₅(r) ≥ π₅(r’)"
  shows "r ⊑ r’"
  unfolding fresher_def using assms by simp

lemma fresherI [intro]:
  assumes "(nsqn, r < nsqn, r’) ∨ (nsqn, r = nsqn, r’ ∧ π₅(r) ≥ π₅(r’))"
  shows "r ⊑ r’"
  unfolding fresher_def using assms .

lemma fresherE [elim]:
  assumes "r ⊑ r’"
  and "nsqn, r < nsqn, r’ ⇒ P r r’" 
  and "nsqn, r = nsqn, r’ ∧ π₅(r) ≥ π₅(r’) ⇒ P r r’"
  shows "P r r’"
  using assms unfolding fresher_def by auto

lemma fresher_refl [simp]: "r ⊑ r"
unfolding fresher_def by simp

lemma fresher_trans [elim, trans]:
  \[ x \sqsubseteq y; y \sqsubseteq z \quad \Rightarrow \quad x \sqsubseteq z \]
  unfolding fresher_def by auto

lemma not_fresher_trans [elim, trans]:
  \[ \neg(x \sqsubseteq y); \neg(z \sqsubseteq x) \quad \Rightarrow \quad \neg(z \sqsubseteq y) \]
  unfolding fresher_def by auto

lemma fresher_dsn_flag_hops_const [simp]:
  fixes dsn dsk dsk' flag hops nhip nhip' pre pre'
  shows \((dsn, dsk, flag, hops, nhip, pre) \sqsubseteq (dsn, dsk', flag, hops, nhip', pre')\)
  unfolding fresher_def by (cases flag) simp_all

lemma addpre_fresher [simp]: "\(\forall npre. r \sqsubseteq (\text{addpre } r \ npre)\)"
  by clarsimp

0.6.3 Comparing routing tables

definition
  \(rt\_fresher:: "ip \Rightarrow rt \Rightarrow rt \Rightarrow bool"\)
  where
  \(rt\_fresher \equiv \lambda dip rt rt'. \text{(the (\(\sigma\_route(rt, dip)\))) \sqsubseteq \text{(the (\(\sigma\_route(rt', dip)\)))}\)"

abbreviation
  \(rt\_fresher\_syn:: "rt \Rightarrow ip \Rightarrow rt \Rightarrow bool" ("_/__") [51, 999, 51] 50\)
  where
  \(\text{"rt1 \sqsubseteq ip \sqrt{rt2}\"}\)

lemma rt_fresher_def':
  \("(rt1 \sqsubseteq ip \sqrt{rt2}) = (\text{nsqn rt1 dip} < \text{nsqn rt2 dip} \quad \lor \quad \text{nsqn rt1 dip} = \text{nsqn rt2 dip} \quad \land \quad \pi_5 \text{(the (dhops rt1 dip))} \geq \text{the (dhops rt2 dip)})\)"
  unfolding rt_fresher_def fresher_def by (rule refl)

lemma single_rt_fresher [intro]:
  assumes \("\text{the (rt1 ip) \sqsubseteq (the (rt2 ip))}\)"
  shows \("\text{rt1 \sqsubseteq (addpre rt1 rt2)}\)"
  using assms unfolding rt_fresher_def .

lemma rt_fresher_single [intro]:
  assumes \("\text{the (rt1 ip) \sqsubseteq (the (rt2 ip))}\)"
  shows \("\text{rt1 \sqsubseteq ip \sqrt{rt2}}\)"
  using assms unfolding rt_fresher_def .

lemma rt_fresher_def2:
  assumes \("\text{dip \in kD(rt1)}\)"
  and \("\text{dip \in kD(rt2)}\)"
  shows \("(rt1 \sqsubseteq \text{ip dip rt2}) = (\text{nsqn rt1 dip} < \text{nsqn rt2 dip} \quad \lor \quad \text{nsqn rt1 dip} = \text{nsqn rt2 dip} \quad \land \quad \text{the (dhops rt1 dip)} \geq \text{the (dhops rt2 dip)})\)"
  using assms unfolding rt_fresher_def fresher_def by (simp add: kD_nsqn proj5_eq_dhops)

lemma rt_fresherI1 [intro]:
  assumes \("\text{dip \in kD(rt1)}\)"
  and \("\text{dip \in kD(rt2)}\)"
  and \("\text{nsqn rt1 dip} < \text{nsqn rt2 dip}\)"
  shows \("\text{rt1 \sqsubseteq dip rt2}\)"
  unfolding rt_fresher_def2 [OF assms(1-2)] using assms(3) by simp

lemma rt_fresherI2 [intro]:
  assumes \("\text{dip \in kD(rt1)}\)"
  and \("\text{dip \in kD(rt2)}\)"
  and \("\text{nsqn rt1 dip} = \text{nsqn rt2 dip}\)"
  unfolding rt_fresher_def2 [OF assms(1-2)] using assms(3) by simp

35
and "the (dhops rt1 dip) ≥ the (dhops rt2 dip)
shows "rt1 ⊑ dip rt2"
unfolding rt_fresher_def2 [OF assms(1-2)] using assms(3-4) by simp

lemma rt_fresherE [elim]:
assumes "rt1 ⊑ dip rt2"
and "dip ∈ kD(rt1)"
and "dip ∈ kD(rt2)"
and ":[ nsqn rt1 dip < nsqn rt2 dip ] =⇒ P rt1 rt2 dip"
and ":[ nsqn rt1 dip = nsqn rt2 dip;
the (dhops rt1 dip) ≥ the (dhops rt2 dip) ] =⇒ P rt1 rt2 dip"
shows "P rt1 rt2 dip"
using assms(1) unfolding rt_fresher_def2 [OF assms(2-3)]
using assms(4-5) by auto

lemma rt_fresher_refl [simp]: "rt ⊑ dip rt"
unfolding rt_fresher_def by simp

lemma rt_fresher_trans [elim, trans]:
assumes "rt1 ⊑ dip rt2"
and "rt2 ⊑ dip rt3"
shows "rt1 ⊑ dip rt3"
using assms unfolding rt_fresher_def by auto

lemma rt_fresher_if_Some [intro!]:
assumes "the (rt dip) ⊑ r"
shows "rt ⊑ dip (λip. if ip = dip then Some r else rt ip)"
using assms unfolding rt_fresher_def by simp

definition rt_fresh_as :: "ip ⇒ rt ⇒ rt ⇒ bool"
where
"rt_fresh_as ≡ λdip rt1 rt2. (rt1 ⊑ dip rt2) ∧ (rt2 ⊑ dip rt1)"

abbreviation rt_fresh_as_syn :: "rt ⇒ ip ⇒ rt ⇒ bool" ("(_/≈_/)"
"[51, 999, 51] 50"
where
"rt1 ≈_1 rt2 ≡ rt_fresh_as i rt1 rt2"

lemma rt_fresh_as_refl [simp]: "⋀rt dip. rt ≈ dip rt" UNFOLDING RT_FRESHER_DEF BY SIMP

lemma rt_fresh_as_trans [simp, intro, trans]:
"⋀rt1 rt2 rt3 dip. [ rt1 ⊑ dip rt2; rt2 ⊑ dip rt3 ] =⇒ rt1 ≈ dip rt3"
unfolding rt_fresh_as_def rt_fresher_def
by (metis (mono_tags) fresher_trans)

lemma rt_fresh_asI [intro!]:
assumes "rt1 ⊑ dip rt2"
and "rt2 ⊑ dip rt1"
shows "rt1 ≈ dip rt2"
using assms unfolding rt_fresh_as_def by simp

lemma rt_fresh_as_fresherI [intro]:
assumes "dip ∈ kD(rt1)"
and "dip ∈ kD(rt2)"
and "the (rt1 dip) ⊑ the (rt2 dip)"
and "the (rt2 dip) ⊑ the (rt1 dip)"
shows "rt1 ≈ dip rt2"
using assms unfolding rt_fresh_as_def
by (clarsimp dest!: single_rt_fresher)

lemma nsqn_rt_fresh_asI:
assumes "dip ∈ kD(rt)"
and "dip ∈ kD(rt')"
and "nsqn rt dip = nsqn rt' dip"
and "π₅(the (rt dip)) = π₅(the (rt' dip))"
s
shows "rt ≈ₕ rt'"

proof
from assms(1-2,4) have dhops': "the (dhops rt' dip) ≤ the (dhops rt dip)"
  by (simp add: proj5_eq_dhops)
with assms(1-3) show "rt ⊑ₕ rt'"
  by (rule rt_fresherI2)

next
from assms(1-2,4) have dhops: "the (dhops rt dip) ≤ the (dhops rt' dip)"
  by (simp add: proj5_eq_dhops)
with assms(2,1) assms(3) [symmetric] show "rt' ⊑ₕ dip rt"
  by (rule rt_fresherI2)

qed

lemma rt_fresh_asE [elim]:
  assumes "rt₁ ≈ₗ dip rt₂"
  shows "rt₁ ⊑ₕ dip rt₂" using assms unfolding rt_fresh_as_def by simp

lemma rt_fresh_asD1 [dest]:
  assumes "rt₁ ≈ₗ dip rt₂"
  shows "rt₁ ⊑ₕ dip rt₂" using assms unfolding rt_fresh_as_def by simp

lemma rt_fresh_asD2 [dest]:
  assumes "rt₁ ≈ₗ dip rt₂"
  shows "rt₂ ⊑ₕ dip rt₁" using assms unfolding rt_fresh_as_def by simp

lemma rt_fresh_as_sym:
  assumes "rt₁ ≈ₗ dip rt₂"
  shows "rt₂ ≈ₗ dip rt₁" using assms unfolding rt_fresh_as_def by simp

lemma not_rt_fresh_asI1 [intro]:
  assumes "¬ (rt₁ ⊑ₕ dip rt₂)"
  shows "¬ (rt₁ ≈ₗ dip rt₂)"
  proof
    assume "rt₁ ≈ₗ dip rt₂"
    hence "rt₁ ⊑ₕ dip rt₂" ..
    with ¬ (rt₁ ⊑ₕ dip rt₂) show False ..
  qed

lemma not_rt_fresh_asI2 [intro]:
  assumes "¬ (rt₂ ⊑ₕ dip rt₁)"
  shows "¬ (rt₁ ≈ₗ dip rt₂)"
  proof
    assume "rt₁ ≈ₗ dip rt₂"
    hence "rt₂ ⊑ₕ dip rt₁" ..
    with ¬ (rt₂ ⊑ₕ dip rt₁) show False ..
  qed

lemma not_single_rt_fresher [elim]:
  assumes "¬(the (rt₁ ip) ⊑ ip the (rt₂ ip))"
  shows "¬(rt₁ ⊑ ip rt₂)"
  proof
    assume "rt₁ ⊑ ip rt₂"
    hence "the (rt₁ ip) ⊑ the (rt₂ ip)" ..
    with ¬(the (rt₁ ip) ⊑ the (rt₂ ip)) show False ..
  qed

lemmas not_single_rt_fresh_asI1 [intro] = not_rt_fresh_asI1 [OF not_single_rt_fresher]
lemmas not_single_rt_fresh_asI2 [intro] = not_rt_fresh_asI2 [OF not_single_rt_fresher]

lemma not_rt_fresher_single [elim]:
asumes "¬ (rt1 ⊑ ip rt2)"
shows "¬ (the (rt1 ip) ⊑ the (rt2 ip))"
proof
assume "the (rt1 ip) ⊑ the (rt2 ip)"
hence "rt1 ⊑ ip rt2" ..
with ¬(rt1 ⊑ ip rt2) show False ..
qed

lemma rt_fresher_as_nsqnr:
asumes "dip ∈ kD(rt1)"
and "dip ∈ kD(rt2)"
and "rt1 ≈ dip rt2"
shows "nsqn r (the (rt2 dip)) = nsqn r (the (rt1 dip))"
using assms(3) unfolding rt_fresher_def
by (auto simp: rt_fresher_def2 [OF ⟨dip ∈ kD(rt1)⟩ ⟨dip ∈ kD(rt2)⟩]
kD_nsqn [OF ⟨dip ∈ kD(rt1)⟩]
kD_nsqn [OF ⟨dip ∈ kD(rt2)⟩])

lemma rt_fresher_mapupd [intro!]:
asumes "dip ∈ kD(rt)"
and "dip ≠ ip"
shows "rt ⊑ dip rt(dip ↦ r)"
using assms unfolding rt_fresher_def by simp

lemma rt_fresher_map_update_other [intro!]:
asumes "dip ∈ kD(rt)"
and "dip ≠ ip"
shows "rt ⊑ dip rt(ip ↦ r)"
using assms unfolding rt_fresher_def by simp

lemma rt_fresher_update_other [simp]:
asumes "dip ∈ kD(rt)"
and "dip ≠ ip"
shows "rt ⊑ dip update rt ip r"
using assms unfolding update_def
by (clarsimp split: option.split) (fastforce)

theorem rt_fresher_update [simp]:
asumes "dip ∈ kD(rt)"
and "the (dhops rt dip) ≥ 1"
and "update_arg_wf r"
shows "rt ⊑ dip update rt ip r"
proof (cases "dip = ip")
assume "dip ≠ ip" with ⟨dip ∈ kD(rt)⟩ show ?thesis by (rule rt_fresher_update_other)
next
assume "dip = ip"

from ⟨dip ∈ kD(rt)⟩ obtain dsn, dsk, f, hops, nhip, pre
where rtn [simp]: "the (rt dip) = (dsn, dsk, f, hops, nhip, pre)"
by (metis prod_cases6)
with ⟨the (dhops rt dip) ≥ 1⟩ and ⟨dip ∈ kD(rt)⟩ have "hops ≥ 1"
by (metis proj5_eq_dhops projs(4))
from ⟨dip ∈ kD(rt)⟩ rtn have [simp]: "sqn rt dip = dsn"
and [simp]: "the (dhops rt dip) = hops"
and [simp]: "the (flag rt dip) = f"
by (simp add: sqn_def proj5_eq_dhops [symmetric]
proj4_eq_flag [symmetric])

from ⟨update_arg_wf r⟩ have "(dsn, dsk, f, hops, nhip, pre)"
proof (rule wf_r_cases)
fix nhip pre
from ⟨hops_n ≥ 1⟩ have "\(d_{n}, d_{sk_n}, f_n, hops_n, nhip_n, pre_n\) " ⊑ (dsn_n, unk, val, Suc 0, nhip, pre')"

unfolding fresher_def sqn_def by (cases f) auto
thus "(dsn_n, dsk_n, f_n, hops_n, nhip_n, pre_n) " ⊑ (dsn_n, unk, val, Suc 0, nhip, pre')" by (rule update_cases_kD, simp_all)

next
fix dsn :: sqn and hops nhip pre
assume "0 < dsn" and "hops < hops_n" thus "(dsn_n, dsk_n, f_n, hops_n, nhip_n, pre_n) " ⊑ (dsn_n, unk, val, Suc 0, nhip, pre')" using ⟨dip ∈ kD(rt)⟩ by (rule update_cases_kD, simp_all add: ⟨0 < dsn⟩)

next
assume "dsn_n = dsn" and "hops < hops_n" thus "(dsn_n, dsk_n, f_n, hops_n, nhip_n, pre_n) " ⊑ (dsn_n, unk, val, Suc 0, nhip, pre')" unfolding fresher_def nsqn_def by simp

next
assume "dsn_n = dsn" with ⟨0 < dsn⟩ show "(dsn_n, dsk_n, inv, hops_n, nhip_n, pre_n) " ⊑ (dsn_n, unk, val, Suc 0, nhip, pre')" unfolding fresher_def by simp

qed

hence "rt ⊑ dip update rt dip r"
by - (rule single_rt_fresher, simp)
with ⟨dip = ip⟩ show ?thesis by simp

qed

theorem rt_fresher_invalidate [simp]:
assumes "dip ∈ kD(rt)"
and indests: "\(\forall rip ∈ dom(dests). rip ∈ vD(rt) \land sqn rt rip < the (dests rip)\)"
shows "rt ⊑ dip invalidate rt dests"
proof (cases "dip ∈ dom(dests)"
assume "dip ∈ dom(dests)"
thus ?thesis using ⟨dip ∈ kD(rt)⟩
by - (rule single_rt_fresher, simp)

next
assume "dip ∈ dom(dests)"
moreover with indests have "dip ∈ vD(rt)"
and "sqn rt dip < the (dests dip)"
by auto
ultimately show ?thesis
unfolding invalidate_def sqn_def
by - (rule single_rt_fresher, auto simp: fresher_def)

qed

lemma nsqn_r_invalidate [simp]:
assumes "dip ∈ kD(rt)"
and "dip ∈ dom(dests)"
s shows "nsqn_r (the (invalidate rt dests dip)) = the (dests dip) - 1"
using assms unfolding invalidate_def by auto

lemma rt_fresh_as_inc_invalidate [simp]:
assumes "dip ∈ kD(rt)"
and "\( \forall \text{rip} \in \text{dom}(\text{dests}). \text{rip} \in vD(\text{rt}) \land \text{the} (\text{dests} \text{rip}) = \text{inc} (\text{sqn} \text{rt} \text{rip}) \)"
shows "\( \text{rt} \approx_{\text{dip}} \text{invalidate} \text{rt} \text{dests} \)"
proof (cases "\( \text{dip} \in \text{dom}(\text{dests}) \)"
assume "\( \text{dip} \in \text{dom}(\text{dests}) \)"
with \( \text{dip} \in kD(\text{rt}) \): have "\( \text{dip} \in kD(\text{invalidate} \text{rt} \text{dests}) \)"
by simp
with \( \text{dip} \in kD(\text{rt}) \): show \( ?\text{thesis} \)
by rule (simp_all add: \( \text{dip} \in \text{dom}(\text{dests}) \))
next
assume "\( \text{dip} \in \text{dom}(\text{dests}) \)"
with assms\( (2) \): have "\( \text{dip} \in vD(\text{rt}) \)"
and "\( \text{the} (\text{dests} \text{dip}) = \text{inc} (\text{sqn} \text{rt} \text{dip}) \)"
by auto
from \( \text{dip} \in vD(\text{rt}) \): have "\( \text{nsqn} \text{rt} \text{dip} = \text{sqn} \text{rt} \text{dip} \)"
by simp
moreover then have "\( \text{dip} \in kD(\text{rt}) \)"
by \( \text{simp} \)
ultimately show \( ?\text{thesis} \)
proof (rule nsqn\_rt\_fresh\_asI)
from \( \text{dip} \in vD(\text{rt}) \): have "\( \text{nsqn} \text{r} \text{rt} \text{dip} = \text{sqn} \text{rt} \text{dip} \)"
also have "\( \text{sqn} \text{rt} \text{dip} = \text{nsqn} \text{r} \text{(the} (\text{invalidate} \text{rt} \text{dests} \text{dip})) \)"
proof -
from \( \text{dip} \in kD(\text{rt}) \): have "\( \text{nsqn} \text{r} \text{(the} (\text{invalidate} \text{rt} \text{dests} \text{dip})) = \text{the} (\text{dests} \text{dip}) - 1 \)"
using \( \text{dip} \in \text{dom}(\text{dests}) \) by (rule nsqn\_r\_invalidate)
with \( \text{the} (\text{dests} \text{dip}) = \text{inc} (\text{sqn} \text{rt} \text{dip}) \)
show "\( \text{sqn} \text{rt} \text{dip} = \text{nsqn} \text{r} \text{(the} (\text{invalidate} \text{rt} \text{dests} \text{dip})) \)"
by simp
qed
also from \( \text{dip} \in kD(\text{invalidate} \text{rt} \text{dests}) \)
have "\( \text{nsqn} \text{r} \text{(the} (\text{invalidate} \text{rt} \text{dests} \text{dip})) = \text{nsqn} \text{(invalidate} \text{rt} \text{dests} \text{dip}) \)"
by simp\( \text{add} : \text{kD}\_\text{nsqn} \)
finally show "\( \text{nsqn} \text{rt} \text{dip} = \text{nsqn} \text{(invalidate} \text{rt} \text{dests} \text{dip}) \)"
qed simp
qed
lemmas \text{rt\_fresher\_inc\_invalidate} \[\text{simp}\] = \text{rt\_fresh\_as\_inc\_invalidate} \[\text{THEN} \text{rt\_fresh\_asD1}\]

lemma \text{rt\_fresh\_as\_addpreRT} \[\text{simp}\]:
assumes "\( \text{ip} \in kD(\text{rt}) \)"
shows "\( \text{rt} \approx_{\text{dip}} \text{the} (\text{addpreRT} \text{rt} \text{ip} \text{npre}) \)"
using assms \[\text{THEN} \text{kD}\_\text{Some}\] by (auto simp: \text{addpreRT}\_\text{def})

lemmas \text{rt\_fresher\_addpreRT} \[\text{simp}\] = \text{rt\_fresh\_as\_addpreRT} \[\text{THEN} \text{rt\_fresh\_asD1}\]

0.6.4 Strictly comparing routing tables

definition \text{rt\_strictly\_fresher} \equiv "\( \text{ip} \Rightarrow \text{rt} \Rightarrow \text{rt} \Rightarrow \text{bool} \)"
where "\( \text{rt\_strictly\_fresher} \equiv \lambda \text{dip} \text{rt1} \text{rt2}. (\text{rt1} \subseteq_{\text{dip}} \text{rt2}) \land \neg(\text{rt1} \approx_{\text{dip}} \text{rt2}) \)"

abbreviation \text{rt\_strictly\_fresher\_syn} :: "\( \text{rt} \Rightarrow \text{ip} \Rightarrow \text{rt} \Rightarrow \text{bool} \)" ("(){/ }{ _ _ }"
[51, 999, 51] 50)
where "\( \text{rt1} \sqsubseteq_{\text{ip}} \text{rt2} \equiv \text{rt\_strictly\_fresher} \text{ip} \text{rt1} \text{rt2} \)"

lemma \text{rt\_strictly\_fresher\_def\'}:
"\( \text{rt1} \sqsubseteq_{\text{ip}} \text{rt2} = ((\text{rt1} \sqsubseteq_{\text{ip}} \text{rt2}) \land \neg(\text{rt2} \sqsubseteq_{\text{ip}} \text{rt1})) \)"
unfolding \text{rt\_strictly\_fresher\_def} \text{rt\_fresh\_as\_def} by auto

lemma \text{rt\_strictly\_fresherI\'} [intro]:
assumes "\( \text{rt1} \sqsubseteq_{\text{ip}} \text{rt2} \)"
and "\( \neg(\text{rt2} \sqsubseteq_{\text{ip}} \text{rt1}) \)"
shows "\( \text{rt1} \sqsubseteq_{\text{ip}} \text{rt2} \)"
using assms unfolding \text{rt\_strictly\_fresher\_def\'} by simp

lemma \text{rt\_strictly\_fresherE\'} [elim]:
assumes "\( \text{rt1} \sqsubseteq_{\text{ip}} \text{rt2} \)"
and "\([ \text{rt1} \sqsubseteq_{\text{ip}} \text{rt2}; \neg(\text{rt2} \sqsubseteq_{\text{ip}} \text{rt1}) ] \Rightarrow \text{P} \text{rt1 rt2 i} \)"
shows "P rt1 rt2 i"
using assms unfolding rt_strictly_fresher_def'' by simp

lemma rt_strictly_fresherI [intro]:
  assumes "rt1 ⊑_i rt2"
  and "¬(rt1 ≈_i rt2)"
  shows "rt1 ⊏_i rt2"
unfolding rt_strictly_fresher_def using assms ..

lemmas rt_strictly_fresher_singleI [elim] = rt_strictly_fresherI [OF single_rt_fresher]

lemma rt_strictly_fresherE [elim]:
  assumes "rt1 ⊏_i rt2"
  and "[ rt1 ⊑_i rt2; ¬(rt1 ≈_i rt2) ] ==> P rt1 rt2 i"
  shows "P rt1 rt2 i"
using assms(1) unfolding rt_strictly_fresher_def
by rule (erule(1) assms(2))

lemma rt_strictly_fresher_def':
"rt1 ⊏_i rt2 =
(nsqr_r (the (rt1 i)) < nsqr_r (the (rt2 i))
 ∨ (nsqr_r (the (rt1 i)) = nsqr_r (the (rt2 i)) ∧ π5(the (rt1 i)) > π5(the (rt2 i))))"
unfolding rt_strictly_fresher_def'' rt_fresher_def fresher_def
by auto

lemma rt_strictly_fresher_fresherD [dest]:
  assumes "rt1 ⊏_i dip rt2"
  shows "the (rt1 dip) ⊑ the (rt2 dip)"
using assms unfolding rt_strictly_fresher_def rt_fresher_def by auto

lemma rt_strictly_fresher_not_fresh_asD [dest]:
  assumes "rt1 ⊏ dip rt2"
  shows "¬(rt1 ≈ dip rt2)"
using assms unfolding rt_strictly_fresher_def by auto

lemma rt_strictly_fresher_trans [elim, trans]:
  assumes "rt1 ⊏_i dip rt2"
  and "rt2 ⊏_i dip rt3"
  shows "rt1 ⊏_i dip rt3"
using assms proof -
from rt1 ⊏_i dip rt2 obtain "the (rt1 dip) ⊑ the (rt2 dip)" by auto
also from rt2 ⊏_i dip rt3 obtain "the (rt2 dip) ⊑ the (rt3 dip)" by auto
finally have "the (rt1 dip) ⊑ the (rt3 dip)" .
moreover have "¬ (rt1 ≈_i dip rt3)"
proof -
from rt1 ⊏_i dip rt2 obtain "¬(the (rt2 dip) ⊑ the (rt1 dip))" by auto
also from rt2 ⊏_i dip rt3 obtain "¬(the (rt3 dip) ⊑ the (rt2 dip))" by auto
finally have "¬(the (rt3 dip) ⊑ the (rt1 dip))" .
thus ?thesis ..
qed ultimately show "rt1 ⊏_i dip rt3" ..
qed

lemma rt_strictly_fresher_irefl [simp]: "¬ (rt ⊏_i dip rt)"
using assms unfolding rt_strictly_fresher_def
by clarsimp

lemma rt_fresher_trans_rt_strictly_fresher [elim, trans]:
  assumes "rt1 ⊏_i dip rt2"
  and "rt2 ⊏_i dip rt3"
  shows "rt1 ⊏_i dip rt3"
proof -
from rt1 ⊏_i dip rt2 have "rt1 ⊏_i dip rt2"
and "¬(rt2 ⊏_i dip rt1)"
unfolding \textit{rt\_strictly\_fresher\_def''} by auto
from this(1) and \(\langle rt2 \sqsubseteq dip rt3 \rangle\) have "rt1 \(\subseteq dip\) rt3" ..

moreover from \(\neg \langle rt3 \sqsubseteq dip rt1 \rangle\) have "\(\neg (rt3 \sqsubseteq dip rt1)\)"
proof (rule contrapos_nn)
assume "rt3 \(\sqsubseteq dip\) rt1"
with \(\langle rt2 \sqsubseteq dip rt3 \rangle\) show "rt2 \(\sqsubseteq dip\) rt1" ..
qed

ultimately show "rt1 \(\sqsubseteq dip\) rt3"
unfolding \textit{rt\_strictly\_fresher\_def''} by auto
qed

\textbf{lemma \textit{rt\_fresher\_trans\_rt\_strictly\_fresher'}} [elim, trans]:
assumes "rt1 \(\sqsubseteq dip\) rt2" and "rt2 \(\sqsubseteq dip\) rt3"
shows "rt1 \(\sqsubseteq dip\) rt3"

proof -
from \(\langle rt2 \sqsubseteq dip rt3 \rangle\) have "rt2 \(\sqsubseteq dip\) rt3" and "\(\neg (rt3 \sqsubseteq dip rt2)\)"
unfolding \textit{rt\_strictly\_fresher\_def''} by auto
from \(\langle rt1 \sqsubseteq dip rt2 \rangle\) and this(1) have "rt1 \(\sqsubseteq dip\) rt3" ..

moreover from \(\neg \langle rt3 \sqsubseteq dip rt2 \rangle\) have "\(\neg (rt3 \sqsubseteq dip rt1)\)"
proof (rule contrapos_nn)
assume "rt3 \(\sqsubseteq dip\) rt1"
thus "rt3 \(\sqsubseteq dip\) rt2" using \(\langle rt1 \sqsubseteq dip rt2 \rangle\) ..
qed

ultimately show "rt1 \(\sqsubseteq dip\) rt3"
unfolding \textit{rt\_strictly\_fresher\_def''} by auto
qed

\textbf{lemma \textit{rt\_fresher\_imp\_nsqn\_le}}:
assumes "rt1 \(\sqsubseteq dip\) rt2" and "ip \(\in\) kD rt1" and "ip \(\in\) kD rt2"
shows "\text{nsqn} rt1 ip \leq nsqn rt2 ip"
using assms(1)
by (auto simp add: \textit{rt\_fresher\_def2} [OF assms(2-3)])

\textbf{lemma \textit{rt\_strictly\_fresher\_ltI}} [intro]:
assumes "\text{dip} \(\in\) kD(rt1)" and "\text{dip} \(\in\) kD(rt2)" and "\text{nsqn} rt1 \text{ dip} < \text{nsqn} rt2 \text{ dip}"
shows "rt1 \(\sqsubseteq dip\) rt2"

proof
from assms show "rt1 \(\sqsubseteq dip\) rt2" ..
next
show "\(\neg (rt1 \approx dip rt2)\)"
proof
assume "rt1 \(\approx dip\) rt2"
hence "rt2 \(\sqsubseteq dip\) rt1" ..
hence "\text{nsqn} rt2 dip \leq \text{nsqn} rt1 dip"
using \(\text{dip} \(\in\) kD(rt2)\) \(\text{dip} \(\in\) kD(rt1)\)
by (rule \textit{rt\_fresher\_imp\_nsqn\_le})
with \(\text{nsqn} rt1 dip \leq \text{nsqn} rt2 dip\) show "False"
by simp
qed

qed

\textbf{lemma \textit{rt\_strictly\_fresher\_eqI}} [intro]:
assumes "i\(\in\)kD(rt1)" and "i\(\in\)kD(rt2)"

42
and "nsqn rt1 i = nsqn rt2 i"
and "π5(the (rt2 i)) < π5(the (rt1 i))"
sows "rt1 ⊏ rt2"
using assms unfolding rt_strictly_fresher_def' by (auto simp add: kD_nsqn)

lemma invalidate_rfsl_f_left [simp]:
"\\dests dip rt rt'. dests dip = None \implies (invalidate rt dests ⊏ dip rt') = (rt ⊏ dip rt')"
unfolding invalidate_def rt_strictly_fresher_def'
by (rule iffI) (auto split: option.split_asm)

lemma vD_invalidate_rfsl_f_strictly_fresher [simp]:
assumes "dip ∈ vD(invalidate rt1 dests)"
shows "(invalidate rt1 dests ⊏ dip rt') = (rt1 ⊏ dip rt')"
proof (cases "dip ∈ dom(dests)")
assume "dip ∉ dom(dests)"
with ⟨dip ∈ vD(invalidate rt1 dests)⟩
ultimately show ?thesis
unfolding invalidate_def rt_strictly_fresher_def'
by auto
next
assume "dip ∉ dom(dests)"
hence "dip ∉ dom(dests)"
unfolding invalidate_def vD_def
by clarsimp (metis (hide_lams, no_types) assms vD_Some vD_invalidate_vD_not_dests)
ultimately show ?thesis
unfolding invalidate_def rt_strictly_fresher_def'
by auto
qed

lemma rt_strictly_fresher_update_other [elim!]:
"\\dests dip ip rt r rt'. [ dip ≠ ip; rt ⊏ dip rt'] \implies update rt ip r ⊏ dip rt'
unfolding rt_strictly_fresher_def' by clarsimp

lemma addpreRF_strictly_fresher [simp]:
assumes "dip ∈ kD(rt)"
shows "(the (addpreRF rt dip npre) ⊏ ip rt2) = (rt ⊏ ip rt2)"
using assms unfolding rt_strictly_fresher_def'

lemma lt_sqn_imp_update_strictly_fresher:
assumes "dip ∈ vD (rt2 nhip)"
and *)) "osn < sqn (rt2 nhip) dip"
and **) "rt ≠ update rt dip (osn, kno, val, hops, nhip, {})"
shows "update rt dip (osn, kno, val, hops, nhip, {}) ⊏ dip rt2 nhip"
unfolding rt_strictly_fresher_def'
proof (rule disjI1)
from **) have "nsqn (update rt dip (osn, kno, val, hops, nhip, {})) dip = osn"
by (rule nsqn_update_changed_kno_val)
with ⟨dip ∈ vD(rt2 nhip)⟩
have "nsqn_r (the (update rt dip (osn, kno, val, hops, nhip, {})) dip) = osn"
by (simp add: kD_nsqn)
also have "osn < sqn (rt2 nhip) dip" by (rule *)
also have "sqn (rt2 nhip) dip = nsqn_r (the (rt2 nhip dip))"
unfolding nsqn_r_def using ⟨dip ∈ vD(rt2 nhip)⟩
by - (metis vD_flag_val proj2_eq_sqn proj4_eq_flag vD_iD_gives_kD(1))
finally show "nsqn_r (the (update rt dip (osn, kno, val, hops, nhip, {})) dip)
< nsqn_r (the (rt2 nhip dip))".
qed

lemma dhops_le_hops_imp_update_strictly_fresher:
assumes "dip ∈ vD(rt2 nhip)"
and sqn: "sqn (rt2 nhip) dip = osn"
and hop: "the (dhops (rt2 nhip) dip) ≤ hops"
and **) "rt ≠ update rt dip (osn, kno, val, Suc hops, nhip, {})"
shows "update rt dip (osn, kno, val, Suc hops, nhip, {}) ⊏ dip rt2 nhip"
unfolding rt_strictly_fresher_def'
proof (rule disjI2, rule conjI)
  from ** have "nsqn (update rt dip (osn, kno, val, Suc hops, nhip, {})) dip = osn"
    by (rule nsqn_update_changed_kno_val)
  with (dip∈vD(rt2 nhip))
    have "nsqₗ (the (update rt dip (osn, kno, val, Suc hops, nhip, {})) dip) = osn"
    by (simp add: kD_nsqn)
  also have "osn = sqn (update rt dip (osn, kno, val, Suc hops, nhip, {})) dip"
    unfolding nsqn_r_def using ⟨
      dip ∈ vD(rt2 nhip)
    ⟩
    by - (metis vD_flag_val proj2_eq_sqn proj4_eq_flag vD_iD_gives_kD(1))
  finally show "nsqn (the (update rt dip (osn, kno, val, Suc hops, nhip, {})) dip) = nsqn (the (rt2 nhip dip))".
next
  have "the (dhops (rt2 nhip) dip) ≤ hops" by (rule hop)
  also have "hops < hops + 1" by simp
  finally have "the (dhops (rt2 nhip) dip) < the (dhops (update rt dip (osn, kno, val, Suc hops, nhip, {})) dip)"
    using ⟨
      dip ∈ vD(rt2 nhip)
    ⟩
    by (simp add: proj5_eq_dhops)
  thus "π₅ (the (rt2 nhip dip)) < π₅ (the (update rt dip (osn, kno, val, Suc hops, nhip, {})) dip)"
    using ⟨
      dip ∈ vD(rt2 nhip)
    ⟩
    by (simp add: proj5_eq_dhops)
qed

lemma nsqn_invalidate:
  assumes "dip ∈ kD(rt)"
  and "∀ ip ∈ dom(dests). ip ∈ vD(rt) ∧ the (dests ip) = inc (sqn rt ip)"
  shows "nsqn (invalidate rt dests) dip = nsqn rt dip"
proof -
  from ⟨
      dip ∈ kD(rt)
    ⟩
    have "dip ∈ kD(invalidate rt dests)" by simp
  from assms have "rt ≃₆ dip invalidate rt dests"
    by (rule rt_fresh_as_inc_invalidate)
  with ⟨
      dip ∈ kD(rt) (dip ∈ kD(invalidate rt dests)) show ?thesis
    ⟩
    by (simp add: kD_nsqn del: invalidate_kD_inv)
    (erule(2) rt_fresh_as_nsqnr)
qed

0.7 Invariant proofs on individual processes

theory Seq_Invariants
imports "./AWN/Invariants" Aodv Aodv_Data Aodv_Predicates Fresher
begin

The proposition numbers are taken from the December 2013 version of the Fehnker et al technical report.

Proposition 7.2

lemma sequence_number_increases:
  "paodv i |=A onll Γ AODV (λ((ξ, _), _, (ξ', _)). sn ξ ≤ sn ξ')"
  by inv_cterms

lemma sequence_number_one_or_bigger:
  "paodv i |= onl Γ AODV (λ(ξ, _). 1 ≤ sn ξ)"
  by (rule onll_step_to_invariantI [OF sequence_number_increases])
  (auto simp: σAODV_def)

We can get rid of the onl/onll if desired...

lemma sequence_number_increases':
  "paodv i |=A (λ((ξ, _), _, (ξ', _)). sn ξ ≤ sn ξ')"
  by (rule step_invariant_weakenE [OF sequence_number_increases]) (auto dest!: onllD)
lemma sequence_number_one_or_bigger':
  "paodv i |= (\lambda(\xi, _). \, 1 \leq sn \xi)"
by (rule invariant_weakenE [OF sequence_number_one_or_bigger]) auto

lemma sip_in_kD:
  "paodv i |= onl \Gamma_{AODV} (\lambda(\xi, 1), l) \in \{\{PAodv-\:7\} \cup \{PAodv-\:5\} \cup \{PRrep-\:0..PRrep-\:1\} \\
\cup \{PRreq-\:0..PRreq-\:3\}) \rightarrow sip \xi \in kD (rt \xi)"
by inv_cterms

lemma rrep_1_update_changes:
  "paodv i |= onl \Gamma_{AODV} (\lambda(\xi, 1), (1 = PRrep-\:1 \rightarrow \\
rt \xi \neq update (rt \xi) (\{\xi\}))" 
by inv_cterms

lemma addpreRT_partly_welldefined:
  "paodv i |= onl \Gamma_{AODV} (\lambda(\xi, 1), (1 \in \{PRreq-\:16..PRreq-\:18\} \cup \{PRreq-\:2..PRreq-\:6\} \rightarrow dip \xi \in kD (rt \xi)) \\
\wedge (1 \in \{PRreq-\:3..PRreq-\:17\} \rightarrow oip \xi \in kD (rt \xi))"
by inv_cterms

Proposition 7.38

lemma includes_nhip:
  "paodv i |= onl \Gamma_{AODV} (\lambda(\xi, 1), \forall dip \in kD (rt \xi). the (nhop (rt \xi) dip) \in kD (rt \xi))"
proof -
{ fix ip and \xi \xi' :: state
  assume "\forall dip \in kD (rt \xi). the (nhop (rt \xi) dip) \in kD (rt \xi)"
  and "\xi' = \xi[\{\{\}]))" 
  hence "\forall dip \in kD (rt \xi).
          the (nhop (rt \xi) dip) = dip
          \lor the (nhop (rt \xi) dip) \in kD (rt \xi)"
  by clarsimp (metis nhop_update_unchanged nhop_update_changed update_another)
} note one_hop = this

{ fix ip sip sn hops and \xi \xi' :: state
  assume "\forall dip \in kD (rt \xi). the (nhop (rt \xi) dip) \in kD (rt \xi)"
  and "\xi' = \xi[\{\{\}]))" 
  and "sip \in kD (rt \xi)"
  hence "\forall dip \in kD (rt \xi).
          the (nhop (rt \xi) dip) = dip
          \lor the (nhop (rt \xi) dip) \in kD (rt \xi)"
  by clarsimp (metis nhop_update_unchanged nhop_update_changed update_another)
} note nhip_is_sip = this

show ?thesis
by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf sip_in_kD]
          onl_invariant_sterms [OF aodv_wf addpreRT_partly_welldefined]
          solve: one_hop nhip_is_sip)

qed

Proposition 7.22: needed in Proposition 7.4

lemma addpreRT_welldefined:
  "paodv i |= onl \Gamma_{AODV} (\lambda(\xi, 1), l \in \{PRreq-\:16..PRreq-\:18\} \rightarrow dip \xi \in kD (rt \xi)) \\
(1 = PRreq-\:17 \rightarrow oip \xi \in kD (rt \xi)) \\
(1 = PRreq-\:5 \rightarrow dip \xi \in kD (rt \xi)) \\
(1 = PRreq-\:6 \rightarrow (the (nhop (rt \xi) dip \xi) \in kD (rt \xi)))"
(is "_ ||= onl \Gamma_{AODV} ?p")

unfolding invariant_def
proof
  fix s
  assume "s \in reachable (paodv i) TT" 
  then obtain \xi p where "s = (\xi, p)"
          and "(\xi, p) \in reachable (paodv i) TT"
  by (metis PairE)

45
have "onl \( \Gamma_{AODV} \) ?P (\( \xi, p \))"
proof (rule onlI)
  fix l
  assume "l \in \text{labels} \( \Gamma_{AODV} \) p"
  with \((\xi, p) \in \text{reachable} \ (\text{paodv } i) \) \( \text{TT} \)
  have I1: "l \in \{PRreq-:16..PRreq-:18\} \rightarrow \text{dip } \xi \in \text{kD} (rt \xi)"
  and I2: "l = PRreq-:17 \rightarrow \text{oip } \xi \in \text{kD} (rt \xi)"
  and I3: "l = PRreq-:6 \rightarrow \text{dip } \xi \in \text{kD} (rt \xi)"
  by (auto dest!: invariantD [OF addpreRT_partly_welldefined])
moreover from \((\xi, p) \in \text{reachable} \ (\text{paodv } i) \) \( \text{TT} \) \( l \in \text{labels} \( \Gamma_{AODV} \) p)\) and I3
  have "l = PRreq-:6 \rightarrow \text{(the (nhop (rt } \xi \text{) (dip } \xi))) \in \text{kD} (rt \xi)"
  by (auto dest!: invariantD [OF includes_nhip])
ultimately show "?P (\( \xi, l \))"
  by simp
qed

with \((s = (\xi, p))\) show "onl \( \Gamma_{AODV} \) ?P s"
by simp
qed

Proposition 7.4

lemma known_destinations_increase:
  "\( \text{paodv } i \) \( \models =\) \( \text{A} \) \( \text{onll} \) \( \Gamma_{AODV} \) \( (\lambda((\xi, \_), \_, (\xi', \_)). \text{kD} \ (rt \xi) \subseteq \text{kD} \ (rt \xi'))\)"
by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf addpreRT_welldefined]
  simp add: subset_insertI)

Proposition 7.5

lemma rreqs_increase:
  "\( \text{paodv } i \) \( \models =\) \( \text{A} \) \( \text{onll} \) \( \Gamma_{AODV} \) \( (\lambda((\xi, \_), \_, (\xi', \_)). \text{rreqs } \xi \subseteq \text{rreqs } \xi')\)"
by (inv_cterms simp add: subset_insertI)

lemma dests_bigger_than_sqn:
  "\( \text{paodv } i \) \( \models \) \( \text{onl} \) \( \Gamma_{AODV} \) \( (\lambda((\xi, l)). l \in \{PAodv-:15..PAodv-:19\}
    \cup \{PPkt-:7..PPkt-:11\}
    \cup \{PRreq-:9..PRreq-:13\}
    \cup \{PRreq-:21..PRreq-:25\}
    \cup \{PRrep-:10..PRrep-:14\}
    \cup \{PRerr-:1..PRerr-:5\}
    \rightarrow \forall ip \in \text{dom}(\text{dests } \xi). \text{ip} \in \text{kD} (rt \xi) \land \text{sqn} (rt \xi) \text{ ip} \leq \text{the} \ (\text{dests } \xi \ \text{ip}))\)"
proof -
  have sqninv:
    "\( \forall \text{dests rt rsn ip.} \)
    \( \forall ip \in \text{dom}(\text{dests}) \). \text{ip} \in \text{kD} (rt \xi) \land \text{sqn} rt \text{ ip} \leq \text{the} \ (\text{dests \ ip}); \text{dests ip = Some rsn} \]
    \( \implies \text{sqn} \ (\text{invalidate \ rt \ \text{dests}) \ \text{ip} \leq \text{rsn})\"
    by (rule sqn_invalidate_in_dests [THEN eq_imp_le], assumption) auto
  have indests:
    "\( \forall \text{dests rt rsn ip.} \)
    \( \forall ip \in \text{dom}(\text{dests})\). \text{ip} \in \text{kD} (rt \xi) \land \text{sqn rt \ \text{ip} \leq \text{the} \ (\text{dests \ \text{ip}); \text{dests ip = Some rsn} \]
    \( \implies \text{ip} \in \text{kD} (rt \xi) \land \text{sqn rt \ \text{ip} \leq \text{rsn})\"
    by (metis domI option.sel)
  show ?thesis
  by inv_cterms
    (clarsimp split: split_if_asm option.split_asm
    elim!: sqninv indests)+
qed

Proposition 7.6

lemma sqns_increase:
  "\( \text{paodv } i \) \( \models =\) \( \text{A} \) \( \text{onll} \) \( \Gamma_{AODV} \) \( (\lambda((\xi, \_), \_, (\xi', \_)). \forall \text{ip. sqn} \ (rt \xi) \text{ ip} \leq \text{sqn} \ (rt \xi') \text{ ip}))\)"
proof -
  { fix \( \xi \) :: \text{state}
    assume \( \forall \text{ip} \in \text{dom}(\text{dests } \xi). \text{ip} \in \text{kD} (rt \xi) \land \text{sqn} (rt \xi) \text{ ip} \leq \text{the} \ (\text{dests } \xi \ \text{ip})\)
    have \( \forall \text{ip. sqn} (rt \xi) \text{ ip} \leq \text{sqn} \ (\text{invalidate \ (rt } \xi \text{) (dests } \xi \text{ ip))) \text{ ip})\)
    proof
      fix ip
      have "\( \text{\forall } \text{ip. sqn} \ (rt \xi) \text{ ip} \leq \text{sqn} \ (\text{invalidate \ (rt } \xi \text{) (dests } \xi \text{ ip))) \text{ ip})\)"
      proof (rule onlI)
        have I1: "l \in \{PRreq-:16..PRreq-:18\} \rightarrow \text{dip } \xi \in \text{kD} (rt \xi)"
        and I2: "l = PRreq-:17 \rightarrow \text{oip } \xi \in \text{kD} (rt \xi)"
        and I3: "l = PRreq-:6 \rightarrow \text{dip } \xi \in \text{kD} (rt \xi)"
        by (auto dest!: invariantD [OF addpreRT_partly_welldefined])
moreover from \((\xi, p) \in \text{reachable} \ (\text{paodv } i) \) \( \text{TT} \) \( l \in \text{labels} \( \Gamma_{AODV} \) p)\) and I3
        have "l = PRreq-:6 \rightarrow \text{(the (nhop (rt } \xi \text{) (dip } \xi))) \in \text{kD} (rt \xi)"
        by (auto dest!: invariantD [OF includes_nhip])
ultimately show "?P (\( \xi, l \))"
          by simp
        qed
      qed
    qed
  qed

46
from * have "ip ∉ dom(dests ξ) ∨ sqn (rt ξ) ip ≤ the (dests ξ ip)" by simp 
thus "sqn (rt ξ) ip ≤ sqn (invalidate (rt ξ) (dests ξ)) ip"
  by (metis domI invalidate_sqn option.sel)
qed
}

note solve_invalidate = this

show ?thesis
  by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf addpreRT_welldefined]
      onl_invariant_sterms [OF aodv_wf dests_bigger_than_sqn]
      simp add: solve_invalidate)
qed

Proposition 7.7

lemma ip_constant:
  "paodv i |= onl Γ_AODV (λ(ξ, _). ip ξ = i)"
by (inv_cterms simp add: σ_AODV_def)

Proposition 7.8

lemma sender_ip_valid':
  "paodv i |= onll Γ_AODV (λ(((ξ, _), a, _). anycast (λm. not_Pkt m −→ msg_sender m = ip ξ)) a)"
by inv_cterms

lemma sender_ip_valid:
  "paodv i |= (recvmsg P −→) onll Γ_AODV (λ(((ξ, _), a, _). anycast (λm. not_Pkt m −→ msg_sender m = i)) a)"
by (rule step_invariant_weaken_with_invariantE [OF ip_constant sender_ip_valid'])
(auto dest!: onlD onllD)

lemma received_msg_inv:
  "paodv i |= (recvmsg P →) onll Γ_AODV (λ((ξ, l). l ∈ {PAodv-:1} −→ P (msg ξ)))"
by inv_cterms

Proposition 7.9

lemma sip_not_ip':
  "paodv i |= (recvmsg (λm. not_Pkt m −→ msg_sender m ≠ i) →) onll Γ_AODV (λ(ξ, _). sip ξ ≠ ip ξ)"
by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]
  onl_invariant_sterms [OF aodv_wf ip_constant [THEN invariant_restrict_inD]]
  simp add: clear_locals_sip_not_ip') clarsimp+

lemma sip_not_ip:
  "paodv i |= (recvmsg (λm. not_Pkt m −→ msg_sender m ≠ i) →) onll Γ_AODV (λ(ξ, _). sip ξ ≠ i)"
by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]
  onl_invariant_sterms [OF aodv_wf ip_constant [THEN invariant_restrict_inD]]
  simp add: clear_locals_sip_not_ip') clarsimp+

Neither sip_not_ip' nor sip_not_ip is needed to show loop freedom.

Proposition 7.10

lemma hop_count_positive:
  "paodv i |= onll Γ_AODV (λ(ξ, _). ∀ip∈kD (rt ξ). the (dhops (rt ξ) ip) ≥ 1)"
by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf addpreRT_welldefined]) auto

lemma rreq_dip_in_vD_dip_eq_ip:
  "paodv i |= onll Γ_AODV (λ(ξ, 1). (1 ∈ {PRreq-:16..PRreq-:18} −→ dip ξ ∈ vD(rt ξ))
  ∧ (1 ∈ {PRreq-:5, PRreq-:6} −→ dip ξ = ip ξ)
  ∧ (1 ∈ {PRreq-:15..PRreq-:18} −→ dip ξ ≠ ip ξ))"
proof (inv_cterms, elim conjE)
  fix 1 ξ pp p'
  assume "((ξ, pp) ∈ reachable (paodv i) TT"
  and "(PRreq-:17)[ξ, rt := (the (addpreRT (rt ξ) (oip ξ) (the (nhop (rt ξ) (dip ξ))))]] p'
    ∈ sters Γ_AODV pp"
  and "1 = PRreq-:17"
  and "dip ξ ∈ vD (rt ξ)"
from this(1-3) have "oip ξ ∈ kD (rt ξ)"
  by (auto dest: onl_invariant_sterms [OF aodv_wf addpreRT_welldefined, where l="PRreq-:17"])
with \( \text{dip } \xi \in vD \text{ (rt } \xi) \) 

show "dip \( \xi \) \in vD \text{ (the (addpreRT (rt } \xi \text{) (oip } \xi \text{) \{the (nhop (rt } \xi \text{) (dip } \xi\}))})" by simp 

qed

Proposition 7.11

lemma anycast_msg_zhops:
"\langle \text{dip } \xi \in vD \text{ (rt } \xi) \rangle \text{ show "dip } \xi \in vD \text{ (the (addpreRT (rt } \xi \text{) (oip } \xi \text{) \{the (nhop (rt } \xi \text{) (dip } \xi\})}))" by simp} \)

qed

\begin{align*}
\text{Proposition 7.11} \\
\text{lemma anycast_msg_zhops:} \\
\text{"dip } \xi \in vD \text{ (rt } \xi) \}\text{ show "dip } \xi \in vD \text{ (the (addpreRT (rt } \xi \text{) (oip } \xi \text{) \{the (nhop (rt } \xi \text{) (dip } \xi\})}))" by simp} \) \\
\end{align*}
by (cases a) simp_all
qed

lemma hop_count_zero_oip_dip_sip':
"paodv i ||= (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) →) onl \(\Gamma_{AODV}\) \((λ(ξ, 1)).
(\{\langle λ \(\xi\), 1 \rangle \mid \xi \in \{4..5\} \cup \{6..7\}\} \cup \{\langle λ \(\xi\), 1 \rangle \mid \xi \in \{8..9\}\}) →
(hops \(\xi = 0 \rightarrow oip \(\xi = sip \(\xi))\)) ∧
((\{λ \(\xi\), 1 \rangle \mid \xi \in \{8..9\}\} \cup \{\langle λ \(\xi\), 1 \rangle \mid \xi \in \{10..11\}\}) →
(hops \(\xi = 0 \rightarrow dip \(\xi = sip \(\xi))))")
proof (rule invariant_weakenE [OF hop_count_zero_oip_dip_sip])
fix a
assume "recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) a"
thus "recvmsg msg_zhops a" by (cases a) simp_all
qed

Proposition 7.12

lemma zero_seq_unk_hops_one':
"paodv i ||= (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) →) onl \(\Gamma_{AODV}\) \((λ(ξ, _)).
∀ dip ∈ kD(rt). (sqn (rt dip) = 0 −→ sqnf (rt dip) = unk) ∧ (sqnf (rt dip) = unk −→ the (dhops (rt dip)) = Suc 0) ∧
(the (dhops (rt dip)) = Suc 0 −→ the (nhop (rt dip)) = dip))")
proof -
{ fix dip and ξ :: state and P
assume "sqn (invalidate (rt ξ) (dests ξ)) dip = 0" and all: "∀ ip. sqn (rt ξ) ip ≤ sqn (invalidate (rt ξ) (dests ξ)) ip" and *: "sqn (rt ξ) dip = 0 −→ P ξ dip" have "P ξ dip" proof -
from all have "sqn (rt ξ) dip ≤ sqn (invalidate (rt ξ) (dests ξ)) dip" ..
with * have "sqn (rt ξ) dip = 0" by simp thus "P ξ dip" by (rule *)
qed }
note sqn_invalidate_zero [elim!] = this

{ fix dsn hops :: nat and sip oip rt and ip dip :: ip
assume "∀ dip ∈ kD(rt). (sqn rt dip = 0 −→ \(\pi₃\) (the (rt dip)) = unk) ∧
(\(\pi₃\) (the (rt dip)) = unk −→ the (dhops rt dip) = Suc 0) ∧
(the (dhops rt dip) = Suc 0 −→ the (nhop rt dip) = dip))") and "hops = 0 −→ sip = dip" and "Suc 0 ≤ dsn" and "ip ≠ dip −→ ip ∈ kD(rt)"
hence "the (dhops (update rt dip (dsn, kno, val, Suc hops, sip, {})) ip) = Suc 0 −→
the (nhop (update rt dip (dsn, kno, val, Suc hops, sip, {})) ip) = ip" by - (rule update_cases, auto simp add: sqn_def dest!: bspec)
} note prreq_ok1 [simp] = this

{ fix ip dsn hops sip oip rt dip
assume "∀ dip ∈ kD(rt). (sqn rt dip = 0 −→ \(\pi₃\) (the (rt dip)) = unk) ∧
(\(\pi₃\) (the (rt dip)) = unk −→ the (dhops rt dip) = Suc 0) ∧
(the (dhops rt dip) = Suc 0 −→ the (nhop rt dip) = dip))") and "Suc 0 ≤ dsn" and "ip ≠ dip −→ ip ∈ kD(rt)"
hence "\(\pi₃\) (the (update rt dip (dsn, kno, val, Suc hops, sip, {})) ip) = unk −→
the (dhops (update rt dip (dsn, kno, val, Suc hops, sip, {})) ip) = Suc 0" by - (rule update_cases, auto simp add: sqn_def sqnf_def dest!: bspec)
} note prreq_ok2 [simp] = this

{ fix ip dsn hops sip oip rt dip
assume "∀ dip ∈ kD(rt). (sqn rt dip = 0 −→ \(\pi₃\) (the (rt dip)) = unk) ∧

\[
\begin{align*}
\pi_3(\text{the } (rt \ dip)) &= \text{unk} \rightarrow \text{the } (dhops \ rt \ dip) = \text{Suc } 0 \\
\text{the } (dhops \ rt \ dip) &= \text{Suc } 0 \rightarrow \text{the } (nhop \ rt \ dip) = \text{dip})
\end{align*}
\]

and "Suc } } 0 \leq \text{dsn}"
and "ip \neq dip \rightarrow \text{ip} \in \text{kD}(rt)"

hence "sqn \ (update \ rt \ dip \ (dsn, kno, val, Suc } } hops, sip, \{\}) \ ip = 0 \rightarrow \\
\pi_3 \ (the \ (update \ rt \ dip \ (dsn, kno, val, Suc } } hops, sip, \{\}) \ ip) = \text{unk}"

by - (rule update_cases, auto simp add: sqn_def dest!: bspec)

} note prreq_ok3 [simp] = this

{ fix \ rt \ sip \ assume \\
\"\forall \ dip \in \text{kD} \ rt. \\
(sqn \ (update \ rt \ dip \ (0, unk, val, Suc } } 0, sip, \{\}) \ dip = 0 \rightarrow \\
\pi_3 \ (the \ (update \ rt \ dip \ (0, unk, val, Suc } } 0, sip, \{\}) \ dip) = \text{unk} \\
\land \ (\pi_3 \ (the \ (update \ rt \ dip \ (0, unk, val, Suc } } 0, sip, \{\}) \ dip)) = \text{unk} \rightarrow \\
\text{the } (dhops \ (update \ rt \ dip \ (0, unk, val, Suc } } 0, sip, \{\}) \ dip) = \text{Suc } 0 \\
\land \ (\text{the } (dhops \ (update \ rt \ dip \ (0, unk, val, Suc } } 0, sip, \{\}) \ dip)) = \text{Suc } 0 \rightarrow \\
\text{the } (nhop \ (update \ rt \ dip \ (0, unk, val, Suc } } 0, sip, \{\}) \ dip) = \text{dip}"

by - (rule update_cases, simp_all add: sqn_def sqn_def)

} note prreq_ok4 [simp] = this

have prreq_ok5 [simp]: \\
\"\forall \ dip \in \text{kD} \ rt. \\
\pi_3 \ (the \ (update \ rt \ dip \ (0, unk, val, Suc } } 0, sip, \{\}) \ dip) = \text{unk} \rightarrow \\
\text{the } (dhops \ (update \ rt \ dip \ (0, unk, val, Suc } } 0, sip, \{\}) \ dip) = \text{Suc } 0"

by (rule update_cases) simp_all

have prreq_ok6 [simp]: \\
\"\forall \ dip \in \text{kD} \ rt. \\
sqn \ (update \ rt \ dip \ (0, unk, val, Suc } } 0, sip, \{\}) \ dip = 0 \rightarrow \\
\pi_3 \ (the \ (update \ rt \ dip \ (0, unk, val, Suc } } 0, sip, \{\}) \ dip) = \text{unk}"

by (rule update_cases) simp_all

show \(\text{thesis}\)

by (inv_cterms inv add: \\
onl_invariant_sterms_TT [OF aodv_wf addpreRT_welldefined] \\
onl_invariant_sterms [OF aodv_wf hop_count_zero_oip_dip_sip'] \\
seq_step_invariant_sterms_TT [OF sqns_increase aodv_wf aodv_trans] \\
onl_invariant_sterms [OF aodv_wf osn_rreq'] \\
onl_invariant_sterms [OF aodv_wf dsn_rrep']))) clarsimp+

qed

lemma zero_seq_unk_hops_one: \\
"paodv i \?thesis \by (rule invariant_weakenE [OF zero_seq_unk_hops_one']) auto

lemma kD_unk_or_atleast_one: \\
"paodv i \?thesis \\
by (rule invariant_weakenE [OF zero_seq_unk_hops_one']) auto

proof -

{ fix \ sip \ rt \ dsn1 \ dsn2 \ dsk1 \ dsk2 \ flag1 \ flag2 \ hops1 \ hops2 \ nhip1 \ nhip2 \ pre1 \ pre2
assume "dsk1 = \text{unk} \lor \text{Suc } } } 0 \leq \text{dsn2}"

hence "\pi_3 \ (the \ (update \ rt \ dip \ (dsn1, dsk1, flag1, hops1, nhip1, pre1) \ sip)) = \text{unk} \lor \text{Suc } } } 0 \leq \text{sqn \ (update \ rt \ dip \ (dsn2, flag2, hops2, nhip2, pre2) \ sip)"

unfolding update_def by (cases "dsk1 = \text{unk}") (clarsimp split: option.split)+

} note fromsip [simp] = this

{ fix \ dip \ sip \ rt \ dsn1 \ dsn2 \ dsk1 \ dsk2 \ flag1 \ flag2 \ hops1 \ hops2 \ nhip1 \ nhip2 \ pre1 \ pre2
assume alld: "\forall \ dip \in \text{kD} \ rt. \ \pi_3 \ (the \ (rt \ dip)) = \text{unk} \lor \text{Suc } } } 0 \leq \text{sqn \ rt \ dip}"

and **: "dsk1 = \text{unk} \lor \text{Suc } } } 0 \leq \text{dsn2}"

have "\forall \ dip \in \text{kD} \ rt. \ \pi_3 \ (the \ (update \ rt \ dip \ (dsn1, dsk1, flag1, hops1, nhip1, pre1) \ dip)) = \text{unk}

50
\[ \forall \text{Suc } 0 \leq \text{sqn} \ (\text{update } \text{rt sip} \ (\text{dsn2, dsk2, flag2, hops2, nhip2, pre2})) \ \text{dip} \]

(is "\(\forall \text{dip} \in \text{kD}(\text{rt})\). \ ?\text{prop dip}\(\))

\text{proof}

fix dip
assume "dip \in \text{kD}(\text{rt})"
thus "?\text{prop dip}"
proof (cases "dip = sip")
assume "dip = sip"
with ** show ?thesis
by simp
next
assume "dip \neq sip"
with \langle dip \in \text{kD}(\text{rt})\rangle allkd show ?thesis
by simp
qed

\{ note solve_update [simp] = this \}

\text{fix } \text{rt dests}
assume *: "\(\forall \text{ip} \in \text{dom(\text{dests})}. \ \text{ip} \in \text{kD}(\text{rt}) \wedge \text{sqn } \text{rt ip} \leq \text{the (\text{dests} \ \text{ip})}\)"
and **: "\(\forall \text{ip} \in \text{kD}(\text{rt}). \ \pi_3(\text{the (\text{rt dip})}) = \text{unk} \lor \text{Suc } 0 \leq \text{sqn } \text{rt dip}\)"

\text{have } "\forall \text{dip} \in \text{kD}(\text{rt}). \ \pi_3(\text{the (\text{rt dip})}) = \text{unk} \lor \text{Suc } 0 \leq \text{sqn } (\text{invalidate rt dests} \ \text{dip})"

\text{proof}

fix dip
assume "dip \in \text{kD}(\text{rt})"
with ** have "\(\pi_3(\text{the (\text{rt dip})}) = \text{unk} \lor \text{Suc } 0 \leq \text{sqn } \text{rt dip}\)" ..
thus "\pi_3 (\text{the (\text{rt dip})}) = \text{unk} \lor \text{Suc } 0 \leq \text{sqn } (\text{invalidate rt dests} \ \text{dip})"

\text{proof}

assume "\pi_3(\text{the (\text{rt dip})}) = \text{unk}" thus ?thesis ..
next
assume "\text{Suc } 0 \leq \text{sqn } \text{rt dip}"

\text{have } "\text{Suc } 0 \leq \text{sqn } (\text{invalidate rt dests} \ \text{dip})"

\text{proof (cases "dip} \in \text{dom(\text{dests})")}
assume "dip} \in \text{dom(\text{dests})")
with * have "\text{sqn } \text{rt dip} \leq \text{the (\text{dests dip})}" by simp
with \langle \text{Suc } 0 \leq \text{sqn } \text{rt dip} \rangle have "\text{Suc } 0 \leq \text{the (\text{dests dip})}" by simp

\text{unfolding invalidate_def sqn_def by auto}

next
assume "dip} \in \text{dom(\text{dests})")
with \langle \text{Suc } 0 \leq \text{sqn } \text{rt dip} \rangle \协会 dip} \in \text{kD}(\text{rt})\rangle \ [\text{THEN kd_Some}] \ \text{show } ?\text{thesis}

\text{unfolding invalidate_def sqn_def by auto}

qed

\{ note solve_invalidate [simp] = this \}

\text{show } ?\text{thesis}


qed

\text{Proposition 7.13}

\text{lemma rreq_rrep_sn_any_step_invariant:}
"\text{paodv } i \models \ A \ \text{recvmsg rreq_rrep_sn} \rightarrow \ \text{onll } \Gamma_{AODV} (\lambda(\_, a, \_). \text{anycast rreq_rrep_sn a})"

\text{proof -}

\text{have sqnf_kno: "\text{paodv } i \models \ \text{onll } \Gamma_{AODV} (\lambda(\xi, 1). (1 \in \{PRreq:-16..PRreq:-18\} \rightarrow \text{sqnf} \ (\text{rt } \xi) \ (\text{dip } \xi) = \text{kno}))"}

\text{by (\text{inv_cterms inv add: onl_invariant_sterms_TT [OF aodv_wf addpreRT_welldefined]})}
show ?thesis
  by (inv_cterms inv add: onl_invariant_sterms_TT [OF aodv_wf addpreRT_welldefined]
      onl_invariant_sterms [OF aodv_wf sequence_number_one_or_bigger
      [THEN invariant_restrict_inD]]
      onl_invariant_sterms [OF aodv_wf kD_unk_or_atleast_one]
      onl_invariant_sterms_TT [OF aodv_wf sqnf_kno]
      onl_invariant_sterms [OF aodv_wf osn_rreq]
      onl_invariant_sterms [OF aodv_wf dsn_rrep])
  (auto simp: proj2_eq_sqn)
qed

Proposition 7.14

lemma rreq_rrep_fresh_any_step_invariant:
  "paodv i |=A onll Γ_AODV (λ((ξ, _), a, _). anycast (rreq_rrep_fresh (rt ξ)) a)"
proof -
  have rreq_oip: "paodv i |= onll Γ_AODV (λ(ξ, l). anycast (rreq_rrep_fresh (rt ξ)) a)
      {PRreq-:3, PRreq-:4, PRreq-:15, PRreq-:27} →
      oip ξ ∈ kD(rt ξ)
      ∨ (sqn (rt ξ) (oip ξ) > (osn ξ)
          ∧ the (dhops (rt ξ) (oip ξ)) ≤ Suc (hops ξ)
          ∧ the (flag (rt ξ) (oip ξ)) = val))"
proof inv_cterms
    fix l l' pp p'
    assume "(ξ, pp) ∈ reachable (paodv i) TT"
    and "{PRreq-:2}\[ξ. ξ[rt :=
    update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ, {}, {})] p' ∈ sterms Γ_AODV pp"
    and "l' = PRreq-:3"
    show "osn ξ < sqn (update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ, {}, {})) (oip ξ)
        ∨ (sqn (update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ, {}, {})) (oip ξ) = osn ξ
            ∧ the (dhops (update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ, {}, {})) (oip ξ)) ≤ Suc (hops ξ)
            ∧ the (flag (update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ, {}, {})) (oip ξ)) = val)"
    unfolding update_def by (clarsimp split: option.split)
      (metis linorder_neqE_nat not_less)
  qed

have rrep_prrep: "paodv i |= onll Γ_AODV (λ(ξ, 1).
    {PRreq-:2..PRreq-:7} →
    (dip ξ ∈ kD(rt ξ)
        ∧ sqn (rt ξ) (dip ξ) = dsn ξ
        ∧ the (dhops (rt ξ) (dip ξ)) = Suc (hops ξ)
        ∧ the (flag (rt ξ) (dip ξ)) = val
        ∧ the (nhop (rt ξ) (dip ξ)) ∈ kD (rt ξ)))"
by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf rrep_1_update_changes]
    onl_invariant_sterms [OF aodv_wf sip_in_kD])

show ?thesis
  by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf rreq_oip]
      onl_invariant_sterms [OF aodv_wf rreq_dip_in_vD_dip_eq_ip]
      onl_invariant_sterms [OF aodv_wf rrep_prrep])
qed

Proposition 7.15

lemma rerr_invalid_any_step_invariant:
  "paodv i |=A onll Γ_AODV (λ((ξ, _), a, _). anycast (rerr_invalid (rt ξ)) a)"
proof -
  have dests_inv: "paodv i |= onll Γ_AODV (λ(ξ, 1).
    {PAodv-:15, PPkt-:7, PRreq-:9, PRreq-:21, PRreq-:10, PRerr-:1} →
    (∀ ip∈dom(dests ξ). ip∈vD(rt ξ)))
    ∧ (1 ∈ {PAodv-:16..PAodv-:19}
        ∪ {PPkt-:8..PPkt-:11}
        ∪ {PRreq-:10..PRreq-:13})"
Some well-definedness obligations are irrelevant for the Isabelle development:

1. In each routing table there is at most one entry for each destination: guaranteed by type.
2. In each store of queued data packets there is at most one data queue for each destination: guaranteed by structure.
3. Whenever a set of pairs \((rip, rsn)\) is assigned to the variable \(dests\) of type \(ip \rightarrow sqn\), or to the first argument of the function \(rerr\), this set is a partial function, i.e., there is at most one entry \((rip, rsn)\) for each destination \(rip\): guaranteed by type.

**Lemma 7.16**

\[ \text{Lemma } \text{dests}_\text{VD}_\text{inc}_\text{sqn}: \]

\[
\text{\textquote{paodv } } \vdash_m \text{onl } \Gamma_{AODV} (\lambda(\xi, \_). (\_ \in \{PAodv\rightarrow:15, PPkt\rightarrow:7, PRreq\rightarrow:9, PRreq\rightarrow:21, PRrep\rightarrow:10\}) \rightarrow (\forall \text{ip} \in \text{dom}(\text{dests } \xi). \text{ip} \in \text{vD}(\text{rt } \xi) \land \text{the } (\text{dests } \xi \text{ ip}) = \text{inc } (\text{sqn } (\text{rt } \xi \text{ ip}))) \land (1 = \text{PRerr}\rightarrow:1 \rightarrow (\forall \text{ip} \in \text{dom}(\text{dests } \xi). \text{ip} \in \text{vD}(\text{rt } \xi) \land \text{the } (\text{dests } \xi \text{ ip}) > \text{sqn } (\text{rt } \xi \text{ ip}))) \]

\[ \text{by invcterms (clarsimp split: split_if_asm option.split_asm simp: domIff)+} \]

**Proposition 7.27**

\[ \text{Lemma } \text{route}_\text{tables}_\text{fresher}: \]

\[
\text{\textquote{paodv } } \vdash_A (\text{recvmsg } \text{rreq}_\text{rrep}_\text{sn} \rightarrow \text{onll } \Gamma_{AODV} (\lambda((\xi, \_), (\_, (\xi', \_))). \forall \text{dip} \in \text{kD}(\text{rt } \xi). \text{rt } \xi \sqsubseteq \text{ip } \text{update } (\text{rt } \xi) (\text{oip } \xi) (\text{osn } \xi, \text{} \text{kn0, val, Suc } (\text{hops } \xi), \text{} \text{sip } \xi, \{})) \]

\[ \text{proof } \text{invcterms inv add: onl_invariant_sterms [OF aodv_wf dests_vD_inc_sqn \[THEN invariant_restrict_inD\]]} \]

\[ \text{fix } \xi \text{ pp p' assume } "(\xi, pp) \in \text{reachable } (\text{paodv } i) (\text{recvmsg } \text{rreq}_\text{rrep}_\text{sn})" \]

\[ \text{and } "(\{PRreq\rightarrow:2\})[\xi. \xi(\text{rt } := \text{update } (\text{rt } \xi) (\text{oip } \xi) (\text{osn } \xi, \text{} \text{kn0, val, Suc } (\text{hops } \xi), \text{} \text{sip } \xi, \{}))] p' \in \text{sterms } \Gamma_{AODV} pp" \]

\[ \text{and } "\text{Suc } 0 \leq \text{osn } \xi" \]

\[ \text{and } *: "\forall \text{ip} \in \text{kD } (\text{rt } \xi). \text{Suc } 0 \leq \text{the } (\text{dhops } (\text{rt } \xi) \text{ ip})" \]

\[ \text{show } "\forall \text{ip} \in \text{kD } (\text{rt } \xi). \text{rt } \xi \sqsubseteq \text{ip } \text{update } (\text{rt } \xi) (\text{oip } \xi) (\text{osn } \xi, \text{} \text{kn0, val, Suc } (\text{hops } \xi), \text{} \text{sip } \xi, \{}))" \]

\[ \text{proof } \text{fix ip assume } "\text{ip} \in \text{kD } (\text{rt } \xi)" \]

\[ \text{moreover with } * \text{ have } "1 \leq \text{the } (\text{dhops } (\text{rt } \xi) \text{ ip})" \text{ by simp} \]

\[ \text{moreover from } \text{Suc } 0 \leq \text{osn } \xi: \text{ have } "\text{update_arg_wf } (\text{osn } \xi, \text{} \text{kn0, val, Suc } (\text{hops } \xi), \text{} \text{sip } \xi, \{}))" .. \]

\[ \text{ultimately show } "\text{rt } \xi \sqsubseteq \text{ip } \text{update } (\text{rt } \xi) (\text{oip } \xi) (\text{osn } \xi, \text{} \text{kn0, val, Suc } (\text{hops } \xi), \text{} \text{sip } \xi, \{}))" \]

\[ \text{by (rule rt_fresher_update)} \]

\[ \text{qed} \]

\[ \text{next} \]

\[ \text{fix } \xi \text{ pp p' assume } "(\xi, pp) \in \text{reachable } (\text{paodv } i) (\text{recvmsg } \text{rreq}_\text{rrep}_\text{sn})" \]

\[ \text{and } "(\{PRreq\rightarrow:2\})[\xi. \xi(\text{rt } := \text{update } (\text{rt } \xi) (\text{dip } \xi) (\text{dsn } \xi, \text{} \text{kn0, val, Suc } (\text{hops } \xi), \text{} \text{sip } \xi, \{}))] p' \in \text{sterms } \Gamma_{AODV} pp" \]

\[ \text{and } "\text{Suc } 0 \leq \text{dsn } \xi" \]

\[ \text{and } *: "\forall \text{ip} \in \text{kD } (\text{rt } \xi). \text{Suc } 0 \leq \text{the } (\text{dhops } (\text{rt } \xi) \text{ ip})" \]
show "∀ ip∈kD (rt ξ). rt ξ ⊑ ip update (rt ξ) (dip ξ) (dsn ξ, kno, val, Suc (hops ξ), sip ξ, {})"

proof

fix ip
assume "ip∈kD (rt ξ)"
moreover with * have "1 ≤ the (dhops (rt ξ) ip)" by simp
moreover from ⟨Suc 0 ≤ dsn ξ⟩ have "update_arg_wf (dsn ξ, kno, val, Suc (hops ξ), sip ξ, {})" ..
ultimately show "rt ξ ⊑ ip update (rt ξ) (dip ξ) (dsn ξ, kno, val, Suc (hops ξ), sip ξ, {})"
by (rule rt_fresher_update)
qed

end

0.8 The quality increases predicate

definition quality_increases :: "state ⇒ state ⇒ bool"
where "quality_increases ξ ξ' ≡ (∀ dip∈kD(rt ξ). dip ∈ kD(rt ξ') ∧ rt ξ ⊑ dip rt ξ') ∧ (∀ dip. sqn (rt ξ) dip ≤ sqn (rt ξ') dip)"

lemma quality_increasesI [intro!]:
assumes "∀ dip. dip ∈ kD(rt ξ) =⇒ dip ∈ kD(rt ξ')"
and "∀ dip. ⟨dip ∈ kD(rt ξ); dip ∈ kD(rt ξ') ⟩ =⇒ rt ξ ⊑ dip rt ξ'"
and "∀ dip. sqn (rt ξ) dip ≤ sqn (rt ξ') dip"
shows "quality_increases ξ ξ'"
unfolding quality_increases_def using assms by clarsimp

lemma quality_increasesE [elim]:
fixes dip
assumes "quality_increases ξ ξ'" and "dip ∈ kD(rt ξ)
and "[ dip ∈ kD(rt ξ); rt ξ ⊑ dip rt ξ'; sqn (rt ξ) dip ≤ sqn (rt ξ') dip ] =⇒ R dip ξ ξ'"
shows "R dip ξ ξ'"
using assms unfolding quality_increases_def by clarsimp

lemma quality_increases_rt_fresherD [dest]:
fixes ip
assumes "quality_increases ξ ξ'" and "ip∈kD(rt ξ)"
shows "rt ξ ⊑ ip rt ξ'"
using assms by auto

lemma quality_increases_sqnE [elim]:
fixes dip
assumes "quality_increases ξ ξ'" and "sqn (rt ξ) dip ≤ sqn (rt ξ') dip =⇒ R dip ξ ξ'"
shows "R dip ξ ξ'"
using assms unfolding quality_increases_def by clarsimp

lemma quality_increases_refl [intro, simp]: "quality_increases ξ ξ"
by rule simp_all

lemma strictly_fresher-quality_increases_right [elim]:
fixes σ σ' dip
assumes "rt (σ i) ⊑ dip rt (σ' nhip)" and qinc: "quality_increases (σ nhip) (σ' nhip)"
and "dip∈kD(rt (σ nhip))"
shows "rt (σ i) ⊑ dip rt (σ' nhip)"
proof -
from qinc have "rt (σ nhip) ⊑ dip rt (σ' nhip)" using ⟨dip∈kD(rt (σ nhip))⟩
by auto
with \texttt{\textlangle}\textit{rt (\sigma i)} \sqsubseteq_{\textit{dip}} \textit{rt (\sigma nhip)}\texttt{\textrangle} show \textit{?thesis} ..
qed

**Lemma kD\_quality\_increases [elim]:**

**assumes** \texttt{"i \in kD(\textit{rt } \xi)"}

and \texttt{"quality\_increases } \xi \xi'\texttt{"}

shows \texttt{"i \in kD(\textit{rt } \xi')"}

using \texttt{assms} by \texttt{auto}

**Lemma kD\_nsqn\_quality\_increases [elim]:**

**assumes** \texttt{"i \in kD(\textit{rt } \xi)"}

and \texttt{"quality\_increases } \xi \xi'\texttt{"}

shows \texttt{\"i \in kD(\textit{rt } \xi)'\"}

proof -
from \texttt{assms} have \texttt{"i \in kD(\textit{rt } \xi)"} ..
moreover with \texttt{assms} have \texttt{"rt } \xi \sqsubseteq_{\textit{i}} \textit{rt } \xi'\texttt{"} by \texttt{auto}
ultimately have \texttt{\"nsqn (\textit{rt } \xi) i \leq nsqn (\textit{rt } \xi') i\"}
using \texttt{i \in kD(\textit{rt } \xi)} by \texttt{- (erule(2) rt\_fresher\_imp\_nsqn\_le)}
with \texttt{i \in kD(\textit{rt } \xi')\texttt{show }?\textit{thesis} ..
qed

**Lemma nsqn\_quality\_increases [elim]:**

**assumes** \texttt{"i \in kD(\textit{rt } \xi)"}

and \texttt{"quality\_increases } \xi \xi'\texttt{"}

shows \texttt{\"nsqn (\textit{rt } \xi) i \leq nsqn (\textit{rt } \xi') i\"}

using \texttt{assms} by (rule kD\_nsqn\_quality\_increases [THEN conjunct2])

**Lemma kD\_nsqn\_quality\_increases\_trans [elim]:**

**assumes** \texttt{"i \in kD(\textit{rt } \xi)"}

and \texttt{"s \leq nsqn (\textit{rt } \xi) i"}

and \texttt{"quality\_increases } \xi \xi'\texttt{"}

shows \texttt{\"i \in kD(\textit{rt } \xi')\"}

proof -
from \texttt{i \in kD(\textit{rt } \xi) and \langle quality\_increases } \xi \xi'\texttt{\rangle show \"i \in kD(\textit{rt } \xi')\" ..
next
from \texttt{i \in kD(\textit{rt } \xi) and \langle quality\_increases } \xi \xi'\texttt{\rangle have \"nsqn (\textit{rt } \xi) i \leq nsqn (\textit{rt } \xi') i\" ..
with \texttt{s \leq nsqn (\textit{rt } \xi) i} show \texttt{\"s \leq nsqn (\textit{rt } \xi') i\"} by (rule le\_trans)
qed

**Lemma nsqn\_quality\_increases\_nsqn\_lt\_lt [elim]:**

**assumes** \texttt{"i \in kD(\textit{rt } \xi)"}

and \texttt{"quality\_increases } \xi \xi'\texttt{"}

and \texttt{"s < nsqn (\textit{rt } \xi) i"}

shows \texttt{\"s < nsqn (\textit{rt } \xi') i\"}

proof -
from \texttt{assms(1-2) have \"nsqn (\textit{rt } \xi) i \leq nsqn (\textit{rt } \xi') i\" ..
with \texttt{s < nsqn (\textit{rt } \xi) i} show \texttt{\"s < nsqn (\textit{rt } \xi') i\"} by \texttt{simp}
qed

**Lemma nsqn\_quality\_increases\_dhops [elim]:**

**assumes** \texttt{"i \in kD(\textit{rt } \xi)"}

and \texttt{"quality\_increases } \xi \xi'\texttt{"}

and \texttt{"nsqn (\textit{rt } \xi) i = nsqn (\textit{rt } \xi') i"}

shows \texttt{\"the (dhops (\textit{rt } \xi) i) \geq the (dhops (\textit{rt } \xi') i)\"}

using \texttt{assms unfolding quality\_increases\_def}
by \texttt{(clarsimp)} \texttt{(drule(1) bspec,clarsimp simp: rt\_fresher\_def2)}

**Lemma nsqn\_quality\_increases\_nsqn\_eq\_le [elim]:**

**assumes** \texttt{"i \in kD(\textit{rt } \xi)"}

and \texttt{"quality\_increases } \xi \xi'\texttt{"}

and \texttt{"s = nsqn (\textit{rt } \xi) i"}

shows \texttt{\"s < nsqn (\textit{rt } \xi') i \lor (s = nsqn (\textit{rt } \xi') i \land \texttt{(dhops (\textit{rt } \xi) i) \geq the (dhops (\textit{rt } \xi') i))\"}

using \texttt{assms by (metis nat\_less\_le nsqn\_quality\_increases nsqn\_quality\_increases\_dhops)}

55
lemma quality_increases_rreq_rrep_props [elim]:
  fixes sn ip hops sip
assumes qinc: "quality_increases (σ sip) (σ' sip)"
  and "1 ≤ sn"
  and *: "ip∈kD(rt (σ sip)) ∧ sn ≤ nsqn (rt (σ sip)) ip
  ∧ (nsqn (rt (σ sip)) ip = sn
  → (the (dhops (rt (σ sip)) ip) ≤ hops
  ∨ the (flag (rt (σ sip)) ip) = inv))" 
shows "ip∈kD(rt (σ' sip)) ∧ sn ≤ nsqn (rt (σ' sip)) ip
  ∧ (nsqn (rt (σ' sip)) ip = sn
  → (the (dhops (rt (σ' sip)) ip) ≤ hops
  ∨ the (flag (rt (σ' sip)) ip) = inv))"
(is "_ ∧ ?nsqnafter")
proof -
from * obtain "ip∈kD(rt (σ sip))" and "sn ≤ nsqn (rt (σ sip)) ip" by auto
from quality_increases (σ sip) (σ' sip)
  have "sqn (rt (σ sip)) ip ≤ sqn (rt (σ' sip)) ip" ..
from quality_increases (σ sip) (σ' sip) and ip∈kD (rt (σ sip))
  have "ip∈kD (rt (σ' sip))" ..
from (sn ≤ nsqn (rt (σ sip)) ip) have ?nsqnafter
proof
  assume "sn < nsqn (rt (σ sip)) ip"
  also from ip∈kD(rt (σ sip))
  have "... ≤ nsqn (rt (σ' sip)) ip" ..
finally have "sn < nsqn (rt (σ' sip)) ip" .
thus ?thesis by simp
next
  assume "sn = nsqn (rt (σ sip)) ip"
  with ip∈kD(rt (σ sip)) and quality_increases (σ sip) (σ' sip)
  have "sn < nsqn (rt (σ' sip)) ip
  ∨ (sn = nsqn (rt (σ' sip)) ip
  ∧ the (dhops (rt (σ' sip)) ip) ≤ the (dhops (rt (σ sip)) ip))" ..
  hence "sn < nsqn (rt (σ' sip)) ip
  ∨ (nsqn (rt (σ' sip)) ip = sn ∧ (the (dhops (rt (σ' sip)) ip) ≤ hops
  ∨ the (flag (rt (σ' sip)) ip) = inv))"
proof
  assume "sn < nsqn (rt (σ' sip)) ip" thus ?thesis ..
next
  assume "sn = nsqn (rt (σ' sip)) ip
  ∧ the (dhops (rt (σ sip)) ip) ≥ the (dhops (rt (σ' sip)) ip)"
  hence "sn = nsqn (rt (σ' sip)) ip"
  and "the (dhops (rt (σ' sip)) ip) ≤ the (dhops (rt (σ sip)) ip)" by auto
from * and (sn = nsqn (rt (σ sip)) ip) have "the (dhops (rt (σ sip)) ip) ≤ hops
  ∨ the (flag (rt (σ sip)) ip) = inv"
  by simp
thus ?thesis
proof
  assume "the (dhops (rt (σ sip)) ip) ≤ hops"
  with the (dhops (rt (σ' sip)) ip) ≤ the (dhops (rt (σ sip)) ip):
  have "the (dhops (rt (σ' sip)) ip) ≤ hops" by simp
  with (sn = nsqn (rt (σ' sip)) ip) show ?thesis by simp
next
  assume "the (flag (rt (σ sip)) ip) = inv"
  with (ip∈kD(rt (σ sip))): have "sqn (rt (σ sip)) ip = sqn (rt (σ sip)) ip - 1" ..
  with (sn ≥ 1) and (sn = nsqn (rt (σ sip)) ip)
  have "sqn (rt (σ sip)) ip > 1" by simp
from (ip∈kD(rt (σ' sip))): show ?thesis
proof (rule vD_or_iD)
assume "ip ∈ ID(rt (σ’ sip))"
hence "the (flag (rt (σ’ sip)) ip) = inv" ..
with \( sn = nsqn (rt (σ’ sip)) \) show ?thesis
  by simp

next

assume "ip ∈ vD(rt (σ’ sip))"
hence "nsqn (rt (σ’ sip)) ip = sqn (rt (σ’ sip)) ip" ..
with \( sn = nsqn (rt (σ’ sip)) \) show "ip ∈ kD(rt (σ’ sip)) ∧ ?nsqnafter" ..

qed

lemma quality_increases_rreq_rrep_props':
  fixes sn ip hops sip
  assumes "∀ j. quality_increases (σ j) (σ’ j)"
  and "1 ≤ sn"
  and \( *: \) "ip ∈ kD(rt (σ sip)) ∧ sn ≤ nsqn (rt (σ sip)) ip
  \( \wedge (nsqn (rt (σ sip)) ip = sn \) → \) (the (dhops (rt (σ sip)) ip) ≤ hops
  \( \lor \) the (flag (rt (σ sip)) ip) = inv))"
  shows "ip ∈ kD(rt (σ’ sip)) ∧ sn ≤ nsqn (rt (σ’ sip)) ip
  \( \wedge (nsqn (rt (σ’ sip)) ip = sn \) → \) (the (dhops (rt (σ’ sip)) ip) ≤ hops
  \( \lor \) the (flag (rt (σ’ sip)) ip) = inv))"

  proof -
  from assms(1) have "quality_increases (σ sip) (σ’ sip)" ..
  thus ?thesis using assms(2-3) by (rule quality_increases_rreq_rrep_props)
  qed

lemma rteq_quality_increases:
  assumes "∀ j. j ≠ i → quality_increases (σ j) (σ’ j)"
  and "rt (σ i) = rt (σ i)"
  shows "∀ j. quality_increases (σ j) (σ’ j)"
  using assms by (clarsimp simp (metis order_refl quality_increasesI rt_fresher_refl))

definition msg_fresh :: "(ip ⇒ state) ⇒ msg ⇒ bool"
where "msg_fresh σ m ≡
  case m of Rreq hopsc _ _ _ oipc osnc sipc ⇒ osnc ≥ 1 ∧ (sipc ≠ oipc → oipc∈kD(rt (σ sipc)) ∧ nsqn (rt (σ sipc)) oipc ≥ osnc
  \( \wedge (nsqn (rt (σ sipc)) oipc = osnc \) → \) (hopsc ≥ the (dhops (rt (σ sipc)) oipc)
  \( \lor \) the (flag (rt (σ sipc)) oipc) = inv))
  \| Rrep dipc dipc dsnc _ sipc ⇒ dsnc ≥ 1 ∧ (sipc ≠ dipc → dipc∈kD(rt (σ sipc)) ∧ nsqn (rt (σ sipc)) dipc ≥ dsnc
  \( \wedge (nsqn (rt (σ sipc)) dipc = dsnc \) → \) (hopsc ≥ the (dhops (rt (σ sipc)) dipc)
  \( \lor \) the (flag (rt (σ sipc)) dipc) = inv))
  \| Rerr destsc sipc ⇒ (∀ ripc∈dom(destsc). ripc∈kD(rt (σ sipc)) ∧ the (destsc ripc) - 1 ≤ nsqn (rt (σ sipc)) ripc)
  \| _ ⇒ True"
lemma msg_fresh [simp]:
"\( \forall \text{hops rreqid dip dsn oip osn sip} \).
\( \text{msg_fresh } \sigma \text{ (Rreq hops rreqid dip dsn oip osn sip)} = \)
\( (\text{osn} \geq 1 \land (\text{sip} \neq \text{oip} \implies \text{oip} \in \text{kD(rt(\sigma \text{ sip})))) \)
\( \land \text{nsqn rt(\sigma \text{ sip})) oip} \geq \text{osn} \)
\( \land (\text{nsqn rt(\sigma \text{ sip})) oip} = \text{osn} \)
\( \implies (\text{hops} \geq \text{the dhops rt(\sigma \text{ sip})) oip}))\)"


lemma msg_fresh_inc_sn [simp, elim]:
"msg_fresh \sigma m = rreq_rrep_sn m"
by (cases m) simp_all

lemma recv_msg_fresh_inc_sn [simp, elim]:
"orecvmsg (msg_fresh) \sigma m = rrecvmsg rreq_rrep_sn m"
by (cases m) simp_all

lemma rreq_nsqn_is_fresh [simp]:
fixes \( \sigma \text{ msg hops rreqid dip dsn oip osn sip} \)
assumes "rreq_rrep_fresh (rt(\sigma \text{ sip})) (Rreq hops rreqid dip dsn oip osn sip)"
and "rreq_rrep_sn (Rreq hops rreqid dip dsn oip osn sip)"
shows "msg_fresh \sigma (Rreq hops rreqid dip dsn oip osn sip)"
(is "msg_fresh \sigma \?msg")
proof -
  let \( ?rt = \text{rt(\sigma \text{ sip})} \)
  from assms(2) have "1 \leq \text{osn}" by simp
thus ?thesis
unfolding msg_fresh_def
proof (simp only: msg.case, intro conjI impI)
  assume "\text{sip} \neq \text{oip}"
  with assms(1) show "\text{oip} \in \text{kD(?rt)}" by simp
next
  assume "\text{sip} \neq \text{oip}"
  and "\text{nsqn ?rt oip} = \text{osn}"
  show "the (dhops ?rt oip) \leq \text{hops} \lor \text{the (flag ?rt oip) = inv}"
  proof (cases "\text{oip} \in \text{vD(?rt)}")
    assume "\text{oip} \in \text{vD(?rt)}"
    hence "\text{nsqn ?rt oip} = \text{sqn ?rt oip} ..
    with (\text{nsqn ?rt oip} = \text{osn}) have "\text{sqn ?rt oip} = \text{osn}" by simp
    with assms(1) and (\text{sip} \neq \text{oip}) have "the (dhops ?rt oip) \leq \text{hops}"
    by simp
    thus ?thesis ..
  next
  assume "\text{oip} \notin \text{vD(?rt)}"
  moreover from assms(1) and (\text{sip} \neq \text{oip}) have "\text{oip} \in \text{kD(?rt)}" by simp
  ultimately have "\text{oip} \in \text{iD(?rt)}" by auto
  hence "\text{the (flag ?rt oip) = inv} ..
  thus ?thesis ..
  qed
next
  assume "\text{sip} \neq \text{oip}"
with assms(1) have "osn ≤ sqn ?rt oip" by auto
thus "osn ≤ nsqn (rt (σ sip)) oip"
proof (rule nat_le_eq_or_lt)
  assume "osn < sqn ?rt oip"
  hence "osn ≤ sqn ?rt oip - 1" by simp
  also have "... ≤ nsqn ?rt oip" by (rule sqn_nsqn)
  finally show "osn ≤ nsqn ?rt oip" .
next
  assume "osn = sqn ?rt oip"
  with assms(1) and ⟨sip ≠ oip⟩ have "oip∈kD(?rt)" and "the (flag ?rt oip) = val"
    by auto
  hence "nsqn ?rt oip = sqn ?rt oip" ..
  thus "osn ≤ nsqn ?rt oip" by simp
qed

lemma rrep_nsqn_is_fresh [simp]:
fixes σ msg hops dip dsn oip sip
assumes "rreq_rrep_fresh (rt (σ sip)) (Rrep hops dip dsn oip sip)"
  and "rreq_rrep_sn (Rrep hops dip dsn oip sip)"
shows "msg_fresh σ (Rrep hops dip dsn oip sip)"
(is "msg_fresh σ ?msg")
proof -
  let ?rt = "rt (σ sip)"
  from assms have "sip ≠ dip → dip∈kD(?rt) ∧ sqn ?rt dip = dsn ∧ the (flag ?rt dip) = val"
    by simp
  hence "sip ≠ dip → dip∈kD(?rt) ∧ nsqn ?rt dip ≥ dsn"
    by clarsimp
  with assms show "msg_fresh σ ?msg"
    by clarsimp
qed

lemma rerr_nsqn_is_fresh [simp]:
fixes σ msg dests sip
assumes "rerr_invalid (rt (σ sip)) (Rerr dests sip)"
shows "msg_fresh σ (Rerr dests sip)"
(is "msg_fresh σ ?msg")
proof -
  let ?rt = "rt (σ sip)"
  from assms have *(∀ rip∈dom(dests). (rip∈iD(rt (σ sip))
      ∧ the (dests rip) = sqn (rt (σ sip)) rip))"
    by clarsimp
  have *(∀ rip∈dom(dests). (rip∈kD(rt (σ sip)))
      ∧ the (dests rip) - 1 ≤ nsqn (rt (σ sip)) rip))"
    by clarsimp
  proof
    fix rip
    assume "rip ∈ dom dests"
    with * have "rip∈iD(rt (σ sip))" and "the (dests rip) = sqn (rt (σ sip)) rip"
      by auto
    from this(2) have "the (dests rip) - 1 = sqn (rt (σ sip)) rip - 1" by simp
    also have "... ≤ nsqn (rt (σ sip)) rip" by (rule sqn_nsqn)
    finally have "the (dests rip) - 1 ≤ nsqn (rt (σ sip)) rip" .
    with ⟨rip∈iD(rt (σ sip))⟩
      show "rip∈kD(rt (σ sip)) ∧ the (dests rip) - 1 ≤ nsqn (rt (σ sip)) rip"
        by clarsimp
    qed
  thus "msg_fresh σ ?msg"
    by simp
qed

lemma
lemma quality_increases_msg_fresh [elim]:
assumes qinc: "∀j. quality_increases (σ j) (σ' j)"
and "msg_fresh σ m"
shows "msg_fresh σ' m"
using assms(2)

proof (cases m)
  fix hops rreqid dip dsn dk oip osn sip
  assume [simp]: "m = Rreq hops rreqid dip dsn dk oip osn sip"
  and "msg_fresh σ m"
  then have "osn ≥ 1" and "sip = oip ∨ (oip∈kD(rt (σ sip)) ∧ osn ≤ nsqn (rt (σ sip))) oip
  ∧ (nsqn (rt (σ sip))) oip = osn
  → (the (dhops (rt (σ sip)) oip) ≤ hops
  ∨ the (flag (rt (σ sip)) oip) = inv))"
  by auto
  from this(2) show ?thesis
proof
  assume "sip = oip" with ⟨osn ≥ 1⟩ show ?thesis by simp
next
  assume "oip∈kD(rt (σ sip)) ∧ osn ≤ nsqn (rt (σ sip))) oip
  ∧ (nsqn (rt (σ sip))) oip = osn
  → (the (dhops (rt (σ sip)) oip) ≤ hops
  ∨ the (flag (rt (σ sip)) oip) = inv))"
moreover from qinc have "quality_increases (σ sip) (σ' sip)"..
ultimately have "oip∈kD(rt (σ' sip)) ∧ osn ≤ nsqn (rt (σ' sip))) oip
  ∧ (nsqn (rt (σ' sip))) oip = osn
  → (the (dhops (rt (σ' sip)) oip) ≤ hops
  ∨ the (flag (rt (σ' sip)) oip) = inv))"
using ⟨osn ≥ 1⟩ by (rule quality_increases_rreq_rrep_props [rotated 2])
with ⟨osn ≥ 1⟩ show "msg_fresh σ' m"
by (clarsimp)
qued

next
  fix hops dip dsn oip sip
  assume [simp]: "m = Rrep hops dip dsn oip sip"
  and "msg_fresh σ m"
  then have "dsn ≥ 1" and "sip = dip ∨ (dip∈kD(rt (σ sip)) ∧ dsn ≤ nsqn (rt (σ sip))) dip
  ∧ (nsqn (rt (σ sip))) dip = dsn
  → (the (dhops (rt (σ sip)) dip) ≤ hops
  ∨ the (flag (rt (σ sip)) dip) = inv))"
  by auto
  from this(2) show "?thesis"
proof
  assume "sip = dip" with ⟨dsn ≥ 1⟩ show ?thesis by simp
next
  assume "dip∈kD(rt (σ sip)) ∧ dsn ≤ nsqn (rt (σ sip)) dip
  ∧ (nsqn (rt (σ sip))) dip = dsn
  → (the (dhops (rt (σ sip)) dip) ≤ hops
  ∨ the (flag (rt (σ sip)) dip) = inv))"
moreover from qinc have "quality_increases (σ sip) (σ' sip)"..
ultimately have "dip∈kD(rt (σ' sip)) ∧ dsn ≤ nsqn (rt (σ' sip))) dip
  ∧ (nsqn (rt (σ' sip))) dip = dsn
  → (the (dhops (rt (σ' sip)) dip) ≤ hops
  ∨ the (flag (rt (σ' sip)) dip) = inv))"
using ⟨dsn ≥ 1⟩ by (rule quality_increases_rreq_rrep_props [rotated 2])
with ⟨dsn ≥ 1⟩ show "msg_fresh σ' m"
by clarsimp
qed

next
  fix dests sip
  assume [simp]: "m = Rerr dests sip"
  and "msg_fresh σ m"
  then have "∀rip∈dom(dests). rip∈kD(rt (σ sip))
  ∧ the (dests rip) - 1 ≤ nsqn (rt (σ sip)) rip"
0.9 The ‘open’ AODV model

theory OAodv
imports Aodv ../AWN/OAWN_SOS_Labels ../AWN/OAWN_Convert
begin

Definitions for stating and proving global network properties over individual processes.

definition σAODV' :: "((ip ⇒ state) × ((state, msg, pseqp, pseqp label) seqp)) set"
where "σAODV' ≡ {λi. aodv_init i, ΓAODV PAdov}"

abbreviation opaodv
:: "ip ⇒ ((ip ⇒ state) × (state, msg, pseqp, pseqp label) seqp, msg seq_action) automaton"
where "opaodv i ≡ (| init = σAODV', trans = oseqp_sos ΓAODV i |)"

lemma initiali_aodv [intro!, simp]: "initiali i (init (opaodv i)) (init (paodv i))"
unfolding σAODV'_def by rule simp_all

lemma oao dov_control_within [simp]: "control_within ΓAODV (init (opaodv i))"
unfolding σAODV'_def by (rule control_withinI) (auto simp del: ΓAODV_sims)

lemma σAODV'_labels [simp]: "(σ, p) ∈ σAODV' =⇒ labels ΓAODV p = {PAodv-:0}" 
unfolding σAODV'_def by simp

lemma oao dov_init_kD_empty [simp]:
"(σ, p) ∈ σAODV' =⇒ kD (rt (σ i)) = {}"
unfolding σAODV'_def kD_def by simp

lemma oao dov_init_vD_empty [simp]:
"(σ, p) ∈ σAODV' =⇒ vD (rt (σ i)) = {}"
unfolding σAODV'_def vD_def by simp

lemma oadov_trans: "trans (opaodv i) = oseqp_sos ΓAODV i"
by simp

declare oseq_invariant_ctermsI [OF aodv_wf oadov_control_within aodv_simple_labels aodv_trans, cterms_intros]
oseq_step_invariant_ctermsI [OF aodv_wf oadov_control_within aodv_simple_labels aodv_trans, cterms_intros]
end

0.10 Global invariant proofs over sequential processes

theory Global_Invariants
imports Seq_Invariants
Aodv_Predicates
Fresher
begin

lemma other_quality_increases [elim]:
  assumes "other quality_increases I σ σ'"
  shows "∀ j. quality_increases (σ j) (σ' j)"
  using assms by (rule, clarsimp) (metis quality_increases_refl)

lemma weaken_otherwith [elim]:
  fixes m
  assumes *: "otherwith P I (orecvmsg Q) σ σ' a" and weakenP: "∀ σ m. P σ m =⇒ P' σ m" and weakenQ: "∀ σ m. Q σ m =⇒ Q' σ m"
  shows "otherwith P' I (orecvmsg Q') σ σ' a"
  proof
    fix j
    assume "j /∈ I"
    with * have "P (σ j) (σ' j)" by auto
    thus "P' (σ j) (σ' j)" by (rule weakenP)
  next
    from * have "orecvmsg Q σ a" by auto
    thus "orecvmsg Q' σ a" by rule (erule weakenQ)
  qed

lemma oreceived_msg_inv:
  assumes other: "∀ σ σ' m. [ P σ m; other Q {i} σ σ' ] =⇒ P σ' m" and local: "∀ σ m. P σ m =⇒ P (σ(i := σ i(msg := m))) m"
  shows "opaodv i |= (otherwith Q {i} (orecvmsg P), other Q {i} →) onl Γ AODV (λ(σ, l). l ∈ {PAodv-:1} −→ P σ (msg (σ i)))"
  proof
    (inv_cterms, intro impI)
    fix σ σ' l
    assume "l = PAodv-:1 −→ P σ (msg (σ i))"
    and "l = PAodv-:1"
    and "other Q {i} σ σ'"
    from this(1-2) have "P σ (msg (σ i))" ..
    hence "P σ' (msg (σ i))" using 'other Q {i} σ σ'" by (rule local)
    moreover from 'other Q {i} σ σ' have "σ' i = σ i" ..
    ultimately show "P σ' (msg (σ' i))" by simp
  next
    fix σ σ' msg
    assume "otherwith Q {i} (orecvmsg P) σ σ' (receive msg)"
    and "σ' i = σ i(msg := msg)"
    from this(1) have "P σ msg"
    and "∀ j. j ≠ i −→ Q (σ j) (σ' j)" by auto
    from this(1) have "P (σ(i := σ i(msg := msg))) msg" by (rule local)
    thus "P σ' msg"
  proof (rule other)
    from "σ' i = σ i(msg := msg)" and ∀ j. j ≠ i −→ Q (σ j) (σ' j)
    show "other Q {i} σ (i := σ i(msg := msg))) σ'" by - (rule otherI, auto)
  qed

 qed

(Equivalent to) Proposition 7.27

lemma local_quality_increases:
  "paodv i |= (recmsg req_rrep_sn →) onll Γ AODV (λ((ξ, _), _, (ξ', _)). quality_increases ξ ξ')"
  proof (rule step_invariantI)
    fix s a s'
    assume sr: "s ∈ reachable (paodv i) (recmsg req_rrep_sn)"
    and tr: "(s, a, s') ∈ trans (paodv i)"
and rm: "recvmsg rreq_rrep_sn a"
from sr have srTT: "s ∈ reachable (paodv i) TT" ..
from route_tables_fresher sr rm have "onll Γ_AODV (λ((ξ, _), (ξ', _)), ∀ dip∈kD (rt ξ, rt ξ'∈dip) (s, a, a'))" by (rule step_invariantD)
moreover from known_destinations_increase srTT tr TT_True have "onll Γ_AODV (λ((ξ, _), (ξ', _)), kD (rt ξ) ⊆ kD (rt ξ')) (s, a, a')" by (rule step_invariantD)
moreover from sqns_increase srTT tr TT_True have "onll Γ_AODV (λ((ξ, _), (ξ', _)), ∀ ip. sqn (rt ξ) ip ≤ sqn (rt ξ') ip) (s, a, a')" by (rule step_invariantD)
ultimately show "onll Γ_AODV (λ((ξ, _), (ξ', _)), quality_increases ξ ξ') (s, a, a')" unfolding onll_def by auto qed
lemmas olocal_quality_increases = open_seq_step_invariant [OF local_quality_increases initiali_aodv oaodv_trans aodv_trans, simplified seqll_onll_swap]

lemma quality_increases:
"opaodv i |=A (otherwith quality_increases {i} (orecvmsg (λ_. rreq_rrep_sn)), other quality_increases {i} →) onll Γ_AODV (λ((ξ, _), (ξ', _)), ∀ j. quality_increases (σ j) (σ' j))"
(is "_ |=A (?S, _ →)")
proof (rule onll_ostep_invariantI, simp)
fix σ p l a σ' p' l'
assume or: "((σ, p) ∈ orreachable (opaodv i) ?S (other quality_increases {i}))"
and ll: "l ∈ labels Γ_AODV p"
and "?S σ σ' a"
and tr: "((σ, p), a, (σ', p')) ∈ oseqp_sos Γ_AODV i"
and ll': "l' ∈ labels Γ_AODV p'"
from this(1-3) have "orecvmsg (λ_. rreq_rrep_sn) σ a" by (auto dest!: oreachable_weakenE [where QS="act (recvmsg rreq_rrep_sn)" and QU="other quality_increases {i}"], otherwith_actionD)
with or have crw: "((σ, p) ∈ orreachable (opaodv i) (act (recvmsg rreq_rrep_sn)) (other quality_increases {i}))"
by (erule orachable_weakenE, auto)
with tr ll ll' and orecvmsg (λ_. rreq_rrep_sn) σ a have "quality_increases (σ i) (σ' i)"
by (drule onll_ostep_invariantD [OF olocal_quality_increases], auto simp: seqll_def)
with (?S σ σ' a) show "∀ j. quality_increases (σ j) (σ' j)"
by (auto dest!: otherwith_syncD)
qed

lemma rreq_rrep_nsqn_fresh_any_step_invariant:
"opaodv i |=A (act (recvmsg rreq_rrep_sn), other A {i} →) onll Γ_AODV (λ((ξ, _), a, _). anycast (msg_fresh σ a))"
proof (rule ostep_invariantI, simp del: act_simp)
fix σ p a σ' p'
assume or: "((σ, p) ∈ orreachable (opaodv i) (act (recvmsg rreq_rrep_sn)) (other A {i}))"
and "((σ, p), a, (σ', p')) ∈ oseqp_sos Γ_AODV i"
and recv: "act (recvmsg rreq_rrep_sn) σ σ' a"
obtain l l' where "l ∈ labels Γ_AODV p" and "l' ∈ labels Γ_AODV p'"
by (metis aodv_ex_label)
from (((σ, p), a, (σ', p')) ∈ oseqp_sos Γ_AODV i) have tr: "((σ, p), a, (σ', p')) ∈ trans (opaodv i)" by simp
have "anycast (rreq_rrep_fresh (rt (σ i))) a" proof - have "opaodv i |=A (act (recvmsg rreq_rrep_sn), other A {i} →)
63
onll $\Gamma_{AODV}$ (seqll $i \ (\lambda((\xi, _), a, _). \text{anycast} \ (\text{rreq}_rrep\_\text{fresh} \ (\text{rt} \ \xi)) \ a))$

by (rule ostep_invariant_weakenE [OF open_seq_step_invariant [OF rreq_rrep_fresh_any_step_invariant initiali_aodv, simplified seqll_onll_swap]]) auto

hence "onll $\Gamma_{AODV}$ (seqll $i \ (\lambda((\xi, _), a, _). \text{anycast} \ (\text{rreq}_rrep\_\text{fresh} \ (\text{rt} \ \xi)) \ a))$

((\sigma, p), a, (\sigma', p'))"

using or tr recv by - (erule(4) ostep_invariantE)

thus ?thesis

using $l \in \text{labels} \ \Gamma_{AODV} \ p$ and $l' \in \text{labels} \ \Gamma_{AODV} \ p'$ by auto

qed

moreover have "anycast (rerr_invalid (rt (\sigma i)) a)"

proof -

have "opaodv $i \models_A \ (\text{act} (\text{recvmsg} \ \text{rreq}_rrep\_\text{sn}), \text{other} \ A \ \{i\} \rightarrow)$

onll $\Gamma_{AODV}$ (seqll $i \ (\lambda((\xi, _), a, _). \text{anycast} \ (\text{rerr}_\text{invalid} \ (\text{rt} \ \xi)) \ a))$

by (rule ostep_invariant_weakenE [OF open_seq_step_invariant [OF rerr_invalid_any_step_invariant initiali_aodv, simplified seqll_onll_swap]]) auto

hence "onll $\Gamma_{AODV}$ (seqll $i \ (\lambda((\xi, _), a, _). \text{anycast} \ (\text{rerr}_\text{invalid} \ (\text{rt} \ \xi)) \ a))$

((\sigma, p), a, (\sigma', p'))"

using or tr recv by - (erule(4) ostep_invariantE)

thus ?thesis

using $l \in \text{labels} \ \Gamma_{AODV} \ p$ and $l' \in \text{labels} \ \Gamma_{AODV} \ p'$ by auto

qed

moreover have "anycast (\text{rreq}_rrep\_\text{sn} a)"

proof -

from or tr recv

have "onll $\Gamma_{AODV}$ (seqll $i \ (\lambda(_, a, _). \text{anycast} \ \text{rreq}_rrep\_\text{sn} \ a)) \ ((\sigma, p), a, (\sigma', p'))"

by (rule ostep_invariantE [OF open_seq_step_invariant [OF rreq_rrep_sn_any_step_invariant initiali_aodv, oadv_trans aodv_trans, simplified seqll_onll_swap]])

thus ?thesis

using $l \in \text{labels} \ \Gamma_{AODV} \ p$ and $l' \in \text{labels} \ \Gamma_{AODV} \ p'$ by auto

qed

moreover have "anycast (\lambda m. \text{not}_\text{Pkt} \ m \rightarrow \text{msg}_\text{sender} \ m = i) a"

proof -

have "opaodv $i \models_A \ (\text{act} (\text{recvmsg} \ \text{rreq}_rrep\_\text{sn}), \text{other} \ A \ \{i\} \rightarrow)$

onll $\Gamma_{AODV}$ (seqll $i \ (\lambda((\xi, _), a, _). \text{anycast} \ (\lambda m. \text{not}_\text{Pkt} \ m \rightarrow \text{msg}_\text{sender} \ m = i) \ a))$

by (rule ostep_invariant_weakenE [OF open_seq_step_invariant [OF sender_ip_valid initiali_aodv, simplified seqll_onll_swap]]) auto

hence "onll $\Gamma_{AODV}$ (seqll $i \ (\lambda((\xi, _), a, _). \text{anycast} \ (\lambda m. \text{not}_\text{Pkt} \ m \rightarrow \text{msg}_\text{sender} \ m = i) \ a))$

((\sigma, p), a, (\sigma', p'))"

using or tr recv by - (erule(3) onll_ostep_invariantD, auto)

thus ?thesis using $l \in \text{labels} \ \Gamma_{AODV} \ p$ and $l' \in \text{labels} \ \Gamma_{AODV} \ p'$

qed

ultimately have "anycast (\text{msg}_\text{fresh} \ \sigma) a"

by (simp_all add: anycast_def del: msg_fresh

split: seq.action.split_asm msg.split_asm) simp_all

thus "onll $\Gamma_{AODV}$ (\lambda((\sigma, _), a, _). \text{anycast} \ (\text{msg}_\text{fresh} \ \sigma) \ a) \ ((\sigma, p), a, (\sigma', p'))"

by auto

qed

lemma oreceived_rreq_rrep_msqn_fresh_inv:

"opaodv $i \models (\text{otherwith} \ \text{quality}_\text{increases} \ \{i\} \ \text{orecvmsg} \ \text{msg}_\text{fresh}), \text{other} \ \text{quality}_\text{increases} \ \{i\} \rightarrow)$

onl $\Gamma_{AODV}$ (\lambda(\sigma, 1). \ l \in \{PAodv:-1\} \rightarrow \text{msg}_\text{fresh} \ \sigma \ (\text{msg} \ (\text{msg} \ (\text{msg}_\text{i}))))"

proof (rule oreceived_msg_inv)

fix $\sigma \ \sigma'$

assume *: "\text{msg}_\text{fresh} \ \sigma \ m"

and "\text{other} \ \text{quality}_\text{increases} \ \{i\} \ \sigma'"
from this(2) have "∀ j. quality_increases (σ j) (σ’ j)" ..
thus "msg_fresh σ m" using * ..

next
fix σ m
assume "msg_fresh σ m"
thus "msg_fresh (σ(ι := σ i|msg := m)) m"
proof (cases m)
fix dests sip
assume "m = Rerr dests sip"
with msg_fresh σ m: show ?thesis
by auto
qed auto

qed

lemma quality_increases_nsqn_fresh:
"opaodv i |→ (otherwith quality_increases {i} (orecvmsg msg_fresh),
other quality_increases {i} →)
onl Γ _AODV _ (λ(_σ, _, _). (σ, _, _)). ∀ j. quality_increases (σ j) (σ’ j)"
by (rule ostep_invariant_weakenE [OF oquality_increases]) auto

lemma oosn_rreq:
"opaodv i |→ (otherwith quality_increases {i} (orecvmsg msg_fresh),
other quality_increases {i} →)
onl Γ _AODV _ (seql _ (λ(l, l). 1 ∈ {PAodv::-4, PAodv::-5} ∪ {PRreq::n ln nTrue} → 1 ≤ osn l))"
by (rule oinvariant_weakenE [OF open_seq_invariant [OF oosn_rreq initiali_aodv]])
(auto simp: seql_onl_swap)

lemma rreq_sip:
"opaodv i |→ (otherwith quality_increases {i} (orecvmsg msg_fresh),
other quality_increases {i} →)
onl Γ _AODV _ (λ(l, ω). l ∈ {PAodv::-4, PAodv::-5, PRreq::-0, PRreq::-2} ∧ sip (σ i) ≠ oip (σ i))
→ oip (σ i) ∈ kD(rt (σ (sip (σ i))))
∧ nsqn (rt (σ (sip (σ i)))) (oip (σ i)) ≥ osn (σ i)
∧ (nsqn (rt (σ (sip (σ i)))) (oip (σ i)) = osn (σ i))
→ (hops (σ i) ≥ the (dhops (rt (σ (sip (σ i)))) (oip (σ i))))
∨ the (flag (rt (σ (sip (σ i)))) (oip (σ i))) = inv)
(is _ _ (λl, ω). l ∈ {PAodv::-4, PAodv::-5, PRreq::-0, PRreq::-2} ∧ sip (σ i) ≠ oip (σ i))"
proof (inv_cterms inv add: oseq_step_invariant_sterms [OF oquality_increases_nsqn_fresh]
aodv_wf oadv_trans)
onl oinvariant_sterms [OF aodv_wf oreceived_rreq_rrep_nsqn_fresh_inv]
onl oinvariant_sterms [OF aodv_wf oosn_rreq]
simp add: seqlsimp
simp del: One_nat_def, rule implI)
fix σ σ’ p l
assume "(σ, p) ∈ reachable (opaodv i) σ? σ?"
and "1 ∈ labels Γ _AODV _ p"
and pre:
"(l = PAodv::-4 ∨ l = PAodv::-5 ∨ l = PRreq::-0 ∨ l = PRreq::-2) ∧ sip (σ i) ≠ oip (σ i)
→ oip (σ i) ∈ kD(rt (σ (sip (σ i))))
∧ osn (σ i) ≤ nsqn (rt (σ (sip (σ i)))) (oip (σ i))
∧ (nsqn (rt (σ (sip (σ i)))) (oip (σ i)) = osn (σ i))
→ the (dhops (rt (σ (sip (σ i)))) (oip (σ i))) ≤ hops (σ i)
∨ the (flag (rt (σ (sip (σ i)))) (oip (σ i))) = inv)"
and "other quality_increases {i} σ σ’"
and hyp: "(l = PAodv::-4 ∨ l = PAodv::-5 ∨ l = PRreq::-0 ∨ l = PRreq::-2) ∧ sip (σ’ i) ≠ oip (σ’ i)"
(is "labels ∧ sip (σ’ i) ≠ oip (σ’ i))"
from this(4) have "σ’ i = σ i" ..
with hyp have hyp’: "labels ∧ sip (σ i) ≠ oip (σ i)" by simp
show "oip (σ i) ∈ kD(rt (σ (sip (σ i))))
∧ osn (σ i) ≤ nsqn (rt (σ (sip (σ i)))) (oip (σ i))
∧ (nsqn (rt (σ (sip (σ i)))) (oip (σ i)) = osn (σ i))
→ the (dhops (rt (σ (sip (σ i)))) (oip (σ i))) ≤ hops (σ i)
∨ the (flag (rt (σ (sip (σ i)))) (oip (σ i))) = inv)"
proof (cases "sip (σ i) = i")
  assume "sip (σ i) ≠ i"
  from other_quality_increases {i} σ σ'
    have "quality_increases (σ (sip (σ i))) (σ' (sip (σ' i)))"
    by (rule otherE) (clarsimp simp: sip (σ i) ≠ i)
  moreover from ⟨(σ, p)⟩ ∈ oreachable (opaodv i) ?S ?U | l ∈ labels Γ_AODV p | and hyp
    have "1 ≤ osn (σ' i)"
    by (auto dest!: onl_oinvariant_weakenD [OF oosn_rreq]
        simp add: seqlsimp (auto simp del: One_nat_def, rule impI)
    from this(4) have "σ' i = σ i"
  ultimately show ?thesis
  by (rule quality_increases_rreq_rrep_props)
next
  assume "sip (σ i) = i" thus ?thesis
  using (σ' i = σ i) hyp and pre by auto
qed

lemma odsn_rrep:
  "opaodv i ⊢ (otherwith quality_increases {i} (orecmsg msg_fresh),
    other_quality_increases {i} →)
  onl Γ_AODV (seql i (λ(τ, l). l ∈ {PAodv-:6, PAodv-:7} ∪ {PRrep-:n|n. True} → 1 ≤ dsn ξ))"
by (rule onl_oinvariant_sterms [OF open_seq_invariant [OF dsn_rrep initial_aodv]])
(auto simp: seql_onl_swap)

lemma rrep_sip:
  "opaodv i ⊢ (otherwith quality_increases {i} (orecmsg msg_fresh),
    other_quality_increases {i} →)
  onl Γ_AODV (λ(σ, l).
    (l ∈ {PAodv-:6, PAodv-:7, PRrep-:0, PRrep-:1} ∧ sip (σ i) ≠ dip (σ i)) →
    dip (σ i) ∈ kD(rt (σ (sip (σ i))))
    ∧ nsqn (rt (σ (sip (σ i)))) (dip (σ i)) ≥ dsn (σ i)
    ∧ (nsqn (rt (σ (sip (σ i)))) (dip (σ i)) = dsn (σ i)) →
    (hops (σ i) ≥ the (dhops rt ((σ i))) (dip (σ i)))
    ∧ (the (flag rt ((σ i))) (dip (σ i))) = inv)"
(is "_ _ (⊤, ⊤) _ _")

proof (intro conjE allE not_allE) assume "(σ, p) ∈ oreachable (opaodv i) ?S ?U"
  and "l ∈ labels Γ_AODV p | and pre: 
    "((l = PAodv-:6 ∨ l = PAodv-:7 ∨ l = PRrep-:0 ∨ l = PRrep-:1) ∧ sip (σ i) ≠ dip (σ i)) →
    dip (σ i) ∈ kD(rt (σ (sip (σ i))))
    ∧ dsn (σ i) ≤ nsqn (rt (σ (sip (σ i)))) (dip (σ i))
    ∧ (nsqn (rt (σ (sip (σ i)))) (dip (σ i)) = dsn (σ i)) →
    (hops (σ i) ≥ the (dhops rt ((σ i))) (dip (σ i)))
    ∧ (the (flag rt ((σ i))) (dip (σ i))) = inv)"
  and "other_quality_increases {i} σ σ'"
  and hyp: "((l = PAodv-:6 ∨ l = PAodv-:7 ∨ l = PRrep-:0 ∨ l = PRrep-:1) ∧ sip (σ' i) ≠ dip (σ' i))" 
(is "?labels ∧ sip (σ' i) ≠ dip (σ' i)")
  from this(4) have "σ' i = σ i" ...
  with hyp have hyp': "?labels ∧ sip (σ i) ≠ dip (σ i)" by simp
  show "dip (σ' i) ∈ kD(rt (σ' (sip (σ' i))))
    ∧ dsn (σ' i) ≤ nsqn (rt (σ' (sip (σ' i)))) (dip (σ' i))"
(cases "sip (σ i) = i")
assume "sip (σ i) ≠ i"
from (other_quality_increases {i} σ σ')
  have "quality_increases (σ (sip (σ i))) (σ (sip (σ i)))"
  by (rule quality_increases_rreq_rrep_props)
moreover from (σ, p) ∈ oreachable (opaodv i) ?S ?U {l ∈ labels Γ_AODV p} and hyp
  have "1 ≤ dsn (σ' i)"
by (auto simp: 'σ' i = σ i)
moreover from 'sip (σ i) ≠ i' '(hyp' and pre
  have "dip (σ i) ∈ kD (rt (σ (sip (σ i))))")
  by (rule quality_increases_rreq_rrep_props)
ultimately show θthesis
next
  assume "sip (σ i) = i" thus θthesis
  using 'σ' i = σ i' hyp and pre by auto
qed

lemma rerr_sip:
"opaodv i = (otherwith quality_increases {i} (orecvmsg msg_fresh), other_quality_increases {i} →) onl Γ_AODV (λ(σ, l).
  l ∈ {PAdv-8, PAdv-9, PRerr-0, PRerr-:1}
  → (∀ ripc∈dom(dests (σ i)). ripc∈kD(rt (σ (sip (σ i)))) ∧
      the (dests (σ i) ripc) - 1 ≤ nsqn (rt (σ (sip (σ i)))) ripc)"
(is "_. = (?S, ?U →) _.")

proof -
{ fix dests rip sip rsn and σ σ' :: "ip ⇒ state"
  assume qinc: "∀ j. quality_increases (σ j) (σ' j)"
  and *: "∀ ripc∈dom dests. rip ∈ kD (rt (σ sip)) ∧
        the (dests ripc) - 1 ≤ nsqn (rt (σ sip)) ripc"
  and "dests rip = Some rsn"
from this(3) have "rip∈dom dests" by auto
with * and 'dests rip = Some rsn' have "rip∈kD(rt (σ sip))"
  and "rsn - 1 ≤ nsqn (rt (σ sip)) rip"
  by (auto dest!: bspec)
from qinc have "quality_increases (σ sip) (σ' sip)" ..
have "rip ∈ kD(rt (σ' sip)) ∧ rsn - 1 ≤ nsqn (rt (σ' sip)) rip"proof
  from 'rip∈kD(rt (σ sip))' and 'quality_increases (σ sip) (σ' sip)'
    show "rip ∈ kD(rt (σ' sip))" ..
next
  from 'rip∈kD(rt (σ sip))' and 'quality_increases (σ sip) (σ' sip)'
    have "nsqn (rt (σ sip)) rip ≤ nsqn (rt (σ' sip)) rip" ..
  with 'rsn - 1 ≤ nsqn (rt (σ sip)) rip' show "rsn - 1 ≤ nsqn (rt (σ' sip)) rip"
  by (rule le_trans)
qed
}

show θthesis
  by (inv_cterms inv add: oseq_step_invariant_sterms [OF quality_increases_nsqn_fresh aodv_wf
    oaodv_trans]
      onl_oinvariant_sterms [OF aodv_wf oreceived_rreq_rrep_nsqn_fresh_inv]
      other_quality_increases_other_localD
      simp del: One_nat_def, intro conj1)
lemma prerr_guard: "paodv i \|= 
onl \Gamma_{AODV} (λ(ξ, l). (l = PRerr→−→ \rightarrow (\forall ip∈dom(dests ξ). ip∈vD(rt ξ) 
∧ the (nhop (rt ξ) ip) = sip ξ 
∧ sqn (rt ξ) ip < the (dests ξ ip))))"
by (inv_cterms) (clarsimp simp: split_if_asm split_asm, erule(2) partial)+

qed

lemmas oaddpreRT_welldefined =
open_seq_invariant [OF addpreRT_welldefined initiali_aodv oaodv_trans aodv_trans, 
simplified seql_onl_swap, 
THEN oinvariant_anyact]

lemmas odests_vD_inc_sqn =
open_seq_invariant [OF dests_vD_inc_sqn initiali_aodv oaodv_trans aodv_trans, 
simplified seql_onl_swap, 
THEN oinvariant_anyact]

lemmas oprerr_guard =
open_seq_invariant [OF prerr_guard initiali_aodv oaodv_trans aodv_trans, 
simplified seql_onl_swap, 
THEN oinvariant_anyact]

Proposition 7.28

lemma seq_compare_next_hop':
"opaodv i \|= \{\text{otherwith quality_increases \{i\} (orecmsg msg_fresh), \nother_quality_increases \{i\} \rightarrow onl \Gamma_{AODV} (λ(σ, _). \n\forall dip. let nhip = the (nhop (rt (σ i)) dip) 
in dip ∈ kD(rt (σ i)) ∧ nhip \neq dip → 
dip ∈ kD(rt (σ nhip)) ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ nhip)) dip)\}"
(is \("_ \|= (?S, ?U → _)\)"
proof -
{
fix nhip and σ σ' :: \"ip \Rightarrow state\"
assume pre: \"\forall dip∈kD(rt (σ i)). nhop dip \neq dip → 
dip∈kD(rt (σ (nhop dip))) ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (nhop dip))) dip\" 
and qinc: \"\forall j. quality_increases (σ j) (σ' j)\"
have \"\forall dip∈kD(rt (σ i)). nhop dip \neq dip → 
dip∈kD(rt (σ' (nhop dip))) ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (nhop dip))) dip\"
proof (intro ballI impI)
fix dip
assume "dip∈kD(rt (σ i))"
and "nhop dip \neq dip"
with pre have "dip∈kD(rt (σ (nhop dip)))"
and "nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (nhop dip))) dip"
by auto
from qinc have qinc_nhop: \"quality_increases (σ (nhop dip)) (σ' (nhop dip))\" ..
with \"dip∈kD(rt (σ (nhop dip)))\" have \"dip∈kD (rt (σ' (nhop dip)))\" ..
moreover have \"nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (nhop dip))) dip\"
proof -
from \"dip∈kD(rt (σ (nhop dip)))\" qinc_nhop
have "nsqn (rt (σ (nhop dip))) dip ≤ nsqn (rt (σ' (nhop dip))) dip" ..
with \"nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (nhop dip))) dip\" show \?thesis
by simp
qed
ultimately show \"dip∈kD(rt (σ' (nhop dip))) 
∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (nhop dip))) dip\" ..
qed
} note basic = this
{ fix nhop and σ σ' :: "ip ⇒ state"
assume pre: "∀ dip ∈ kD(rt (σ i)). nhop dip ≠ dip → dip ∈ kD(rt (σ (nhop dip)))
∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (nhop dip))) dip"
and ndest: "∀ ripc ∈ dom (dests (σ i)). ripc ∈ kD (rt (σ (sip (σ i))))
∧ the (dests (σ i)) ripc - 1 ≤ nsqn (rt (σ (sip (σ i)))) ripc"
and issip: "∀ ip ∈ dom (dests (σ i)). nhop ip = sip (σ i)"
and qinc: "∀ (dests (σ i)) dip ≤ nsqn (rt (σ' (nhop dip))) dip"
have "∀ dip ∈ kD(rt (σ i)). nhop dip ≠ dip → dip ∈ kD (rt (σ' (nhop dip)))
∧ nsqn (invalidate (rt (σ i)) (dests (σ i))) dip ≤ nsqn (rt (σ' (nhop dip))) dip"
proof (intro ballI impI)
fix dip
assume "dip ∈ kD(rt (σ i))"
and "nhop dip ≠ dip" with pre and qinc
have "dip ∈ kD (rt (σ' (nhop dip)))" and "nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (nhop dip))) dip"
proof (cases "dip ∈ dom (dests (σ i))")
assume "dip ∈ dom (dests (σ i))"
with ⟨dip ∈ kD (rt (σ i))⟩ obtain dsn where "dests (σ i) dip = Some dsn"
by auto
with ⟨dip ∈ kD (rt (σ i))⟩ have "nsqn (invalidate (rt (σ i)) (dests (σ i))) dip = dsn - 1"
by (rule nsqn_invalidate_eq)
moreover have "dsn - 1 ≤ nsqn (rt (σ' (nhop dip))) dip"
proof -
from ⟨dests (σ i) dip = Some dsn⟩ have "the (dests (σ i) dip) = dsn" by simp
with ⟨dests (σ i) dip = Some dsn⟩ have "dip ∈ kD (rt (σ (sip (σ i))))"
"dsn - 1 ≤ nsqn (rt (σ (sip (σ i)))) dip" by simp auto
moreover from issip and ⟨dip ∈ dom (dests (σ i))⟩ have "nhop dip = sip (σ i)" ..
ultimately have "dip ∈ kD (rt (σ (nhop dip)))" and "dsn - 1 ≤ nsqn (rt (σ (nhop dip))) dip" by auto
with qinc show "dsn - 1 ≤ nsqn (rt (σ' (nhop dip))) dip" by simp auto
qed with ⟨dip ∈ kD (rt (σ' (nhop dip)))⟩
show "dip ∈ kD (rt (σ' (nhop dip)))" and "dsn - 1 ≤ nsqn (rt (σ' (nhop dip))) dip" by simp auto
qed
ultimately show ?thesis by simp
next assume "dip ≠ dom (dests (σ i))"
with ⟨dip ∈ kD (rt (σ i))⟩
have "nsqn (invalidate (rt (σ i)) (dests (σ i))) dip = nsqn (rt (σ i)) dip"
by (rule nsqn_invalidate_other)
with nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (nhop dip))) dip) show ?thesis by simp
qed
show "dip ∈ kD (rt (σ' (nhop dip)))" and "dsn - 1 ≤ nsqn (rt (σ' (nhop dip))) dip" by simp auto
qed }

note basic_prerr = this

{ fix σ σ' :: "ip ⇒ state"
assume a1: "∀ dip ∈ kD (rt (σ i)). the (nhop (rt (σ i)) dip) ≠ dip →
dip ∈ kD (rt (σ (the (nhop (rt (σ i)) dip))))
∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (the (nhop (rt (σ i)) dip)))) dip" and a2: "∀ j. quality_increases (σ j) (σ' j)"
have "∀ dip ∈ kD (rt (σ i)).
the (nhop (update (rt (σ i)) (sip (σ i)) (0, unk, val, Suc 0, sip (σ i), {})) dip) ≠ dip →
dip ∈ kD (rt (σ' (the (nhop (update (rt (σ i)) (sip (σ i))
(0, unk, val, Suc 0, sip (σ i), {})) dip))))
∧ nsqn (update (rt (σ i)) (sip (σ i)) (0, unk, val, Suc 0, sip (σ i), {})) dip
≤ nsqn (rt (σ' (the (nhop (update (rt (σ i)) (sip (σ i))
(0, unk, val, Suc 0, sip (σ i), {})) dip)))
 dip" (is "\(\forall \text{dip} \in kD(\text{rt } (\sigma i)). \ ?P \ \text{dip}\)"

proof
fix dip
assume "\(\text{dip} \in kD(\text{rt } (\sigma i))\)"
with a1 and a2
have "\((\text{the (nhop (rt } (\sigma i)) \text{dip}) \neq \text{dip} \rightarrow \text{dip} \in kD(\text{rt } (\sigma ' (\text{the (nhop (rt } (\sigma i)) \text{dip}))))\)"
\(\land \ \text{nsqn (rt } (\sigma i)) \text{ dip} \leq \text{nsqn (rt } (\sigma ' (\text{the (nhop (rt } (\sigma i)) \text{dip}))))\) dip"
by - (drule(1) basic, auto)
thus "\(?P \ \text{dip}\)" by (cases "\(\text{dip} = \text{sip } (\sigma i)\)") auto
qed

} note nhop_update_sip = this

{ fix \(\sigma \ \sigma ' \ \text{oip} \ \text{sip} \ \text{osn} \ \text{hops}\)
assume pre: "\(\forall \text{dip} \in kD (\text{rt } (\sigma i)). \ \text{the (nhop (rt } (\sigma i)) \text{dip}) \neq \text{dip} \rightarrow \text{dip} \in kD(\text{rt } (\sigma ' (\text{the (nhop (rt } (\sigma i)) \text{dip}))))\)"
\(\land \ \text{nsqn (rt } (\sigma i)) \text{ dip} \leq \text{nsqn (rt } (\sigma ' (\text{the (nhop (rt } (\sigma i)) \text{dip}))))\) dip"
and qinc: "\(\forall j. \text{ quality_increases } (\sigma j) (\sigma ' j)\)"
and *: "\(\text{sip} \neq \text{oip} \rightarrow \text{oip} \in kD(\text{rt } (\sigma sip))\)"
\(\land \ \text{osn } \leq \text{nsqn (rt } (\sigma sip)) \text{ oip}\)
\(\land \ \text{(nsqn } (\sigma \text{ sip}) \text{ oip} = \text{osn}\)"
\(\rightarrow \ \text{the (dhops } (\sigma \text{ sip}) \text{ oip) } \leq \text{hops}\)
\(\lor \ \text{the (flag } (\sigma \text{ sip}) \text{ oip) = inv}\)"
from pre and qinc
have pre': "\(\forall \text{dip} \in kD (\text{rt } (\sigma i)). \ \text{the (nhop (rt } (\sigma i)) \text{dip}) \neq \text{dip} \rightarrow \text{dip} \in kD(\text{rt } (\sigma ' (\text{the (nhop (rt } (\sigma i)) \text{dip}))))\)"
\(\land \ \text{nsqn (rt } (\sigma i)) \text{ dip} \leq \text{nsqn (rt } (\sigma ' (\text{the (nhop (rt } (\sigma i)) \text{dip}))))\) dip"
by (rule basic)
have "\((\text{the (nhop (update (rt } (\sigma i)) \text{oip} (\text{osn, kno, val, Suc hops, sip, {}}) \text{oip}) \neq \text{oip}\)"
\(\rightarrow \ \text{oip} \in kD(\text{rt } (\sigma ' (\text{the (nhop (update (rt } (\sigma i)) \text{oip})))) \text{oip}\)"
\(\land \ \text{(osn, kno, val, Suc hops, sip, {}}) \text{oip})\)])
\(\land \ \text{nsqn (update (rt } (\sigma i)) \text{oip} (\text{osn, kno, val, Suc hops, sip, {}}) \text{oip})\)"
\(\leq \ \text{nsqn (rt } (\sigma ' (\text{the (nhop (update (rt } (\sigma i)) \text{oip})))) \text{oip}\)"
\(\land \ \text{(osn, kno, val, Suc hops, sip, {}}) \text{oip})\)]) \text{oip})\)"
(is "\(\text{?nhop} \neq \text{oip} \rightarrow \text{?oip} \in kD \land \ ?\text{nsqn} \leq \text{nsqn}\)"

proof (rule, split update_rt_split_asm)
assume "\(\text{rt } (\sigma i) = \text{update (rt } (\sigma i)) \text{oip} (\text{osn, kno, val, Suc hops, sip, {}})\)"
and "\(\text{the (nhop (rt } (\sigma i)) \text{oip}) \neq \text{oip}\)"
with pre' show "\(\text{?oip} \in kD \land ?\text{nsqn} \leq \text{nsqn}\)" by auto
next
assume rtnot: "\(\text{rt } (\sigma i) \neq \text{update (rt } (\sigma i)) \text{oip} (\text{osn, kno, val, Suc hops, sip, {}})\)"
and notoip: ?nhop_not_oip
with * qinc have ?oip_in_kD
by auto
moreover with * pre qinc rtnot notoip have ?nsqn_le_nsqn
by simp (metis kD_nsqn_quality_increases_trans)
ultimately show "\(\text{?oip} \in kD \land ?\text{nsqn} \leq \text{nsqn}\)" ..
qed
}

} note update1 = this

{ fix \(\sigma \ \sigma ' \ \text{oip} \ \text{sip} \ \text{osn} \ \text{hops}\)
assume pre: "\(\forall \text{dip} \in kD (\text{rt } (\sigma i)). \ \text{the (nhop (rt } (\sigma i)) \text{dip}) \neq \text{dip} \rightarrow \text{dip} \in kD(\text{rt } (\sigma ' (\text{the (nhop (rt } (\sigma i)) \text{dip}))))\)"
\(\land \ \text{nsqn (rt } (\sigma i)) \text{ dip} \leq \text{nsqn (rt } (\sigma ' (\text{the (nhop (rt } (\sigma i)) \text{dip}))))\) dip"
and qinc: "\(\forall j. \text{ quality_increases } (\sigma j) (\sigma ' j)\)"
and *: "\(\text{sip} \neq \text{oip} \rightarrow \text{oip} \in kD(\text{rt } (\sigma sip))\)"
\(\land \ \text{osn } \leq \text{nsqn (rt } (\sigma sip)) \text{oip}\)
\(\land \ \text{(nsqn (rt } (\sigma sip)) \text{oip} = \text{osn}\)"
\(\rightarrow \ \text{the (dhops } (\sigma \text{ sip}) \text{oip) } \leq \text{hops}\)
\(\lor \ \text{the (flag } (\sigma \text{ sip}) \text{oip) = inv}\)"
from pre and qinc
have pre': "\(\forall \text{dip} \in kD (\text{rt } (\sigma i)). \ \text{the (nhop (rt } (\sigma i)) \text{dip}) \neq \text{dip} \rightarrow \text{dip} \in kD(\text{rt } (\sigma ' (\text{the (nhop (rt } (\sigma i)) \text{dip}))))\)"
\(\land \ \text{nsqn (rt } (\sigma i)) \text{ dip} \leq \text{nsqn (rt } (\sigma ' (\text{the (nhop (rt } (\sigma i)) \text{dip}))))\) dip"
by (rule basic)
have "∀ dip ∈ kD (rt (σ i)). the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})) dip) ≠ dip
⇒ dip ∈ kD (rt (σ' (the (nhop (update (rt (σ i)) oip
(osn, kno, val, Suc hops, sip, {})) dip))))
∧ nsqn (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})) dip
≤ nsqn (rt (σ' (the (nhop (update (rt (σ i)) oip
(osn, kno, val, Suc hops, sip, {})) dip)))
(is "∀ dip ∈ kD (rt (σ i)). _ ⇒ dip ∈ kD dip ∧ nsqn dip"
proof (intro ballI impI, split update_rt_split_asm)
fix dip
assume "dip ∈ kD (rt (σ i))"
and "the (nhop (rt (σ i)) dip) ≠ dip"
and "rt (σ i) ≠ update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})"
with pre' show "dip ∈ kD dip ∧ nsqn dip" by simp
next
fix dip
assume "dip ∈ kD (rt (σ i))"
and notdip: "the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})) dip) ≠ dip"
and rtnot: "rt (σ i) ≠ update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})"
show "dip ∈ kD dip ∧ nsqn dip" proof (cases "dip = oip")
assume "dip = oip"
with pre' ⟨dip ∈ kD (rt (σ i))⟩ notdip *
have "dip ∈ kD dip" by simp (metis kD_quality_increases)
moreover from ⟨dip = oip⟩ rtnot qinc ⟨dip ∈ kD (rt (σ i))⟩ notdip *
have "nsqn dip" by simp (metis kD_nsqn_quality_increases_trans)
ultimately show ?thesis ..
qed
qed

note update2 = this

have "opao i = (?S, ?U → onl Γ AODV (λ (σ, _). ∀ dip ∈ kD (rt (σ i)). the (nhop (rt (σ i)) dip) ≠ dip
⇒ dip ∈ kD (rt (σ (the (nhop (rt (σ i)) dip))))
∧ nsqn (rt (σ i) dip) ≤ nsqn (rt (σ (the (nhop (rt (σ i)) dip)))) dip)"
by (inv_cterms inv add: oseq_step_invariant_sterms [OF oquality_increases_nsqn_fresh aodv_wf
aoadv_trans]
onl_oinvariant_sterms [OF aodv_wf oaddpreRT_welldefined]
onl_oinvariant_sterms [OF aodv_wf odests_vD_inc_sqn]
onl_oinvariant_sterms [OF aodv_wf oprerr_guard]
onl_oinvariant_sterms [OF aodv_wf rreq sip]
onl_oinvariant_sterms [OF aodv_wf rrep sip]
onl_oinvariant_sterms [OF aodv_wf rerr sip]
other_quality_increases
other_localD
solve: basic basic_prerr
simp add: seqlsimp nsqn_invalidate nhop_update_sip
simp del: One_nat_def)
(rule conjI, erule(2) update1, erule(2) update2)+
thus ?thesis unfolding Let_def by auto
qed

Proposition 7.30
lemmas okD_unk_or_atleast_one =
open_seq_invariant [OF kD_unk_or_atleast_one initiali_aodv,
simplified seql_onl_swap]
lemmas ozero_seq_unk_hops_one =
  open_seq_invariant [OF zero_seq_unk_hops_one initiali_aodv,
  simplified seql_onl_swap]

lemma ororeachable_fresh_okD_unk_or_atleast_one:
  fixes dip
  assumes "((σ, p) ∈ ororeachable (opaodv i)
             (otherwith (op=) {i} (orecmsg (λσ m. msg_fresh σ m
              ∧ msg_zhops m))))
             (other quality_increases {i})"
  and "dip∈kD(rt (σ i))"
  shows "π3 (the (rt (σ i) dip)) = unk ∨ 1 ≤ π2 (the (rt (σ i) dip))"
  (is "?P dip")
proof -
  have "∃l. l∈labels Γ AODV p" by (metis aodv_ex_label)
  with assms(1) have "∀dip∈kD(rt (σ i)). ?P dip"
    by - (drule oinvariant_weakenD [OF okD_unk_or_atleast_one [OF oaodv_trans aodv_trans]],
         auto dest!: onlD otherwith_actionD simp: seqlsimp)
  with ⟨dip∈kD(rt (σ i))⟩ show ?thesis by simp
qed

lemma ororeachable_fresh_ozero_seq_unk_hops_one:
  fixes dip
  assumes "((σ, p) ∈ ororeachable (opaodv i)
             (otherwith (op=) {i} (orecmsg (λσ m. msg_fresh σ m
              ∧ msg_zhops m))))
             (other quality_increases {i})"
  and "dip∈kD(rt (σ i))"
  shows "sqn (rt (σ i)) dip = 0 → sqnf (rt (σ i)) dip = unk ∧
              the (dhops (rt (σ i)) dip) = 1 ∧
              the (nhop (rt (σ i)) dip) = dip"
  (is "?P dip")
proof -
  have "∃l. l∈labels Γ AODV p" by (metis aodv_ex_label)
  with assms(1) have "∀dip∈kD(rt (σ i)). ?P dip"
    by - (drule oinvariant_weakenD [OF ozero_seq_unk_hops_one [OF oaodv_trans aodv_trans]],
         auto dest!: onlD otherwith_actionD simp: seqlsimp)
  with ⟨dip∈kD(rt (σ i))⟩ show ?thesis by simp
qed

lemma seq_nhop_quality_increases':
  shows "opaodv i |= (otherwith (op=) {i}
              (orecmsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
              other quality_increases {i} →)
              onl Γ AODV (λ(σ, _). ∀dip. let nthp = the (nhop (rt (σ i)) dip)
              in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nthp))
              ∧ nthp ≠ dip → (rt (σ i) □ dip (rt (σ nthp))))"
  (is "_ |= (?S i, _ →) _")
proof -
  have weaken:
    "∀P I Q R P. P |= (otherwith quality_increases I (orecmsg Q), other quality_increases I →) P
     ⇒ P |= (otherwith (op=) I (orecmsg (λσ m. Q σ m ∧ R σ m)), other quality_increases I →) P"
    by auto
  
  fix i a and σ σ' :: "ip ⇒ state"
  assume ai: "∀dip. dip∈VD(rt (σ i))
              ∧ dip∈VD(rt (σ the (nhop (rt (σ i)) dip))))
              ∧ (the (nhop (rt (σ i)) dip)) ≠ dip →
              rt (σ i) □ dip rt (σ the (nhop (rt (σ i)) dip)))"
  and ov: "?S i σ σ' a"
  have "∀dip. dip∈VD(rt (σ i))
              ∧ dip∈VD (rt (σ' (the (nhop (rt (σ i)) dip))))"
proof clarify

fix dip
assume a2: "dip ∈ VD(rt (σ i))"
  and a3: "dip ∈ VD (rt (σ' (the (nhop (rt (σ i)) dip))))"
  and a4: "(the (nhop (rt (σ i)) dip)) ≠ dip"
from ow have "∀ j. j ≠ i → σ j = σ' j" by auto
show "rt (σ i) ⊏ dip rt (σ' (the (nhop (rt (σ i)) dip)))"
proof (cases "(the (nhop (rt (σ i)) dip)) = i")
  assume "(the (nhop (rt (σ i)) dip)) = i"
  with ⟨dip ∈ VD(rt (σ i))⟩ have "dip ∈ VD(rt (σ (the (nhop (rt (σ i)) dip))))" by simp
  with a1 a2 a4 have "rt (σ i) ⊏ dip rt (σ (the (nhop (rt (σ i)) dip)))" by simp
  with ⟨(the (nhop (rt (σ i)) dip)) = i⟩ have "rt (σ i) ⊏ dip rt (σ i)" by simp
  hence False by simp
  thus ?thesis ..
next
assume "(the (nhop (rt (σ i)) dip)) ≠ i"
with ⟨∀ j. j ≠ i → σ j = σ' j⟩
  have *: "σ (the (nhop (rt (σ i)) dip)) = σ' (the (nhop (rt (σ i)) dip))" by simp
  with ⟨dip ∈ VD(rt (σ' (the (nhop (rt (σ i)) dip))))⟩
    have "dip ∈ VD(rt (σ (the (nhop (rt (σ i)) dip))))" by simp
  with a1 a2 a4 have "rt (σ i) ⊏ dip rt (σ (the (nhop (rt (σ i)) dip)))" by simp
  with * show ?thesis by simp
qed
qed

} note basic = this

{ fix σ σ' a dip sip i
  assume a1: "∀ dip. dip ∈ VD(rt (σ i))
    ∧ dip ∈ VD (rt (σ (the (nhop (rt (σ i)) dip))))
    ∧ the (nhop (rt (σ i)) dip) ≠ dip
    → rt (σ i) ⊏ dip rt (σ (the (nhop (rt (σ i)) dip))))"
  and ow: "∃? i. σ i = σ' a"
  have "∀ dip. dip ∈ VD(update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {}))
    ∧ dip ∈ VD (rt (σ' (the (nhop (update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {})) dip))))
    ∧ the (nhop (update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {})) dip) ≠ dip
    → update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {})
    ⊏ dip rt (σ' (the (nhop (update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {})) dip))))"
proof clarify

fix dip
assume a2: "dip ∈ VD (update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {}))"
  and a3: "dip ∈ VD (rt (σ' (the (nhop (update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {})) dip))))"
  and a4: "the (nhop (update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {})) dip) ≠ dip"
show "update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {})
    ⊏ dip rt (σ' (the (nhop (update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {})) dip))))"
proof (cases "dip = sip")
  assume "dip = sip"
  with ⟨the (nhop (update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {})) dip) ≠ dip⟩
    have False by simp
  thus ?thesis ..
next
assume [simp]: "dip ≠ sip"
from a2 have "dip ∈ VD (rt (σ i)) ∨ dip = sip"
  by (rule vD_update_val)
with ⟨dip ≠ sip⟩ have "dip ∈ VD (rt (σ i))" by simp
moreover from a3 have "dip ∈ VD (rt (σ' (the (nhop (rt (σ i)) dip))))" by simp
moreover from a4 have "the (nhop (rt (σ i)) dip) ≠ dip" by simp
ultimately have "rt (σ i) ⊏ dip rt (σ' (the (nhop (rt (σ i)) dip)))"
  using a1 ow by (drule(1) basic, simp)
with ⟨dip ≠ sip⟩ show ?thesis
  by (erule rt_strictly_fresher_update_other, simp)
qed
\begin{proof}
\fix \sigma \ a \ \sigma' \ \text{nhop}
\begin{ass}
\pre: "\forall \text{dip}. \text{dip} \in \text{Vd} (rt (\sigma \ i)) \ \land \ \text{dip} \in \text{Vd} (rt (\sigma' (\text{nhop} \ \text{dip}))) \ \land \ \text{nhop} \ \text{dip} \neq \ \text{dip}
\rightarrow rt (\sigma \ i) \ \sqsubseteq_{\text{dip}} \ rt (\sigma' (\text{nhop} \ \text{dip}))"
\end{ass}
\begin{owl}
"?S i \ a \ \sigma' \ a"
\end{owl}
\begin{have}
"\forall \text{dip}. \text{dip} \in \text{Vd} (\text{invalidate} (rt (\sigma \ i)) \ (\text{dests} (\sigma \ i))) \\
\land \ \text{dip} \in \text{Vd} (rt (\sigma' (\text{nhop} \ \text{dip}))) \ \land \ \text{nhop} \ \text{dip} \neq \ \text{dip}
\rightarrow rt (\sigma \ i) \ \sqsubseteq_{\text{dip}} \ rt (\sigma' (\text{nhop} \ \text{dip}))"
\end{have}
\begin{proof}
\clarify \fix \text{dip}
\begin{ass}
"\text{dip} \in \text{Vd} (\text{invalidate} (rt (\sigma \ i)) \ (\text{dests} (\sigma \ i)))"
\end{ass}
\begin{ass}
"\text{dip} \in \text{Vd} (rt (\sigma' (\text{nhop} \ \text{dip})))"
\end{ass}
\begin{ass}
"\text{nhop} \ \text{dip} \neq \ \text{dip}"
\end{ass}
\begin{from}
\text{this} (1)
\end{from}
\begin{have}
"\text{dip} \in \text{Vd} (rt (\sigma \ i))"
\end{have}
\begin{by}
\text{clarsimp}:
\text{Vd}\_\text{invalidate}\_\text{not}\_\text{dests}
\end{by}
\begin{moreover}
from \text{ow}
\begin{have}
"\forall j. j \neq i \ \rightarrow \ \sigma j = \sigma' j" \ by \ \text{auto}
\end{have}
\end{moreover}
\begin{ultimately}
ultimately\ have
"rt (\sigma \ i) \ \sqsubseteq_{\text{dip}} \ rt (\sigma' (\text{nhop} \ \text{dip})))"
\begin{by}
\text{metis}
\end{by}
\begin{with}
\forall j. j \neq i \ \rightarrow \ \sigma j = \sigma' j
\end{with}
\begin{show}
show
"rt (\sigma \ i) \ \sqsubseteq_{\text{dip}} \ rt (\sigma' (\text{nhop} \ \text{dip})))"
\begin{by}
\text{metis} \ rt\_\text{strictly}\_\text{fresher}\_\text{irefl}
\end{by}
\end{show}
\end{ultimately}
\end{proof}
\end{proof}
\end{proof}
\begin{proof}
\fix \sigma \ a \ \sigma' \ \text{dip} \ \text{oip} \ \text{osn} \ \text{sip} \ \text{hops} \ \text{i}
\begin{ass}
\pre: "\forall \text{dip}. \text{dip} \in \text{Vd} (rt (\sigma \ i)) \\
\land \ \text{dip} \in \text{Vd} (rt (\sigma' (\text{nhop} \ \text{dip}))) \\
\land \ \text{the} (\text{nhop} (rt (\sigma \ i)) \ \text{dip}) \neq \ \text{dip}
\rightarrow rt (\sigma \ i) \ \sqsubseteq_{\text{dip}} \ rt (\sigma' (\text{nhop} \ \text{dip})))"
\end{ass}
\begin{owl}
"?S i \ a" \ \sigma' \ a"
\end{owl}
\begin{ass}
"\text{Suc} 0 \leq \text{osn}"
\end{ass}
\begin{ass}
"\text{sip} \neq \text{oip} \ \rightarrow \ \text{oip} \in \text{kD} (rt (\sigma \ \text{sip}))
\land \ \text{osn} \leq \text{nsqn} (rt (\sigma \ \text{sip})) \ \text{oip}
\land \ \text{nsqn} (rt (\sigma \ \text{sip})) \ \text{oip} = \text{osn}
\rightarrow \ \text{the} (\text{dhops} (rt (\sigma \ \text{sip}))) \ \text{oip} \leq \text{hops}
\lor \ \text{the} (\text{flag} (rt (\sigma \ \text{sip}))) \ \text{oip} = \text{inv}"
\end{ass}
\begin{and}
\after:
"\sigma' \ i = \sigma \ i\ (rt := \text{update} (rt (\sigma \ i)) \ \text{oip} (\text{osn}, \ \text{kno}, \ \text{val}, \ \text{Suc} \ \text{hops}, \ \text{sip}, \ \{\}))"
\end{and}
\begin{have}
\forall \text{dip}. \text{dip} \in \text{Vd} (\text{update} (rt (\sigma \ i)) \ \text{oip} (\text{osn}, \ \text{kno}, \ \text{val}, \ \text{Suc} \ \text{hops}, \ \text{sip}, \ \{\}))
\land \ \text{dip} \in \text{Vd} (rt (\sigma' (\text{nhop} (\text{update} (rt (\sigma \ i)) \ \text{oip} \\
(\text{osn}, \ \text{kno}, \ \text{val}, \ \text{Suc} \ \text{hops}, \ \text{sip}, \ \{\}))) \ \text{dip}))))
\land \ \text{the} (\text{nhop} (\text{update} (rt (\sigma \ i)) \ \text{oip} (\text{osn}, \ \text{kno}, \ \text{val}, \ \text{Suc} \ \text{hops}, \ \text{sip}, \ \{\}))) \ \text{dip} \neq \ \text{dip}
\rightarrow \ \text{update} (rt (\sigma \ i)) \ \text{oip} (\text{osn}, \ \text{kno}, \ \text{val}, \ \text{Suc} \ \text{hops}, \ \text{sip}, \ \{\})))
\sqsubseteq_{\text{dip}}
rt (\sigma' (\text{the} (\text{nhop} (\text{update} (rt (\sigma \ i)) \ \text{oip} (\text{osn}, \ \text{kno}, \ \text{val}, \ \text{Suc} \ \text{hops}, \ \text{sip}, \ \{\}) \ \text{dip})))")
\end{have}
\begin{proof}
\clarify \fix \text{dip}
\begin{ass}
"\text{dip} \in \text{Vd} (\text{update} (rt (\sigma \ i)) \ \text{oip} (\text{osn}, \ \text{kno}, \ \text{val}, \ \text{Suc} \ \text{hops}, \ \text{sip}, \ \{\}))"
\end{ass}
\begin{ass}
"\text{dip} \in \text{Vd} (rt (\sigma' (\text{the} (\text{nhop} (\text{update} (rt (\sigma \ i)) \ \text{oip} (\text{osn}, \ \text{kno}, \ \text{val}, \ \text{Suc} \ \text{hops}, \ \text{sip}, \ \{\}))) \ \text{dip})))")"
\end{ass}
\begin{ass}
"\text{the} (\text{nhop} (\text{update} (rt (\sigma \ i)) \ \text{oip} (\text{osn}, \ \text{kno}, \ \text{val}, \ \text{Suc} \ \text{hops}, \ \text{sip}, \ \{\}))) \ \text{dip} \neq \ \text{dip}"
\end{ass}
\begin{from}
\text{ow}
\begin{have}
"\text{a5}:
\forall j. j \neq i \ \rightarrow \ \sigma j = \sigma' j" \ by \ \text{auto}
\end{have}
\end{from}
\begin{show}
show
"\text{update} (rt (\sigma \ i)) \ \text{oip} (\text{osn}, \ \text{kno}, \ \text{val}, \ \text{Suc} \ \text{hops}, \ \text{sip}, \ \{\}) = rt (\sigma \ i)"
\begin{by}
\text{metis} \ \text{a5} \ \forall \text{j.} \ \sigma \ j = \sigma' \ j"
\end{by}
\end{show}
\end{proof}
\end{proof}
from a2 have "dip\in vD \ (rt \ (\sigma \ i))" by simp
moreover from a3 have "dip\in vD(rt \ (\sigma \ (\text{the} \ (\text{nhop} \ (rt \ (\sigma \ i)) \ \text{dip}))))"
  using nochange and \( \forall j. \ \sigma \ j = \sigma' \ j \) by clarsimp
moreover from a4 have "the (\text{nhop} \ (rt \ (\sigma \ i)) \ \text{dip}) \neq dip" by simp
ultimately have "rt \ (\sigma \ i) \sqsubseteq dip rt \ (\sigma \ (the \ (\text{nhop} \ (rt \ (\sigma \ i)) \ \text{dip})))"
  using pre by simp

hence "rt \ (\sigma \ i) \sqsubseteq dip rt' \ (the \ (\text{nhop} \ (rt \ (\sigma \ i)) \ \text{dip})))"
  using \( \forall j. \ \sigma \ j = \sigma' \ j \) by simp
thus "?thesis" by simp

next
assume change: "?rt1 \neq rt \ (\sigma \ i)"
from after a2 have "dip\in kD(rt \ (\sigma' \ i))" by auto
show ?thesis
proof (cases "dip = oip")
  assume "dip \neq oip"

with a2 have "dip\in vD \ (rt \ (\sigma \ i))" by auto
moreover with a3 a5 after and \( \text{dip} \neq oip \)
  have "dip\in vD(rt \ (\sigma \ (\text{the} \ (\text{nhop} \ (rt \ (\sigma \ i)) \ \text{dip}))))"
    by simp metis
moreover from a4 and \( \text{dip} \neq oip \) have "the (\text{nhop} \ (rt \ (\sigma \ i)) \ \text{dip}) \neq dip" by simp
ultimately have "rt \ (\sigma \ i) \sqsubseteq dip rt \ (\sigma \ (the \ (\text{nhop} \ (rt \ (\sigma \ i)) \ \text{dip})))"
  using pre by simp

with after and a5 and \( \text{dip} \neq oip \) show ?thesis
  by simp (metis rt_strictly_fresher_update_other rt_strictly_fresher_irefl)

next
assume "dip = oip"

with a4 and change have "sip \neq oip" by simp
with a6 have "oip\in kD(rt \ (\sigma \ sip))"
  and "osn \leq nsqn \ (rt \ (\sigma \ sip)) \ oip" by auto
from a3 change \( \text{dip} = oip \) have "oip\in vD(\text{rt} \ (\sigma' \ sip))" by simp
hence "the (\text{flag} \ (rt \ (\sigma' \ sip)) \ oip) = val" by simp

from \( oip\in kD(\text{rt} \ (\sigma \ sip)) \)
have "osn < nsqn \ (rt \ (\sigma \ sip)) \ oip \lor (osn = nsqn \ (rt \ (\sigma' \ sip)) \ oip \land the (dhops \ (rt \ (\sigma' \ sip)) \ oip) \leq hops)"

proof
  assume "oip\in vD(rt \ (\sigma \ sip))"
  hence "the (\text{flag} \ (rt \ (\sigma \ sip)) \ oip) = val" by simp
with a6 \( \text{sip} \neq oip \) have "nsqn \ (rt \ (\sigma \ sip)) \ oip = osn \rightarrow the (dhops \ (rt \ (\sigma \ sip)) \ oip) \leq hops"
    by simp
show ?thesis
proof (cases "sip = i")
  assume "sip \neq i"
with a5 have "\sigma \ sip = \sigma' \ sip" by simp
with \( osn \leq nsqn \ (rt \ (\sigma \ sip)) \ oip \)
  and \( nsqn \ (rt \ (\sigma \ sip)) \ oip = osn \rightarrow the (dhops \ (rt \ (\sigma \ sip)) \ oip) \leq hops \)
show ?thesis by auto
next
  — alternative to using sip_not_ip
assume [simp]: "sip = i"
have "?rt1 = rt \ (\sigma \ i)"
proof (rule update_cases_kD, simp_all)
  from \(Suc \ 0 \leq osn \) show "0 < osn" by simp
next
  from \( oip\in kD(\text{rt} \ (\sigma \ sip)) \) and \( \text{sip} = i \) show "oip\in kD(\text{rt} \ (\sigma \ i))"
    by simp
next
assume "sqn (rt (σ i)) oip < osn"
also from ⟨osn ≤ nsqn (rt (σ sip)) oip⟩
    have "... ≤ nsqn (rt (σ i)) oip" by simp
also have "... ≤ sqn (rt (σ i)) oip"
    by (rule nsqn_sqn)
finally have "sqn (rt (σ i)) oip < sqn (rt (σ i)) oip".
hence False by simp
thus "(λa. if a = oip then Some (osn, kno, val, Suc hops, i, π7 (the (rt (σ i) oip)))
        else rt (σ i) a) = rt (σ i)". ..

next
assume "sqn (rt (σ i)) oip = osn"
    and "Suc hops < (dhops (rt (σ i)) oip)"
from this(1) and ⟨oip ∈ vD (rt (σ sip))⟩
    have "nsqn (rt (σ i)) oip = osn" by simp
with ⟨nsqn (rt (σ i)) oip = osn ⟩
    have "the (dhops (rt (σ i)) oip) ≤ hops" by simp
with ⟨Suc hops < (dhops (rt (σ i)) oip)⟩
    have False by simp
thus "(λa. if a = oip then Some (osn, kno, val, Suc hops, i, π7 (the (rt (σ i) oip)))
        else rt (σ i) a) = rt (σ i)". ..

next
assume "(the (flag (rt (σ i)) oip) = inv"
    with ⟨the (flag (rt (σ sip)) oip) = val ⟩
    and a5
    have "sip = i" by (metis f.distinct(1) iD_flag_is_inv)
from ⟨oip ∈ iD(rt (σ sip))⟩
    show "(λa. if a = oip then Some (the (rt (σ i) oip)) else rt (σ i) a) = rt (σ i)"
        by (auto dest!: kD_Some)
    with change have False ..
thus ?thesis ..

next
assume "oip ∈ iD(rt (σ sip))"
    with ⟨the (flag (rt (σ sip)) oip) = val ⟩
    and a5 have "sip = i"
        by (metis f.distinct(1) iD_flag_is_inv)
from ⟨oip ∈ iD(rt (σ sip))⟩
    have "the (flag (rt (σ sip)) oip) = inv" by auto
with ⟨sip = i; Suc 0 ≤ osn ⟩
    change after ⟨oip ∈ kD(rt (σ sip))⟩
    have "nsqn (rt (σ sip)) oip < nsqn (rt (σ' sip)) oip"
        unfolding update_def
            by (clarsimp split: option.split_asm split_if_asm)
        (auto simp: sqn_def)
    with ⟨osn ≤ nsqn (rt (σ sip)) oip⟩
        have "osn < nsqn (rt (σ' sip)) oip"
            by simp
    thus ?thesis ..

qed

thus ?thesis
proof
assume osnlt: "osn < nsqn (rt (σ' sip)) oip"
from ⟨dip ∈ kD(rt (σ' i))⟩
    and ⟨dip = oip⟩
    have "dip ∈ kD (?rt1)" by simp
moreover from a3
    have "dip ∈ kD(?rt2 dip)" by simp
moreover have "nsqn ?rt1 dip < nsqn (?rt2 dip) dip"
    proof
        have "nsqn ?rt1 oip = osn"
            by (simp add: ⟨dip = oip⟩; nsqn_update_changed_kno_val [OF change [THEN not_sym]])
        also have "... < nsqn (rt (σ' sip)) oip" using osnlt .
        also have "... = nsqn (?rt2 oip) oip" by (simp add: change)
        finally have ?thesis
            using ⟨dip = oip⟩ by simp
        qed
ultimately show ?thesis

76
by (rule rt_strictly_fresher_ltI)

next
assume osneq: "osn = nsqn (rt (σ' sip)) oip ∧ the (dhops (rt (σ' sip)) oip) ≤ hops"

have "oip∈kD(?rt1)" by simp
moreover from a3 ⟨dip = oip⟩ have "oip∈kD(?rt2 oip)" by simp

moreover have "nsqn ?rt1 oip = nsqn (?rt2 oip) oip"
proof
  from osneq have "osn = nsqn ?rt1 oip"
  also have "osn = nsqn ?rt2 oip"
    by (simp add: ⟨dip = oip⟩)
  finally show ?thesis .
qed

moreover have "π5(the (?rt2 oip oip)) < π5(the (?rt1 oip))"
proof
  from osneq have "the (dhops (rt (σ' sip)) oip) ≤ hops"
    by auto
  ultimately have "π5(the (rt (σ' sip) oip)) ≤ hops"
    by (auto simp add: proj5_eq_dhops)
  finally have "π5(the (rt (σ' i) oip)) < π5(the (?rt1 oip))" .
with change after show ?thesis by simp
qed

ultimately have "?rt1 ⊑oip ?rt2 oip"
  by (rule rt_strictly_fresher_eqI)
with ⟨dip = oip⟩ show ?thesis by simp
qed

qed

qed

note rreq_rrep_update = this

have "opaodv i ⪯ (otherwith (op=) {i} (orecmsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
other quality_increases {i} →)
onl ΓAODV
(λ(σ, _). ∀dip. dip ∈ vD (rt (σ i)) ∩ vD (rt (σ (the (nhop (rt (σ i)) dip)))))
∧ the (nhop (rt (σ i)) dip) ≠ dip
→ rt (σ i) ⊑dip rt (σ (the (nhop (rt (σ i)) dip))))"
proof (inv_cterms inv add: onl_oinvariant_sterms [OF aodv_wf rreq_sip [THEN weaken]]
onl_oinvariant_sterms [OF aodv_wf rrep_sip [THEN weaken]]
onl_oinvariant_sterms [OF aodv_wf rerr_sip [THEN weaken]]
onl_oinvariant_sterms [OF aodv_wf oosn_rreq [THEN weaken]]
onl_oinvariant_sterms [OF aodv_wf odsn_rrep [THEN weaken]]
onl_oinvariant_sterms [OF aodv_wf oaddpreRT_welldefined]
solve: basic update_0_unk invalidate rreq_rrep_update
simp add: seqlsimp)

fix σ σ' p l
assume or: "(σ, p) ∈ oreachable (opaodv i) (?S i) (other quality_increases {i})"
  and "other quality_increases {i} σ σ'"
and ll: "l ∈ labels ΓAODV p"
and pre: "∀dip. dip∈vD (rt (σ i))
∧ dip∈vD(rt (σ (the (nhop (rt (σ i)) dip))))
∧ the (nhop (rt (σ i)) dip) ≠ dip
→ rt (σ i) ⊑dip rt (σ (the (nhop (rt (σ i)) dip))))"
from this(1-2)
have or': "(σ', p) ∈ oreachable (opaodv i) (?S i) (other quality_increases {i})"
  by - (rule oreachable_other')
from or and ll have next_hop: "∀ dip. let nhip = the (nhop (rt (σ i)) dip)
  in dip ∈ kD(rt (σ i)) ∧ nhip ≠ dip
  −→ dip ∈ kD(rt (σ nhip))
  ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ nhip)) dip"
  by (auto dest!: onl_oinvariant_weakenD [OF seq_compare_next_hop'])

from or and ll have unk_hops_one: "∀ dip ∈ kD (rt (σ i)). sqn (rt (σ i)) dip = 0
  −→ sqnf (rt (σ i)) dip = unk ∧ the (dhops (rt (σ i)) dip) = 1
  ∧ the (nhop (rt (σ i)) dip) ≠ dip"
by (auto dest!: onl_oinvariant_weakenD [OF ozero_seq_unk_hops_one [OF oaodv_trans aodv_trans]]
otherwith_actionD simp: seqlsimp)

from ⟨other quality_increases {i} σ σ'⟩
have "σ' i = σ i" by auto
hence "quality_increases (σ i) (σ' i)"
by - (erule otherE, metis singleton_iff)

show "∀ dip. dip ∈ vD (rt (σ' i))
  ∧ dip ∈ vD (rt (σ' (the (nhop (rt (σ' i)) dip))))
  ∧ the (nhop (rt (σ' i)) dip) ≠ dip
  −→ rt (σ' i) ⊑ dip rt (σ' (the (nhop (rt (σ' i)) dip)))"
proof clarify
fix dip
assume "dip∈vD(rt (σ' i))"
and "dip∈vD(rt (σ' (the (nhop (rt (σ' i)) dip))))"
and "the (nhop (rt (σ' i)) dip) ≠ dip"
from this(1) and (σ' i = σ i) have "dip∈vD(rt (σ i))"
  and "dip∈kD(rt (σ i))"
by auto
from (the (nhop (rt (σ' i)) dip) ≠ dip) and (σ' i = σ i)
have "the (nhop (rt (σ i)) dip) ≠ dip" (is "?nhip ≠ _") by simp
with (dip∈kD(rt (σ i))) and next_hop
have "dip∈kD(rt (σ (?nhip)))"
  and nsqns: "nsqn (rt (σ i)) dip ≤ nsqn (rt (σ ?nhip)) dip"
  by (auto simp: Let_def)

have "0 < sqn (rt (σ i)) dip"
proof (rule neq0_conv [THEN iffD1, OF notI])
  assume "sqn (rt (σ i)) dip = 0"
  with (dip∈kD(rt (σ i))) and unk_hops_one
  have "?nhip = dip" by simp
  with (?nhip ≠ dip) show False ..
  qed
also have "... = nsqn (rt (σ i)) dip"
by (rule vD_nsqn_sqn [OF (dip∈vD(rt (σ i))), THEN sym])
also have "... ≤ nsqn (rt (σ ?nhip)) dip"
by (rule nsqns)
also have "... ≤ sqn (rt (σ ?nhip)) dip"
by (rule nsqn_sqn)
finally have "0 < sqn (rt (σ ?nhip)) dip".

have "rt (σ i) ⊑ dip rt (σ' ?nhip)"
proof (cases "dip∈vD(rt (σ' ?nhip))")
  assume "dip∈vD(rt (σ' ?nhip))"
  with prem (dip∈vD(rt (σ i))) and (?nhip ≠ dip)
  have "rt (σ i) ⊑ dip rt (σ' ?nhip)" by auto
  moreover from "∀ j. quality_increases (σ j) (σ' j)"
  have "quality_increases (σ ?nhip) (σ' ?nhip)" ..
  ultimately show ?thesis
using ⟨dip∈kD(rt (σ ?nhip)) ⟩
by (rule strictly_fresher_quality_increases_right)

next
assume "dip∈vD(rt (σ ?nhip))"
with ⟨dip∈kD(rt (σ ?nhip)) ⟩ have "dip∈iD(rt (σ ?nhip))" ..
hence "the (flag (rt (σ ?nhip)) dip) = inv"
by auto
have "nsqn (rt (σ i)) dip ≤ nsqn (rt (σ ?nhip)) dip"
by (rule nsqns)
also from ⟨dip∈iD(rt (σ ?nhip)) ⟩ have "⟨dip∈kD(rt (σ ?nhip)) ⟩"
have "dip∈iD(rt (σ ?nhip)) dip - 1" ..
also have "⟨dip∈iD(rt (σ ?nhip)) ⟩"
proof -
from ∀j. quality_increases (σ j) (σ' j) have "quality_increases (σ ?nhip) (σ' ?nhip)" ..
hence "∀ip. sqn (rt (σ ?nhip)) ip ≤ sqn (rt (σ' ?nhip)) ip" by auto
hence "sqn (rt (σ ?nhip)) dip = sqn (rt (σ' ?nhip)) dip - 1" ..
also have "... < sqn (rt (σ' ?nhip)) dip"
proof (rule vD_nsqn_sqn [THEN sym])
from ⟨dip∈vD(rt (σ' (the (nhop (rt (σ' i)) dip)))) ⟩
and ⟨σ' i = σ i ⟩ show "dip∈vD(rt (σ' ?nhip))" by simp
qed
finally have "nsqn (rt (σ i)) dip < nsqn (rt (σ' ?nhip)) dip" .
moreover from ⟨dip∈vD(rt (σ' (the (nhop (rt (σ' i)) dip)))) ⟩
and ⟨σ' i = σ i ⟩ have "dip∈kD(rt (σ' ?nhip))" by auto
ultimately show "rt (σ i) ⊏ dip rt (σ' ?nhip)"
using ⟨dip∈kD(rt (σ i)) ⟩ by - (rule rt_strictly_fresher_ltI)
with ⟨σ' i = σ i ⟩ show "rt (σ i) ⊏ dip rt (σ' (the (nhop (rt (σ' i)) dip)))" by simp
qed

thus ?thesis unfolding Let_def .
qed

lemma seq_nhop_quality_increases:
shows "opaodv i |= (otherwith (op=) {i})
    (orecvmsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
    other quality_increases {i} →
    global (λσ. ∀dip. let nhip = the (nhop (rt (σ' i)) dip)
        in dip∈vD (rt (σ i)) ∩ vD (rt (σ' nhip)) ∧ nhip ≠ dip
        ⟷ (rt (σ i)) ⊏ dip (rt (σ nhp)))"
by (rule oinvariant_weakenE [OF seq_nhop_quality_increases'] [auto dest!: onlD])

end

0.11 Routing graphs and loop freedom

theory Loop_Freedom
imports Aodv_Predicates Fresher
begin

Define the central theorem that relates an invariant over network states to the absence of loops in the associate routing graph.

definition rt_graph :: "(ip ⇒ state) ⇒ ip ⇒ ip rel"
where
"rt_graph σ = (λdip.
    {(ip, ip') | ip ip' dsn dsk hops pre.
       ip ≠ dip ∧ rt (σ ip) dip = Some (dsn, dsk, val, hops, ip', pre)})"
Given the state of a network $\sigma$, a routing graph for a given destination IP address $\text{dip}$ abstracts the details of routing tables into nodes (IP addresses) and vertices (valid routes between IP addresses).

**lemma rt_graphE [elim]:**

fixes $n$ dip ip ip'

assumes "$(ip, ip') \in rt_graph \sigma \text{ dip}\n
shows "ip $\neq$ dip $\land$ (exists r. rt $\sigma$ ip) = r $\land$ (exists den dsk hops pre. r dip = Some (dsn, dsk, val, hops, ip', pre))"

using assms unfolding rt_graph_def by auto

**lemma rt_graph_vD [dest]:**

"$\forall$ ip ip' $\sigma$. dip. $(ip, ip') \in rt_graph \sigma$ dip $\implies$ dip $\in vD(rt (\sigma ip))"

unfolding rt_graph_def vD_def by auto

**lemma rt_graph_vD_trans [dest]:**

"$\forall$ ip ip' $\sigma$. dip. $(ip, ip') \in (rt_graph \sigma$ dip)$^+$$\implies$ dip $\in vD(rt (\sigma ip))"

by (erule converse_transclE) auto

**lemma rt_graph_not_dip [dest]:**

"$\forall$ ip ip' $\sigma$. dip. $(ip, ip') \in rt_graph \sigma$ dip $\implies ip \neq dip"

unfolding rt_graph_def by auto

**lemma rt_graph_not_dip_trans [dest]:**

"$\forall$ ip ip' $\sigma$. dip. $(ip, ip') \in (rt_graph \sigma$ dip)$^+$$\implies ip \neq dip"

by (erule converse_transclE) auto

NB: the property below cannot be lifted to the transitive closure

**lemma rt_graph_nhip_is_nhop [dest]:**

"$\forall$ ip ip' $\sigma$. dip. $(ip, ip') \in rt_graph \sigma$ dip $\implies ip' = the (nhop (rt (\sigma ip)) dip)"

unfolding rt_graph_def by auto

**theorem inv_to_loop_freedom:**

assumes "$\forall$ dip. let nhip = the (nhop (rt (\sigma i)) dip) $\implies$ in dip $\in vD (rt (\sigma i)) \land vD (rt (\sigma nhip)) \land nhip \neq dip $\implies$ (rt (\sigma i)) $\sqsubseteq$ dip $\sqsubseteq rt (\sigma nhip)"

shows "$\forall$ dip. irrefl ((rt_graph $\sigma$ dip)$^+)"

using assms proof (intro allI)

fix $\sigma :: "ip \Rightarrow state"$ and dip

assume inv: "$\forall$ dip. $\exists nhip = the (nhop (rt (\sigma i)) dip) $\implies$ in dip $\in vD(rt (\sigma i)) \land vD(rt (\sigma nhip)) \land nhip \neq dip $\implies$ rt (\sigma i) $\sqsubseteq$ dip rt (\sigma nhip)"

{ fix ip ip'

  assume "$(ip, ip') \in (rt_graph \sigma$ dip)$^+"
  and "dip $\in vD(rt (\sigma ip'))"
  and "ip' \neq dip"
  hence "rt (\sigma ip) $\sqsubseteq$ dip rt (\sigma ip')"

  proof induction

    fix nhip

    assume "$(ip, nhip) \in rt_graph \sigma$ dip"

    and "dip $\in vD(rt (\sigma nhip))"

    and "nhip \neq dip"

    from $(ip, nhip) \in rt_graph \sigma$ dip have "dip $\in vD(rt (\sigma ip))"

    and "nhip = the (nhop (rt (\sigma ip)) dip)"

    by auto

    from $(ip, nhip) \in rt_graph \sigma$ dip have "dip $\in vD(rt (\sigma nhip))" ..

    with nhip = the (nhop (rt (\sigma ip)) dip)

    and nhip $\neq$ dip

    and inv

    show "rt (\sigma ip) $\sqsubseteq$ dip rt (\sigma nhip)"

    by (clarsimp simp: Let_def)

  next

    fix nhip nhip'

    assume "$(ip, nhip) \in (rt_graph \sigma$ dip)$^+"


and \( (\text{nhip}, \text{nhip}') \in \text{rt}\_\text{graph} \sigma \text{dip} \)
and \( \text{IH: } [\dip \in \mathcal{V}(\text{rt} (\sigma \text{nhip})); \text{nhip} \neq \dip ] \implies \text{rt} (\sigma \text{ip}) \sqsubseteq_{\text{dip}} \text{rt} (\sigma \text{nhip}) \)
and \( \text{dip} \in \mathcal{V}(\text{rt} (\sigma \text{nhip}')) \)
and \( \text{nhip}' \neq \text{dip} \)
from \( (\text{nhip}, \text{nhip}') \in \text{rt}\_\text{graph} \sigma \text{dip} \) have 1: \( \text{dip} \in \mathcal{V}(\text{rt} (\sigma \text{nhip})) \)
and 2: \( \text{nhip} \neq \text{dip} \)
and \( \text{nhip}' = \text{the} (\text{nhop} (\text{rt} (\sigma \text{nhip}))) \text{dip} \)
by auto
from 1 2 have \( \text{rt} (\sigma \text{ip}) \sqsubseteq_{\text{dip}} \text{rt} (\sigma \text{nhip}) \)
proof (rule IH)
also have \( \text{rt} (\sigma \text{nhip}) \sqsubseteq_{\text{dip}} \text{rt} (\sigma \text{nhip}') \)
proof -
from \( \dip \in \mathcal{V}(\text{rt} (\sigma \text{nhip})) \) and \( \dip \in \mathcal{V}(\text{rt} (\sigma \text{nhip}')) \) have \( \dip \in \mathcal{V}(\text{rt} (\sigma \text{nhip})) \cap \mathcal{V}(\text{rt} (\sigma \text{nhip}')) \) ..
with \( \text{nhip}' \neq \dip \)
and \( \text{nhip}' = \text{the} (\text{nhop} (\text{rt} (\sigma \text{nhip}))) \text{dip} \)
and inv
show \( \text{rt} (\sigma \text{nhip}) \sqsubseteq_{\text{dip}} \text{rt} (\sigma \text{nhip}') \)
by (clarsimp simp: Let_def)
qed 
finally show \( \text{rt} (\sigma \text{ip}) \sqsubseteq_{\text{dip}} \text{rt} (\sigma \text{nhip}') \).
qed 

show "irrefl ((\text{rt}\_\text{graph} \sigma \text{dip})\+)")
unfolding irrefl_def proof (intro allI notI)
fix \text{ip}
assume "((\text{ip}, \text{ip}) \in (\text{rt}\_\text{graph} \sigma \text{dip})\+)" 
moreover then have "\text{dip} \in \mathcal{V}(\text{rt} (\sigma \text{ip}))"
and "\text{ip} \neq \text{dip}"
by auto
ultimately have "\text{rt} (\sigma \text{ip}) \sqsubseteq_{\text{dip}} \text{rt} (\sigma \text{ip})" by (rule fresher)
thus False by simp
qed

end

0.12 Lift and transfer invariants to show loop freedom

theory Aodv_Loop_Freedom
imports "../AWN/OClosed_Transfer" "../AWN/Qmsg_Lifting" Global_Invariants Loop_Freedom
begin

0.12.1 Lift to parallel processes with queues

lemma par_step_no_change_on_send_or_receive:
fixes \( \sigma \; s \; a \; \sigma' \; s' \)
assumes \( (((\sigma, s), a, (\sigma', s')) \in \text{oopar}_s i (\text{oseqp}_s \Gamma_{\text{AODV}} i) (\text{seqp}_s \Gamma_{\text{QMSG}}))" 
and "a \neq \tau"
shows "\sigma' \; i = \sigma \; i"
using assms by (rule qmsg_no_change_on_send_or_receive)

lemma par_nhop_quality_increases:
shows "\sigma \; \text{msg} \models (\text{otherwith (op=} i) (\text{orecmsg} (\lambda \sigma \; m. \text{msg}\_\text{fresh} \; \sigma \; m \land \text{msg}\_\text{zhops} \; m)), \text{other quality increases} \; \{i\} \implies \text{global} (\lambda \sigma. \forall \dip. \text{let} \; \text{nhip} = \text{the} (\text{nhop} \; (\text{rt} \; (\sigma \; i))) \; \text{dip} \; \text{in} \; \dip \in \mathcal{V} \; (\text{rt} \; (\sigma \; i)) \land \mathcal{V} \; (\text{rt} \; (\sigma \; \text{nhip})) \land \text{nhip} \neq \dip \rightarrow \text{rt} \; (\sigma \; i) \sqsubseteq_{\text{dip}} \text{rt} \; (\sigma \; \text{nhip})))" 
proof (rule lift_into_qmsg [OF seq_nhop_quality_increases])
show "\sigma \; \text{msg} \models A (\text{otherwith (op=} i) (\text{orecmsg} (\lambda \sigma \; m. \text{msg}\_\text{fresh} \; \sigma \; m \land \text{msg}\_\text{zhops} \; m)), \text{other quality increases} \; \{i\} \implies \text{globala} (\lambda (\sigma, \_, \sigma'). \text{quality increases} \; (\sigma \; i) \; (\sigma' \; i))" 
proof (rule ostep_invariant_weakenE [OF oquality_increases], simp_all)
fix t :: "((nat ⇒ state) × (state, msg, pseqp, pseqp label) seqp), msg seq_action) transition"
assume "onll Γ_AODV \((λ(((), _), _, (', _)). ∀j. quality_increases (σ j) (σ' j))\)
thus "quality_increases (fst (fst t) i) (fst (snd (snd t)) i)"
  by (cases t) (clarsimp dest!: onllD, metis aodv_ex_label)
next
fix σ σ' a
assume "otherwith (op=) {i}
  (orecvmsg (λσ m. msg_fresh σ m ∧ msg_zhops m)) σ σ' a"
thus "otherwith quality_increases (i) (orecvmsg (λ_. rreq_rrep_sn)) σ σ' a"
  by (erule weaken_otherwith, auto)
qed

lemma par_rreq_rrep_sn_quality_increases:
  "opaodv i \(((i qmsg =A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ), other (λ_. _ True) \{i\} →)
  glabal (λ(σ, _, σ'). quality_increases (σ i) (σ' i)))"
proof -
  have "opaodv i \(\{i\} =A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ), other (λ_. _ True) \{i\} →)
  glabal (λ(σ, _, σ'). quality_increases (σ i) (σ' i))"
  by (rule ostep_invariant_weakenE)
  (auto dest!: ostep_invariant_weakenE [OF olocal_quality_increases])
  (auto dest!: onllD seq1D elim!: aodv_ex_labelE)
  hence "opaodv i \(((i qmsg =A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ), other (λ_. _ True) \{i\} →)
  glabal (λ(σ, _, σ'). quality_increases (σ i) (σ' i)))"
  by (rule lift_step_into_qmsg_statelessassm) simp_all
thus ?thesis by rule auto
qed

lemma par_rreq_rrep_nsqn_fresh_any_step:
  shows "opaodv i \(((i qmsg =A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ), other (λ_. _ True) \{i\} →)
  glabal (λ(σ, a, σ'). ancast (msg_fresh σ) a))"
proof -
  have "opaodv i \(\{i\} =A (λσ _. (orecvmsg (λ_. rreq_rrep_sn) σ), other (λ_. _ True) \{i\} →)
  glabal (λ(σ a, σ'). ancast (msg_fresh σ) a)"
  by (rule ostep_invariant_weakenE)
  (cases t) (clarsimp dest!: onllD, metis aodv_ex_label)
  hence "opaodv i \(((i qmsg =A (λσ _. (orecvmsg (λ_. rreq_rrep_sn) σ), other (λ_. _ True) \{i\} →)
  glabal (λ(σ a, σ'). ancast (msg_fresh σ) a))"
  by (rule lift_step_into_qmsg_statelessassm) simp_all
thus ?thesis by rule auto
qed

lemma par_anycast_msg_zhops:
  shows "opaodv i \(((i qmsg =A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ), other (λ_. _ True) \{i\} →)
  glabal (λ(_, a, _). ancast msg_zhops a))"
proof -
  from anycast_msg_zhops initail_aodv oadv_trans oadv_trans
  have "opaodv i \(\{i\} =A (act TT, other (λ_. _ True) \{i\} →)
  seq1l \(\{onll Γ_AODV (λ(_, a, _). ancast msg_zhops a))"\)" 
  by (rule open_seq_step_invariant)
  hence "opaodv i \(\{i\} =A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ), other (λ_. _ True) \{i\} →)
  glabal (λ(_, a, _). ancast msg_zhops a)"
  by (rule ostep_invariant_weakenE)
  (cases t) (clarsimp dest!: seq1lD onllD, metis aodv_ex_label)
  qed simp_all
  hence "opaodv i \(((i qmsg =A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ), other (λ_. _ True) \{i\} →)
  glabal (λ(_, a, _). ancast msg_zhops a))"
by (rule lift_step_into_qmsg_statelessassm) simp_all
thus ?thesis by rule auto
qed

0.12.2 Lift to nodes

lemma node_step_no_change_on_send_or_receive:
assumes "((σ, NodeS i P R), a, (σ', NodeS i' P' R')) ∈ onode_sos
(oparp_sos i (oseqp_sos Γ AODV i) (seqp_sos Γ QMSG))"
and "a ≠ τ"
shows "σ' i = σ i"
using assms by (cases a) (auto elim!: par_step_no_change_on_send_or_receive)

lemma node_nhop_quality_increases:
shows "⟨i : opaodv i ((i qmsg : R) ∪
(oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
other quality_increases (i)}
→ global (λσ. ∀ dip. let nhip = the (nhop (rt (σ i)) dip)
in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
→ (rt (σ i)) ⊏ dip (rt (σ nhip)))⟩"
by (rule node_lift [OF par_nhop_quality_increases]) auto

lemma node_quality_increases:
shows "⟨i : opaodv i ((i qmsg : R) ∪
(oarrivemsg (λσ _. oarrivemsg (λ_. rreq_rrep_sn) σ,
other (λ_. True) {i}) →
globala (λ(σ, a, σ') . quality_increases (σ i) (σ' i)))⟩"
by (rule node_lift_step_statelessassm [OF par_rreq_rrep_sn_quality_increases]) simp

lemma node_rreq_rrep_nsqn_fresh_any_step:
shows "⟨i : opaodv i ((i qmsg : R) ∪
(oarrivemsg (λσ _. oarrivemsg (λ_. rreq_rrep_sn) σ,
other ((λ_ _. True) {i} →
globala (λ(_, a, _). castmsg (msg_fresh σ) a))
⟩
rreq_rrep_nsqn_fresh (λσ _. rreq_rrep_sn) σ a)⟩
by (rule node_lift_anycast_statelessassm [OF par_rreq_rrep_nsqn_fresh_any_step])

lemma node_anycast_msg_zhops:
shows "⟨i : opaodv i ((i qmsg : R) ∪
(oarrivemsg (λσ _. oarrivemsg (λ_. rreq_rrep_sn) σ,
other ((λ_ _. True) {i} →
globala (λ(_, a, _). castmsg msg_zhops a))
⟩
rreq_rrep_nsqn_fresh (λσ _. rreq_rrep_sn) σ a)⟩
by (rule node_lift_anycast_statelessassm [OF par_anycast_msg_zhops])

lemma node_silent_change_only:
shows "⟨i : opaodv i ((i qmsg : R) ∪
(oarrivemsg (λσ _. oarrivemsg (λ_. rreq_rrep_sn) σ,
other ((λ_ _. True) {i} →
globala (λ(σ, a, σ') . a ≠ τ → σ' i = σ i))⟩"
proof (rule ostep_invariantI, simp (no_asm), rule impl)
fix σ ζ a σ' ζ'
assume or: "(σ, ζ) ∈ oreachable ((i : opaodv i ((i qmsg : R)∪
(oarrivemsg (λσ_. oarrivemsg (λ_. True) σ)
(or (λ_. True) {i})))
and tr: "((σ, ζ), a, (σ', ζ')) ∈ trans ((i : opaodv i ((i qmsg : R)∪
and "a ≠ τ n"
from or obtain p R where "ζ = NodeS i p R"
by - (drule node_net_state, metis)
with tr have "((σ, NodeS i p R), a, (σ', ζ'))
∈ onode_sos (oparp_sos i (trans (opaodv i)) (trans qmsg))"
by simp
thus "σ' i = σ i" using "a ≠ τ n"
by (cases rule: onode_sos_cases)
(auto elim: qmsg_no_change_on_send_or_receive)
qed
0.12.3 Lift to partial networks

lemma arrive_rreq_rrep_nsqn_fresh_inc_sn [simp]:
assumes "oarrivemsg (λσ . msg_fresh σ m ∧ P σ m) σ m"
shows "oarrivemsg (λ_. rreq_rrep_sn) σ m"
using assms by (cases m) auto

lemma opnet_nhop_quality_increases:
shows "opnet (λi . opaodv i ⟨⟨i qmsg : R⟩⟩ p =
(otherwith (op=) (net_tree_ips p)
(oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
other_quality_increases (net_tree_ips p) →)
global (λσ. ∀ i∈net_tree_ips p. ∀ dip.
let nhip = the (nhop (rt (σ i)) dip)
in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
→ (rt (σ i)) ⊆ dip (rt (σ nhip)))"
proof (rule pnet_lift [OF node_nhop_quality_increases])
next
simp_all
qed

proof (rule ostep_invariantI, simp (no_asm))
next
simp_all
qed

thus "oarrivemsg (λ_. rreq_rrep_sn) σ, other (λ_. True) {i} →) globala (λ(σ, a, σ').
castmsg (λm. msg_fresh σ m ∧ msg_zhops m) a"
proof (rule ostep_invariantD [OF node_rreq_rrep_nsqn_fresh_any_step])

and from or tr am have "castmsg (msg_fresh σ) a"
by (auto dest!: ostep_invariantD [OF node_rreq_rrep_nsqn_fresh_any_step])

next
next

and am: "oarrivemsg (λ_. rreq_rrep_sn) σ a"
from or tr am have "castmsg (msg_zhops m) a"
by (auto dest!: ostep_invariantD [OF node_anycast_msg_zhops])

ultimately show "castmsg (λm. msg_fresh σ m ∧ msg_zhops m) a"
by (case_tac a) auto

qed

thus "(i : opaodv i ⟨⟨i qmsg : R⟩⟩ o =_A (λσ _. oarrivemsg (λ_. rreq_rrep_sn) σ, other (λ_. True) {i})
other_quality_increases {i} →) globala (λ(σ, a, _).
castmsg (λm. msg_fresh σ m ∧ msg_zhops m) a)"

by rule auto

next

fix i R

show "(i : opaodv i ⟨⟨i qmsg : R⟩⟩ o =_A (λσ _. oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m) σ, other_quality_increases {i} →) globala (λ(σ, a, _).
castmsg (λm. msg_fresh σ m ∧ msg_zhops m) a)

by (rule ostep_invariant_weakenE [OF node_silent_change_only]) auto

next

fix i R

show "(i : opaodv i ⟨⟨i qmsg : R⟩⟩ o =_A (λσ _. oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m) σ, other_quality_increases {i} →) globala (λ(σ, a, _).
a ≠ τ ∧ (∀ i d. a ≠ i:deliver(d) → σ i = σ' i)"
by (rule ostep_invariant_weakenE [OF node_quality_increases]) auto

qed simp_all

0.12.4 Lift to closed networks

lemma onet_nhop_quality_increases:
shows "onet (open (λi . opaodv i ⟨⟨i qmsg⟩⟩ p) =
(λ_. True, other quality_increases (net_tree_ips p) →)
global (λσ. ∀ i∈net_tree_ips p. ∀ dip.
let nhip = the (nhop (rt (σ i)) dip)
in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
→ (rt (σ i)) ⊆ dip (rt (σ nhip)))"
0.12.5 Transfer into the standard model

interpretation aodv_openproc: openproc paodv opaodv id
where "aodv_openproc.initmissing = initmissing"

proof -
  show "openproc paodv opaodv id"
proof unfold_locales
  fix i :: ip
  have "{(σ, ζ). (σ i, ζ) ∈ σ_AODV i ∧ (∀ j. j ≠ i → σ j ∈ fst ' σ_AODV j)} ⊆ σ_AODV'
unfolding σ_AODV_def σ_AODV'_def
proof (rule equalityD1)
  show "∀ f p. {(σ, ζ). (σ i, ζ) ∈ {(f i, p)} ∧ (∀ j. j ≠ i → σ j ∈ fst ' {(f j, p)})} = {(f, p)}" by (rule set_eqI) auto
qed
  thus "{(σ i, q) | σ ζ s. s ∈ init (paodv i) ∧ (σ i, ζ) = id s ∧ (∀ j. j ≠ i → σ j ∈ (fst o id) ' init (paodv j))} ⊆ init (opaodv i)"
  by simp
next
  show "∀ j. init (paodv j) ≠ {}"
unfolding σ_AODV_def by simp
next
  fix i s a s' σ σ'
  assume "σ i = fst (id s)"
and "σ' i = fst (id s')"
and "(s, a, s') ∈ trans (paodv i)"
then obtain q q' where "s = (σ i, q)"
  and "s' = (σ' i, q')"
  and "((σ i, q), a, (σ' i, q')) ∈ trans (paodv i)"
  by (cases s, cases s') auto
from this(3) have "((σ, q), a, (σ', q')) ∈ trans (opaodv i)"
  by simp (rule open_seqp_action [OF aodv_wf])
with ⟨s = (σ i, q)⟩ and ⟨s' = (σ' i, q')⟩
  show "((σ, snd (id s)), a, (σ', snd (id s')))) ∈ trans (opaodv i)"
  by simp
qed
then interpret op: openproc paodv opaodv id .
have [simp]: "∀ i. (SOME x. x ∈ (fst o id) ' init (paodv i)) = aodv_init i"
unfolding σ_AODV_def by simp
hence "∀ i. openproc.initmissing paodv id i = initmissing i"
unfolding op.initmissing_def op.someinit_def initmissing_def
by (auto split: option.split)
thus "openproc.initmissing paodv id = initmissing" ..
qed

interpretation aodv_openproc_par_qmsg: openproc_parq paodv opaodv id qmsg
where "aodv_openproc_par_qmsg.netglobal = netglobal"
and "aodv_openproc_par_qmsg.initmissing = initmissing"
proof -
show "openproc_parq paodv opaodv id qmsg"
by (unfold_locales) simp
then interpret opq: openproc_parq paodv opaodv id qmsg .
have im: "∀ σ. openproc.initmissing (λi. paodv i (λ (p, q). (fst (id p), snd (id p), q)) σ = initmissing σ"
unfolding opq.initmissing_def opq.someinit_def initmissing_def
unfolding σ_AODV_def σ_QMSG_def
by (clarsimp cong: option.case_cong)
thus "openproc.initmissing (λi. paodv i (λ (p, q). (fst (id p), snd (id p), q))) = initmissing" by (rule ext)
have "∀ P σ. openproc.netglobal (λi. paodv i (λ (p, q). (fst (id p), snd (id p), q))) P σ = netglobal P σ"
unfolding opq.netglobal_def netglobal_def opq.initmissing_def initmissing_def opq.someinit_def
unfolding σ_AODV_def σ_QMSG_def
by (clarsimp cong: option.case_cong simp del: One_nat_def simp add: fst_initmissing_netgmap_default_aodv_init_netlift [symmetric, unfolded initmissing_def])
thus "openproc.netglobal (λi. paodv i (λ (p, q). (fst (id p), snd (id p), q))) = netglobal" by auto
qed

lemma net_nhop_quality_increases:
assumes "wf_net_tree n"
shows "closed (pnet (λi. paodv i (λ (qmsg) n)) |= netglobal
(λσ. ∀ i dip. let nhip = the (nhop (rt (σ i)) dip)
in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
→ (rt (σ i)) ⊏ dip (rt (σ nhip)))")
(is "_ |= netglobal (λσ. ∀ i. ?inv σ i))"
proof -
from "wf_net_tree n"
have proto: "closed (pnet (λi. paodv i (λ (qmsg) n)) |= netglobal (λσ. ∀ i ∈ net_tree_ips n. ∀ dip. let nhip = the (nhop (rt (σ i)) dip)
in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
→ (rt (σ i)) ⊏ dip (rt (σ nhip)))")
by (rule aodv_openproc_par_qmsg.close_opnet [OF _ onet_nhop_quality_increases])
show ?thesis
unfolding invariant_def opnet_sos.opnet_tau1
proof (rule, simp only: aodv_openproc_par_qmsg.netglobalsimp
fst_initmissing_netgmap_pair_fst, rule allI)
fix σ i
assume sr: "σ ∈ reachable (closed (pnet (λi. paodv i (λ (qmsg) n)) TT"
hence "∀ i ∈ net_tree_ips n. ?inv (fst (initmissing (netgmap fst σ))) i"
by - (drule invariantD [OF proto],
simp only: aodv_openproc_par_qmsg.netglobalsimp
fst_initmissing_netgmap_pair_fst)
thus "?inv (fst (initmissing (netgmap fst σ))) i" by cases "i ∈ net_tree_ips n"
assume "i ∉ net_tree_ips n"
proof cases "i ∈ net_tree_ips n"
assume "i ∉ net_tree_ips n"
from sr have "σ ∈ reachable (pnet (λi. paodv i (λ (qmsg) n)) TT" ..
hence "net_ips σ = net_tree_ips n" ..
with (i ∉ net_tree_ips n) have "i ∉ net_ips σ" by simp
86
hence "(fst (initmissing (netgmap fst σ))) i = aodv_init i"
  by simp
thus thesis by simp
qed metis
qed

0.12.6 Loop freedom of AODV

theorem aodv_loop_freedom:
  assumes "wf_net_tree n"
  shows "closed (pnet (λi. paodv i □ qmsg) n) |= netglobal (λσ. ∀ dip. irrefl ((rt_graph σ dip)⁺))"
  using assms by (rule aodv_openproc_par_qmsg.netglobal_weakenE
       [OF net_nhop_quality_increases inv_to_loop_freedom])
end
Chapter 1

Variant A: Skipping the RREQ ID

Explanation [4, §10.1]: AODV does not need the route request identifier. This number, in combination with the IP address of the originator, is used to identify every RREQ message in a unique way. This variant shows that the combination of the originator’s IP address and its sequence number is just as suited to uniquely determine the route request to which the message belongs. Hence, the route request identifier field is not required. This can then reduce the size of the RREQ message.

1.1 Predicates and functions used in the AODV model

theory A_Aodv_Data
imports A_Norreqid
begin

1.1.1 Sequence Numbers

Sequence numbers approximate the relative freshness of routing information.

definition inc :: "sqn ⇒ sqn"
  where "inc sn ≡ if sn = 0 then sn else sn + 1"

lemma less_than_inc [simp]: "x ≤ inc x"
  unfolding inc_def by simp

lemma inc_minus_suc_0 [simp]: "inc x - Suc 0 = x"
  unfolding inc_def by simp

lemma inc_never_one' [simp, intro]: "inc x ̸= Suc 0"
  unfolding inc_def by simp

lemma inc_never_one [simp, intro]: "inc x ̸= 1"
  by simp

1.1.2 Modelling Routes

A route is a 6-tuple, \((dsn, dsk, flag, hops, nhip, pre)\) where \(dsn\) is the ‘destination sequence number’, \(dsk\) is the ‘destination-sequence-number status’, \(flag\) is the route status, \(hops\) is the number of hops to the destination, \(nhip\) is the next hop toward the destination, and \(pre\) is the set of ‘precursor nodes’ those interested in hearing about changes to the route.

type_synonym r = "sqn × k × f × nat × ip × ip set"

definition proj2 :: "r ⇒ sqn" ("\(\pi_2\)"
  where "\(\pi_2 \equiv \lambda(dsn, _, _, _, _, _). dsn\)"

definition proj3 :: "r ⇒ k" ("\(\pi_3\)"
  where "\(\pi_3 \equiv \lambda(_, dsk, _, _, _, _). dsk\)"

definition proj4 :: "r ⇒ f" ("\(\pi_4\)"

where "\( \pi_4 \equiv \lambda(\_, \_, \text{flag}, \_, \_, \_). \text{flag} \)"
definition \text{proj5} :: "r \Rightarrow \text{nat}" ("\( \pi_5 \)"
  where "\( \pi_5 \equiv \lambda(\_, \_, \_, \text{hops}, \_, \_). \text{hops} \)"

definition \text{proj6} :: "r \Rightarrow \text{ip}" ("\( \pi_6 \)"
  where "\( \pi_6 \equiv \lambda(\_, \_, \_, \_, \text{nhip}, \_). \text{nhip} \)"

definition \text{proj7} :: "r \Rightarrow \text{ip set}" ("\( \pi_7 \)"
  where "\( \pi_7 \equiv \lambda(\_, \_, \_, \_, \_, \text{pre}). \text{pre} \)"

text{lemma} \text{projs [simp]}:
  "\( \pi_2(dsn, dsk, \text{flag}, \text{hops}, \text{nhip}, \text{pre}) = dsn \)"
  "\( \pi_3(dsn, dsk, \text{flag}, \text{hops}, \text{nhip}, \text{pre}) = dsk \)"
  "\( \pi_4(dsn, dsk, \text{flag}, \text{hops}, \text{nhip}, \text{pre}) = \text{flag} \)"
  "\( \pi_5(dsn, dsk, \text{flag}, \text{hops}, \text{nhip}, \text{pre}) = \text{hops} \)"
  "\( \pi_6(dsn, dsk, \text{flag}, \text{hops}, \text{nhip}, \text{pre}) = \text{nhip} \)"
  "\( \pi_7(dsn, dsk, \text{flag}, \text{hops}, \text{nhip}, \text{pre}) = \text{pre} \)"
  by (clarsimp simp: \text{proj2_def} \text{proj3_def} \text{proj4_def} \text{proj5_def} \text{proj6_def} \text{proj7_def})+

lemma \text{proj3_pred [intro]}: "[ [ \[ \text{kno}; \text{unk} \] ] \Rightarrow P (\pi_3 \ x) "
  by (rule \text{k.induct})

lemma \text{proj4_pred [intro]}: "[ [ \[ \text{val}; \text{inv} \] ] \Rightarrow P (\pi_4 \ x) "
  by (rule \text{f.induct})

lemma \text{proj6_pair_snd [simp]}:
  fixes \( \text{dsn'} \ r \)
  shows "\( \pi_6(\text{dsn'}, \text{snd}(\ r)) = \pi_6(\ r) \)"
  by (cases \( r \)) simp

1.1.3 Routing Tables
Routing tables map ip addresses to route entries.

type synonym \( \text{rt} = \text{ip} \Rightarrow r \) 

syntax 
  "\_Sigma_route" :: "\text{rt} \Rightarrow \text{ip} \Rightarrow r" ("\sigma_{\text{route}}(\_, \_')")

translations 
  "\sigma_{\text{route}}(\text{rt}, \text{dip})" \Rightarrow "\text{rt} \ \text{dip}"

definition \text{sqn} :: "\text{rt} \Rightarrow \text{ip} \Rightarrow \text{sqn}"
  where "\text{sqn} \text{rt} \ \text{dip} \equiv \text{case } \sigma_{\text{route}}(\text{rt}, \text{dip}) \ \text{of } \text{Some } \text{r} \ \Rightarrow \pi_2(\text{r}) \ | \ \text{None} \ \Rightarrow 0"

definition \text{sqnf} :: "\text{rt} \Rightarrow \text{ip} \Rightarrow k"
  where "\text{sqnf} \text{rt} \ \text{dip} \equiv \text{case } \sigma_{\text{route}}(\text{rt}, \text{dip}) \ \text{of } \text{Some } \text{r} \ \Rightarrow \pi_3(\text{r}) \ | \ \text{None} \ \Rightarrow \text{unk}"

abbreviation \text{flag} :: "\text{rt} \Rightarrow \text{ip} \Rightarrow f"
  where "\text{flag} \text{rt} \ \text{dip} \equiv \text{map_option} \ \pi_4 \ (\sigma_{\text{route}}(\text{rt}, \text{dip}))"

abbreviation \text{dhops} :: "\text{rt} \Rightarrow \text{ip} \Rightarrow \text{nat}"
  where "\text{dhops} \text{rt} \ \text{dip} \equiv \text{map_option} \ \pi_5 \ (\sigma_{\text{route}}(\text{rt}, \text{dip}))"

abbreviation \text{nhop} :: "\text{rt} \Rightarrow \text{ip} \Rightarrow \text{ip}"
  where "\text{nhop} \text{rt} \ \text{dip} \equiv \text{map_option} \ \pi_6 \ (\sigma_{\text{route}}(\text{rt}, \text{dip}))"

abbreviation \text{precs} :: "\text{rt} \Rightarrow \text{ip} \Rightarrow \text{ip set}"
  where "\text{precs} \text{rt} \ \text{dip} \equiv \text{map_option} \ \pi_7 \ (\sigma_{\text{route}}(\text{rt}, \text{dip}))"

definition \text{vD} :: "\text{rt} \Rightarrow \text{ip set}"
  where "\text{vD} \text{rt} \equiv \{\text{dip. flag rt dip = Some val}\}"
definition \(iD : \text{rt} \Rightarrow \text{ip set}\)
  where "\(iD \text{ rt} \equiv \{\text{dip. flag rt dip = Some inv}\}\)"

definition \(kD : \text{rt} \Rightarrow \text{ip set}\)
  where "\(kD \text{ rt} \equiv \{\text{dip. rt dip \neq None}\}\)"

lemma \(kD\_is\_vD\_and\_iD\): "\(kD \text{ rt} = vD \text{ rt} \cup iD \text{ rt}\)"
  unfolding kD_def vD_def iD_def by auto

lemma \(vD\_iD\_gives\_kD\) [simp]:
  \[
  \forall ip \text{ rt}. \, ip \in vD \text{ rt} \Rightarrow ip \in kD \text{ rt}
  \]
  unfolding kD_is_vD_and_iD by simp_all

lemma \(kD\_Some\) [dest]:
  fixes dip rt
  assumes "dip \in kD \text{ rt}"
  shows "\(\exists dsn \, dsk \, flag \, hops \, nhip \, pre. \sigma\_route(\text{rt}, dip) = Some (dsn, dsk, flag, hops, nhip, pre)\)"
  using assms unfolding kD_def by simp

lemma \(kD\_None\) [dest]:
  fixes dip rt
  assumes "dip /\in kD \text{ rt}"
  shows "\(\sigma\_route(\text{rt}, dip) = None\)"
  using assms unfolding kD_def by (metis (mono_tags) mem_Collect_eq)

lemma \(vD\_Some\) [dest]:
  fixes dip rt
  assumes "dip \in vD \text{ rt}"
  shows "\(\exists dsn \, dsk \, flag \, hops \, nhip \, pre. \sigma\_route(\text{rt}, dip) = Some (dsn, dsk, flag, hops, nhip, pre)\)"
  using assms unfolding vD_def by simp

lemma \(vD\_empty\) [simp]: "vD \text{ Map.empty} = {}"
  unfolding vD_def by simp

lemma \(iD\_Some\) [dest]:
  fixes dip rt
  assumes "dip \in iD \text{ rt}"
  shows "\(\exists dsn \, dsk \, flag \, hops \, nhip \, pre. \sigma\_route(\text{rt}, dip) = Some (dsn, dsk, flag, hops, nhip, pre)\)"
  using assms unfolding iD_def by simp

lemma \(val\_is\_vD\) [elim]:
  fixes ip rt
  assumes "ip \in kD(\text{rt})"
  and "the (flag rt ip) = val"
  shows "ip \in vD(\text{rt})"
  using assms unfolding vD_def by auto

lemma \(inv\_is\_iD\) [elim]:
  fixes ip rt
  assumes "ip \in kD(\text{rt})"
  and "the (flag rt ip) = inv"
  shows "ip \in iD(\text{rt})"
  using assms unfolding iD_def by auto

lemma \(iD\_flag\_is\_inv\) [elim, simp]:
  fixes ip rt
  assumes "ip \in iD(\text{rt})"
  shows "the (flag rt ip) = inv"
  proof -
from \( \text{ip} \in \text{iD}(\text{rt}) \) have "\( \text{ip} \in \text{kD}(\text{rt}) \)" by auto
with assms show ?thesis unfolding \( \text{id} \_\text{def} \) by auto
Qed

lemma \( \text{kD\_but\_not\_vD\_is\_iD} \) [elim]:
fixes ip rt
assumes "\( \text{ip} \in \text{kD}(\text{rt}) \)"
and "\( \text{ip} \notin \text{vD}(\text{rt}) \)"
shows "\( \text{ip} \in \text{iD}(\text{rt}) \)"
proof -
from \( \text{ip} \in \text{kD}(\text{rt}) \) obtain dsn dsk f hops nhop pre
where rtip: "\( \text{rt} \, \text{ip} = \text{Some (dsn, dsk, f, hops, nhop, pre)} \)"
by (metis kD_Some)
from \( \text{ip} \notin \text{vD}(\text{rt}) \) have "\( f \neq \text{val} \)"
proof (rule contrapos_nn)
assume "\( f = \text{val} \)"
with rtip have "\( \text{the (flag rt ip)} = \text{val} \)" by simp
with rtip have "\( \text{ip} \in \text{iD}(\text{rt}) \)" ..
Qed

lemma \( \text{vD\_or\_iD} \) [elim]:
fixes ip rt
assumes "\( \text{ip} \in \text{kD}(\text{rt}) \)"
and "\( \text{ip} \in \text{vD}(\text{rt}) \) = \( \Rightarrow \text{P rt ip} \)"
and "\( \text{ip} \in \text{iD}(\text{rt}) \) = \( \Rightarrow \text{P rt ip} \)"
shows "\( \text{P rt ip} \)"
proof -
from \( \text{ip} \in \text{kD}(\text{rt}) \) have "\( \text{ip} \in \text{vD}(\text{rt}) \cup \text{iD}(\text{rt}) \)"
by (simp add: kD_is_vD_and_iD)
thus ?thesis by (auto elim: assms(2-3))
Qed

lemma \( \text{proj5\_eq\_dhops} \) : "\( \text{\( \bigwedge \text{dip rt. dip} \in \text{kD}(\text{rt}) \) = \( \Rightarrow \pi_5(\text{the (rt dip)}) = \text{the (dhops rt dip)} \)""
unfolding sqn_def by (drule kD_Some) clarsimp

lemma \( \text{proj4\_eq\_flag} \) : "\( \text{\( \bigwedge \text{dip rt. dip} \in \text{kD}(\text{rt}) \) = \( \Rightarrow \pi_4(\text{the (rt dip)}) = \text{the (flag rt dip)} \)""
unfolding sqn_def by (drule kD_Some) clarsimp

lemma \( \text{proj2\_eq\_sqn} \) : "\( \text{\( \bigwedge \text{dip rt. dip} \in \text{kD}(\text{rt}) \) = \( \Rightarrow \pi_2(\text{the (rt dip)}) = \text{sqn rt dip} \)""
unfolding sqn_def by (drule kD_Some) clarsimp

lemma \( \text{kD\_sqnf\_is\_proj3} \) [simp]:
"\( \forall \text{ip rt. ip} \in \text{kD}(\text{rt}) \Rightarrow \text{sqnf rt ip} = \pi_3(\text{the (rt ip)}) \)"
unfolding sqnf_def by auto

lemma \( \text{vD\_flag\_val} \) [simp]:
"\( \forall \text{dip rt. dip} \in \text{vD (rt)} \Rightarrow \text{the (flag rt dip)} = \text{val} \)"
unfolding vD_def by clarsimp

lemma \( \text{kD\_update} \) [simp]:
"\( \forall \text{rt nip v. kD (rt(nip \mapsto v)) = insert nip (kD rt)} \)"
unfolding kD_def by auto

lemma \( \text{kD\_empty} \) [simp]: "\( \text{kD Map.empty = \{\}} \)"
unfolding kD_def by simp

lemma \( \text{ip\_equal\_or\_known} \) [elim]:
fixes rt ip ip'
assumes "\( \text{ip} = \text{ip'} \lor \text{ip} \in \text{kD}(\text{rt}) \)"
and "\( \text{ip} = \text{ip'} \Rightarrow \text{P rt ip ip'} \)"
and "\( [ \text{ip} \neq \text{ip'}; \text{ip} \in \text{kD}(\text{rt})]\) = \( \Rightarrow \text{P rt ip ip'} \)"
1.1.4 Updating Routing Tables

Routing table entries are modified through explicit functions. The properties of these functions are important in invariant proofs.

Updating Precursor Lists

definition addpre :: "r ⇒ ip set ⇒ r"
  where "addpre r npre ≡ let (dsn, dsk, flag, hops, nhip, pre) = r in
   (dsn, dsk, flag, hops, nhip, pre ∪ npre)"

text{lemma proj2_addpre:
fixes v pre
shows "π2(addpre v pre) = π2(v)"}
unfolding addpre_def by (cases v) simp

lemma proj3_addpre:
fixes v pre
shows "π3(addpre v pre) = π3(v)"
unfolding addpre_def by (cases v) simp

lemma proj4_addpre:
fixes v pre
shows "π4(addpre v pre) = π4(v)"
unfolding addpre_def by (cases v) simp

lemma proj5_addpre:
fixes v pre
shows "π5(addpre v pre) = π5(v)"
unfolding addpre_def by (cases v) simp

lemma proj6_addpre:
fixes dsn dsk flag hops nhip pre npre
shows "π6(addpre v npre) = π6(v)"
unfolding addpre_def by (cases v) simp

lemma proj7_addpre:
fixes dsn dsk flag hops nhip pre npre
shows "π7(addpre v npre) = π7(v) ∪ npre"
unfolding addpre_def by (cases v) simp

lemma addpre_empty: "addpre r {} = r"
unfolding addpre_def by simp

lemmas addpre_simps [simp] = proj2_addpre proj3_addpre proj4_addpre proj5_addpre proj6_addpre proj7_addpre addpre_empty addpre_r

definition addpreRT :: "rt ⇒ ip ⇒ ip set ⇒ rt"
  where "addpreRT rt dip npre ≡
    map_option (λs. rt (dip ↦→ addpre s npre)) (σ_route(rt, dip))"

lemma snd_addpre [simp]:
  "snd (dsn dsk' v pre. (dsn, snd(addpre (dsn', v) pre)) = addpre (dsn, v) pre"
unfolding addpre_def by clarsimp

lemma proj2_addpreRT [simp]:
  fixes ip rt ip' npre
assumes "ip ∈ kD rt"
and "ip' ∈ kD rt"
shows "π_2 (the (the (addpreRT rt ip' npre) ip)) = π_2 (the (rt ip))"
using assms [THEN kD_Some] unfolding addpreRT_def by clarsimp

lemma proj3_addpreRT [simp]:
fixes ip rt ip'
pre
assumes "ip ∈ kD rt"
and "ip' ∈ kD rt"
shows "π_3 (the (the (addpreRT rt ip' npre) ip)) = π_3 (the (rt ip))"
using assms [THEN kD_Some] unfolding addpreRT_def by clarsimp

lemma proj5_addpreRT [simp]:
"∀ rt dip ip npre. dip ∈ kD (rt) ⟹ π_5 (the (the (addpreRT rt dip npre) ip)) = π_5 (the (rt ip))"
unfolding addpreRT_def by auto

lemma flag_addpreRT [simp]:
fixes rt pre ip dip
assumes "dip ∈ kD rt"
shows "flag (the (addpreRT rt dip pre)) ip = flag rt ip"
unfolding addpreRT_def
using assms [THEN kD_Some] by (clarsimp)

lemma kD_addpreRT [simp]:
fixes rt dip npre
assumes "dip ∈ kD rt"
shows "kD (the (addpreRT rt dip npre)) = kD rt"
unfolding kD_def addpreRT_def
using assms [THEN kD_Some] by clarsimp blast

lemma vD_addpreRT [simp]:
fixes rt dip npre
assumes "dip ∈ kD rt"
shows "vD (the (addpreRT rt dip npre)) = vD rt"
unfolding vD_def addpreRT_def
using assms [THEN kD_Some] by clarsimp auto

lemma iD_addpreRT [simp]:
fixes rt dip npre
assumes "dip ∈ kD rt"
shows "iD (the (addpreRT rt dip npre)) = iD rt"
unfolding iD_def addpreRT_def
using assms [THEN kD_Some] by clarsimp auto

lemma nhop_addpreRT [simp]:
fixes rt pre ip dip
assumes "dip ∈ kD rt"
shows "nhop (the (addpreRT rt dip pre)) ip = nhop rt ip"
unfolding sqn_def addpreRT_def
using assms [THEN kD_Some] by (clarsimp)

lemma sqn_addpreRT [simp]:
fixes rt pre ip dip
assumes "dip ∈ kD rt"
shows "sqn (the (addpreRT rt dip pre)) ip = sqn rt ip"
unfolding sqn_def addpreRT_def
using assms [THEN kD_Some] by (clarsimp)

lemma dhops_addpreRT [simp]:
fixes rt pre ip dip
assumes "dip ∈ kD rt"
shows "dhops (the (addpreRT rt dip pre)) ip = dhops rt ip"
unfolding addpreRT_def
using assms [THEN kD_Some] by (clarsimp)

lemma sqnf_addpreRT [simp]:
"\( \forall \text{ip dip. } \text{ip} \in \text{kd(\text{rt} \, \xi)} \rightarrow \text{sqnf (the (addpreRT (\text{rt} \, \xi) \, \text{ip} \, \text{npre})) dip} = \text{sqnf (\text{rt} \, \xi) dip} \)"
unfolding sqnf_def addpreRT_def by auto

Updating route entries

lemma in_kD_case [simp]:
fixes dip rt
assumes "dip \in \text{kD(rt)}"
shows "(case rt dip of None \Rightarrow en | Some r \Rightarrow es r) = es (the (rt dip))"
using assms [THEN kD_Some] by auto

lemma not_in_kD_case [simp]:
fixes dip rt
assumes "dip /\in \text{kD(rt)}"
shows "(case rt dip of None \Rightarrow en | Some r \Rightarrow es r) = en"
using assms [THEN kD_None] by auto

lemma rt_Some_sqn [dest]:
fixes rt and ip dsn dsk flag hops nhip pre
assumes "rt ip = Some (dsn, dsk, flag, hops, nhip, pre)"
shows "sqn rt ip = dsn"
unfolding sqn_def using assms by simp

lemma not_kD_sqn [simp]:
fixes dip rt
assumes "dip /\in \text{kD(rt)}"
shows "sqn rt dip = 0"
using assms unfolding sqn_def by simp

definition update_arg_wf :: "r \Rightarrow \text{bool}"
where "update_arg_wf r \equiv \pi_4(r) = \text{val} \land 
(\pi_2(r) = 0) = (\pi_3(r) = \text{unk}) \land 
(\pi_3(r) = \text{unk} \rightarrow \pi_5(r) = 1)"

lemma update_arg_wf_gives_cases:
"\( \forall r. \text{update_arg_wf r} \Rightarrow (\pi_2(r) = 0) = (\pi_3(r) = \text{unk}) \)"
unfolding update_arg_wf_def by simp

lemma update_arg_wf_tuples [simp]:
"\( \forall nhip \, \text{pre. } \text{update_arg_wf} \, (0, \text{unk}, \text{val}, \text{Suc 0}, \text{nhip}, \text{pre}) \)"
"\( \forall \text{n hops nhip pre. } \text{update_arg_wf} \, (\text{Suc n}, \text{kno}, \text{val}, \text{hops}, \text{nhip}, \text{pre}) \)"
unfolding update_arg_wf_def by auto

lemma update_arg_wf_tuples' [elim]:
"\( \forall \text{n hops nhip pre. } \text{Suc 0} \leq \text{n} \rightarrow \text{update_arg_wf} \, (\text{n}, \text{kno}, \text{val}, \text{hops}, \text{nhip}, \text{pre}) \)"
unfolding update_arg_wf_def by auto

lemma wf_r_cases [intro]:
fixes P r
assumes "update_arg_wf r" and cl: "\( \forall \text{nhip pre. } \text{P} \, (0, \text{unk}, \text{val}, \text{Suc 0}, \text{nhip}, \text{pre}) \)"
and c2: "\( \forall \text{dsn hops nhip pre. } \text{dsn} > 0 \rightarrow \text{P} \, (\text{dsn}, \text{kno}, \text{val}, \text{hops}, \text{nhip}, \text{pre}) \)"
shows "P r"
proof -
obtain dsn dsk flag hops nhip pre where *: "r = (dsn, dsk, flag, hops, nhip, pre)" by (cases r)
with update_arg_wf r have wf1: "flag = val"
and wf2: "(dsn = 0) = (dsk = unk)"
and wf3: "dsk = unk \rightarrow (hops = 1)"
unfolding update_arg_wf_def by auto
have "P (dsn, dsk, flag, hops, nhip, pre)"

proof (cases dsk)
  assume "dsk = unk"
  moreover with wf2 wf3 have "dsn = 0" and "hops = Suc 0" by auto
  ultimately show ?thesis using ⟨flag = val⟩ by simp (rule c1)
next
  assume "dsk = kno"
  moreover with wf2 have "dsn > 0" by simp
  ultimately show ?thesis using ⟨flag = val⟩ by simp (rule c2)
qed

with * show "P r" by simp
qed

definition update :: "rt ⇒ ip ⇒ r ⇒ rt"
where
"update rt ip r ≡
case σ Route(rt, ip) of
  None ⇒ rt (ip ↦→ r)
| Some s ⇒
  if π₂(s) < π₂(r) then rt (ip ↦→ addpre r (π₇(s)))
  else if π₂(s) = π₂(r) ∧ (π₅(s) > π₅(r) ∨ π₁(s) = inv)
  then rt (ip ↦→ addpre r (π₇(s)))
  else if π₃(r) = unk
  then rt (ip ↦→ (π₂(s), snd (addpre r (π₇(s)))))
  else if π₃(s) = unk
  then rt (ip ↦→ addpre s (π₇(r)))

lemma update_simps [simp]:
fixes r s nr nr' ns rt ip
defines "s ≡ the σ Route(rt, ip)"
and "nr ≡ addpre r (π₇(s))"
and "nr' ≡ (π₃(s), π₄(nr), π₅(nr), π₆(nr), π₇(nr))"
and "ns ≡ addpre s (π₇(r))"
shows
"[ip /∈ kD(rt)]" ⟹ update rt ip r = rt (ip ↦→ r)"
"[ip ∈ kD(rt); sqn rt ip < π₂(r)]" ⟹ update rt ip r = rt (ip ↦→ nr)"
"[ip ∈ kD(rt); sqn rt ip = π₂(r);
  the (dhops rt ip) > π₅(r)]" ⟹ update rt ip r = rt (ip ↦→ nr)"
"[ip ∈ kD(rt); sqn rt ip = π₂(r);
  flag rt ip = Some inv]
  ⟹ update rt ip r = rt (ip ↦→ nr)"
"[ip ∈ kD(rt); π₃(r) = unk; (π₂(r) = 0) = (π₃(r) = unk)]" ⟹ update rt ip r = rt (ip ↦→ nr')"
"[ip ∈ kD(rt); sqn rt ip ≥ π₂(r); π₃(r) = kno;
  sqn rt ip = π₂(r) ⟹ the (dhops rt ip) ≤ π₅(r) ∧ the (flag rt ip) = val ]
  ⟹ update rt ip r = rt (ip ↦→ ns)"
proof -
  assume "ip /∈ kD(rt)"
  hence "σ Route(rt, ip) = None" ..
  thus "update rt ip r = rt (ip ↦→ r)"
unfolding update_def by simp
next
  assume "ip ∈ kD(rt)"
  and "sqn rt ip < π₂(r)"
from this(1) obtain dsn dsk fl hops nhip pre
  where "rt ip = Some (dsn, dsk, fl, hops, nhip, pre)"
    by (metis kD_Some)
with ⟨sqn rt ip < π₂(r)⟩ show "update rt ip r = rt (ip ↦→ nr)"
unfolding update_def nr_def s_def by auto
next
  assume "ip ∈ kD(rt)"
  and "sqn rt ip = π₂(r)"
  and "the (dhops rt ip) > π₅(r)"
from this(1) obtain dsn dsk fl hops nhip pre
  where "rt ip = Some (dsn, dsk, fl, hops, nhip, pre)"
    by (metis kD_Some)
with ⟨sqn rt ip = π₂(r), ⟩ (the (dhops rt ip) > π₅(r))
show "update rt ip r = rt (ip ↦ nr)"
    unfolding update_def nr_def s_def by auto
next
assume "ip ∈ kD(rt)"
    and "sqn rt ip = π₂(r)"
    and "flag rt ip = Some inv"
from this(1) obtain dsn dsk fl hops nhip pre
    where "rt ip = Some (dsn, dsk, fl, hops, nhip, pre)"
    by (metis kD_Some)
with (sqn rt ip = π₂(r)) and (flag rt ip = Some inv)
show "update rt ip r = rt (ip ↦ nr)"
    unfolding update_def nr_def s_def by auto
next
assume "ip ∈ kD(rt)"
    and "(π₂(r) = 0) = (π₃(r) = unk)"
from this(1) obtain dsn dsk fl hops nhip pre
    where "rt ip = Some (dsn, dsk, fl, hops, nhip, pre)"
    by (metis kD_Some)
with ((π₂(r) = 0) = (π₃(r) = unk)) and (π₃(r) = unk)
show "update rt ip r = rt (ip ↦ nr')"
    unfolding update_def nr'_def nr_def s_def
by (cases r) simp
next
assume "ip ∈ kD(rt)"
    and "π₃(r) = kno"
    and "(π₂(r) = 0) = (π₃(r) = unk)"
from this(1) obtain dsn dsk fl hops nhip pre
    where "rt ip = Some (dsn, dsk, fl, hops, nhip, pre)"
    by (metis kD_Some)
with otherassms show "update rt ip r = rt (ip ↦ ns)"
    unfolding update_def ns_def s_def by auto
qed

lemma update_cases [elim]:
assumes "((π₂(r) = 0) = (π₃(r) = unk))"
    and c1: "[[ip /∈ kD(rt)] ▹ P (rt (ip ↦ r))]
    and c2: "[[ip ∈ kD(rt); sqn rt ip < π₂(r)] ▹ P (rt (ip ↦ addpre r (π₇(the σroute(rt, ip))))]
    and c3: "[[ip ∈ kD(rt); sqn rt ip = π₂(r); the (dhops rt ip) > π₅(r)] ▹ P (rt (ip ↦ addpre r (π₇(the σroute(rt, ip))))]
    and c4: "[[ip ∈ kD(rt); sqn rt ip = π₂(r); the (flag rt ip) = inv] ▹ P (rt (ip ↦ addpre r (π₇(the σroute(rt, ip))))]
    and c5: "[[ip ∈ kD(rt); π₃(r) = unk] ▹ P (rt (ip ↦ (π₃(the σroute(rt, ip)), π₃(r), π₄(r), π₅(r), π₆(r), π₇(addpre r (π₇(the σroute(rt, ip)))))))]
    and c6: "[[ip ∈ kD(rt); sqn rt ip ≥ π₂(r); π₃(r) = kno; sqn rt ip = π₃(r) ▹ the (dhops rt ip) ≤ π₅(r) ∧ the (flag rt ip) = val] ▹ P (rt (ip ↦ addpre (the σroute(rt, ip)) (π₇(r))))"
shows "(P (update rt ip r))"
proof (cases "ip /∈ kD(rt)")
assume "ip /∈ kD(rt)"
with c1 show ?thesis
by simp
next
assume "ip ∈ kD(rt)"
moreover then obtain dsn dsk fl hops nhip pre
    where rteq: "rt ip = Some (dsn, dsk, fl, hops, nhip, pre)"
    by (metis kD_Some)
moreover obtain dsn' dsk' fl' hops' nhip' pre'
    where req: "r = (dsn', dsk', fl', hops', nhip', pre')" by (cases r) metis
ultimately show \( \text{thesis} \)

using \( (\pi_2(r) = 0) = (\pi_3(r) = \text{unk}) \)

\[
c2 \ [\text{OF} \ ip \in kD(rt)]
c3 \ [\text{OF} \ ip \in kD(rt)]
c4 \ [\text{OF} \ ip \in kD(rt)]
c5 \ [\text{OF} \ ip \in kD(rt)]
c6 \ [\text{OF} \ ip \in kD(rt)]
\]

unfolding update_def sqn_def by auto

due

lemma update_cases_kD:
assumes \( (\pi_2(r) = 0) = (\pi_3(r) = \text{unk}) \)
and \( ip \in kD(rt) \)
and \( c2: \sqn rt ip < \pi_2(r) \Rightarrow P (rt (ip \mapsto \text{addpre} r (\pi_7(\sigma_{\text{route}(rt, ip))]))) \)
and \( c3: [\sqn rt ip = \pi_2(r); \text{the (dhops rt ip)} > \pi_5(r)] \Rightarrow P (rt (ip \mapsto \text{addpre} r (\pi_7(\sigma_{\text{route}(rt, ip))))) \)
and \( c4: [\sqn rt ip = \pi_2(r); \text{the (flag rt ip)} = \text{inv}] \Rightarrow P (rt (ip \mapsto \text{addpre} r (\pi_7(\sigma_{\text{route}(rt, ip))))) \)
and \( c5: \pi_3(r) = \text{unk} \Rightarrow P (rt (ip \mapsto (\pi_2(\text{the } \sigma_{\text{route}(rt, ip)}), \pi_3(r), \pi_4(r), \pi_5(r), \pi_6(r), \pi_7(\text{addpre} r (\pi_7(\sigma_{\text{route}(rt, ip)))))))) \)
and \( c6: [\sqn rt ip \geq \pi_2(r); \pi_3(r) = \text{kno}; \sqn rt ip = \pi_2(r) \Rightarrow \text{the (dhops rt ip)} \leq \pi_5(r) \land \text{the (flag rt ip)} = \text{val}] \Rightarrow P (rt (ip \mapsto \text{addpre} (\text{the } \sigma_{\text{route}(rt, ip)} (\pi_7(r))))) \)

shows \( (P (\text{update rt ip r})) \)

using assms(1) proof (rule update_cases)
assume \( \sqn rt ip < \pi_2(r) \)
thus \( P (rt(ip \mapsto \text{addpre} r (\pi_7(\text{the (rt ip))}))) \)
by (rule c2)

next

assume \( \sqn rt ip = \pi_2(r) \)
and \( \text{the (dhops rt ip)} > \pi_5(r) \)
thus \( P (rt(ip \mapsto \text{addpre} r (\pi_7(\text{the (rt ip)}))) \)
by (rule c3)

next

assume \( \sqn rt ip = \pi_2(r) \)
and \( \text{the (flag rt ip)} = \text{inv} \)
thus \( P (rt(ip \mapsto \text{addpre} r (\pi_7(\text{the (rt ip)}))) \)
by (rule c4)

next

assume \( \pi_3(r) = \text{unk} \)
thus \( P (rt (ip \mapsto (\pi_2(\text{the } \sigma_{\text{route}(rt, ip)}), \pi_3(r), \pi_4(r), \pi_5(r), \pi_6(r), \pi_7(\text{addpre} r (\pi_7(\text{the (rt ip)}))))) \)
by (rule c5)

next

assume \( \sqn rt ip \geq \pi_2(r) \)
and \( \pi_3(r) = \text{kno} \)
and \( \sqn rt ip = \pi_2(r) \Rightarrow \text{the (dhops rt ip)} \leq \pi_5(r) \land \text{the (flag rt ip)} = \text{val} \)
thus \( P (rt (ip \mapsto \text{addpre} (\text{the (rt ip)})) (\pi_7(r))) \)
by (rule c6)

due (simp add: \( \langle ip \in \text{kD(rt)} \rangle \))

lemma in_kD_after_update [simp]:
fixes rt nip dsn dsk flag hops nhip pre
shows \( \text{kD (update rt nip (dsn, dsk, flag, hops, nhip, pre)) = insert nip (kD rt)} \)
unfolding update_def
by (cases "rt nip")

lemma nhop_of_update [simp]:
fixes rt dip dsn dsk flag hops nhip
assumes \( rt \neq \text{update rt dip (dsn, dsk, flag, hops, nhip, \{\})} \)
shows \( \text{the (nhop (update rt dip (dsn, dsk, flag, hops, nhip, \{\})) dip) = nhip} \)
proof -
from assms
have update_neq: "\( \forall v. \text{rt dip = Some v} \)"
update rt dip (dsn, dsk, flag, hops, nhip, {}) 
≠ rt(dip ⇀ addpre (the (rt dip)) (π7 (dsn, dsk, flag, hops, nhip, {})))

by auto
show ?thesis
proof (cases "rt dip = None")
  assume "rt dip = None"
  thus "?thesis" unfolding update_def by clarsimp
next
  assume "rt dip ≠ None"
  then obtain v where "rt dip = Some v" by (metis not_None_eq)
  with update_neq [OF this] show ?thesis
    unfolding update_def by auto
qed

lemma sqn_if_updated:
  fixes rip v rt ip
  shows "sqn (λx. if x = rip then Some v else rt x) ip
       = (if ip = rip then π2(v) else sqn rt ip)"
  unfolding sqn_def by simp

lemma update_sqn [simp]:
  fixes rt dip rip dsn dsk hops nhip pre
  assumes "(dsn = 0) = (dsk = unk)"
  shows "sqn rt dip ≤ sqn (update rt rip (dsn, dsk, val, hops, nhip, pre)) dip"
  proof (rule update_cases)
    show "(π2 (dsn, dsk, val, hops, nhip, pre) = 0) = (π3 (dsn, dsk, val, hops, nhip, pre) = unk)"
      by simp (rule assms)
  qed (clarsimp simp: sqn_if_updated sqn_def)+

lemma sqn_update_bigger [simp]:
  fixes rt ip ip' dsn dsk flag hops nhip pre
  assumes "1 ≤ hops"
  shows "sqn rt ip ≤ sqn (update rt ip' (dsn, dsk, flag, hops, nhip, pre)) ip"
  using assms unfolding update_def sqn_def
  by (clarsimp split: option.split) auto

lemma dhops_update [intro]:
  fixes rt dsn dsk flag hops ip rip nhip pre
  assumes ex: "∀ip∈kD rt. the (dhops rt ip) ≥ 1"
          and ip: "((ip = rip ∧ Suc 0 ≤ hops) ∨ (ip ≠ rip ∧ ip∈kD rt))"
  shows "Suc 0 ≤ the (dhops (update rt rip (dsn, dsk, flag, hops, nhip, pre)) ip)"
  using ip proof
    assume "ip = rip ∧ Suc 0 ≤ hops" thus ?thesis
    unfolding update_def using ex
    by (cases "rip ∈ kD rt") (drule(1) bspec, auto)
next
  assume "ip ≠ rip ∧ ip∈kD rt" thus ?thesis
  using ex unfolding update_def
  by (cases "rip∈kD rt") auto
qed

lemma update_another [simp]:
  fixes dip ip rt dsn dsk flag hops nhip pre
  assumes "ip ≠ dip"
  shows "((update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = rt ip"
  using assms unfolding update_def
  by (clarsimp split: option.split)

lemma nhop_update_another [simp]:
  fixes dip ip rt dsn dsk flag hops nhip pre
  assumes "ip ≠ dip"
  shows "nhop (update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = nhop rt ip"
  using assms unfolding update_def
by (clarsimp split: option.split)

lemma dhops_update_another [simp]:
  fixes dip ip rt dan dsk flag hops nhip pre
  assumes "ip ≠ dip"
  shows "dhops (update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = dhops rt ip"
  using assms unfolding update_def
  by (clarsimp split: option.split)

lemma sqn_update_same [simp]:
  "∀ rt ip dsn dsk flag hops nhip pre. sqn (rt(ip ↦→ v)) ip = π₂(v)"
  unfolding sqn_def by simp

lemma dhops_update_changed [simp]:
  fixes rt dip osn hops nhip
  assumes "rt ≠ update rt dip (osn, kno, val, hops, nhip, {})"
  shows "the (dhops (update rt dip (osn, kno, val, hops, nhip, {})) dip) = hops"
  using assms unfolding update_def
  by (clarsimp split: option.split_asm option.split split_if_asm) auto

lemma nhop_update_unk_val [simp]:
  "∀ rt dip ip dsn hops npre.
  the (nhop (update rt dip (dsn, unk, val, hops, ip, npre)) dip) = ip"
  unfolding update_def by (clarsimp split: option.split)

lemma nhop_update_changed [simp]:
  fixes rt dip dsn dsk flg hops sip
  assumes "update rt dip (dsn, dsk, flg, hops, sip, {}) ≠ rt"
  shows "the (nhop (update rt dip (dsn, dsk, flg, hops, sip, {})) dip) = sip"
  using assms unfolding update_def
  by (clarsimp split: option.splits split_if_asm) auto

lemma update_rt_split_asm:
  "∀ rt ip dsn dsk flag hops sip.
  P (update rt ip (dsn, dsk, flag, hops, sip, {})) =
  (∼(rt = update rt ip (dsn, dsk, flag, hops, sip, {})) ∧ ∼P rt
  ∨ rt ≠ update rt ip (dsn, dsk, flag, hops, sip, {})
  ∧ ∼P (update rt ip (dsn, dsk, flag, hops, sip, {}))))"
  by auto

lemma sqn_update [simp]: "∀ rt dip dsn flg hops sip.
  rt ≠ update rt dip (dsn, kno, flg, hops, sip, {})
  ⇒ sqn (update rt dip (dsn, kno, flg, hops, sip, {})) dip = dsn"
  unfolding update_def by (clarsimp split: option.split split_if_asm) auto

lemma sqnf_update [simp]: "∀ rt dip dsn flg hops sip.
  rt ≠ update rt dip (dsn, dsk, flg, hops, sip, {})
  ⇒ sqnf (update rt dip (dsn, dsk, flg, hops, sip, {})) dip = dsk"
  unfolding update_def sqnf_def
  by (clarsimp split: option.splits split_if_asm) auto

lemma update_kno_dsn_greater_zero:
  "∀ rt dip ip dsn hops npre. 1 ≤ dsn ⇒ 1 ≤ (sqn (update rt dip (dsn, kno, val, hops, ip, npre)) dip)"
  unfolding update_def
  by (clarsimp split: option.splits)

lemma proj3_update [simp]: "∀ rt dip dsn flg hops sip.
  rt ≠ update rt dip (dsn, dsk, flg, hops, sip, {})
  ⇒ π₃(the (update rt dip (dsn, dsk, flg, hops, sip, {})) dip) = dsk"
  unfolding update_def sqnf_def
  by (clarsimp split: option.splits split_if_asm) auto

lemma nhop_update_changed_kno_val [simp]: "∀ rt ip dsn dsk hops nhip.
  inequalities and propositions about routing updates and their implications on network status.
rt ≠ update rt ip (dsn, kno, val, hops, nhip, {}) 
⇒ the (nhop (update rt ip (dsn, kno, val, hops, nhip, {})) ip) = nhip" 
using assms unfolding update_def 
by (clarsimp split: option.split_asm option.split split_if_asm) auto

lemma flag_update [simp]: "∀ rt dip dsn flg hops sip. 
rt ≠ update rt dip (dsn, kno, flg, hops, sip, {}) 
⇒ the (flag (update rt dip (dsn, kno, flg, hops, sip, {})) dip) = flg" 
unfolding update_def 
by (clarsimp split: option.split split_if_asm) auto

lemma the_flag_Some [dest!]: 
fixes ip rt 
assumes "the (flag rt ip) = x" 
and "ip ∈ kD rt" 
shows "flag rt ip = Some x" 
using assms by auto

lemma kD_update_unchanged [dest]: 
fixes rt dip dsn dsk flag hops nhip pre 
assumes "rt = update rt dip (dsn, dsk, flag, hops, nhip, pre)" 
shows "dip ∈ kD(rt)" 
proof - 
have "dip ∈ kD(update rt dip (dsn, dsk, flag, hops, nhip, pre))" by simp 
with assms show ?thesis by simp 
qed

lemma nhop_update [simp]: "∀ rt dip dsn dsk flag hops sip. 
rt ≠ update rt dip (dsn, dsk, flag, hops, sip, {}) 
⇒ the (nhop (update rt dip (dsn, dsk, flag, hops, sip, {})) dip) = sip" 
unfolding update_def sqnf_def 
by (clarsimp split: option.split split_if_asm) auto

lemma sqn_update_another [simp]: 
fixes dip ip rt dsn dsk flag hops nhip pre 
assumes "ip ≠ dip" 
shows "sqn (update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = sqn rt ip" 
using assms unfolding update_def sqn_def 
by (clarsimp split: option.splits) auto

lemma sqnf_update_another [simp]: 
fixes dip ip rt dsn dsk flag hops nhip pre 
assumes "ip ≠ dip" 
shows "sqnf (update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = sqnf rt ip" 
using assms unfolding update_def sqnf_def 
by (clarsimp split: option.splits) auto

lemma vD_update_val [dest]: 
"∀ dip rt dip’ dsn dsk hops nhip pre. 
dip ∈ vD(update rt dip’ (dsn, dsk, val, hops, nhip, pre)) ⇒ (dip ∈ vD(rt) ∨ dip = dip’)" 
unfolding update_def vD_def by (clarsimp split: option.split_asm split_if_asm)

Invalidating route entries

definition invalidate :: "rt ⇒ (ip ⇀ sqn) ⇒ rt" 
where "invalidate rt dests ≡ λip. case (rt ip, dests ip) of 
(None, _) ⇒ None 
| (Some s, None) ⇒ Some s 
| (Some (_, dsk, _, hops, nhip, pre), Some rsn) ⇒ 
Some (rsn, dsk, inv, hops, nhip, pre)"

lemma proj3_invalidate [simp]: 
"∀ dip. π₃(the ((invalidate rt dests) dip)) = π₃(the (rt dip))"
unfolding invalidate_def by (clarsimp split: option.split)

lemma proj5_invalidate [simp]:
"\( dip. \pi_5(\text{the } (\text{invalidate } rt \text{ dests} \text{ dip})) = \pi_5(\text{the } (rt \text{ dip})) \)"
unfolding invalidate_def by (clarsimp split: option.split)

lemma proj6_invalidate [simp]:
"\( dip. \pi_6(\text{the } (\text{invalidate } rt \text{ dests} \text{ dip})) = \pi_6(\text{the } (rt \text{ dip})) \)"
unfolding invalidate_def by (clarsimp split: option.split)

lemma proj7_invalidate [simp]:
"\( dip. \pi_7(\text{the } (\text{invalidate } rt \text{ dests} \text{ dip})) = \pi_7(\text{the } (rt \text{ dip})) \)"
unfolding invalidate_def by (clarsimp split: option.split)

1.1.5 Route Requests

lemma invalidate_kD_inv [simp]:
"\( rt \text{ dests}. \ kD (\text{invalidate } rt \text{ dests}) = kD rt \)"
unfolding invalidate_def kD_def
by (simp split: option.split)

lemma invalidate_sqn:
fixes rt dip dests
assumes "\( \forall rsn. \text{ dests dip = Some rsn } \rightarrow \text{ sqn rt dip } \leq rsn \)"
shows "\( \text{ sqn rt dip } \leq \text{ sqn (invalidate rt dests) dip } \)"
proof
  (cases "\( dip \notin kD(rt) \)"
    assume "\( \neg dip \notin kD(rt) \)"
    hence "\( dip \in kD(rt) \)" by simp
    then obtain dsn dsk flag hops nhip pre where "\( rt dip = \text{ Some (dsn, dsk, flag, hops, nhip, pre)} \)"
      by (metis kD_Some)
    with assms show "\( \text{ sqn rt dip } \leq \text{ sqn (invalidate rt dests) dip } \)"
      by (cases "\( \text{ dests dip } \)" (auto simp add: invalidate_def sqn_def))
  qed simp

lemma sqn_invalidate_in_dests [simp]:
fixes dests ipa rsn rt
assumes "\( \text{ dests ipa = Some rsn } \)" and "\( \text{ ipa} \in kD(rt) \)"
shows "\( \text{ sqn (invalidate rt dests) ipa = rsn } \)"
unfolding invalidate_def sqn_def
using assms(1) assms(2) [THEN kD_Some]
by clarsimp

lemma dhops_invalidate [simp]:
"\( dip. \text{ the } (\text{dhops (invalidate rt dests) dip}) = \text{ the } (dhops rt dip) \)"
unfolding invalidate_def by (clarsimp split: option.split)

lemma sqnf_invalidate [simp]:
"\( dip. \text{ sqnf (invalidate (rt } \xi \text{) (dests } \xi \text{)) dip = sqnf (rt } \xi \text{) dip} \)"
unfolding sqnf_def invalidate_def by (clarsimp split: option.split)

lemma nhop_invalidate [simp]:
"\( dip. \text{ the } (\text{nhop (invalidate (rt } \xi \text{) (dests } \xi \text{)) dip}) = \text{ the } (nhop (rt } \xi \text{) dip} \)"
unfolding invalidate_def by (clarsimp split: option.split)

lemma invalidate_other [simp]:
fixes rt dests dip
assumes "\( dip \notin dom(dests) \)"
shows "\( \text{ invalidate rt dests dip = rt dip } \)"
using assms unfolding invalidate_def
by (clarsimp split: option.split_asm)

lemma invalidate_none [simp]:
fixes rt dests dip

assumes "dip \in kD(rt)"
shows "invalidate rt dests dip = None"
using assms unfolding invalidate_def by clarsimp

lemma vD_invalidate_vD_not_dests:
"\forall dip rt dests. dip \in vD(invalidate rt dests) \Rightarrow dip \in vD(rt) \land dests dip = None"
unfolding invalidate_def vD_def
by (clarsimp split: option.split_asm)

lemma sqn_invalidate_not_in_dests [simp]:
fixes dests dip rt
assumes "dip /\in dom(dests)"
shows "sqn (invalidate rt dests) dip = sqn rt dip"
using assms unfolding sqn_def by simp

lemma invalidate_changes:
fixes rt dests dip dsn dsk flag hops nhip pre
assumes "invalidate rt dests dip = Some (dsn, dsk, flag, hops, nhip, pre)"
shows "dsn = (case dests dip of None \Rightarrow \pi_2 (the (rt dip)) | Some rsn \Rightarrow rsn)
\land dsk = \pi_3 (the (rt dip))
\land flag = (if dests dip = None then \pi_4 (the (rt dip)) else inv)
\land hops = \pi_5 (the (rt dip))
\land nhip = \pi_6 (the (rt dip))
\land pre = \pi_7 (the (rt dip))"
using assms unfolding invalidate_def by (cases "rt dip", clarsimp, cases "dests dip") auto

lemma proj3_inv: "\forall dip rt dests. dip \in kD (rt) = \Rightarrow \pi_3 (the (invalidate rt dests dip)) = \pi_3 (the (rt dip))"
by (clarsimp simp: invalidate_def kD_def split: option.split)

lemma dests_iD_invalidate [simp]:
assumes "dests ip = Some rsn"
and "ip \in kD(rt)"
shows "ip \in iD(invalidate rt dests)"
using assms(1) assms(2) [THEN kD_Some] unfolding invalidate_def kD_def split: option.split
by (clarsimp split: option.split)

1.1.6 Queued Packets

Functions for sending data packets.

type_synonym store = "ip \Rightarrow (p \times data list)"

definition sigma_queue :: "store \Rightarrow ip \Rightarrow data list" ("\sigma_queue(_\_, _\_)")
where "\sigma_queue(store, dip) \equiv case store dip of None \Rightarrow [] | Some (p, q) \Rightarrow q"

definition qD :: "store \Rightarrow ip set"
where "qD \equiv dom"

definition add :: "data \Rightarrow ip \Rightarrow store \Rightarrow store"
where "add d dip store \equiv case store dip of
    None \Rightarrow store (dip \mapsto (req, [d]))
    | Some (p, q) \Rightarrow store (dip \mapsto (p, q @ [d]))"

lemma qD_add [simp]:
fixes dip store
shows "qD(add d dip store) = insert dip (qD store)"
unfolding add_def Let_def qD_def
by (clarsimp split: option.split)

definition drop :: "ip \Rightarrow store \Rightarrow store"
where "drop dip store \equiv
    map_option (\lambda(p, q). if tl q = [] then store (dip := None)"
definition sigma_p_flag :: 
  "store ⇒ ip → p" ("σ_p-flag'(_, _, _)")
where "σ_p-flag(store, dip) = map_option fst (store dip)"

definition unsetRRF :: "store ⇒ ip ⇒ store"
where "unsetRRF store dip = case store dip of
  None ⇒ store
  | Some (p, q) ⇒ store (dip ↦→ (noreq, q))"

definition setRRF :: "store ⇒ (ip ↦ sqn) ⇒ store"
where "setRRF store dests = λ dip. if dests dip = None then store dip
  else map_option (λ(_, q). (req, q)) (store dip)"

1.1.7 Comparison with the original technical report

The major differences with the AODV technical report of Fehnker et al are:

1. nhop is partial, thus a ‘the’ is needed, similarly for dhops and addpreRT.
2. precs is partial.
3. σ_p-flag(store, dip) is partial.
4. The routing table (rt) is modelled as a map (ip ⇒ r option) rather than a set of 7-tuples, likewise, the r
  is a 6-tuple rather than a 7-tuple, i.e., the destination ip-address (dip) is taken from the argument to the
  function, rather than a part of the result. Well-definedness then follows from the structure of the type and
  more related facts are available automatically, rather than having to be acquired through tedious proofs.
5. Similar remarks hold for the dests mapping passed to invalidate, and store.

end

1.2 AODV protocol messages

theory A_Aodv_Message
imports A_Norreqid
begin

datatype msg =
  Rreq nat ip sqn k ip sqn ip
  | Rrep nat ip sqn ip ip
  | Rerr "ip ↦ sqn" ip
  | Newpkt data ip
  | Pkt data ip ip

instantiation msg :: msg
begin

definition rreq_def [simp]: "rreq ≡ λ(hops, dip, dsn, dsk, oip, osn, sip).
  Rreq hops dip dsn dsk oip osn sip"

lemma rreq_simp [simp]:
  "rreq(hops, dip, dsn, dsk, oip, osn, sip) = Rreq hops dip dsn dsk oip osn sip"

end
unfolding rreq_def by simp

definition rrep :: "nat × ip × sqn × ip × ip ⇒ msg"
  where "rrep ≡ λ(hops, dip, dsn, oip, sip). Rrep hops dip dsn oip sip"

lemma rrep_simps [simp]:
  "rrep(hops, dip, dsn, oip, sip) = Rrep hops dip dsn oip sip"
unfolding rrep_def by simp

definition rerr :: "ip ⇒ sqn ⇒ msg"
  where "rerr ≡ λ(dests, sip). Rerr dests sip"

lemma rerr_simps [simp]:
  "rerr(dests, sip) = Rerr dests sip"
unfolding rerr_def by simp

lemma not_eq_newpkt_rreq [simp]: "¬eq_newpkt (Rreq hops dip dsn dsk oip osn sip)"
unfolding eq_newpkt_def by simp

lemma not_eq_newpkt_rrep [simp]: "¬eq_newpkt (Rrep hops dip dsn oip sip)"
unfolding eq_newpkt_def by simp

lemma not_eq_newpkt_rerr [simp]: "¬eq_newpkt (Rerr dests sip)"
unfolding eq_newpkt_def by simp

lemma not_eq_newpkt_pkt [simp]: "¬eq_newpkt (Pkt d dip sip)"
unfolding eq_newpkt_def by simp

definition pkt :: "data × ip × ip ⇒ msg"
  where "pkt ≡ λ(d, dip, sip). Pkt d dip sip"

lemma pkt_simps [simp]:
  "pkt(d, dip, sip) = Pkt d dip sip"
unfolding pkt_def by simp

end

1.3 The AODV protocol

theory A_Aodv
imports A_Aodv_Data A_Aodv_Message
  "../../../AWN/AWN_SOS_Labels" "../../../AWN/AWN_Invariants"
begin

1.3.1 Data state

record state =
  ip :: "ip"
  sn :: "sqn"
  rt :: "rt"
  rreqs :: "(ip × sqn) set"
  store :: "store"

  msg :: "msg"
  data :: "data"
  dests :: "ip ⇒ sqn"
  pre :: "ip set"
  dip :: "ip"
  oip :: "ip"
  hops :: "nat"
  dsn :: "sqn"
  dsk :: "k"
  osn :: "sqn"
  sip :: "ip"

104
abbreviation aodv_init :: "ip ⇒ state"
where "aodv_init i ≡ |
ip = i,
    sn = 1,
    rt = empty,
    rreqs = {},
    store = empty,
    msg = (SOME x. True),
    data = (SOME x. True),
    dests = (SOME x. True),
    pre = (SOME x. True),
    oip = (SOME x. True),
    hops = (SOME x. True),
    dip = (SOME x. True),
    dsn = (SOME x. True),
    dsk = (SOME x. True),
    osn = (SOME x. True),
    sip = (SOME x. x ≠ i)
|"

lemma some_neq_not_eq [simp]: "¬((SOME x :: nat. x ≠ i) = i)"
by (subst some_eq_ex) (metis zero_neq_numeral)

definition clear_locals :: "state ⇒ state"
where "clear_locals ξ = ξ |
    msg := (SOME x. True),
    data := (SOME x. True),
    dests := (SOME x. True),
    pre := (SOME x. True),
    oip := (SOME x. True),
    hops := (SOME x. True),
    dip := (SOME x. True),
    dsn := (SOME x. True),
    dsk := (SOME x. True),
    osn := (SOME x. True),
    sip := (SOME x. x ≠ ip ξ)
|"

lemma clear_locals_sip_not_ip [simp]: "¬(sip (clear_locals ξ) = ip ξ)"
unfolding clear_locals_def by simp

lemma clear_locals_but_not_globals [simp]:
"ip (clear_locals ξ) = ip ξ"
"sn (clear_locals ξ) = sn ξ"
"rt (clear_locals ξ) = rt ξ"
"rreqs (clear_locals ξ) = rreqs ξ"
"store (clear_locals ξ) = store ξ"
unfolding clear_locals_def by auto

1.3.2 Auxilliary message handling definitions

definition is_newpkt
where "is_newpkt ξ ≡ case msg ξ of
    Newpkt data' dip' ⇒ { ξ[data := data', dip := dip']|
   _| ⇒ {}}"

definition is_pkt
where "is_pkt ξ ≡ case msg ξ of
    Pkt data' dip' oip' ⇒ { ξ[data := data', dip := dip', oip := oip']|
   _| ⇒ {}}"

definition is_rreq
where "is_rreq \xi \equiv case \text{msg} \xi of \begin{align*}
& \text{Rreq hops' dip' dsn' dsk' oip' osn' sip'} \\
& \{ \xi(\mid hops := \text{hops'}, dip := \text{dip'}, dsn := \text{dsn'}, \\
& \quad dsk := \text{dsk'}, oip := \text{oip'}, osn := \text{osn'}, sip := \text{sip'}) \} \\
& | _ \Rightarrow {} \end{align*}"

lemma is_rreq_asm [dest!]:
\begin{itemize}
\item assumes "\xi' \in is_rreq \xi" \\
\item shows "(\exists \text{hops'} \text{rreqid'} \text{dip'} \text{dsn'} \text{dsk'} \text{oip'} \text{osn'} \text{sip'}.
\text{msg} \xi = \text{Rreq hops'} \text{dip'} \text{dsn'} \text{dsk'} \text{oip'} \text{osn'} \text{sip'} \\
\quad \xi' = \xi(\mid \text{hops := hops'}, \text{dip := dip'}, \text{dsn := dsn'}, \\
\quad \text{dsk := dsk'}, \text{oip := oip'}, \text{osn := osn'}, \text{sip := sip'}))"
\end{itemize}
using assms unfolding is_rreq_def
by (cases "\text{msg} \xi") simp_all

definition is_rrep
where "is_rrep \xi \equiv case \text{msg} \xi of \begin{align*}
& \text{Rrep hops' dip' dsn' oip' sip'} \\
& \{ \xi(\mid \text{hops := hops'}, \text{dip := dip'}, \text{dsn := dsn'}, \text{dsk := dsk'}, \\
& \quad \text{oip := oip'}, \text{osn := osn'}, \text{sip := sip'}) \} \\
& | _ \Rightarrow {} \end{align*}"

lemma is_rrep_asm [dest!]:
\begin{itemize}
\item assumes "\xi' \in is_rrep \xi" \\
\item shows "(\exists \text{hops'} \text{dip'} \text{dsn' oip' sip'}. \\
\text{msg} \xi = \text{Rrep hops'} \text{dip'} \text{dsn'} \text{oip'} \text{sip'} \\
\quad \xi' = \xi(\mid \text{hops := hops'}, \text{dip := dip'}, \text{dsn := dsn'}, \text{dsk := dsk'}, \\
\quad \text{oip := oip'}, \text{osn := osn'}, \text{sip := sip'}))"
\end{itemize}
using assms unfolding is_rrep_def
by (cases "\text{msg} \xi") simp_all

definition is_rerr
where "is_rerr \xi \equiv case \text{msg} \xi of \begin{align*}
& \text{Rerr dests' sip'} \\
& \{ \xi(\mid \text{dests := dests'}, \text{sip := sip'}) \} \\
& | _ \Rightarrow {} \end{align*}"

lemma is_rerr_asm [dest!]:
\begin{itemize}
\item assumes "\xi' \in is_rerr \xi" \\
\item shows "(\exists \text{dests'} \text{sip'}. \\
\text{msg} \xi = \text{Rerr dests'} \text{sip'} \\
\quad \xi' = \xi(\mid \text{dests := dests'}, \text{sip := sip'}))"
\end{itemize}
using assms unfolding is_rerr_def
by (cases "\text{msg} \xi") simp_all

lemmas is_msg_defs =
is_rerr_def is_rrep_def is_rreq_def is_pkt_def is_newpkt_def

lemma is_msg_inv_ip [simp]:
\begin{itemize}
\item "\xi' \in is_rerr \xi \quad \Rightarrow \text{ip} \xi' = \text{ip} \xi" \\
\item "\xi' \in is_rrep \xi \quad \Rightarrow \text{ip} \xi' = \text{ip} \xi" \\
\item "\xi' \in is_rreq \xi \quad \Rightarrow \text{ip} \xi' = \text{ip} \xi" \\
\item "\xi' \in is_pkt \xi \quad \Rightarrow \text{ip} \xi' = \text{ip} \xi" \\
\item "\xi' \in is_newpkt \xi \quad \Rightarrow \text{ip} \xi' = \text{ip} \xi"
\end{itemize}
unfolding is_msg_defs
by (cases "\text{msg} \xi", clarsimp)+

lemma is_msg_inv_sn [simp]:
\begin{itemize}
\item "\xi' \in is_rerr \xi \quad \Rightarrow \text{sn} \xi' = \text{sn} \xi" \\
\item "\xi' \in is_rrep \xi \quad \Rightarrow \text{sn} \xi' = \text{sn} \xi" \\
\item "\xi' \in is_rreq \xi \quad \Rightarrow \text{sn} \xi' = \text{sn} \xi" \\
\item "\xi' \in is_pkt \xi \quad \Rightarrow \text{sn} \xi' = \text{sn} \xi" \\
\item "\xi' \in is_newpkt \xi \quad \Rightarrow \text{sn} \xi' = \text{sn} \xi"
\end{itemize}
unfolding is_msg_defs
by (cases "\text{msg} \xi", clarsimp)+

lemma is_msg_inv_rt [simp]:
\[
\xi' \in \text{is}_rerr \xi \implies \text{rt} \, \xi' = \text{rt} \, \xi
\]
\[
\xi' \in \text{is}_rrep \xi \implies \text{rt} \, \xi' = \text{rt} \, \xi
\]
\[
\xi' \in \text{is}_rreq \xi \implies \text{rt} \, \xi' = \text{rt} \, \xi
\]
\[
\xi' \in \text{is}_pkt \xi \implies \text{rt} \, \xi' = \text{rt} \, \xi
\]
\[
\xi' \in \text{is}_newpkt \xi \implies \text{rt} \, \xi' = \text{rt} \, \xi
\]

unfolding is_msg_defs
by (cases "msg \xi", clarsimp+)+

lemma is_msg_inv_rreqs [simp]:
\[
\xi' \in \text{is}_rerr \xi \implies \text{rreqs} \, \xi' = \text{rreqs} \, \xi
\]
\[
\xi' \in \text{is}_rrep \xi \implies \text{rreqs} \, \xi' = \text{rreqs} \, \xi
\]
\[
\xi' \in \text{is}_rreq \xi \implies \text{rreqs} \, \xi' = \text{rreqs} \, \xi
\]
\[
\xi' \in \text{is}_pkt \xi \implies \text{rreqs} \, \xi' = \text{rreqs} \, \xi
\]
\[
\xi' \in \text{is}_newpkt \xi \implies \text{rreqs} \, \xi' = \text{rreqs} \, \xi
\]

unfolding is_msg_defs
by (cases "msg \xi", clarsimp+)+

lemma is_msg_inv_store [simp]:
\[
\xi' \in \text{is}_rerr \xi \implies \text{store} \, \xi' = \text{store} \, \xi
\]
\[
\xi' \in \text{is}_rrep \xi \implies \text{store} \, \xi' = \text{store} \, \xi
\]
\[
\xi' \in \text{is}_rreq \xi \implies \text{store} \, \xi' = \text{store} \, \xi
\]
\[
\xi' \in \text{is}_pkt \xi \implies \text{store} \, \xi' = \text{store} \, \xi
\]
\[
\xi' \in \text{is}_newpkt \xi \implies \text{store} \, \xi' = \text{store} \, \xi
\]

unfolding is_msg_defs
by (cases "msg \xi", clarsimp+)+

lemma is_msg_inv_sip [simp]:
\[
\xi' \in \text{is}_pkt \xi \implies \text{sip} \, \xi' = \text{sip} \, \xi
\]
\[
\xi' \in \text{is}_newpkt \xi \implies \text{sip} \, \xi' = \text{sip} \, \xi
\]

unfolding is_msg_defs
by (cases "msg \xi", clarsimp+)+

1.3.3 The protocol process

datatype pseqp =
  PAodv |
  PNewPkt |
  PPkt |
  PRreq |
  PRrep |
  PRerr

fun nat_of_seqp :: "pseqp \Rightarrow nat"
where
  "nat_of_seqp PAodv = 1"
|  "nat_of_seqp PPkt = 2"
|  "nat_of_seqp PNewPkt = 3"
|  "nat_of_seqp PRreq = 4"
|  "nat_of_seqp PRrep = 5"
|  "nat_of_seqp PRerr = 6"

instantiation "pseqp" :: ord
begin
definition less_eq_seqp [iff]: "\leq_1 = (\text{nat_of_seqp} \, \leq_1 \leq_2)"
definition less_seqp [iff]: "\less_1 = (\text{nat_of_seqp} \, \less_1 \less_2)"
instance..
end

abbreviation AODV
where
  "AODV \equiv \lambda_. \text{[clear_locals] call(PAodv)}"

abbreviation PKT
where
"PKT args ≡

ξ. let (data, dip, oip) = args ξ in
  (clear_locals ξ) [] data := data, dip := dip, oip := oip []
call(PPkt)"

abbreviation NEWPKT
where
"NEWPKT args ≡
ξ. let (data, dip) = args ξ in
  (clear_locals ξ) [] data := data, dip := dip []
call(PNewPkt)"

abbreviation RREQ
where
"RREQ args ≡
ξ. let (hops, dip, dsn, dsk, oip, osn, sip) = args ξ in
call(PRreq)"

abbreviation RREP
where
"RREP args ≡
ξ. let (hops, dip, dsn, oip, sip) = args ξ in
  (clear_locals ξ) [] hops := hops, dip := dip, dsn := dsn, oip := oip, sip := sip []
call(PRrep)"

abbreviation RERR
where
"RERR args ≡
ξ. let (dests, sip) = args ξ in
  (clear_locals ξ) [] dests := dests, sip := sip []
call(PRerr)"

fun Γ AODV :: "(state, msg, pseqp, pseqp label) seqp_env"
where
"Γ AODV PAodv = labelled PAodv (λmsg . ξ () msg := msg') (λ (is_newpkt) NEWPKT(λξ. (data ξ, ip ξ)))
  ⊕ (is_pkt) PKT(λξ. (data ξ, dip ξ, oip ξ))
  ⊕ (is_rreq)
    [ξ. ξ () rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ, {}) []]
    RREQ(λξ. (hops ξ, dip ξ, dsn ξ, dsk ξ, oip ξ, osn ξ, sip ξ))
  ⊕ (is_rrep)
    [ξ. ξ () rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ, {}) []]
    RREP(λξ. (hops ξ, dip ξ, dsn ξ, oip ξ, sip ξ))
  ⊕ (is_rerr)
    [ξ. ξ () rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ, {}) []]
    RERR(λξ. (dests ξ, sip ξ))
)
  ⊕ (λξ. {ξ () dip := dip []} [] dip. dip ∈ qD(store ξ) ∩ vD(rt ξ) [])
  ⊕ (λξ. {ξ () data := hd(σqueue(store ξ, dip ξ)) []} [] unicast(λξ. the (nhop (rt ξ) (dip ξ)), λξ. pkt(data ξ, dip ξ, ip ξ)).
    λξ. {ξ () store := the (drop (dip ξ) (store ξ)) []} [] AODV())
  ⊕ (λξ. {ξ () dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (dip ξ))
     then Some (inc (sqn (rt ξ) rip)) else None) []} []
    λξ. {ξ () rt := invalidate (rt ξ) (dests ξ) []}
    λξ. {ξ () store := setRRF (store ξ) (dests ξ) []} []
    λξ. {ξ () pre := (λξ. the (precs (rt ξ) rip) [] rip. rip ∈ dom (dests ξ) [] } []
  ⊕ (λξ. {ξ () dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ {}) then (dests ξ) rip else None) []}

108
\[
groupcast(λξ.\ pre\ ξ, λξ.\ rerr(dests\ ξ, ip\ ξ)).\ AODV()
\]

\[
\{ξ.\ dip\ ξ = dip\ |
\{dip.\ dip\ ∈\ qD(store\ ξ) \land \neg\ vD(rt\ ξ) \land \neg\ (\sigma_{p-flag}(store\ ξ, dip) = req)\}
\]

\[
[ξ.\ ξ \mid store := \text{unsetRRF (store \ ξ)} (dip \ ξ) \}]
\]

\[
[ξ.\ ξ \mid sn := \text{inc (sn \ ξ)} \}]
\]

\[
[ξ.\ ξ \mid rreqs := \{\{ip\ ξ, sn\ ξ\} \} \}]
\]

\[
\text{broadcast}(λξ.\ rreq(0, dip\ ξ, sqn\ rt\ ξ) (dip\ ξ), sqnf\ rt\ ξ) (dip\ ξ), ip\ ξ, sn\ ξ, ip\ ξ)).\ AODV()"
\]

\[
\Gamma_{AODV} \ PNewPkt = labelled\ PNewPkt (\langle ξ.\ dip\ ξ = ip\ ξ\rangle)
\]

\[
\text{deliver}(λξ.\ data\ ξ).AODV()
\]

\[
\langleξ.\ dip\ ξ \neq ip\ ξ\rangle
\]

\[
\{dip.\ dip\ ∈\ vD (rt\ ξ)\}
\]

\[
\text{unicast}(λξ.\ the\ (nhop\ (rt\ ξ)) (dip\ ξ)), λξ.\ pkt(data\ ξ, dip\ ξ, oip\ ξ).\ AODV()
\]

\[
\{ξ.\ dip\ ξ \neq vD (rt\ ξ)\}
\]

\[
\text{groupcast}(λξ.\ the\ (nhop\ (rt\ ξ)) (dip\ ξ)), λξ.\ pkt(data\ ξ, dip\ ξ, oip\ ξ).\ AODV()
\]

\[
\{ξ.\ dip\ ξ \neq vD (rt\ ξ)\}
\]

\[
\Gamma_{AODV} \ PPkt = labelled\ PPkt (\langleξ.\ dip\ ξ = ip\ ξ\rangle)
\]

\[
\text{deliver}(λξ.\ data\ ξ).AODV()
\]

\[
\langleξ.\ dip\ ξ \neq ip\ ξ\rangle
\]

\[
\{dip.\ dip\ ∈\ iD (rt\ ξ)\}
\]

\[
\text{unicast}(λξ.\ the\ (nhop\ (rt\ ξ)) (dip\ ξ)), λξ.\ pkt(data\ ξ, dip\ ξ, oip\ ξ).\ AODV()
\]

\[
\langleξ.\ dip\ ξ \neq iD (rt\ ξ)\}
\]

\[
\text{groupcast}(λξ.\ the\ (nhop\ (rt\ ξ)) (dip\ ξ)), λξ.\ pkt(data\ ξ, dip\ ξ, oip\ ξ).\ AODV()
\]

\[
\{ξ.\ dip\ ξ \neq iD (rt\ ξ)\}
\]

\[
\Gamma_{AODV} \ PRreq = labelled\ PRreq (\langleξ.\ (oip\ ξ, osn\ ξ) \in\ rreqs\ ξ\rangle)
\]

\[
\text{AODV()}
\]

\[
\langleξ.\ (oip\ ξ, osn\ ξ) \neq rreqs\ ξ\rangle
\]

\[
[ξ.\ ξ \mid rt := \text{update (rt \ ξ) (oip \ ξ) (osn \ ξ, kno, val, hops \ ξ + 1, sip \ ξ, \{\}) \}]
\]

\[
[ξ.\ ξ \mid rreqs := rreqs \cup \{(oip \ ξ, osn \ ξ)\} \}]
\]

\[
[ξ.\ ξ \mid \{dsn\ ξ ≤ sqn\ (rt\ ξ) (dip\ ξ) \land sqnf\ (rt\ ξ) (dip\ ξ) = kno\}]
\]
\[
\begin{align*}
\text{PRrep} & = \text{labelled PRrep (AODV)} \\
\text{unicast(} & \lambda \xi. \text{the (nhop (rt \xi) (oip \xi)), } \lambda \xi. \text{rrep(th the (nhops (rt \xi) (dip \xi)), dip \xi, sqn (rt \xi) (dip \xi), oip \xi, ip \xi)).}
\end{align*}
\]

\[
AODV()
\]

\[
\begin{align*}
\xi & \mid \text{dests := (} \lambda \text{rip. if (rip } \epsilon \text{ vD (rt} \xi) \wedge \text{nhop (rt} \xi) \text{ rip } = \text{nhop (rt} \xi) \text{ (oip} \xi) \text{) then Some (inc (sqn (rt} \xi) \text{ rip)) else None) } \\
\xi & \mid \text{rt := invalidate (rt} \xi) \text{ (dests} \xi) \text{)} \\
\xi & \mid \text{store := setRRF (store} \xi) \text{ (dests} \xi)\text{)} \\
\xi & \mid \text{pre := } \cup \{ \text{the (prec} \xi \text{ (rt} \xi) \text{ rip) } \mid \text{rip. rip } \epsilon \text{ dom (dests} \xi) \} \\
\xi & \mid \text{dests := (} \lambda \text{rip. if (} \text{dests} \xi \text{ rip } \neq \text{None } \wedge \text{the (prec} \xi \text{ (rt} \xi) \text{ rip}) \neq \{\} \text{ then (dests} \xi \text{ rip else None) )}
\end{align*}
\]

\[
\text{groupcast(} & \lambda \xi. \text{ pre } \xi, \lambda \xi. \text{ rerr(dests} \xi, \text{ ip} \xi)). AODV()
\]

\[
\begin{align*}
\xi & \mid \text{dip} \xi \notin \text{ vD (rt} \xi) \vee \text{sqn (rt} \xi) \text{ (dip} \xi) < \text{dsn} \xi \vee \text{sqnf (rt} \xi) \text{ (dip} \xi) = \text{unk)} \\
\text{broadcast(} & \lambda \xi. \text{ rreq(hops} \xi + 1, \text{ dip} \xi, \text{ max (sqn (rt} \xi) \text{ (dip} \xi) \text{) (dsn} \xi), \\
\text{dsk} \xi, \text{ oip} \xi, \text{ osn} \xi, \text{ ip} \xi)).
\end{align*}
\]

\[
AODV()
\]

\[
\begin{align*}
\text{rt } & = \text{ the (addpreRT (rt} \xi) \text{ (dip} \xi) \text{ (sip} \xi)) \\
\text{rt } & = \text{ the (addpreRT (rt} \xi) \text{ (oip} \xi) \text{ (the (nhop (rt} \xi) \text{ (dip} \xi)))}
\end{align*}
\]

\[
\text{unicast(} & \lambda \xi. \text{the (nhop (rt} \xi) \text{ (oip} \xi)), \lambda \xi. \text{rrep(hops} \xi + 1, \text{ dip} \xi, \text{ max (sqn (rt} \xi) \text{ (dip} \xi) \text{) (dsn} \xi), \\
\text{dsk} \xi, \text{ oip} \xi, \text{ osn} \xi, \text{ ip} \xi)).
\]

\[
AODV()
\]

\[
\begin{align*}
\xi & \mid \text{dests := (} \lambda \text{rip. if (rip } \epsilon \text{ vD (rt} \xi) \wedge \text{nhop (rt} \xi) \text{ rip } = \text{nhop (rt} \xi) \text{ (oip} \xi) \text{) then Some (inc (sqn (rt} \xi) \text{ rip)) else None) } \\
\xi & \mid \text{rt := invalidate (rt} \xi) \text{ (dests} \xi) \text{)} \\
\xi & \mid \text{store := setRRF (store} \xi) \text{ (dests} \xi)\text{)} \\
\xi & \mid \text{pre := } \cup \{ \text{the (prec} \xi \text{ (rt} \xi) \text{ rip) } \mid \text{rip. rip } \epsilon \text{ dom (dests} \xi) \} \\
\xi & \mid \text{dests := (} \lambda \text{rip. if (} \text{dests} \xi \text{ rip } \neq \text{None } \wedge \text{the (prec} \xi \text{ (rt} \xi) \text{ rip}) \neq \{\} \text{ then (dests} \xi \text{ rip else None) )}
\end{align*}
\]

\[
\text{groupcast(} & \lambda \xi. \text{ pre } \xi, \lambda \xi. \text{ rerr(dests} \xi, \text{ ip} \xi)). AODV()
\]

\[
\begin{align*}
\xi & \mid \text{rt } \xi = \text{ update (rt} \xi) \text{ (dip} \xi) \text{ (dsn} \xi, \text{ kno, val, hops} \xi + 1, \text{ sip} \xi, \{\}) \} \\
\text{AODV()}
\end{align*}
\]

\[
\begin{align*}
\text{dests := (} & \lambda \text{rip. case (dests} \xi \text{ rip of None } \Rightarrow \text{ None) } \Rightarrow \text{ Some rsn } \Rightarrow \text{ if rip } \epsilon \text{ vD (rt} \xi) \wedge \text{the (nhop (rt} \xi) \text{ rip) = sip} \xi \wedge \text{sqn (rt} \xi) \text{ rip } < \text{rsn then Some rsn else None) } \} \\
\xi & \mid \text{rt := invalidate (rt} \xi) \text{ (dests} \xi) \text{)} \\
\xi & \mid \text{store := setRRF (store} \xi) \text{ (dests} \xi)\text{)} \\
\xi & \mid \text{pre := } \cup \{ \text{the (prec} \xi \text{ (rt} \xi) \text{ rip) } \mid \text{rip. rip } \epsilon \text{ dom (dests} \xi) \} \\
\xi & \mid \text{dests := (} \lambda \text{rip. if (} \text{dests} \xi \text{ rip } \neq \text{None } \wedge \text{the (prec} \xi \text{ (rt} \xi) \text{ rip}) \neq \{\} \text{ then (dests} \xi \text{ rip else None) )}
\end{align*}
\]

\[
\text{groupcast(} & \lambda \xi. \text{ pre } \xi, \lambda \xi. \text{ rerr(dests} \xi, \text{ ip} \xi)). AODV()
\]

\[
\begin{align*}
\text{dests := (} & \lambda \text{rip. case (dests} \xi \text{ rip of None } \Rightarrow \text{ None) } \Rightarrow \text{ Some rsn } \Rightarrow \text{ if rip } \epsilon \text{ vD (rt} \xi) \wedge \text{the (nhop (rt} \xi) \text{ rip) = sip} \xi \wedge \text{sqn (rt} \xi) \text{ rip } < \text{rsn then Some rsn else None) } \} \\
\xi & \mid \text{rt := invalidate (rt} \xi) \text{ (dests} \xi) \text{)} \\
\xi & \mid \text{store := setRRF (store} \xi) \text{ (dests} \xi)\text{)} \\
\xi & \mid \text{pre := } \cup \{ \text{the (prec} \xi \text{ (rt} \xi) \text{ rip) } \mid \text{rip. rip } \epsilon \text{ dom (dests} \xi) \} \\
\xi & \mid \text{dests := (} \lambda \text{rip. if (} \text{dests} \xi \text{ rip } \neq \text{None } \wedge \text{the (prec} \xi \text{ (rt} \xi) \text{ rip}) \neq \{\} \text{ then (dests} \xi \text{ rip else None) )}
\end{align*}
\]

\[
\text{groupcast(} & \lambda \xi. \text{ pre } \xi, \lambda \xi. \text{ rerr(dests} \xi, \text{ ip} \xi)). AODV()"
declare $\Gamma_{AODV}.\text{sims}$ [simp del, code del]
lemmas $\Gamma_{AODV}.\text{sims}$ [simp, code] = $\Gamma_{AODV}.\text{sims}$ [simplified]

fun $\Gamma_{AODV}.\text{skeleton}$
where
  "$\Gamma_{AODV}.\text{skeleton} \text{ PAodv} = \text{seqp_skeleton} (\Gamma_{AODV} \text{ PAodv})"
/ "$\Gamma_{AODV}.\text{skeleton} \text{ PNewPkt} = \text{seqp_skeleton} (\Gamma_{AODV} \text{ PNewPkt})"
/ "$\Gamma_{AODV}.\text{skeleton} \text{ PPkt} = \text{seqp_skeleton} (\Gamma_{AODV} \text{ PPkt})"
/ "$\Gamma_{AODV}.\text{skeleton} \text{ PRreq} = \text{seqp_skeleton} (\Gamma_{AODV} \text{ PRreq})"
/ "$\Gamma_{AODV}.\text{skeleton} \text{ PRrep} = \text{seqp_skeleton} (\Gamma_{AODV} \text{ PRrep})"
/ "$\Gamma_{AODV}.\text{skeleton} \text{ PRerr} = \text{seqp_skeleton} (\Gamma_{AODV} \text{ PRerr})"

lemma $\Gamma_{AODV}.\text{skeleton}\_wf$ [simp]:
  "wellformed $\Gamma_{AODV}.\text{skeleton}"
proof (rule, intro allI)
  fix pn pn'
  show "call(pn') \notin \text{stermsl} (\Gamma_{AODV}.\text{skeleton} pn)"
  by (cases pn) simp_all
qed

declare $\Gamma_{AODV}.\text{skeleton}\_sims$ [simp del, code del]
lemmas $\Gamma_{AODV}.\text{skeleton}\_sims$ [simp, code]
  = $\Gamma_{AODV}.\text{skeleton}\_sims$ [simplified $\Gamma_{AODV}.\text{sims}$ seqp_skeleton.simps]

lemma aodv.proc_cases [dest]:
  fixes pn
  shows "p \in \text{ctermsl} (\Gamma_{AODV} \ pn) \imp
    (p \in \text{ctermsl} (\Gamma_{AODV} \ \text{PAodv}) \lor
     p \in \text{ctermsl} (\Gamma_{AODV} \ \text{PNewPkt}) \lor
     p \in \text{ctermsl} (\Gamma_{AODV} \ \text{PPkt}) \lor
     p \in \text{ctermsl} (\Gamma_{AODV} \ \text{PRreq}) \lor
     p \in \text{ctermsl} (\Gamma_{AODV} \ \text{PRrep}) \lor
     p \in \text{ctermsl} (\Gamma_{AODV} \ \text{PRerr}))"
  using assms
  by (cases pn) simp_all

definition $\sigma_{AODV}$ :: "ip \Rightarrow (\text{state} \times (\text{state, msg, pseqp, pseqp label}) \ \text{seqp}) \ \text{set}"
where "$\sigma_{AODV} \ i = \{(\text{aodv.init} \ i, \Gamma_{AODV} \ \text{PAodv})\}"

abbreviation paodv
  :: "ip \Rightarrow (\text{state} \times (\text{state, msg, pseqp, pseqp label}) \ \text{seqp, msg seq_action}) \ \text{automaton}"
where
  "paodv i = (\text{init} = \sigma_{AODV} \ i, \ \text{trans} = \text{seqp_sos} \ \Gamma_{AODV} )"

lemma aodv.trans: "\text{trans} (paodv \ i) = \text{seqp_sos} \ \Gamma_{AODV}"
  by simp

lemma aodv.control_within [simp]: "\text{control_within} \ \Gamma_{AODV} \ (\text{init} \ (paodv \ i))"
  unfolding $\sigma_{AODV}\_\text{def}$
  by (rule control_withinI) (auto simp del: $\Gamma_{AODV}.\text{sims}$)

lemma aodv_wf [simp]:
  "wellformed \ \Gamma_{AODV}"
proof (rule, intro allI)
  fix pn pn'
  show "call(pn') \notin \text{stermsl} (\Gamma_{AODV} \ pn)"
  by (cases pn) simp_all
qed

lemmas aodv.labels_not_empty [simp] = labels_not_empty [OF aodv_wf]

lemma aodv.of_label [intro]: "\exists l. l \in \text{labels} \ \Gamma_{AODV} \ p"
  by (metis aodv.labels_not_empty all_not_in_conv)
lemma aodv_ex_labelE [elim]:
assumes "\( \forall l \in \text{labels} \Gamma_{AODV} p. \ P l p \)"
and "\( \exists p. l. \ P l p \Rightarrow Q \)"
shows "Q"
using assms by (metis aodv_ex_label)

lemma aodv_simple_labels [simp]: "simple_labels \( \Gamma_{AODV} \)"
proof
fix pn p
assume "p \in \text{subterms}(\Gamma_{AODV} pn)"
thus "\( \exists! l. \text{labels} \Gamma_{AODV} p = \{l\} \)"
by (cases pn) (simp_all cong: seqp_congs | elim disjE)+
qed

lemma \( \sigma_{AODV}\_\text{labels} \) [simp]: "(\xi, p) \in \sigma_{AODV} i \Rightarrow \text{labels} \Gamma_{AODV} p = \{\text{PAodv-}:0\}"
unfolding \( \sigma_{AODV}\_\text{def} \) by simp

lemma aodv_init_kD_empty [simp]:
"(\xi, p) \in \sigma_{AODV} i \Rightarrow \text{kD (rt } \xi) = \{\}\"
unfolding \( \sigma_{AODV}\_\text{def} \) \( \text{kD}\_\text{def} \) by simp

lemma aodv_init_sip_not_ip [simp]: "\neg(\text{sip (aodv_init i) = i})" by simp

lemma aodv_init_sip_not_ip' [simp]:
assumes "(\xi, p) \in \sigma_{AODV} i"
shows "\text{sip } \xi \neq \text{ip } \xi"
using assms unfolding \( \sigma_{AODV}\_\text{def} \) by simp

lemma aodv_init_sip_not_i [simp]:
assumes "(\xi, p) \in \sigma_{AODV} i"
shows "\text{sip } \xi \neq i"
using assms unfolding \( \sigma_{AODV}\_\text{def} \) by simp

lemma clear_locals_sip_not_ip':
assumes "\text{ip } \xi = i"
shows "\neg(\text{sip (clear_locals } \xi) = i)"
using assms by auto

Stop the simplifier from descending into process terms.

declare seqp_congs [cong]

Configure the main invariant tactic for AODV.

declare 
\Gamma_{AODV}\_\text{simps} [cterms_env]
\text{aodv}\_\text{proc_cases} [ctermsl_cases]
\text{seq\_invariant\_ctermsI} [OF aodv_wf aodv\_control\_within aodv\_simple_labels aodv\_trans, cterms_intros]
\text{seq\_step\_invariant\_ctermsI} [OF aodv_wf aodv\_control\_within aodv\_simple_labels aodv\_trans, cterms_intros]

end

1.4 Invariant assumptions and properties

theory A_Aodv_Predicates
imports A_Aodv
begin

Definitions for expression assumptions on incoming messages and properties of outgoing messages.

abbreviation \text{not}\_Pkt :: "msg \Rightarrow \text{bool}"
where "\text{not}\_Pkt m \equiv \text{case } m \text{ of Pkt \_ \_ \_ } \Rightarrow \text{False} \ | \_ \Rightarrow \text{True}"

definition \text{msg}\_sender :: "msg \Rightarrow \text{ip}"

112
where "msg_sender m ≡ case m of Rreq _ _ _ _ _ _ ipc ⇒ ipc
| Rep _ _ _ _ ipc ⇒ ipc
| Rerr _ ipc ⇒ ipc
| Pkt _ _ ipc ⇒ ipc"

lemma msg_sender_simps [simp]:
"∀ hops dip dsn oip osn sip.
  msg_sender (Rreq hops dip dsn oip osn sip) = sip"
"∀ hops dip dsn oip sip. msg_sender (Rreq hops dip dsn oip sip) = sip"
"∀ dests sip. msg_sender (Rerr dests sip) = sip"
"∀ d dip. msg_sender (Pkt d dip sip) = sip"
unfolding msg_sender_def by simp_all

definition msg_zhops :: "msg ⇒ bool"
where "msg_zhops m ≡ case m of
  Rreq hopsc dipc _ _ _ _ oipc _ sipc ⇒ hopsc = 0 −→ oipc = sipc
  | Rrep hopsc dipc _ _ _ _ sipc ⇒ hopsc = 0 −→ dipc = sipc
  | _ ⇒ True"

lemma msg_zhops_simps [simp]:
"∀ hops dip dsn oip osn sip.
  msg_zhops (Rreq hops dip dsn oip osn sip) = (hops = 0 −→ oip = sip)"
"∀ hops dip dsn oip sip. msg_zhops (Rreq hops dip dsn oip sip) = (hops = 0 −→ dip = sip)"
"∀ dests sip. msg_zhops (Rerr dests sip) = True"
"∀ d dip. msg_zhops (Newpkt d dip) = True"
"∀ d dip. msg_zhops (Pkt d dip sip) = True"
unfolding msg_zhops_def by simp_all

definition rreq_rrep_sn :: "msg ⇒ bool"
where "rreq_rrep_sn m ≡ case m of
  Rreq hopsc dipc _ _ _ _ _ osnc _ ⇒ osnc ≥ 1
  | Rrep hopsc dipc _ _ _ _ _ sipc ⇒ dipc = sipc
  | _ ⇒ True"

lemma rreq_rrep_sn_simps [simp]:
"∀ hops dip dsn oip osn sip.
  rreq_rrep_sn (Rreq hops dip dsn oip osn sip) = (osn ≥ 1)"
"∀ hops dip dsn oip sip. rreq_rrep_sn (Rreq hops dip dsn oip sip) = (dsn ≥ 1)"
"∀ dests sip. rreq_rrep_sn (Rerr dests sip) = True"
"∀ d dip. rreq_rrep_sn (Newpkt d dip) = True"
"∀ d dip. rreq_rrep_sn (Pkt d dip sip) = True"
unfolding rreq_rrep_sn_def by simp_all

definition rreq_rrep_fresh :: "rt ⇒ msg ⇒ bool"
where "rreq_rrep_fresh crt m ≡ case m of
  Rreq hopsc dipc _ _ _ _ _ oipc osnc ipcc ⇒ (ipcc ≠ oipc −→ oipc ∈ kD(crt) ∧ (sqn crt oipc > osnc
  ∨ (sqn crt oipc = osnc
  ∧ the (dhops crt oipc) ≤ hopsc
  ∧ the (flag crt oipc) = val)))
  | Rrep hopsc dipc _ _ _ _ _ ipcc ⇒ (ipcc ≠ dipc −→ dipc ∈ kD(crt)
  ∧ sqn crt dipc = dsnc
  ∧ the (dhops crt dipc) = hopsc
  ∧ the (flag crt dipc) = val)
  | _ ⇒ True"

lemma rreq_rrep_fresh [simp]:
"∀ hops dip dsn oip osn sip.
  rreq_rrep_fresh crt (Rreq hops dip dsn oip osn sip) = (sip ≠ oip −→ oip ∈ kD(crt)
  ∧ (sqn crt oip > osn
  ∨ (sqn crt oip = osn
  ∧ the (dhops crt oip) ≤ hops
  ∧ the (flag crt oip) = val)))"
"∀ hops dip dsn oip sip. rreq_rrep_fresh crt (Rrep hops dip dsn oip sip) =
(sip ≠ dip → dip∈kD(crt)
   ∧ sqn crt dip = dsn
   ∧ the (dhops crt dip) = hops
   ∧ the (flag crt dip) = val)"

"∧dests sip.  rreq_rrep_fresh crt (Rerr dests sip) = True"
"∧d dip.     rreq_rrep_fresh crt (Newpkt d dip) = True"
"∧d dip sip.  rreq_rrep_fresh crt (Pkt d dip sip) = True"

unfolding rreq_rrep_fresh_def by simp_all

definition rerr_invalid :: "rt ⇒ msg ⇒ bool"
where "rerr_invalid crt m = case m of Rerr destsc _ ⇒
(∀ripc ∈ iD(crt). the (destsc ripc) = sqn crt ripc)
| _ ⇒ True"

lemma rerr_invalid [simp]:
"∧hops dip dsn dsk oip osn sip.
  rerr_invalid crt (Rreq hops dip dsn dsk oip osn sip) = True"
"∧hops dip dsn oip sip. rerr_invalid crt (Rrep hops dip dsn oip sip) = True"
"∧dests sip. rerr_invalid crt (Rerr dests sip) = (∀ripc ∈ iD(crt). ripc ≤ iD(crt) ∧ the (dests ripc) = sqn crt ripc)"
"∧d dip.     rerr_invalid crt (Newpkt d dip) = True"
"∧d dip sip. rerr_invalid crt (Pkt d dip sip) = True"

unfolding rerr_invalid_def by simp_all

definition initmissing :: "(nat ⇒ state option) × 'a ⇒ (nat ⇒ state) × 'a"
where "initmissing σ = (λi. case (fst σ) i of None ⇒ aodv_init i | Some s ⇒ s, snd σ)"

lemma not_in_net_ips_fst_init_missing [simp]:
assumes "i /∈ net_ips σ"
shows "fst (initmissing (netmap fst σ)) i = aodv_init i"
using assms unfolding initmissing_def by simp

lemma fst_initmissing_netmap_pair_fst [simp]:
"fst (initmissing (netmap (λ(p, q). (fst (id p), snd (id p), q)) s))
  = fst (initmissing (netmap fst s))"

unfolding initmissing_def by auto

We introduce a streamlined alternative to initmissing with netmap to simplify invariant statements and thus facilitate their comprehension and presentation.

lemma fst_initmissing_netmap_default_aodv_init_netlift:
"fst (initmissing (netmap fst s)) = default aodv_init (netlift fst s)"

unfolding initmissing_def default_def
by (simp add: fst_netmap_netlift del: One_nat_def)

definition netglobal :: "((nat ⇒ state) ⇒ bool) ⇒ ((state × 'b) × 'c) net_state ⇒ bool"
where "netglobal P ≡ (λs. P (default aodv_init (netlift fst s)))"
end

1.5 Quality relations between routes

thory A_Fresher
imports A_Aodv_Data
begin

1.5.1 Net sequence numbers

On individual routes

definition
nsqn \_r :: "r \Rightarrow sqn"

where
"nsqn, r \equiv \Pi_4(r) = val \lor \Pi_2(r) = 0 \text{ then } \Pi_2(r) \text{ else } (\Pi_2(r) \ - 1)"

lemma nsqnr\_def':
"nsqn, r = (if \Pi_4(r) = inv then \Pi_2(r) - 1 else \Pi_2(r))"

unfolding nsqn\_def by simp

lemma nsqn\_zero [simp]:
"\forall dsn dsk flag hops nhip pre. nsqn, (0, dsk, flag, hops, nhip, pre) = 0"

unfolding nsqn\_def by clarsimp

lemma nsqn\_val [simp]:
"\forall dsn dsk hops nhip pre. nsqn, (dsn, dsk, val, hops, nhip, pre) = dsn"

unfolding nsqn\_def by clarsimp

lemma nsqn\_inv [simp]:
"\forall dsn dsk hops nhip pre. nsqn, (dsn, dsk, inv, hops, nhip, pre) = dsn - 1"

unfolding nsqn\_def by clarsimp

lemma nsqn\_lte_dsn [simp]:
"\forall dsn dsk flag hops nhip pre. nsqn, (dsn, dsk, flag, hops, nhip, pre) \leq dsn"

unfolding nsqn\_def by clarsimp

On routes in routing tables

definition
nsqn :: "rt \Rightarrow ip \Rightarrow sqn"

where
"nsqn \equiv \lambda rt dip. \ case \ \sigma_{route}(rt, dip) \ of None \Rightarrow 0 | \ Some r \Rightarrow nsqn, (r)"

lemma nsqn\_sqn\_def:
"\forall rt dip. nsqn rt dip = (if flag rt dip = Some val \lor sqn rt dip = 0
then sqn rt dip else sqn rt dip - 1)"

unfolding nsqn\_def sqn\_def by (clarsimp split: option.split) auto

lemma not\_in\_kD\_nsqn [simp]:
assumes "dip /\in kD(rt)"
shows "nsqn rt dip = 0"
using assms unfolding nsqn\_def by simp

lemma kD\_nsqn:
assumes "dip \in kD(rt)"
shows "nsqn rt dip = nsqn, (the \ (\sigma_{route}(rt, dip)))"
using assms [THEN kD\_Some] unfolding nsqn\_def by clarsimp

lemma nsqn\_r\_flag\_pred [simp, intro]:
fixes dsn dsk flag hops nhip pre
assumes "P (nsqn, (dsn, dsk, val, hops, nhip, pre))"
and "P (nsqn, (dsn, dsk, inv, hops, nhip, pre))"
shows "P (nsqn, (dsn, dsk, flag, hops, nhip, pre))"
using assms by (cases flag) auto

lemma nsqn\_addpre\_RT\_inv [simp]:
"\forall rt dip npre dip', dip \in kD(rt) \Rightarrow
nsqn, (the \ (the \ (addpre\_RT rt dip npre) dip')) = nsqn, (the \ (rt dip'))"

unfolding addpre\_RT\_def nsqn\_def
by (frule kD\_Some) (clarsimp split: option.split)

lemma sqn\_nsqn:
"\forall rt dip. sqn rt dip - 1 \leq nsqn rt dip"

unfolding sqn\_def nsqn\_def by (clarsimp split: option.split)

lemma nsqn\_sqn: "nsqn rt dip \leq sqn rt dip"
unfolding sqn_def nsqn_def by (cases "rt dip") auto

lemma val_nsqn_sqn [elim, simp]:
  assumes "ip ∈ kD(rt)"
  and "the (flag rt ip) = val"
  shows "nsqn rt ip = sqn rt ip"
using assms unfolding nsqn_sqn_def by auto

lemma vD_nsqn_sqn [elim, simp]:
  assumes "ip ∈ vD(rt)"
  shows "nsqn rt ip = sqn rt ip"
proof -
  from ⟨ip ∈ vD(rt)⟩ have "ip ∈ kD(rt)"
  and "the (flag rt ip) = val" by auto
  thus ?thesis ..
qed

lemma inv_nsqn_sqn [elim, simp]:
  assumes "ip ∈ kD(rt)"
  and "the (flag rt ip) = inv"
  shows "nsqn rt ip = sqn rt ip - 1"
using assms unfolding nsqn_sqn_def by auto

lemma iD_nsqn_sqn [elim, simp]:
  assumes "ip ∈ iD(rt)"
  shows "nsqn rt ip = sqn rt ip - 1"
proof -
  from ⟨ip ∈ iD(rt)⟩ have "ip ∈ kD(rt)"
  and "the (flag rt ip) = inv" by auto
  thus ?thesis ..
qed

lemma nsqn_update_changed_kno_val [simp]: "∀ rt ip dsn dsk hops nhip. rt ≠ update rt ip (dsn, kno, val, hops, nhip, {}) ⇒ nsqn (update rt ip (dsn, kno, val, hops, nhip, {})) ip = dsn"
using assms unfolding nsqn_def update_def
by (clarsimp simp: kD_nsqn split: option.split_asm option.split split_if_asm)
  (metis fun_upd_triv)

lemma nsqn_addpreRT_inv [simp]:
  "∀ rt dip npre dip'. dip ∈ kD(rt) ⇒ nsqn (the (addpreRT rt dip npre)) dip' = nsqn rt dip'"
unfolding addpreRT_def nsqn_def nsqn_-def
by (frule kD_Some) (clarsimp simp: split: option.split)

lemma nsqn_update_other [simp]:
  fixes dsn dsk flag hops dip nhip pre rt ip
  assumes "dip ≠ ip"
  shows "nsqn (update rt ip (dsn, dsk, flag, hops, nhip, pre)) dip = nsqn rt dip"
  using assms unfolding nsqn_def
by (clarsimp simp: split: option.split)

lemma nsqn_invalidate_eq:
  assumes "dip ∈ kD(rt)"
  and "dests dip = Some rsn"
  shows "nsqn (invalidate rt dests) dip = rsn - 1"
using assms
proof -
  from assms obtain dsk hops nhip pre
    where "invalidate rt dests dip = Some (rsn, dsk, inv, hops, nhip, pre)"
    unfolding invalidate_def
    by auto
  moreover from ⟨dip ∈ kD(rt)⟩ have "dip ∈ kD(invalidate rt dests)" by simp
  ultimately show ?thesis
lemma nsqn_invalidate_other [simp]:
  assumes "dip ∈ kD(rt)"
  and "dip ∉ dom dests"
  shows "nsqn (invalidate rt dests) dip = nsqn rt dip"
  using assms by (clarsimp simp add: kD_nsqn)

1.5.2 Comparing routes

definition fresher :: "r ⇒ r ⇒ bool" ("(_/ ⊑ _)"
  [51, 51] 50)
where
  "fresher r r' ≡ ((nsqn r < nsqn r') ∨ (nsqn r = nsqn r' ∧ π5(r) ≥ π5(r')))"

lemma fresherI1 [intro]:
  assumes "nsqn r r < nsqn r r'"
  shows "r ⊑ r'"
  unfolding fresher_def using assms by simp

lemma fresherI2 [intro]:
  assumes "nsqn r r = nsqn r r'" and "π5(r) ≥ π5(r')"
  shows "r ⊑ r'"
  unfolding fresher_def using assms by simp

lemma fresherI [intro]:
  assumes "(nsqn r r < nsqn r r') ∨ (nsqn r = nsqn r' ∧ π5(r) ≥ π5(r'))"
  shows "r ⊑ r'"
  unfolding fresher_def using assms.

lemma fresherE [elim]:
  assumes "r ⊑ r'" and "nsqn r r < nsqn r r' =⇒ P r r'"
  and "nsqn r r = nsqn r r' ∧ π5(r) ≥ π5(r') =⇒ P r r'"
  shows "P r r'"
  using assms unfolding fresher_def by auto

lemma fresher_refl [simp]: "r ⊑ r"
  unfolding fresher_def by simp

lemma fresher_trans [elim, trans]:
  "[ x ⊑ y; y ⊑ z ] =⇒ x ⊑ z"
  unfolding fresher_def by auto

lemma not_fresher_trans [elim, trans]:
  "[ ¬(x ⊑ y); ¬(z ⊑ x) ] =⇒ ¬(z ⊑ y)"
  unfolding fresher_def by auto

lemma fresher_dsn_flag_hops_const [simp]:
  fixes dsn dsk dsk' flag hops nhip nhip' pre pre'
  shows "(dsn, dsk, flag, hops, nhip, pre) ⊑ (dsn, dsk', flag, hops, nhip', pre')"
  unfolding fresher_def by (cases flag) simp_all

lemma addpre_fresher [simp]: "∀ r npre. r ⊑ (addpre r npre)"
  by clarsimp

1.5.3 Comparing routing tables

definition rt_fresher :: "ip ⇒ rt ⇒ rt ⇒ bool"
where
  "rt_fresher ≡ λdip rt rt’. (the (σroute(rt, dip))) ⊑ (the (σroute(rt’, dip)))"
abbreviation 
  rt_fresher_syn :: "rt ⇒ ip ⇒ rt ⇒ bool" ("(_/ ⊑ _)"
  [51, 999, 51] 50)
where 
  "rt1 ⊑ dip rt2 ≡ rt_fresher i rt1 rt2"

lemma rt_fresher_def':
  "(rt1 ⊑ i rt2) = (nsqn_r (the (rt1 i)) < nsqn_r (the (rt2 i)) ∨
    nsqn_r (the (rt1 i)) = nsqn_r (the (rt2 i)) ∧ π5 (the (rt2 i)) ≤ π5 (the (rt1 i)))"
unfolding rt_fresher_def fresher_def by (rule refl)

lemma single_rt_fresher [intro]:
  assumes "the (rt1 ip) ⊑ the (rt2 ip)"
  shows "rt1 ⊑ ip rt2"
using assms unfolding rt_fresher_def.

lemma rt_fresher_single [intro]:
  assumes "rt1 ⊑ ip rt2"
  shows "the (rt1 ip) ⊑ the (rt2 ip)"
using assms unfolding rt_fresher_def.

lemma rt_fresher_def2:
  assumes "dip ∈ kD(rt1)"
  and "dip ∈ kD(rt2)"
  shows "(rt1 ⊑ dip rt2) = (nsqn rt1 dip < nsqn rt2 dip
    ∨ (nsqn rt1 dip = nsqn rt2 dip
    ∧ the (dhops rt1 dip) ≥ the (dhops rt2 dip)))"
using assms unfolding rt_fresher_def2 [OF assms(1-2)] using assms(3) by simp

lemma rt_fresherI1 [intro]:
  assumes "dip ∈ kD(rt1)"
  and "dip ∈ kD(rt2)"
  and "nsqn rt1 dip < nsqn rt2 dip"
  shows "rt1 ⊑ dip rt2"
unfolding rt_fresher_def2 [OF assms(1-2)] using assms(3) by simp

lemma rt_fresherI2 [intro]:
  assumes "dip ∈ kD(rt1)"
  and "dip ∈ kD(rt2)"
  and "nsqn rt1 dip = nsqn rt2 dip"
  and "the (dhops rt1 dip) ≥ the (dhops rt2 dip)"
  shows "rt1 ⊑ dip rt2"
unfolding rt_fresher_def2 [OF assms(1-2)] using assms(3-4) by simp

lemma rt_fresherE [elim]:
  assumes "rt1 ⊑ dip rt2"
  and "dip ∈ kD(rt1)"
  and "dip ∈ kD(rt2)"
  and "[ nsqn rt1 dip < nsqn rt2 dip ] ⟹ P rt1 rt2 dip"
  and "[ nsqn rt1 dip = nsqn rt2 dip; the (dhops rt1 dip) ≥ the (dhops rt2 dip) ] ⟹ P rt1 rt2 dip"
  shows "P rt1 rt2 dip"
using assms(1) unfolding rt_fresher_def2 [OF assms(2-3)]
using assms(4-5) by auto

lemma rt_fresher_refl [simp]: "rt ⊑ dip rt"
unfolding rt_fresher_def by simp

lemma rt_fresher_trans [elim, trans]:
  assumes "rt1 ⊑ dip rt2"
  and "rt2 ⊑ dip rt3"
  shows "rt1 ⊑ dip rt3"
using assms unfolding rt_fresher_def by auto
lemma rt_fresher_if_Some [intro!]:
  assumes "the (rt dip) ⊑ r"
  shows "rt ⊑ dip (λip. if ip = dip then Some r else rt ip)"
  using assms unfolding rt_fresher_def by simp

definition rt_fresh_as :: "ip ⇒ rt ⇒ rt ⇒ bool"
where
  "rt_fresh_as ≡ λdip rt1 rt2. (rt1 ⊑ dip rt2) ∧ (rt2 ⊑ dip rt1)"

abbreviation
  rt_fresh_as_syn :: "rt ⇒ ip ⇒ rt ⇒ bool" ("(_)≈_") [51, 999, 51] 50
where
  "rt1 ≈_i rt2 ≡ rt_fresh_as i rt1 rt2"

lemma rt_fresh_as_refl [simp]: "⋀rt dip. rt ≈ dip rt"
unfolding rt_fresh_as_def by simp

lemma rt_fresh_as_trans [simp, intro, trans]:
  "⋀rt1 rt2 rt3 dip. [ rt1 ≈ dip rt2; rt2 ≈ dip rt3 ] ⇒ rt1 ≈ dip rt3"
unfolding rt_fresh_as_def rt_fresher_def
by (metis (mono_tags) fresher_trans)

lemma rt_fresh_asI [intro!]:
  assumes "rt1 ⊑ dip rt2"
  and "rt2 ⊑ dip rt1"
  shows "rt1 ≈ dip rt2"
  using assms unfolding rt_fresh_as_def by simp

lemma rt_fresh_as_fresherI [intro]:
  assumes "dip ∈ kD(rt1)"
  and "dip ∈ kD(rt2)"
  and "the (rt1 dip) ⊑ the (rt2 dip)"
  and "the (rt2 dip) ⊑ the (rt1 dip)"
  shows "rt1 ≈ dip rt2"
  using assms unfolding rt_fresh_as_def
by (clarsimp dest!: single_rt_fresher)

lemma nsqn_rt_fresh_asI:
  assumes "dip ∈ kD(rt)"
  and "dip ∈ kD(rt')"
  and "nsqn rt dip = nsqn rt' dip"
  and "π5(the (rt dip)) = π5(the (rt' dip))"
  shows "rt ≈ dip rt'"
proof
  from assms(1-2,4) have dhops': "the (dhops rt' dip) ≤ the (dhops rt dip)"
    by (simp add: proj5_eq_dhops)
  with assms(1-3) show "rt ⊑ dip rt'"
    by (rule rt_fresherI2)
  next
  from assms(1-2,4) have dhops: "the (dhops rt dip) ≤ the (dhops rt' dip)"
    by (simp add: proj5_eq_dhops)
  with assms(2,1) assms(3) [symmetric] show "rt' ⊑ dip rt"
    by (rule rt_fresherI2)
qed

lemma rt_fresh_asE [elim]:
  assumes "rt1 ≈ dip rt2"
  and "[ rt1 ⊑ dip rt2; rt2 ⊑ dip rt1 ] ⇒ P rt1 rt2 dip"
  shows "P rt1 rt2 dip"
  using assms unfolding rt_fresh_as_def by simp

lemma rt_fresh_asD1 [dest]:
  assumes "rt1 ≈ dip rt2"
  shows "rt1 ⊑ dip rt2"
using assms unfolding rt_fresh_as_def by simp

lemma rt_fresh_asD2 [dest]:
  assumes "rt1 \approx dip rt2"
  shows "rt2 \subseteq dip rt1"
using assms unfolding rt_fresh_as_def by simp

lemma rt_fresh_as_sym:
  assumes "rt1 \approx dip rt2"
  shows "rt2 \approx dip rt1"
using assms unfolding rt_fresh_as_def by simp

lemma not_rt_fresh_asI1 [intro]:
  assumes "\neg (rt1 \subseteq dip rt2)"
  shows "\neg (rt1 \approx dip rt2)"
proof
  assume "rt1 \approx dip rt2"
  hence "rt1 \subseteq dip rt2" ..
  with \neg (rt1 \subseteq dip rt2) show False ..
qed

lemma not_rt_fresh_asI2 [intro]:
  assumes "\neg (rt2 \subseteq dip rt1)"
  shows "\neg (rt1 \approx dip rt2)"
proof
  assume "rt1 \approx dip rt2"
  hence "rt2 \subseteq dip rt1" ..
  with \neg (rt2 \subseteq dip rt1) show False ..
qed

lemma not_single_rt_fresher [elim]:
  assumes "\neg ((the (rt1 ip) \subseteq the (rt2 ip)))"
  shows "\neg (rt1 \subseteq ip rt2)"
proof
  assume "rt1 \subseteq ip rt2"
  hence "the (rt1 ip) \subseteq the (rt2 ip)" ..
  with \neg (the (rt1 ip) \subseteq the (rt2 ip)) show False ..
qed

lemmas not_single_rt_fresh_asI1 [intro] = not_rt_fresh_asI1 [OF not_single_rt_fresher]
lemmas not_single_rt_fresh_asI2 [intro] = not_rt_fresh_asI2 [OF not_single_rt_fresher]

lemma not_rt_fresher_single [elim]:
  assumes "\neg (rt1 \subseteq ip rt2)"
  shows "\neg ((the (rt1 ip) \subseteq the (rt2 ip)))"
proof
  assume "the (rt1 ip) \subseteq the (rt2 ip)"
  hence "rt1 \subseteq ip rt2" ..
  with \neg (the (rt1 ip) \subseteq the (rt2 ip)) show False ..
qed

lemma rt_fresh_as_nsqnr:
  assumes "dip \in kD(rt1)"
  and "dip \in kD(rt2)"
  and "rt1 \approx dip rt2"
  shows "nsqn r (the (rt2 dip)) = nsqn r (the (rt1 dip))"
using assms(3) unfolding rt_fresh_as_def
by (auto simp: rt_fresher_def2 [OF \dip \in kD(rt1) \dip \in kD(rt2)\])
rt_fresher_def2 [OF \dip \in kD(rt2) \dip \in kD(rt1)\]
kD_nsqn [OF \dip \in kD(rt1)\]
kD_nsqn [OF \dip \in kD(rt2)\])

lemma rt_fresher_mapupd [intro!]:
  assumes "dip \in kD(rt)"

and "the (rt dip) ⊆ r"
  shows "rt ⊑ dip rt(dip → r)"
using assms unfolding rt_fresher_def by simp

lemma rt_fresher_map_update_other [intro!]:
  assumes "dip ∈ kD(rt)"
  and "dip ≠ ip"
  shows "rt ⊑ dip rt(ip → r)"
using assms unfolding rt_fresher_def by simp

lemma rt_fresher_update_other [simp]:
  assumes inkD: "dip ∈ kD(rt)"
  and "dip ≠ ip"
  shows "rt ⊑ dip update rt ip r"
using assms unfolding update_def
by (clarsimp split: option.split) (fastforce)

theorem rt_fresher_update [simp]:
  assumes "dip ∈ kD(rt)"
  and "the (dhops rt dip) ≥ 1" 
  and "update_arg_wf r"
  shows "rt ⊑ dip update rt dip r"
proof (cases "dip = ip")
  assume "dip ≠ ip"
  with ⟨dip ∈ kD(rt)⟩
  show ?thesis
  by (rule rt_fresher_update_other)
next
  assume "dip = ip"
  from ⟨dip ∈ kD(rt)⟩
  have 
  "dsn rt dip = dsn,
  dsk rt dip = dsk,
  f rt dip = f,
  hops rt dip = hops,
  nhip rt dip = nhip,
  pre rt dip = pre" 
  by (simp add: sqn_def proj5_eq_dhops [symmetric] proj4_eq_flag [symmetric])
  from ⟨update_arg_wf r⟩
  have "(dsn, dsk, f, hops, nhip, pre) ⊑ the (update rt dip r) dip)"
  proof (rule wf_r_cases)
    fix nhip pre
    from ⟨hops ≥ 1⟩
    have "\pre'. (dsn, dsk, f, hops, nhip, pre) ⊑ (dsn, unk, val, Suc 0, nhip, pre')"
    unfolding fresher_def sqn_def by (cases f) auto
    thus "(dsn, dsk, f, hops, nhip, pre) ⊑ the (update rt dip (0, unk, val, Suc 0, nhip, pre) dip)"
    using ⟨dip ∈ kD(rt)⟩ by - (rule update_cases_kD, simp_all)
  next
  fix dsn :: sqn and hops nhip pre
  assume "0 < dsn"
  show "(dsn, dsk, f, hops, nhip, pre) ⊑ the (update rt dip (dsn, kno, val, hops, nhip, pre) dip)"
  proof (rule update_cases_kD [OF _ ⟨dip ∈ kD(rt)⟩], simp_all add: '0 < dsn)
    assume "dsn < dsn"
    thus "(dsn, dsk, f, hops, nhip, pre) ⊑ (dsn, kno, val, hops, nhip, pre)"
    unfolding fresher_def by auto
  next
  assume "dsn = dsn"
  and "hops < hops"
  thus "(dsn, dsk, f, hops, nhip, pre) ⊑ (dsn, kno, val, hops, nhip, pre)"
unfolding fresher_def nsqn_r_def by simp

next
assume "dsn_n = dsn"
with (0 < dsn)
show "(dsn, dsk_n, inv, hops_n, neph_n, pre_n)
\subseteq (dsn, kno, val, hops, neph, pre \cup pre_n)"
unfolding fresher_def by simp

qed

hence "rt \subseteq dip update rt dip r"
by - (rule single_rt_fresher, simp)
with ⟨dip = ip⟩ show ?thesis
by simp

qed

theorem rt_fresher_invalidate [simp]:
assumes "dip \in kD(rt)"
and indests: "\forall rip \in \domain(dests). rip \in vD(rt) \land sqn rt rip < \text{the (dests rip)}"
shows "rt \subseteq dip invalidate rt dests"
proof (cases "dip \in \domain(dests)")
assume "dip \in \domain(dests)"
thus ?thesis using ⟨dip \in kD(rt)⟩
by - (rule single_rt_fresher, simp)

next
assume "dip \in \domain(dests)"
moreover with indests have "dip \in vD(rt)"
and "sqn rt dip < \text{the (dests dip)}"
by auto
ultimately show ?thesis
using assms unfolding invalidate_def sqn_def
by - (rule single_rt_fresher, auto simp: fresher_def)

qed

lemma nsqn_r_invalidate [simp]:
assumes "dip \in kD(rt)"
and "dip \in \domain(dests)"
shows "nsqn_r (\text{the (invalidate rt dests dip)}) = \text{the (dests dip)} - 1"
using assms unfolding invalidate_def by auto

lemma rt_fresh_as_inc_invalidate [simp]:
assumes "dip \in kD(rt)"
and "\forall rip \in \domain(dests). rip \in vD(rt) \land \text{the (dests rip)} = \text{inc (sqn rt rip)}"
shows "rt \approx dip invalidate rt dests"
proof (cases "dip \in \domain(dests)")
assume "dip \in \domain(dests)"
with ⟨dip \in kD(rt)⟩ have "dip \in kD(invalidate rt dests)"
by simp
ultimately show ?thesis
proof (rule nsqn_rt_fresh_asI)
from ⟨dip \in vD(rt)⟩ have "nsqn rt dip = sqn rt dip" by simp
also have "sqn rt dip = nsqn_r (\text{the (invalidate rt dests dip)})"
proof -
from ⟨dip \in kD(rt)⟩ have "nsqn_r (\text{the (invalidate rt dests dip)}) = \text{the (dests dip)} - 1" 
using ⟨dip \in kD(dests)⟩ by (rule nsqn_r_invalidate)
with ⟨the (dests dip) = \text{inc (sqn rt dip)}⟩
show "sqn rt dip = nsqn_r (\text{the (invalidate rt dests dip)})" by simp
qed
also from \( dip \in \text{kd}(\text{invalidate rt dests}) \)

have "\( \text{nsqn}_r \ (\text{the} \ (\text{invalidate rt dests} \ dip)) = \text{nsqn} \ (\text{invalidate rt dests} \ dip) \)"

by (simp add: kd_nsqn)

finally show "\( \text{nsqn rt dip} = \text{nsqn} \ (\text{invalidate rt dests} \ dip) \)" .

qed simp

lemmas rt_fresher_inc_invalidate [simp] = rt_fresh_as_inc_invalidate [THEN rt_fresh_asD1]

lemma rt_fresh_as_addpreRT [simp]:

assumes "ip \in \text{kd}(rt)"

shows "rt \approx dip \ (\text{addpreRT} \ rt \ ip \ npre)"

using assms [THEN kd_Some] by (auto simp: addpreRT_def)

lemmas rt_fresher_addpreRT [simp] = rt_fresh_as_addpreRT [THEN rt_fresh_asD1]

1.5.4 Strictly comparing routing tables

definition rt_strictly_fresher :: "ip \Rightarrow rt \Rightarrow rt \Rightarrow bool"

where

"rt\_strictly\_fresher \equiv \lambda dip \ rt1 \ rt2. \ (rt1 \sqsubseteq dip \ rt2) \land \neg (rt1 \approx dip \ rt2)"

abbreviation

rt_strictly_fresher_syn :: "rt \Rightarrow ip \Rightarrow rt \Rightarrow bool" ("(_/ \sqsubseteq_/ _)" [51, 999, 51] 50)

where

"rt1 \sqsubseteq i \ rt2 \equiv rt\_strictly\_fresher \ i \ rt1 \ rt2"

lemma rt_strictly_fresher_def'':

"rt1 \sqsubseteq i \ rt2 = ((rt1 \sqsubseteq i \ rt2) \land \neg (rt2 \sqsubseteq i \ rt1))"

unfolding rt_strictly_fresher_def'' by auto

lemma rt_strictly_fresherI' [intro]:

assumes "rt1 \sqsubseteq i \ rt2"

and "\neg (rt2 \sqsubseteq i \ rt1)"

shows "rt1 \sqsubseteq i \ rt2"

using assms unfolding rt_strictly_fresher_def'' by simp

lemma rt_strictly_fresherE' [elim]:

assumes "rt1 \sqsubseteq i \ rt2"

and "[ rt1 \sqsubseteq i \ rt2; \neg (rt2 \sqsubseteq i \ rt1) ] \Longrightarrow P \ rt1 \ rt2 \ i"

shows "P \ rt1 \ rt2 \ i"

using assms unfolding rt_strictly_fresher_def'' by simp

lemma rt_strictly_fresherI [intro]:

assumes "rt1 \sqsubseteq i \ rt2"

and "\neg (rt1 \approx i \ rt2)"

shows "rt1 \sqsubseteq i \ rt2"

unfolding rt_strictly_fresher_def using assms ..

lemmas rt_strictly_fresher_singleI [elim] = rt_strictly_fresherI [OF single_rt_fresher]

lemma rt_strictly_fresherE [elim]:

assumes "rt1 \sqsubseteq i \ rt2"

and "[ rt1 \sqsubseteq i \ rt2; \neg (rt1 \approx i \ rt2) ] \Longrightarrow P \ rt1 \ rt2 \ i"

shows "P \ rt1 \ rt2 \ i"

using assms(1) unfolding rt_strictly_fresher_def

by rule (erule(1) assms(2))

lemma rt_strictly_fresher_def':

"rt1 \sqsubseteq i \ rt2 =

(\text{nsqn}_r \ (\text{the} \ (rt1 \ i)) < \text{nsqn}_r \ (\text{the} \ (rt2 \ i)) \ 
\lor \ (\text{nsqn}_r \ (\text{the} \ (rt1 \ i)) = \text{nsqn}_r \ (\text{the} \ (rt2 \ i)) \land \pi_5(\text{the} \ (rt1 \ i)) > \pi_5(\text{the} \ (rt2 \ i))))""

unfolding rt_strictly_fresher_def'' rt_fresher_def by auto
lemma rt_strictly_fresher_fresherD [dest]:
  assumes "rt1 ⊏ dip rt2"
  shows "the (rt1 dip) ⊑ the (rt2 dip)"
  using assms unfolding rt_strictly_fresher_def rt_fresher_def by auto

lemma rt_strictly_fresher_not_fresh_asD [dest]:
  assumes "rt1 ⊏ dip rt2"
  shows "¬ rt1 ≈ dip rt2"
  using assms unfolding rt_strictly_fresher_def by auto

lemma rt_strictly_fresher_trans [elim, trans]:
  assumes "rt1 ⊏ dip rt2"
  and "rt2 ⊑ dip rt3"
  shows "rt1 ⊏ dip rt3"
  using assms proof -
    from ⟨rt1 ⊏ dip rt2⟩ obtain "the (rt1 dip) ⊑ the (rt2 dip)" by auto
    also from ⟨rt2 ⊑ dip rt3⟩ obtain "the (rt2 dip) ⊑ the (rt3 dip)" by auto
    finally have "the (rt1 dip) ⊑ the (rt3 dip)".
    moreover have "¬ (rt1 ≈ dip rt3)"
    proof -
      from ⟨rt1 ⊏ dip rt2⟩ obtain "¬ (the (rt2 dip) ⊑ the (rt1 dip))" by auto
      also from ⟨rt2 ⊑ dip rt3⟩ obtain "¬ (the (rt3 dip) ⊑ the (rt2 dip))" by auto
      finally have "¬ (the (rt3 dip) ⊑ the (rt1 dip))".
      thus ?thesis ..
    qed
    ultimately show "rt1 ⊏ dip rt3" ..
    qed

lemma rt_strictly_fresher_irefl [simp]: "¬ (rt ⊏ dip rt)"
  using assms unfolding rt_strictly_fresher_def by clarsimp

lemma rt_fresher_trans_rt_strictly_fresher [elimsimp]
  assumes "rt1 ⊑ dip rt2"
  and "rt2 ⊏ dip rt3"
  shows "rt1 ⊏ dip rt3"
  proof -
    from ⟨rt2 ⊏ dip rt3⟩ have "¬ (rt3 ⊑ dip rt1)"
    unfolding rt_strictly_fresher_def'' by auto
    from this(1) and ⟨rt2 ⊑ dip rt3⟩ have "rt1 ⊏ dip rt3" ..
    moreover from ⟨¬ (rt2 ⊏ dip rt1)⟩ have "¬ (rt3 ⊑ dip rt1)"
    proof (rule contrapos_nn)
      assume "rt3 ⊏ dip rt1"
      with ⟨rt2 ⊑ dip rt3⟩ show "rt2 ⊏ dip rt1" ..
      qed
    ultimately show "rt1 ⊏ dip rt3"
    unfolding rt_strictly_fresher_def'' by auto
    qed

lemma rt_fresher_trans_rt_strictly_fresher' [elim, trans]:
  assumes "rt1 ⊑ dip rt2"
  and "rt2 ⊏ dip rt3"
  shows "rt1 ⊏ dip rt3"
  proof -
    from ⟨rt2 ⊏ dip rt3⟩ have "¬ (rt3 ⊏ dip rt1)"
    unfolding rt_strictly_fresher_def'' by auto
    from ⟨rt1 ⊏ dip rt2⟩ and this(1) have "rt1 ⊏ dip rt3" ..
    moreover from ⟨¬ (rt3 ⊏ dip rt1)⟩ have "¬ (rt3 ⊏ dip rt1)"

124
proof (rule contrapos_nn)
  assume "rt3 ⊑ dip rt1"
  thus "rt3 ⊑ dip rt2" using (rt1 ⊑ dip rt2) ..
qed

ultimately show "rt1 ⊑ dip rt3"
  unfolding rt_strictly_fresher_def'' by auto
qed

lemma rt_fresher_imp_nsqn_le:
  assumes "rt1 ⊑ ip rt2"
  and "ip ∈ kD rt1"
  and "ip ∈ kD rt2"
  shows "nsqn rt1 ip ≤ nsqn rt2 ip"
using assms(1)
  by (auto simp add: rt_fresher_def2 [OF assms(2-3)])

lemma rt_strictly_fresher_ltI [intro]:
  assumes "dip ∈ kD(rt1)"
  and "dip ∈ kD(rt2)"
  and "nsqn rt1 dip < nsqn rt2 dip"
  shows "rt1 ⊑ dip rt2"
proof
  from assms show "rt1 ⊑ dip rt2" ..
next
  show "¬ (rt1 ≈ dip rt2)"
  proof
    assume "rt1 ≈ dip rt2"
    hence "rt2 ⊑ dip rt1" ..
    hence "nsqn rt2 dip ≤ nsqn rt1 dip"
      using ⟨dip ∈ kD(rt2)⟩ ⟨dip ∈ kD(rt1)⟩
      by (rule rt_fresher_imp_nsqn_le)
    with ⟨nsqn rt1 dip < nsqn rt2 dip⟩ show "False"
      by simp
  qed
qed

lemma rt_strictly_fresher_eqI [intro]:
  assumes "i ∈ kD(rt1)"
  and "i ∈ kD(rt2)"
  and "nsqn rt1 i = nsqn rt2 i"
  and "π₅(the (rt2 i)) < π₅(the (rt1 i))"
  shows "rt1 ⊏ i rt2"
using assms unfolding rt_strictly_fresher_def' by (auto simp add: kD_nsqn)

lemma invalidate_rtsf_left [simp]:
  "⋀ dests dip rt rt'. dests dip = None ⟹ (invalidate rt dests ⊑ dip rt') = (rt ⊑ dip rt')"
unfolding invalidate_def rt_strictly_fresher_def'
  by (rule iffI) (auto split: option.split_asm)

lemma vD_invalidate_rt_strictly_fresher [simp]:
  assumes "dip ∈ vD(invalidate rt1 dests)"
  shows "(invalidate rt1 dests ⊑ dip rt') = (rt ⊑ dip rt')"
proof (cases "dip ∈ dom(dests)")
  assume "dip ∈ dom(dests)"
  hence "dip ∉ vD(invalidate rt1 dests)"
    unfolding invalidate_def vD_def
    by clarsimp (metis assms option.simps(3) vD_invalidate_vD_not_dests)
  with ⟨dip ∈ vD(invalidate rt1 dests)⟩ show ?thesis by simp
next
  assume "dip ∉ dom(dests)"
  hence "dests dip = None" by auto
  moreover with ⟨dip ∈ vD(invalidate rt1 dests)⟩ have "dip ∈ vD(rt1)"
    unfolding invalidate_def vD_def
ultimately show \( \forall \text{thesis} \)
unfolding invalidate_def rt_strictly_fresher_def' by auto

lemma rt_strictly_fresher_update_other [elim!]:
"\( \forall \text{ dip ip rt rt' } [ \text{ dip } \neq \text{ ip; rt } \sqsubseteq_{\text{ dip}} \text{ rt'} ] \implies \text{ update rt ip r } \sqsubseteq_{\text{ dip}} \text{ rt'} \)"
unfolding rt_strictly_fresher_def' by clarsimp

lemma addpreRT_strictly_fresher [simp]:
assumes "\( \text{ dip } \in \text{ kD(rt)} \)"
shows "\( (\text{ the (addpreRT rt dip npre)} ) \sqsubseteq_{\text{ ip}} \text{ rt2} ) = (\text{ rt } \sqsubseteq_{\text{ ip}} \text{ rt2}) \)"
using assms unfolding rt_strictly_fresher_def' by clarsimp

lemma lt_sqn_imp_update_strictly_fresher:
assumes "\( \text{ dip } \in \text{ vD } \text{ (rt2 nhip)} \)" and *: "\( \text{ osn } < \text{ sqn } \text{ (rt2 nhip)} \) dip" and **: "\( \text{ rt } \neq \text{ update rt dip (osn, kno, val, hops, nhip, {}} \)"
shows "\( \text{ update rt dip (osn, kno, val, hops, nhip, {}} ) \sqsubseteq_{\text{ dip}} \text{ rt2 nhip} \)"
unfolding rt_strictly_fresher_def' proof (rule disjI1)
from ** have "\( \text{ nsqn } \text{ (update rt dip (osn, kno, val, hops, nhip, {}} ) \text{ dip } = \text{ osn} \)"
  by (rule nsqn_update_changed_kno_val)
  with (\( \text{ dip } \in \text{ vD } \text{ (rt2 nhip)} \))
  have "\( \text{ nsqn } \text{ (the (update rt dip (osn, kno, val, hops, nhip, {}} ) \text{ dip}) = \text{ osn} \)"
    by (simp add: kD_nsqn)
  also have "\( \text{ osn } < \text{ sqn } \text{ (rt2 nhip)} \) dip" by (rule *)
  also have "\( \text{ nsqn } \text{ (rt2 nhip)} \text{ dip } = \text{ nsqn } \text{ (the (rt2 nhip dip)}) \)"
    unfolding nsqn_def using (\( \text{ dip } \in \text{ vD } \text{ (rt2 nhip)} \))
    by - (metis vD_flag_val proj2_eq_sqn proj4_eq_flag vD_iD_gives_kD(1))
  finally show "\( \text{ nsqn } \text{ (the (update rt dip (osn, kno, val, hops, nhip, {}} ) \text{ dip}) } < \text{ nsqn } \text{ (the (rt2 nhip dip))} \) .
qed

lemma dhops_le_hops_imp_update_strictly_fresher:
assumes "\( \text{ dip } \in \text{ vD } \text{ (rt2 nhip)} \)" and sqn: "\( \text{ sqn } \text{ (rt2 nhip)} \text{ dip } = \text{ osn} \)" and hop: "\( \text{ the (dhops (rt2 nhip) dip } \leq \text{ hops} \)"
and **: "\( \text{ rt } \neq \text{ update rt dip (osn, kno, val, Suc hops, nhip, {}} \)"
shows "\( \text{ update rt dip (osn, kno, val, Suc hops, nhip, {}} ) \sqsubseteq_{\text{ dip}} \text{ rt2 nhip} \)"
unfolding rt_strictly_fresher_def' proof (rule disjI2, rule conjI)
from ** have "\( \text{ nsqn } \text{ (update rt dip (osn, kno, val, Suc hops, nhip, {}} ) \text{ dip } = \text{ osn} \)"
  by (rule nsqn_update_changed_kno_val)
  with (\( \text{ dip } \in \text{ vD } \text{ (rt2 nhip)} \))
  have "\( \text{ nsqn } \text{ (the (update rt dip (osn, kno, val, Suc hops, nhip, {}} ) \text{ dip}) } = \text{ osn} \)"
    by (simp add: kD_nsqn)
  also have "\( \text{ osn } = \text{ sqn } \text{ (rt2 nhip)} \) dip" by (rule sqn [symmetric])
  also have "\( \text{ nsqn } \text{ (rt2 nhip)} \text{ dip } = \text{ nsqn } \text{ (the (rt2 nhip dip)}) \)"
    unfolding nsqn_def using (\( \text{ dip } \in \text{ vD } \text{ (rt2 nhip)} \))
    by - (metis vD_flag_val proj2_eq_sqn proj4_eq_flag vD_iD_gives_kD(1))
  finally have "\( \text{ the (dhops (rt2 nhip) dip } < \text{ the (dhops (update rt dip (osn, kno, val, Suc hops, nhip, {}} ) \text{ dip})} \) .
next
have "\( \text{ the (dhops (rt2 nhip) dip } \leq \text{ hops} \)" by (rule hop)
also have "\( \text{ hops } < \text{ hops } + 1 \)" by simp
also have "\( \text{ hops } + 1 = \text{ the (dhops (update rt dip (osn, kno, val, Suc hops, nhip, {}} ) \text{ dip})} \)"
  using ** by simp
finally have "\( \text{ the (dhops (rt2 nhip) dip } < \text{ the (dhops (update rt dip (osn, kno, val, Suc hops, nhip, {}} ) \text{ dip})} \) .
thus "\( \pi_5 \text{ (the (rt2 nhip dip)) } < \pi_5 \text{ (the (update rt dip (osn, kno, val, Suc hops, nhip, {}} ) \text{ dip})} \)"
  using (\( \text{ dip } \in \text{ vD } \text{ (rt2 nhip)} \)) by (simp add: proj5_eq_dhops)
qed
lemma nsqn_invalidate:
  assumes "dip ∈ kD(rt)"
  and "∀ ip∈dom(dests). ip ∈ vD(rt) ∧ the (dests ip) = inc (sqn rt ip)"
  shows "nsqn (invalidate rt dests) dip = nsqn rt dip"
proof -
  from ⟨dip ∈ kD(rt)⟩ have "dip ∈ kD(invalidate rt dests)" by simp
from assms have "rt ≈ dip invalidate rt dests"
  by (rule rt_fresh_as_inc_invalidate)
  with ⟨dip ∈ kD(rt)⟩ ⟨dip ∈ kD(invalidate rt dests)⟩
  show ?thesis by (simp add: kD_nsqn del: invalidate_kD_inv)
  (erule(2) rt_fresh_as_nsqnr)
qed

1.6 Invariant proofs on individual processes

theory A_Seq_Invariants
imports ../../../AWN/Invariants A_Aodv A_Aodv_Data A_Aodv_Predicates A_Fresher
begin
The proposition numbers are taken from the December 2013 version of the Fehnker et al technical report.

Proposition 7.2

lemma sequence_number_increases:
  "paodv i ||= A onll Γ_AODV (λ((ξ, _), _, (ξ', _)). sn ξ ≤ sn ξ')"
by inv_cterms

lemma sequence_number_one_or_bigger:
  "paodv i ||= onl Γ_AODV (λ(ξ, _). 1 ≤ sn ξ)"
by (rule onll_step_to_invariantI [OF sequence_number_increases])
(auto simp: σ_AODV_def)

We can get rid of the onl/onll if desired...

lemma sequence_number_increases':
  "paodv i ||= A (λ((ξ, _), _, (ξ', _)). sn ξ ≤ sn ξ')"
by (rule step_invariant_weakenE [OF sequence_number_increases])
(auto dest!: onllD)

lemma sequence_number_one_or_bigger':
  "paodv i ||= (λ(ξ, _). 1 ≤ sn ξ)"
by (rule invariant_weakenE [OF sequence_number_one_or_bigger]) auto

lemma sip_in_kD:
  "paodv i ||= onl Γ_AODV (λ(ξ, l). l ∈ {{PAodv-:7} ∪ {PAodv-:5} ∪ {PRrep-:0..PRrep-:1} 
  ∪ {PRreq-:0..PRreq-:3}} → sip ξ ∈ kD (rt ξ))"
by inv_cterms

lemma rrep_1_update_changes:
  "paodv i ||= onl Γ_AODV (λ(ξ, l). l = PRrep-:1 → 
  rt ξ ≠ update (rt ξ) (dip) (dsn ξ, kno, val, hops ξ + 1, sip ξ, {})))"
by inv_cterms

lemma addpreRT_partly_welldefined:
  "paodv i ||= onl Γ_AODV (λ(ξ, l). l ∈ {{PRreq-:16..PRreq-:18} ∪ {PRrep-:2..PRrep-:6} → dip ξ ∈ kD (rt ξ)) 
  ∧ (l ∈ {PRreq-:3..PRreq-:17} → oip ξ ∈ kD (rt ξ)))"
by inv_cterms

Proposition 7.38

lemma includes_nhip:
  "paodv i ||= onl Γ_AODV (λ(ξ, l). ∀ dip∈kD(rt ξ). the (nhop (rt ξ) dip)∈kD(rt ξ))"
proof -
{ fix ip and ξ ξ' :: state
  assume "∀ dip ∈ kD (rt ξ). the (nhop (rt ξ) dip) ∈ kD (rt ξ)"
  and "ξ' = ξ(rt := update (rt ξ) ip (0, unk, val, Suc 0, ip, {}))"
  hence "∀ dip ∈ kD (rt ξ).
    the (nhop (update (rt ξ) ip (0, unk, val, Suc 0, ip, {})) dip) = ip
    ∨ the (nhop (update (rt ξ) ip (0, unk, val, Suc 0, ip, {})) dip) ∈ kD (rt ξ)"
    by clarsimp (metis nhop_update_unk_val update_another)
}

note one_hop = this
{ fix ip sip sn hops and ξ ξ' :: state
  assume "∀ dip ∈ kD (rt ξ). the (nhop (rt ξ) dip) ∈ kD (rt ξ)"
  and "ξ' = ξ(rt := update (rt ξ) ip (sn, kno, val, Suc hops, sip, {}))"
  and "sip ∈ kD (rt ξ)"
  hence "(the (nhop (update (rt ξ) ip (sn, kno, val, Suc hops, sip, {})) ip) = ip
    ∨ the (nhop (update (rt ξ) ip (sn, kno, val, Suc hops, sip, {})) ip) ∈ kD (rt ξ))"
    ∧ "∀ dip ∈ kD (rt ξ).
      the (nhop (update (rt ξ) ip (sn, kno, val, Suc hops, sip, {})) dip) = ip
      ∨ the (nhop (update (rt ξ) ip (sn, kno, val, Suc hops, sip, {})) dip) ∈ kD (rt ξ)"
    by (metis kD_update_unchanged nhop_update_changed update_another)
}

note nhip_is_sip = this


Proposition 7.22: needed in Proposition 7.4

lemma addpreRT_welldefined:
"paodv i ||= onl Γ AODV (λ((ξ, p), (_, _)). kD (rt ξ) ⊆ kD (rt ξ'))"
(is "-_ ||= onl Γ AODV ?P")
unfolding invariant_def
proof
  fix s
  assume "s ∈ reachable (paodv i) TT"
  then obtain ξ p where "s = ((ξ, p), (_, _))"
  and "(ξ, p) ∈ reachable (paodv i) TT"
  by (metis PairE)
  have "onl Γ AODV ?P ((ξ, p), (_, _))"
  proof (rule onlI)
    fix onll
    fix 1
    assume "1 ∈ labels Γ AODV p"
    with ((ξ, p), (_, _)) ∈ reachable (paodv i) TT
    have I1: "1 ∈ {PRreq::16..PRreq::18} ||= dip ξ ∈ kD (rt ξ)"
    and I2: "1 = PRreq::17 ||= oip ξ ∈ kD (rt ξ)"
    and I3: "1 = PRreq::5 ||= dip ξ ∈ kD (rt ξ)"
    and I4: "1 = PRreq::6 ||= (the (nhop (rt ξ) (dip ξ))) ∈ kD (rt ξ)"
    by (auto dest!: invariantD [OF addpreRT_partly_welldefined])
    moreover from ((ξ, p), (_, _)) ∈ reachable (paodv i) TT 1 ∈ labels Γ AODV p and I3
    have "1 = PRreq::6 ||= (the (nhop (rt ξ) (dip ξ))) ∈ kD (rt ξ)"
    by (auto dest!: invariantD [OF includes_nhip])
    ultimately show "?P ((ξ, 1), (_, _))"
    by simp
    with (s = ((ξ, p), (_, _))) show "onl Γ AODV ?P s"
    by simp
  qed

Proposition 7.4

lemma known_destinations_increase:
"paodv i ||= onll Γ AODV (λ((ξ, _, _), (ξ', _)). kD (rt ξ) ⊇ kD (rt ξ'))"
by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf addpreRT_partly_welldefined] simp add: subset_insertI)
Proposition 7.5

```
lemma rreqs_increase:  
  "paodv i \|= \Gamma_{AODV} (\lambda((\xi, _), _, (\xi', _)). \text{rreqs} \xi \subseteq \text{rreqs} \xi')" 
  by (inv_cterms simp add: subset_insertI)
```

```
lemma dests_bigger_than_sqn: 
  "paodv i \|= onl \Gamma_{AODV} (\lambda((\xi, _), _, (\xi', _)). \text{rreqs} \xi \subseteq \text{rreqs} \xi')" 
  proof - 
    have sqninv: 
      "\forall \text{dests rt rsn ip.} 
        [ \forall ip \in \text{dom(dests)}. ip \in \text{kD}(rt) \land \text{sqn}(rt \xi) ip \leq \text{the}(\text{dests} \xi ip)] 
        \Rightarrow \text{sqn}(\text{invalidate}(rt \xi) \text{dests}) ip \leq \text{rsn}"
      by (rule sqn_invalidate_in_dests [THEN eq_imp_le], assumption) auto
    have indests: 
      "\forall \text{dests rt rsn ip.} 
        [ \forall ip \in \text{dom(dests)}. ip \in \text{kD}(rt) \land \text{sqn}(rt \xi) ip \leq \text{the}(\text{dests} \xi ip)] 
        \Rightarrow ip \in \text{kD}(rt) \land \text{sqn}(rt \xi) ip \leq \text{rsn}"
      by (metis domI option.sel)
    show ?thesis 
      by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf addpreRT_welldefined] 
        onl_invariant_sterms [OF aodv_wf dests_bigger_than_sqn] 
        simp add: sqninv indests+)
  qed
```

Proposition 7.6

```
lemma sqns_increase: 
  "paodv i \|= onl \Gamma_{AODV} (\lambda((\xi, _), _, (\xi', _)). \text{sqn}(rt \xi) ip \leq \text{sqn}(rt \xi') ip)" 
  proof - 
    { 
      fix \xi :: state 
      assume *: "\forall ip \in \text{dom(dests} \xi). ip \in \text{kD}(rt) \land \text{sqn}(rt \xi) ip \leq \text{the}(\text{dests} \xi ip)"
      have "\forall ip. \text{sqn}(rt \xi) ip \leq \text{sqn}(\text{invalidate}(rt \xi) (\text{dests} \xi ip))" 
        proof 
          fix ip 
          from * have "ip \in \text{dom(dests} \xi) \lor \text{sqn}(rt \xi) ip \leq \text{the}(\text{dests} \xi ip)" by simp 
          thus "\text{sqn}(rt \xi) ip \leq \text{sqn}(\text{invalidate}(rt \xi) (\text{dests} \xi) ip)"
            by (metis domI invalidate_sqn option.sel)
        qed 
    } 
    note solve_invalidate = this 
    show ?thesis 
      by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf addpreRT_welldefined] 
        onl_invariant_sterms [OF aodv_wf dests_bigger_than_sqn] 
        simp add: solve_invalidate)
  qed
```

Proposition 7.7

```
lemma ip_constant: 
  "paodv i \|= onl \Gamma_{AODV} (\lambda(\xi, _). ip \xi = i)"
  by (inv_cterms simp add: \sigma_{AODV}_def)
```

Proposition 7.8

```
lemma sender_ip_valid': 
  "paodv i \|= onl \Gamma_{AODV} (\lambda((\xi, _), a, _). \text{anycast}(\lambda m. \text{not_Pkt} m \rightarrow \text{msg_sender} m = ip \xi) a)"
  by inv_cterms
```

```
lemma sender_ip_valid: 
  "paodv i \|= onl \Gamma_{AODV} (\lambda((\xi, _), a, _). \text{anycast}(\lambda m. \text{not_Pkt} m \rightarrow \text{msg_sender} m = i) a)"
  by (rule step_invariant_weaken_with_invariantE [OF ip_constant sender_ip_valid'])
```

lemma received_msg_inv:
  "paodv i |= (recvmsg P -->) onl \( \Gamma_{AODV}(\lambda(\xi, 1). l \in \{PAodv--:1\} \rightarrow P \text{ msg } \xi)\)"
by inv_cterms

Proposition 7.9

lemma sip_not_ip':
  "paodv i |= (recvmsg (\lambda m. not_Pkt m --> msg_sender m \neq i) -->) onl \( \Gamma_{AODV}(\lambda(\xi, _). \text{ sip } \xi \neq \text{ ip } \xi)\)"
by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]
onl_invariant_sterms [OF aodv_wf ip_constant [THEN invariant_restrict_inD]]) simp add: clear_locals_sip_not_ip' clarsimp+

lemma sip_not_ip:
  "paodv i |= (recvmsg (\lambda m. not_Pkt m --> msg_sender m \neq i) -->) onl \( \Gamma_{AODV}(\lambda(\xi, _). \text{ sip } \xi \neq i)\)"
by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]
onl_invariant_sterms [OF aodv_wf ip_constant [THEN invariant_restrict_inD]]) simp add: clear_locals_sip_not_ip' clarsimp+

Neither sip_not_ip' nor sip_not_ip is needed to show loop freedom.

Proposition 7.10

lemma hop_count_positive:
  "paodv i |= onl \( \Gamma_{AODV}(\lambda(\xi, _). \forall \Gamma \in kD (rt \xi). \text{ the (dhops (rt \xi) ip) } \geq 1)\)"
by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf addpreRT_welldefined]) auto

lemma rreq_dip_in_vD_dip_eq_ip:
  "paodv i |= onl \( \Gamma_{AODV}(\lambda(\xi, 1). l \in \{PRreq--:16..PRreq--:18\} \rightarrow \text{ dip } \xi \in vD(rt \xi))\)\n\wedge (l \in \{PRreq--:5, PRreq--:6\} \rightarrow \text{ dip } \xi = \text{ ip } \xi)\n\wedge (l \in \{PRreq--:15..PRreq--:18\} \rightarrow \text{ dip } \xi \neq \text{ ip } \xi))"

proof (inv_cterms, elim conjE)
fix l \xi pp p'
assume "((\xi, pp) \in \text{ reachable } (paodv i) TT"
  and "\{PRreq--:17\}(\lambda \xi. \xi[rt := \text{ the (addpreRT (rt \xi) (oip \xi) \{the (nhop (rt \xi) (dip \xi)\})\}))]\ p' \in \text{ sterms } \Gamma_{AODV} pp"
  and "l = PRreq--:17"
  and "\text{ dip } \xi \in vD (rt \xi)"

from this(1-3) have "oip \xi \in kD (rt \xi)"
by (auto dest: onl_invariant_sterms [OF aodv_wf addpreRT_welldefined, where l="PRreq--:17"])

with \text{ dip } \xi \in vD (rt \xi)\n
show "\text{ dip } \xi \in vD (the (addpreRT (rt \xi) (oip \xi) \{the (nhop (rt \xi) (dip \xi)\})))" by simp
qed

Proposition 7.11

lemma anycast_msg_zhops:
"A rreqid dip dsn ds k oip osn sip.
paodv i |=A onl \( \Gamma_{AODV}(\lambda(_, a, _). \text{ anycast msg_zhops a})\)"

proof (inv_cterms inv add:
onl_invariant_sterms [OF aodv_wf rreq_dip_in_vD_dip_eq_ip [THEN invariant_restrict_inD]]
onl_invariant_sterms [OF aodv_wf hop_count_positive [THEN invariant_restrict_inD]], elim conjE)
fix l \xi a pp p' pp'
assume "((\xi, pp) \in \text{ reachable } (paodv i) TT"
  and "\{PRreq--:18\}unicast(\lambda. \text{ the (nhop (rt \xi) (oip \xi)),}
  \lambda. \text{ Rrep (the (dhops (rt \xi) (dip \xi))) (dip \xi) (sqn (rt \xi) (dip \xi)) (oip \xi) (ip \xi)).}
  p' > pp' \in \text{ sterms } \Gamma_{AODV} pp"
  and "l = PRreq--:18"
  and "a = unicast (the (nhop (rt \xi) (oip \xi))
(Rrep (the (dhops (rt \xi) (dip \xi))) (dip \xi) (sqn (rt \xi) (dip \xi)) (oip \xi) (ip \xi)))"
  and *: "\forall \text{ ip } \in kD (rt \xi). Suc 0 \leq \text{ the (dhops (rt \xi) ip})"
  and "\text{ dip } \xi \in vD (rt \xi)"

from \text{ dip } \xi \in vD (rt \xi)\ have "\text{ dip } \xi \in kD (rt \xi)"
by (rule vD_iD_gives_kD(1))

with * have "Suc 0 \leq \text{ the (dhops (rt \xi) (dip \xi))}" ..
thus "0 < the (dhops (rt ξ) (dip ξ))" by simp
qed

lemma hop_count_zero_oip_dip_sip:
"paodv i ⊨ (recvmsg msg_zhops →) onl Γ_AODV (λ(ξ, 1).
  (l∈{PAodv-:4..PAodv-:5} ∪ {PRreq-n|n. True} →
    (hops ξ = 0 → oip ξ = sip ξ))
  ∧
  ((l∈{PAodv-:6..PAodv-:7} ∪ {PRrep-n|n. True} →
    (hops ξ = 0 → dip ξ = sip ξ)))"
by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]) auto

lemma osn_rreq:
"paodv i ⊨ (recvmsg rreq_rrep_sn →) onl Γ_AODV (λ(ξ, 1).
  l ∈ {PAodv-:4, PAodv-:5} ∪ {PRreq-n|n. True} → 1 ≤ osn ξ)
by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]) clarsimp

lemma osn_rreq':
"paodv i ⊨ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) →) onl Γ_AODV (λ(ξ, 1).
  l ∈ {PAodv-:4, PAodv-:5} ∪ {PRreq-n|n. True} → 1 ≤ osn ξ)
proof (rule invariant_weakenE [OF osn_rreq])
fix a
assume "recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) a"
thus "recvmsg rreq_rrep_sn a"
  by (cases a) simp_all
qed

lemma dsn_rrep:
"paodv i ⊨ (recvmsg rreq_rrep_sn →) onl Γ_AODV (λ(ξ, 1).
  l ∈ {PAodv-:6, PAodv-:7} ∪ {PRrep-n|n. True} → 1 ≤ dsn ξ)
by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]) clarsimp

lemma dsn_rrep':
"paodv i ⊨ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) →) onl Γ_AODV (λ(ξ, 1).
  l ∈ {PAodv-:6, PAodv-:7} ∪ {PRrep-n|n. True} → 1 ≤ dsn ξ)
proof (rule invariant_weakenE [OF dsn_rrep])
fix a
assume "recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) a"
thus "recvmsg rreq_rrep_sn a"
  by (cases a) simp_all
qed

lemma hop_count_zero_oip_dip_sip':
"paodv i ⊨ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) →) onl Γ_AODV (λ(ξ, 1).
  (l∈{PAodv-:4..PAodv-:5} ∪ {PRreq-n|n. True} →
    (hops ξ = 0 → oip ξ = sip ξ))
  ∧
  ((l∈{PAodv-:6..PAodv-:7} ∪ {PRrep-n|n. True} →
    (hops ξ = 0 → dip ξ = sip ξ)))"
proof (rule invariant_weakenE [OF hop_count_zero_oip_dip_sip])
fix a
assume "recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) a"
thus "recvmsg msg_zhops a"
  by (cases a) simp_all
qed

Proposition 7.12

lemma zero_seq_unk_hops_one':
"paodv i ⊨ (recvmsg rreq_rrep_sn →) onl Γ_AODV (λ(ξ, _).
  (∀ dip∈kD(rt ξ). (sqn (rt ξ) dip = 0 → sqnf (rt ξ) dip = unk)
   ∧ (sqnf (rt ξ) dip = unk → the (dhops (rt ξ) dip) = 1)
   ∧ (the (dhops (rt ξ) dip) = 1 → the (nhop (rt ξ) dip) = dip))"
proof -
{ fix dip and ξ :: state and P
assume "sqn (invalidate (rt ξ) (dests ξ)) dip = 0"
and all: "∀ip. sqn (rt ξ) ip ≤ sqn (invalidate (rt ξ) (dests ξ)) ip"
and #: "sqn (rt ξ) dip = 0 ==> P ξ dip"
have "P ξ dip"

proof -
from all have "sqn (rt ξ) dip ≤ sqn (invalidate (rt ξ) (dests ξ)) dip" ...
with "sqn (invalidate (rt ξ) (dests ξ)) dip = 0" have "sqn (rt ξ) dip = 0" by simp thus "P ξ dip" by (rule *)
qed

note sqn_invalidate_zero [elim!]= this

fix dsn hops :: nat and sip oip rt and ip dip :: ip
assume "∀dip ∈ kD (rt).
    (sqn rt dip = 0 <-> π₃ (the (rt dip)) = unk) ∧
    (π₃ (the (rt dip)) = unk -> the (dhops rt dip) = Suc 0) ∧
    (the (dhops rt dip) = Suc 0 -> the (nhop rt dip) = dip)"
and "hops = 0 -> sip = dip"
and "Suc 0 ≤ dsn"
and "ip ≠ dip -> ip ∈ kD (rt)"
hence "the (dhops (update rt dip (dsn, kno, val, Suc hops, sip, {})) ip) = Suc 0 ->
    the (nhop (update rt dip (dsn, kno, val, Suc hops, sip, {})) ip) = ip"
by - (rule update_cases, auto simp add: sqn_def dest!: bspec)

note prreq_ok1 [simp] = this

fix ip dsn hops sip oip rt dip
assume "∀dip ∈ kD (rt).
    (sqn rt dip = 0 <-> π₃ (the (rt dip)) = unk) ∧
    (π₃ (the (rt dip)) = unk -> the (dhops rt dip) = Suc 0) ∧
    (the (dhops rt dip) = Suc 0 -> the (nhop rt dip) = dip)"
and "Suc 0 ≤ dsn"
and "ip ≠ dip -> ip ∈ kD (rt)"
hence "π₃ (the (update rt dip (dsn, kno, val, Suc hops, sip, {})) ip) = unk ->
    the (dhops (update rt dip (dsn, kno, val, Suc hops, sip, {})) ip) = Suc 0"
by - (rule update_cases, auto simp add: sqn_def sqnf_def dest!: bepec)

note prreq_ok2 [simp] = this

fix ip dsn hops sip oip rt dip
assume "∀dip ∈ kD (rt).
    (sqn rt dip = 0 <-> π₃ (the (rt dip)) = unk) ∧
    (π₃ (the (rt dip)) = unk -> the (dhops rt dip) = Suc 0) ∧
    (the (dhops rt dip) = Suc 0 -> the (nhop rt dip) = dip)"
and "Suc 0 ≤ dsn"
and "ip ≠ dip -> ip ∈ kD (rt)"
hence "sqn (update rt dip (dsn, kno, val, Suc hops, sip, {})) ip = 0 ->
    π₃ (the (update rt dip (dsn, kno, val, Suc hops, sip, {})) ip) = unk"
by - (rule update_cases, auto simp add: sqn_def sqnf_def dest!: bepec)

note prreq_ok3 [simp] = this

fix rt sip
assume "∀dip ∈ kD (rt).
    (sqn rt dip = 0 <-> π₃ (the (rt dip)) = unk) ∧
    (π₃ (the (rt dip)) = unk -> the (dhops rt dip) = Suc 0) ∧
    (the (dhops rt dip) = Suc 0 -> the (nhop rt dip) = dip)"
hence "∀dip ∈ kD (rt).
    (sqn (update rt sip (0, unk, val, Suc 0, sip, {})) dip = 0 ->
    π₃ (the (update rt sip (0, unk, val, Suc 0, sip, {})) dip) = unk)
∧ (∀dip ∈ kD (rt).
    (sqn (update rt sip (0, unk, val, Suc 0, sip, {})) dip = 0 ->
    the (dhops (update rt sip (0, unk, val, Suc 0, sip, {})) dip) = Suc 0)
∧ (∀dip ∈ kD (rt).
    (sqn (update rt sip (0, unk, val, Suc 0, sip, {})) dip = 0 ->
    the (nhop (update rt sip (0, unk, val, Suc 0, sip, {})) dip) = dip))"
by - (rule update_cases, simp_all add: sqn_def sqnf_def)

note prreq_ok4 [simp] = this

have prreq_ok5 [simp]: "∀sip rt.
\(\pi_3(\text{the \ (update \ rt \ sip \ (0, \ unk, \ val, \ Suc 0, \ sip, \ {}) \ sip}) = \text{unk} \rightarrow\)
\(\text{the \ (dhops \ (update \ rt \ sip \ (0, \ unk, \ val, \ Suc 0, \ sip, \ {}) \ sip}) = \text{Suc 0}"
by (rule update_cases) simp_all

have prreq_ok6 [simp]: "/\, \text{sip \ rt.}\
\text{sqn \ (update \ rt \ sip \ (0, \ unk, \ val, \ Suc 0, \ sip, \ {}) \ sip} = 0 \rightarrow\
\pi_3 \ (\text{the \ (update \ rt \ sip \ (0, \ unk, \ val, \ Suc 0, \ sip, \ {}) \ sip}) = \text{unk}"
by (rule update_cases) simp_all

show ?thesis
by (inv_cterms inv add: onl_invariant_sterms_TT [OF aodv_wf addpreRT_welldefined]
onl_invariant_sterms [OF aodv_wf hop_count_zero_oip_dip_sip']
seq_step_invariant_sterms_TT [OF sqns_increase aodv_wf aodv_trans]
onl_invariant_sterms [OF aodv_wf osn_rreq']
onl_invariant_sterms [OF aodv_wf dsn_rrep']) clarsimp+
qed

lemma zero_seq_unk_hops_one:
"paodv i \|= (recvmsg (\lambda m. rreq_rrep_sn m \land msg_zhops m) \rightarrow) \text{onl } \Gamma_{AODV} \ (\lambda(x, _).\)
\forall dip \in kD(rt x). (sqn (rt x) dip = 0 \rightarrow (sqnf (rt x) dip = \text{unk} \land\\
the (dhops (rt x) dip) = 1 \land the (nhop (rt x) dip) = dip)))"
by (rule invariant_weakenE [OF zero_seq_unk_hops_one']) auto

lemma kD_unk_or_atleast_one:
"paodv i \|= (recvmsg rreq_rrep_sn \rightarrow) \text{onl } \Gamma_{AODV} \ (\lambda(x, n).\)
\forall dip \in kD(rt x). \pi_3(\text{the (rt x dip)}) = \text{unk} \lor 1 \leq \pi_2(\text{the (rt x dip)})"
proof -
{ fix sip rt dsn1 dsn2 dsk1 dsk2 flag1 flag2 hops1 hops2 nhip1 nhip2 pre1 pre2
assume "dsk1 = unk \lor Suc 0 \leq dsn2"
hence "\pi_3(\text{the \ (update \ rt \ sip \ (dsn1, \ dsk1, \ flag1, \ hops1, \ nhip1, \ pre1) \ sip}) = \text{unk} \lor Suc 0 \leq \text{sqn \ (update \ rt \ sip \ (dsn2, \ dsk2, \ flag2, \ hops2, \ nhip2, \ pre2) \ sip})"
unfolding update_def by (cases "dsk1 = unk") (clarsimp split: option.split)+
} note fromsip [simp] = this

{ fix dip sip rt dsn1 dsn2 dsk1 dsk2 flag1 flag2 hops1 hops2 nhip1 nhip2 pre1 pre2
assume allkd: "\forall dip \in kD(rt). \pi_3(\text{the \ (rt dip)}) = \text{unk} \lor Suc 0 \leq \text{sqn \ (invalidate \ rt \ dests})
and **: "dsk1 = unk \lor Suc 0 \leq dsn2"
have "\forall dip \in kD(rt). \pi_3(\text{the \ (update \ rt \ sip \ (dsn1, \ dsk1, \ flag1, \ hops1, \ nhip1, \ pre1) \ dip}) = \text{unk} \lor Suc 0 \leq \text{sqn \ (update \ rt \ sip \ (dsn2, \ dsk2, \ flag2, \ hops2, \ nhip2, \ pre2) \ dip})"
(is "\forall dip \in kD(rt). ?prop dip")
proof
fix dip
assume "dip \in kD(rt)"
thus "?prop dip"
proof (cases "dip = sip")
assume "dip = sip"
with ** show ?thesis
by simp
next
assume "dip \neq sip"
with \(dip \in kD(rt)\) allkd show ?thesis
by simp
qed
qed
} note solve_update [simp] = this

{ fix dip rt dests
assume *: "\forall dip \in \text{dom(dests).} \, ip \in kD(rt) \land \text{sqn \ rt \ dip} \leq \text{the \ (dests ip)}"
and **: "\forall dip \in kD(rt). \pi_3(\text{the \ (rt ip)}) = \text{unk} \lor Suc 0 \leq \text{sqn \ (invalidate \ rt \ dests) \ dip}"
have "\forall dip \in kD(rt). \pi_3(\text{the \ (rt dip)}) = \text{unk} \lor Suc 0 \leq \text{sqn \ (invalidate \ rt \ dests) \ dip}"
proof
fix dip
assume "dip \in kD(rt)"

133
with \*\* have \( \pi_3(\text{the (rt dip)}) = \text{unk} \lor \text{Suc 0} \leq \text{sqn rt dip} \) ..
thus \( \pi_3(\text{the (rt dip)}) = \text{unk} \lor \text{Suc 0} \leq \text{sqn (invalidate rt dests) dip} \)
proof
assume \( \pi_3(\text{the (rt dip)}) = \text{unk} \) thus \( \text{?thesis} \) ..
next
assume \( \text{Suc 0} \leq \text{sqn rt dip} \)
proof (cases \( \text{dip} \in \text{dom(dests)} \))
assume \( \text{dip} \in \text{dom(dests)} \)
with \* have \( \text{sqn rt dip} \leq \text{the (dests dip)} \) by simp
with \( \text{Suc 0} \leq \text{sqn rt dip} \) have \( \text{Suc 0} \leq \text{the (dests dip)} \) by simp
with \( \text{dip} \in \text{dom(dests)} \) \( \text{dip} \in \text{kD(rt)} \) \( \text{THEN kD_Some} \) show \( \text{?thesis} \)

unfolding invalidate_def sqn_def by auto

next
assume \( \text{dip} \not\in \text{dom(dests)} \)
with \( \text{Suc 0} \leq \text{sqn rt dip} \) \( \text{dip} \in \text{kD(rt)} \) \( \text{THEN kD_Some} \) show \( \text{?thesis} \)
unfolding invalidate_def sqn_def by auto

qed
thus \( \text{?thesis} \) by (rule disjI2)
qed

} note solve_invalidate [simp] = this

show \( \text{?thesis} \)
by (inv_cterms inv add: onl_invariant_sterms_TT [OF aodv_wf addpreRT_welldefined]
onl_invariant_sterms [OF aodv_wf dests_bigger_than_sqn]
[THEN invariant_restrict_inD])
onl_invariant_sterms [OF aodv_wf osn_rreq]
onl_invariant_sterms [OF aodv_wf dsn_rrep]
simp add: proj3_inv proj2_eq_sqn)

qed

Proposition 7.13
lemma rreq_rrep_sn_any_step_invariant:
"\text{paodv} i \models_{A} (\text{recvmsg} \text{ rreq_rrep_sn} \rightarrow) \text{onll} \Gamma_{AODV} (\lambda(\xi, a, \_). \text{anycast} \text{ rreq_rrep_sn} a)"
proof -
have sqnf_kno: "\text{paodv} i \models_{A} \Gamma_{AODV} (\lambda(\xi, 1).\n(1 \in \{\text{PRreq-:16..PRreq-:18}\} \rightarrow \text{sqnf (rt } \xi) (\text{dip} \xi) = \text{kno}))"
by (inv_cterms inv add: onl_invariant_sterms_TT [OF aodv_wf addpreRT_welldefined])
show \( \text{?thesis} \)
by (inv_cterms inv add: onl_invariant_sterms_TT [OF aodv_wf sequence_number_one_or_bigger]
[THEN invariant_restrict_inD])
onl_invariant_sterms [OF aodv_wf kD_unk_or_atleast_one]
onl_invariant_sterms_TT [OF aodv_wf sqnf_kno]
onl_invariant_sterms [OF aodv_wf osn_rreq]
onl_invariant_sterms [OF aodv_wf dsn_rrep])
(auto simp: proj2_eq_sqn)

qed

Proposition 7.14
lemma rreq_rrep_fresh_any_step_invariant:
"\text{paodv} i \models_{A} \text{onll} \Gamma_{AODV} (\lambda(\xi, a, \_). \text{anycast} \text{ (rreq_rrep_fresh (rt } \xi)) a)"
proof -
have rreq_oip: "\text{paodv} i \models_{A} \Gamma_{AODV} (\lambda(\xi, 1).\n(1 \in \{\text{PRreq-:3, PRreq-:4, PRreq-:15, PRreq-:27}\} \rightarrow \text{oip } \xi \in \text{kD(rt } \xi)\n\land (\text{sqn (rt } \xi) (\text{oip } \xi) > \text{osn } \xi)\n\lor (\text{sqn (rt } \xi) (\text{oip } \xi) = \text{osn } \xi)\n\land \text{the (dhops (rt } \xi) (\text{oip } \xi)) \leq \text{Suc (hops } \xi)\n\land \text{the (flag (rt } \xi) (\text{oip } \xi)) = \text{val})))"
proof inv_cterms
fix 1 \xi l' pp p'
assume "(\xi, pp) \in \text{reachable} (\text{paodv} i) \text{ TT}"

134
and \[\{\text{PRreq-:2}\}\lambda\xi.\ (\xi|rt :=
\quad\text{update (rt }\xi) (\text{oip }\xi) (\text{osn }\xi, \text{kno, val, Suc (hops }\xi), \text{sip }\xi, \{\})\}]\ p' \in \text{sterms }\Gamma_{AODV} \text{ pp}\
and \[\{\text{PRreq-:3}\}\]
show \[\begin{align*}\
\text{osn }\xi &< sqn (\text{update (rt }\xi) (\text{oip }\xi) (\text{osn }\xi, \text{kno, val, Suc (hops }\xi), \text{sip }\xi, \{\})) (\text{oip }\xi) \\
\lor & (\text{sqn (update (rt }\xi) (\text{oip }\xi) (\text{osn }\xi, \text{kno, val, Suc (hops }\xi), \text{sip }\xi, \{\})) (\text{oip }\xi) = \text{osn }\xi \\
& \wedge \text{the (dhops (rt }\xi) (\text{oip }\xi) (\text{osn }\xi, \text{kno, val, Suc (hops }\xi), \text{sip }\xi, \{\})) (\text{oip }\xi) \\
& \leq \text{Suc (hops }\xi) \\
& \wedge \text{the (flag (rt }\xi) (\text{oip }\xi) (\text{osn }\xi, \text{kno, val, Suc (hops }\xi), \text{sip }\xi, \{\})) (\text{oip }\xi) \\
& = \text{val}\end{align*}\]
\]

unfolding \text{update_def} by \(\text{(clarsimp split: option.split)}\)
\(\text{(metis linorder_neqE_nat not_less)}\)
qued

have \text{rrep_prrep}: \[\begin{align*}\
\text{paodv }i &|\Rightarrow\text{ onl }\Gamma_{AODV} (\lambda(\xi, 1). \\
(1 &\in \{\text{PRrep-:2..PRrep-:7}\}) \\
&\wedge \text{sqn (rt }\xi) (\text{dip }\xi) = \text{dsn }\xi \\
&\wedge \text{the (dhops (rt }\xi) (\text{dip }\xi)) = \text{Suc (hops }\xi) \\
&\wedge \text{the (flag (rt }\xi) (\text{dip }\xi)) = \text{val} \\
&\wedge \text{the (nhop (rt }\xi) (\text{dip }\xi)) \in kD (rt }\xi))\)
\]
by \(\text{(inv_cterms inv add: onl_invariant_sterms [OF aodv_wf rrep_1_update_changes] onl_invariant_sterms [OF aodv_wf sip_in_kD])}\)

show \text{?thesis}
qued

Proposition 7.15

\text{lemma rerr_invalid_any_step_invariant}:
\[\begin{align*}\
\text{paodv }i &|\Rightarrow\text{ onl }\Gamma_{AODV} (\lambda(\xi, a, \_). \text{anycast (rerr_invalid (rt }\xi)) a)\)
\]
proof -

have \text{dests_inv}: \[\begin{align*}\
\text{paodv }i &|\Rightarrow\text{ onl }\Gamma_{AODV} (\lambda(\xi, 1). (1 &\in \{\text{PAodv-:15, PPkt-:7, PRreq-:9, PRreq-:21, PRrep-:10, PRerr-:1}\}) \\
&\wedge (1 &\in \{\text{PAodv-:16..PAodv-:19} \cup \{\text{PPkt-:8..PPkt-:11} \cup \{\text{PRreq-:10..PRreq-:13} \cup \{\text{PRreq-:22..PRreq-:25} \cup \{\text{PRrep-:11..PRrep-:14} \cup \{\text{PRerr-:2..PRerr-:5}\} \\
&\quad \rightarrow (\forall ip \in \text{dom(dests }\xi). ip \in \text{iD(rt }\xi))) \\
&\quad \wedge (1 = \text{PPkt-:14} \rightarrow \text{dip }\xi \in \text{iD(rt }\xi)))\)
\]
by \text{inv_cterms (clarsimp split: split_if_asm option.split_asm simp: domIff)+}
show \text{?thesis}
by \text{(inv_cterms inv add: onl_invariant_sterms [OF aodv_wf dests_inv])}\)
qued

Proposition 7.16

Some well-definedness obligations are irrelevant for the Isabelle development:

1. In each routing table there is at most one entry for each destination: guaranteed by type.
2. In each store of queued data packets there is at most one data queue for each destination: guaranteed by structure.
3. Whenever a set of pairs \((rip, rsn)\) is assigned to the variable \(\text{dests}\) of type \(ip \rightarrow sqn\), or to the first argument of the function \(\text{rerr}\), this set is a partial function, i.e., there is at most one entry \((rip, rsn)\) for each destination \(rip\): guaranteed by type.

\text{lemma dests_vD_inc_sqn}:
\[\begin{align*}\
\text{paodv }i &|\Rightarrow\end{align*}\]
1.7 The quality increases predicate

theory A_Quality_Increases

imports A_Aodv_Predicates A_Fresher

begin

definition quality_increases :: "state ⇒ state ⇒ bool"
where "quality_increases ξ ξ' ≡ (∀ dip∈kD(rt ξ). dip ∈ kD(rt ξ') ∧ rt ξ ⊑ dip rt ξ') 
∧ (∀ dip. sqn (rt ξ) dip ≤ sqn (rt ξ') dip)"

lemma quality_increasesI [intro!]:
assumes 
- \( \wedge \text{dip. } \text{dip } \in \text{kD}(rt \xi) \Rightarrow \text{dip } \in \text{kD}(rt \xi') \)
- \( \wedge \text{dip. } \text{dip } \in \text{kD}(rt \xi); \text{dip } \in \text{kD}(rt \xi') \] \( \Rightarrow \text{rt } \xi \sqsubseteq_{\text{dip}} \text{rt } \xi' \)
- \( \wedge \text{dip. } \text{sqn } (rt \xi) \text{ dip } \leq \text{sqn } (rt \xi') \text{ dip} \)

shows "quality_increases \( \xi \xi' \)"

unfolding quality_increases_def using assms by clarsimp

lemma quality_increasesE [elim]:
  fixes dip
  assumes "quality_increases \( \xi \xi' \)"
  and "dip \in \text{kD}(rt \xi)"
  and "[ \text{dip } \in \text{kD}(rt \xi'); \text{rt } \xi \sqsubseteq_{\text{dip}} \text{rt } \xi' ]; \text{sqn } (rt \xi) \text{ dip } \leq \text{sqn } (rt \xi') \text{ dip} ] \Rightarrow \text{R } \text{dip } \xi \xi'"
  using assms unfolding quality_increases_def by clarsimp

lemma quality_increases_rt_fresherD [dest]:
  fixes ip
  assumes "quality_increases \( \xi \xi' \)"
  and "ip \in \text{kD}(rt \xi)"
  shows "rt \( \xi \sqsubseteq_{\text{ip}} \text{rt } \xi' \)"
  using assms by auto

lemma quality_increases_sqnE [elim]:
  fixes dip
  assumes "quality_increases \( \xi \xi' \)"
  and "\text{sqn } (rt \xi) \text{ dip } \leq \text{sqn } (rt \xi') \text{ dip} \Rightarrow \text{R dip } \xi \xi'"
  using assms unfolding quality_increases_def by clarsimp

lemma quality_increases_refl [intro, simp]: "quality_increases \( \xi \xi \)"
  by rule simp_all

lemma strictly_fresher_quality_increases_right [elim]:
  fixes \( \sigma \sigma' \text{ dip} \)
  assumes "rt (\sigma i) \sqsubseteq_{\text{dip}} rt (\sigma' \text{ nhip})"
  and qinc: "quality_increases (\sigma nhip) (\sigma' \text{ nhip})"
  and "dip \in \text{kD}(rt (\sigma' \text{ nhip}))"
  shows "rt (\sigma i) \sqsubseteq_{\text{dip}} rt (\sigma' \text{ nhip})"
  proof -
    from qinc have "rt (\sigma nhip) \sqsubseteq_{\text{dip}} rt (\sigma' \text{ nhip})" using \( \text{dip } \in \text{kD}(rt (\sigma' \text{ nhip})) \)
    by auto
    with \( rt (\sigma i) \sqsubseteq_{\text{dip}} rt (\sigma' \text{ nhip}) \) show ?thesis ..
  qed

lemma kD_quality_increases [elim]:
  assumes "i \in \text{kD}(rt \xi)"
  and "quality_increases \( \xi \xi' \)"
  shows "i \in \text{kD}(rt \xi')"
  using assms by auto

lemma kD_nsqn_quality_increases [elim]:
  assumes "i \in \text{kD}(rt \xi)"
  and "quality_increases \( \xi \xi' \)"
  shows "i \in \text{kD}(rt \xi') \land \text{nsqn } (rt \xi) i \leq \text{nsqn } (rt \xi') i"
  proof -
    from assms have "i \in \text{kD}(rt \xi')" ..
    moreover with assms have "rt \xi \sqsubseteq_{i} rt \xi'" by auto
    ultimately have "\text{nsqn } (rt \xi) i \leq \text{nsqn } (rt \xi') i"
    using \( i \in \text{kD}(rt \xi) \) by - (erule(2) rt_fresher_imp_nsqn_le)
    with \( i \in \text{kD}(rt \xi') \) show ?thesis ..
  qed

lemma nsqn_quality_increases [elim]:
  assumes "i \in \text{kD}(rt \xi)"
  and "quality_increases \( \xi \xi' \)"
shows "nsqn (rt ξ) i ≤ nsqn (rt ξ') i"
using assms by (rule kD_nsqn_quality_increases [THEN conjunct2])

lemma kD_nsqn_quality_increases_trans [elim]:
assumes "i ∈ kD(rt ξ)"
and "s ≤ nsqn (rt ξ) i"
and "quality_increases ξ ξ'"
shows "i ∈ kD(rt ξ') ∧ s ≤ nsqn (rt ξ') i"

proof
  from ⟨i ∈ kD(rt ξ)⟩ and ⟨quality_increases ξ ξ'⟩ show "i ∈ kD(rt ξ')" ..
next
  from ⟨i ∈ kD(rt ξ)⟩ and ⟨quality_increases ξ ξ'⟩ have "nsqn (rt ξ) i ≤ nsqn (rt ξ') i" ..
  with ⟨s ≤ nsqn (rt ξ) i⟩ show "s ≤ nsqn (rt ξ') i" by (rule le_trans)
qed

lemma nsqn_quality_increases_nsqn_lt_lt [elim]:
assumes "i ∈ kD(rt ξ)"
and "quality_increases ξ ξ'"
and "s < nsqn (rt ξ) i"
shows "s < nsqn (rt ξ') i"

proof -
  from assms(1-2) have "nsqn (rt ξ) i ≤ nsqn (rt ξ') i" ..
  with ⟨s < nsqn (rt ξ) i⟩ show "s < nsqn (rt ξ') i" by simp
qed

lemma nsqn_quality_increases_dhops [elim]:
assumes "i ∈ kD(rt ξ)"
and "quality_increases ξ ξ'"
and "nsqn (rt ξ) i = nsqn (rt ξ') i"
shows "the (dhops (rt ξ) i) ≥ the (dhops (rt ξ') i)"

using assms unfolding quality_increases_def
by (clarsimp) (drule(1) bspec, clarsimp simp: rt_fresher_def2)

lemma nsqn_quality_increases_eq_le [elim]:
assumes "i ∈ kD(rt ξ)"
and "quality_increases ξ ξ'"
and "s = nsqn (rt ξ) i"
shows "s < nsqn (rt ξ') i ∨ (s = nsqn (rt ξ') i ∧ the (dhops (rt ξ) i) ≥ the (dhops (rt ξ') i))"

using assms by (metis nat_less_le nsqn_quality_increases nsqn_quality_increases_dhops)

lemma quality_increases_rreq_rrep_props [elim]:
fixes sn ip hops sip
assumes qinc: "quality_increases (σ sip) (σ' sip)"
and "1 ≤ sn"
and *: "ip ∈ kD(rt (σ sip)) ∧ sn ≤ nsqn (rt (σ sip)) ip
∧ (nsqn (rt (σ sip)) ip = sn → (the (dhops (rt (σ sip)) ip) ≤ hops
∨ the (flag (rt (σ sip)) ip) = inv))"
shows "ip ∈ kD(rt (σ' sip)) ∧ sn ≤ nsqn (rt (σ' sip)) ip
∧ (nsqn (rt (σ' sip)) ip = sn → (the (dhops (rt (σ' sip)) ip) ≤ hops
∨ the (flag (rt (σ' sip)) ip) = inv))"
(is "_ ∧ ?nsqnafter")

proof -
  from * obtain "ip ∈ kD(rt (σ sip))" and "sn ≤ nsqn (rt (σ sip)) ip" by auto
  from quality_increases (σ sip) (σ' sip)
    have "sqn (rt (σ sip)) ip ≤ sqn (rt (σ' sip)) ip" ..
  from quality_increases (σ sip) (σ' sip) and ip ∈ kD (rt (σ sip))
    have "ip ∈ kD (rt (σ' sip))" ..
  from (sn ≤ nsqn (rt (σ sip)) ip) have ?nsqnafter
proof
  assume "sn < nsqn (rt (σ sip)) ip"
also from 
\[ \langle \text{ip} \in kD(\text{rt} (\sigma \text{ sip})) \rangle \text{ and } \langle \text{quality}\_\text{increases} (\sigma \text{ sip}) (\sigma' \text{ sip}) \rangle \]

have "\( \ldots \leq \text{nsqn} (\text{rt} (\sigma' \text{ sip})) \text{ ip} \)" ..

finally have "\( \text{sn} < \text{nsqn} (\text{rt} (\sigma' \text{ sip})) \text{ ip} \)" .

thus \( ?\text{thesis} \) by \text{simp}

next

assume "\( \text{sn} = \text{nsqn} (\text{rt} (\sigma' \text{ sip})) \text{ ip} \)"

with \( \langle \text{ip} \in kD(\text{rt} (\sigma \text{ sip})) \rangle \) and \( \langle \text{quality}\_\text{increases} (\sigma \text{ sip}) (\sigma' \text{ sip}) \rangle \)

have "\( \text{sn} < \text{nsqn} (\text{rt} (\sigma' \text{ sip})) \text{ ip} \)
\[ \lor (\text{sn} = \text{nsqn} (\text{rt} (\sigma' \text{ sip})) \text{ ip}) \]
\[ \land (\text{the} (\text{dhops} (\text{rt} (\sigma' \text{ sip})) \text{ ip}) \leq \text{hops}) \]
\[ \lor (\text{the} (\text{flag} (\text{rt} (\sigma' \text{ sip})) \text{ ip}) = \text{inv})" \]

hence "\( \text{sn} < \text{nsqn} (\text{rt} (\sigma' \text{ sip})) \text{ ip} \)
\[ \lor (\text{nsqn} (\text{rt} (\sigma' \text{ sip})) \text{ ip} = \text{sn} \land (\text{the} (\text{dhops} (\text{rt} (\sigma' \text{ sip})) \text{ ip}) \leq \text{hops}) \]
\[ \lor (\text{the} (\text{flag} (\text{rt} (\sigma' \text{ sip})) \text{ ip}) = \text{inv})" \]

proof

assume "\( \text{sn} < \text{nsqn} (\text{rt} (\sigma' \text{ sip})) \text{ ip} \)" thus \( ?\text{thesis} \) ..

next

assume "\( \text{sn} = \text{nsqn} (\text{rt} (\sigma' \text{ sip})) \text{ ip} \)
\[ \land (\text{the} (\text{dhops} (\text{rt} (\sigma' \text{ sip})) \text{ ip}) \geq \text{hops}) \]

hence "\( \text{sn} = \text{nsqn} (\text{rt} (\sigma' \text{ sip})) \text{ ip} \)
\[ \land (\text{the} (\text{dhops} (\text{rt} (\sigma' \text{ sip})) \text{ ip}) \leq \text{hops}) \]
\[ \lor (\text{the} (\text{flag} (\text{rt} (\sigma' \text{ sip})) \text{ ip}) = \text{inv})" \]

from \* and \( \langle \text{sn} = \text{nsqn} (\text{rt} (\sigma' \text{ sip})) \text{ ip} \rangle \) have "\( \text{the} (\text{dhops} (\text{rt} (\sigma' \text{ sip})) \text{ ip}) \leq \text{hops} \)
\[ \lor (\text{the} (\text{flag} (\text{rt} (\sigma' \text{ sip})) \text{ ip}) = \text{inv})" \]

by \text{simp}

thus \( ?\text{thesis} \)

proof

assume "\( \text{the} (\text{dhops} (\text{rt} (\sigma' \text{ sip})) \text{ ip}) \leq \text{hops} \)

with \( \langle \text{sn} = \text{nsqn} (\text{rt} (\sigma' \text{ sip})) \text{ ip} \rangle \) have "\( \text{the} (\text{dhops} (\text{rt} (\sigma' \text{ sip})) \text{ ip}) \leq \text{hops} \) by \text{simp}

with \( \langle \text{sn} = \text{nsqn} (\text{rt} (\sigma' \text{ sip})) \text{ ip} \rangle \) show \( ?\text{thesis} \) by \text{simp}

next

assume "\( \text{the} (\text{flag} (\text{rt} (\sigma' \text{ sip})) \text{ ip}) = \text{inv} \)

with \( \langle \text{ip} \in kD(\text{rt} (\sigma \text{ sip})) \rangle \) have "\( \text{nsqn} (\text{rt} (\sigma \text{ sip})) \text{ ip} = \text{sqn} (\text{rt} (\sigma \text{ sip})) \text{ ip} - 1 \)" ..

with \( \langle \text{sn} \geq 1 \rangle \) and \( \langle \text{sn} = \text{nsqn} (\text{rt} (\sigma \text{ sip})) \text{ ip} \rangle \)

have "\( \text{sqn} (\text{rt} (\sigma \text{ sip})) \text{ ip} \geq 1 \) by \text{simp}

from \( \langle \text{ip} \in kD(\text{rt} (\sigma \text{ sip})) \rangle \) show \( ?\text{thesis} \)

proof (rule \text{vD_or_iD})

\[ \langle \text{ip} \in iD(\text{rt} (\sigma' \text{ sip})) \rangle \]

hence "\( \text{the} (\text{flag} (\text{rt} (\sigma' \text{ sip})) \text{ ip}) = \text{inv} \)" ..

with \( \langle \text{sn} = \text{nsqn} (\text{rt} (\sigma' \text{ sip})) \text{ ip} \rangle \) show \( ?\text{thesis} \) by \text{simp}

next

assume "\( \langle \text{ip} \in \text{vD}(\text{rt} (\sigma' \text{ sip})) \rangle \)

hence "\( \text{nsqn} (\text{rt} (\sigma' \text{ sip})) \text{ ip} = \text{sqn} (\text{rt} (\sigma' \text{ sip})) \text{ ip} \)" ..

with \( \langle \text{sn} = \text{nsqn} (\text{rt} (\sigma' \text{ sip})) \text{ ip} \rangle \)

have "\( \text{nsqn} (\text{rt} (\sigma' \text{ sip})) \text{ ip} \geq \text{sqn} (\text{rt} (\sigma \text{ sip})) \text{ ip} \)" by \text{simp}

with \( \langle \text{sn} = \text{nsqn} (\text{rt} (\sigma' \text{ sip})) \text{ ip} \rangle \) have "\( \text{nsqn} (\text{rt} (\sigma' \text{ sip})) \text{ ip} > \text{sn} \)" by \text{simp}

thus \( ?\text{thesis} \) ..

qed

qed

thus \( ?\text{thesis} \) by \text{(metis (mono_tags) le_cases not_le)}

qed

with \( \langle \text{ip} \in kD (\text{rt} (\sigma' \text{ sip})) \rangle \) show "\( \text{ip} \in kD (\text{rt} (\sigma' \text{ sip})) \wedge \text{?nsqnafter} \)" ..

qed
lemma quality_increases_rreq_rrep_props':
    fixes sn ip hops sip
    assumes "∀j. quality_increases (σ j) (σ' j)")
    and "1 ≤ sn"
    and *: "ip ∈ kD(rt (σ sip)) ∧ sn ≤ nsqn (rt (σ sip)) ip
    ∧ (nsqn (rt (σ sip)) ip = sn → (the (dhops (rt (σ sip)) ip) ≤ hops
    ∨ the (flag (rt (σ sip)) ip) = inv))"
    shows "ip ∈ kD(rt (σ' sip)) ∧ sn ≤ nsqn (rt (σ' sip)) ip
    ∧ (nsqn (rt (σ' sip)) ip = sn → (the (dhops (rt (σ' sip)) ip) ≤ hops
    ∨ the (flag (rt (σ' sip)) ip) = inv))"
proof -
    from assms(1) have "quality_increases (σ sip) (σ' sip)"
    .. thus ?thesis using assms(2-3) by (rule quality_increases_rreq_rrep_props)
qed

lemma rteq_quality_increases:
    assumes "∀j. j ≠ i → quality_increases (σ j) (σ' j)"
    and "rt (σ' i) = rt (σ i)"
    shows "∀j. quality_increases (σ j) (σ' j)"
    using assms by clarsimp (metis order_refl quality_increasesI rt_fresher_refl)

definition msg_fresh :: "(ip ⇒ state) ⇒ msg ⇒ bool"
where "msg_fresh σ m ≡
case m of
  Rreq hopsc oipc osnc sipc ⇒ osnc ≥ 1 ∧ (sipc ≠ oipc −→ oipc ∈ kD(rt (σ sipc))
  ∧ nsqn (rt (σ sipc)) oipc ≥ osnc
  ∧ (nsqn (rt (σ sipc)) oipc = osnc → (hopsc ≥ the (dhops (rt (σ sipc)) oipc)
  ∨ the (flag (rt (σ sipc)) oipc) = inv)))
| Rrep hopsc dipc dsnc sipc ⇒ dsnc ≥ 1 ∧ (sipc ≠ dipc −→ dipc ∈ kD(rt (σ sipc))
  ∧ nsqn (rt (σ sipc)) dipc ≥ dsnc
  ∧ (nsqn (rt (σ sipc)) dipc = dsnc → (hopsc ≥ the (dhops (rt (σ sipc)) dipc)
  ∨ the (flag (rt (σ sipc)) dipc) = inv)))
| Rerr destsc sipc ⇒ (∀ripc ∈ dom(destsc). (ripc ∈ kD(rt (σ sipc))
  ∧ the (destsc ripc) - 1 ≤ nsqn (rt (σ sipc)) ripc))
| _ ⇒ True"

lemma msg_fresh [simp]:
"∀hops dip dsn dsk oip osn sip.
  msg_fresh σ (Rreq hopsc oipc osnc sipc) =
  osn ≥ 1 ∧ (sipc ≠ oipc −→ oipc ∈ kD(rt (σ sipc))
  ∧ nsqn (rt (σ sipc)) oipc ≥ osn
  ∧ (nsqn (rt (σ sipc)) oipc = osn → (hopsc ≥ the (dhops (rt (σ sipc)) oipc)
  ∨ the (flag (rt (σ sipc)) oipc) = inv)))"

"∀hops dip dsn oip sip.
  msg_fresh σ (Rreq hopsc oipc osnc sipc) =
  (dsn ≥ 1 ∧ (sipc ≠ dipc −→ dipc ∈ kD(rt (σ sipc))
  ∧ nsqn (rt (σ sipc)) dipc ≥ dsn
  ∧ (nsqn (rt (σ sipc)) dipc = dsn → (hopsc ≥ the (dhops (rt (σ sipc)) dipc)
  ∨ the (flag (rt (σ sipc)) dipc) = inv)))"

"∀dests sipc.
  msg_fresh σ (Rreq hopsc oipc osnc sipc) =
  (∀ripc ∈ dom(dests). (ripc ∈ kD(rt (σ sipc))
  ∧ the (dests ripc) - 1 ≤ nsqn (rt (σ sipc)) ripc))
  ∧ (the (dests ripc) - 1 ≤ nsqn (rt (σ sipc)) ripc))"

"∀d dip.
  msg_fresh σ (Newpkt d dip) = True"

"∀d dip. msg_fresh σ (Pkt d dip sip) = True"

unfolding msg_fresh_def by simp_all

lemma msg_fresh_inc_sn [simp, elim]:
"∀m.
  msg_fresh σ m =⇒ rreq_rrep_sn m"
by (cases m) simp_all
lemma recv_msg_fresh_inc_sn [simp, elim]:
"orecvmsg (msg_fresh) σ m ⇒ recvmsg rreq_rrep_sn m"
by (cases m) simp_all

lemma rreq_nsqn_is_fresh [simp]:
fixes σ msg hops dip dsn dsk oip osn sip
assumes "rreq_rrep_fresh (rt (σ sip)) (Rreq hops dip dsn dsk oip osn sip)"
and "rreq_rrep_sn (Rreq hops dip dsn dsk oip osn sip)"
shows "msg_fresh σ (Rreq hops dip dsn dsk oip osn sip)"
(is "msg_fresh σ ?msg")
proof -
let ?rt = "rt (σ sip)"
from assms(2) have "1 ≤ osn" by simp
thus ?thesis
unfolding msg_fresh_def
proof (simp only: msg.case, intro conjI impI)
  assume "sip ≠ oip"
  with assms(1) show "oip ∈ kD(?rt)" by simp
next
  assume "sip ≠ oip"
  and "nsqn ?rt oip = osn"
  show "the (dhops ?rt oip) ≤ hops ∨ the (flag ?rt oip) = inv"
proof (cases "oip∈vD(?rt)"
  assume "oip∈vD(?rt)"
  hence "nsqn ?rt oip = sqn ?rt oip" ..
  with assms(1) and ⟨sip ≠ oip⟩ have "sqn ?rt oip = osn" by simp
  thus ?thesis ..
next
  assume "oip∉vD(?rt)"
  moreover from assms(1) and ⟨sip ≠ oip⟩ have "oip∈kD(?rt)" by simp
  ultimately have "oip∈iD(?rt)" by auto
  hence "the (flag ?rt oip) = inv" ..
  thus ?thesis ..
qed

next
  assume "sip ≠ oip"
  with assms(1) have "osn ≤ sqn ?rt oip" by auto
  thus "osn ≤ nsqn (rt (σ sip)) oip"
proof (rule nat_le_eq_or_lt)
  assume "osn < sqn ?rt oip"
  hence "osn ≤ sqn ?rt oip - 1" by simp
  also have "... ≤ nsqn ?rt oip" by (rule sqn_nsqn)
  finally show "osn ≤ nsqn ?rt oip" .
next
  assume "osn = sqn ?rt oip"
  with assms(1) and ⟨sip ≠ oip⟩ have "oip∈kD(?rt)"
  and "the (flag ?rt oip) = val"
  by auto
  hence "nsqn ?rt oip = sqn ?rt oip" ..
  with ⟨osn = sqn ?rt oip⟩ have "nsqn ?rt oip = osn" by simp
  thus "osn ≤ nsqn ?rt oip" by simp
qed

lemma rrep_nsqn_is_fresh [simp]:
fixes σ msg hops dip dsn oip sip
assumes "rreq_rrep_fresh (rt (σ sip)) (Rrep hops dip dsn oip sip)"
and "rreq_rrep_sn (Rrep hops dip dsn oip sip)"
shows "msg_fresh σ (Rrep hops dip dsn oip sip)"
(is "msg_fresh σ ?msg")
proof -
let ?rt = "rt (σ sip)"
from assms have "sip ≠ dip → dip ∈ kD(?rt) ∧ sqn ?rt dip = dsn ∧ the (flag ?rt dip) = val"
  by simp
hence "sip ≠ dip → dip ∈ kD(?rt) ∧ nsqn ?rt dip ≥ dsn"
by clarsimp
with assms show "msg_fresh σ ?msg"
  by clarsimp
qed

lemma rerr_nsqn_is_fresh [simp]:
fixes σ msg dests sip
assumes "rerr_invalid (rt (σ sip)) (Rerr dests sip)"
shows "msg_fresh σ (Rerr dests sip)"
(is "msg_fresh σ ?msg")
proof -
let ?rt = "rt (σ sip)"
from assms have *: "(∀ rip ∈ dom(dests). (rip ∈ iD(rt (σ sip)))
∧ the (dests rip) = sqn (rt (σ sip)) rip))"
by clarsimp
have "(∀ rip ∈ dom(dests). (rip ∈ kD(rt (σ sip)))
∧ the (dests rip) - 1 ≤ nsqn (rt (σ sip)) rip)"
proof
fix rip
assume "rip ∈ dom dests"
with * have "rip ∈ iD(rt (σ sip))" and "the (dests rip) = sqn (rt (σ sip)) rip"
  by auto
from this(2) have "rip ∈ kD(rt (σ sip))"
  show "rip ∈ kD(rt (σ sip)) ∧ the (dests rip) - 1 ≤ nsqn (rt (σ sip)) rip"
    by clarsimp
qed
thus "msg_fresh σ ?msg"
  by simp
qed

lemma quality_increases_msg_fresh [elim]:
assumes qinc: "∀ j. quality_increases (σ j) (σ' j)"
and "msg_fresh σ m"
shows "msg_fresh σ' m"
using assms(2)
proof (cases m)
  fix hops rreqid dip dsn dsk oip osn sip
  assume [simp]: "m = Rreq hops dip dsn dsk oip osn sip"
  and "msg_fresh σ m"
  then have "osn ≥ 1" and "sip = oip ∨ (oip ∈ kD(rt (σ sip)) ∧ osn ≤ nsqn (rt (σ sip)) oip
∧ (nsqn (rt (σ sip)) oip = osn
→ (the (dhops (rt (σ sip)) oip) ≤ hops
∨ the (flag (rt (σ sip)) oip) = inv)))"
    by auto
from this(2) show ?thesis
proof
  assume "sip = oip" with "osn ≥ 1" show ?thesis by simp
next
  assume "oip ∈ kD(rt (σ sip)) ∧ osn ≤ nsqn (rt (σ sip)) oip
∧ (nsqn (rt (σ sip)) oip = osn
→ (the (dhops (rt (σ sip)) oip) ≤ hops
∨ the (flag (rt (σ sip)) oip) = inv))"
moreover from qinc have "quality_increases (σ sip) (σ' sip)" ..
ultimately have "oip ∈ kD(rt (σ' sip)) ∧ osn ≤ nsqn (rt (σ' sip)) oip
∧ (nsqn (rt (σ' sip)) oip = osn
   → (the (dhops (rt (σ' sip)) oip) ≤ hops
       ∨ the (flag (rt (σ' sip)) oip) = inv))"

using ⟨osn ≥ 1⟩ by (rule quality_increases_rreq_rrep_props [rotated 2])
with ⟨osn ≥ 1⟩ show "msg_fresh σ' m"
by (clarsimp)
qed
next
fix hops dip dsn oip sip
assume [simp]: "m = Rrep hops dip dsn oip sip"
and "msg_fresh σ m"
then have "dsn ≥ 1" and "sip = dip ∨ (dip ∈ kD(rt (σ sip)) ∧ dsn ≤ nsqn (rt (σ sip)) dip
∧ (nsqn (rt (σ sip)) dip = dsn
   → (the (dhops (rt (σ sip)) dip) ≤ hops
       ∨ the (flag (rt (σ sip)) dip) = inv)))"
by auto
from this(2) show "?thesis"
proof
assume "sip = dip" with ⟨dsn ≥ 1⟩ show ?thesis by simp
next
assume "dip ∈ kD(rt (σ sip)) ∧ dsn ≤ nsqn (rt (σ sip)) dip
∧ (nsqn (rt (σ sip)) dip = dsn
   → (the (dhops (rt (σ sip)) dip) ≤ hops
       ∨ the (flag (rt (σ sip)) dip) = inv))"
moreover from qinc have "quality_increases (σ sip) (σ' sip)"..
ultimately have "dip ∈ kD(rt (σ' sip)) ∧ dsn ≤ nsqn (rt (σ' sip)) dip
∧ (nsqn (rt (σ' sip)) dip = dsn
   → (the (dhops (rt (σ' sip)) dip) ≤ hops
       ∨ the (flag (rt (σ' sip)) dip) = inv))"
using ⟨dsn ≥ 1⟩ by (rule quality_increases_rreq_rrep_props [rotated 2])
with ⟨dsn ≥ 1⟩ show "msg_fresh σ' m"
by clarsimp
qed
next
fix dests sip
assume [simp]: "m = Rerr dests sip"
and "msg_fresh σ m"
then have *: "∀ rip ∈ dom(dests). rip ∈ kD(rt (σ sip))
∧ the (dests rip) - 1 ≤ nsqn (rt (σ sip)) rip"
proof
fix rip
assume "rip ∈ dom(dests)"
with * have "rip ∈ kD(rt (σ sip))" and "the (dests rip) - 1 ≤ nsqn (rt (σ sip)) rip"
by -(drule(1) bspec, clarsimp)+
moreover from qinc have "quality_increases (σ sip) (σ' sip)" by simp
ultimately show "rip ∈ kD(rt (σ' sip)) ∧ the (dests rip) - 1 ≤ nsqn (rt (σ' sip)) rip"..
qed
thus ?thesis by simp
qed simp_all
end

1.8 The ‘open’ AODV model

theory A_OAodv
imports A_Aodv ../../../AWN/OAWN_SOS_Labels ../../../AWN/OAWN_Convert
begin
Definitions for stating and proving global network properties over individual processes.
definition σ_AODV' :: "((ip ⇒ state) × ((state, msg, pseqp, pseqp label) seqp)) set"
where "σ_AODV' ≡ {λi. aodv_init i, Γ_AODV PAodv}"

143
abbreviation \( \text{opaodv} \) :: 

\[
\text{"ip } \Rightarrow \text{(ip } \Rightarrow \text{state) } \times \text{(state, msg, pseqp, pseqp label) seqp, msg seq_action) automaton}"
\]

where 

\[
\text{"opaodv } i \equiv \{} \text{ init } = \sigma_{\text{AODV}}', \text{ trans } = \text{oseqp sos } \Gamma_{\text{AODV}} i \ \\}
\]

lemma initiali_aodv [intro!, simp]: "initiali i (init (opaodv i)) (init (paodv i))"

unfolding \( \sigma_{\text{AODV}}\_def \) \( \sigma_{\text{AODV}}'\_def \) by rule simp_all

lemma oaoadv_control_within [simp]: "control_within \( \Gamma_{\text{AODV}} \) (init (opaodv i))"

unfolding \( \sigma_{\text{AODV}}'\_def \) by (rule control_withinI) (auto simp del: \( \Gamma_{\text{AODV}} \_simps \))

lemma \( \sigma_{\text{AODV}}'\_labels \) [simp]: "\( \sigma, p \) \in \sigma_{\text{AODV}}' \Rightarrow \text{labels } \Gamma_{\text{AODV}} p = \{\text{PAodv-:0}\}"

unfolding \( \sigma_{\text{AODV}}'\_def \) by simp

lemma oaoadv_init_kD_empty [simp]: 

\[
\text{"(\sigma, p) } \in \sigma_{\text{AODV}}' \Rightarrow kD (rt (\sigma i)) = \{\}
\]

unfolding \( \sigma_{\text{AODV}}'\_def \) \( \text{kD}_\text{def} \) by simp

lemma oaoadv_init_vD_empty [simp]: 

\[
\text{"(\sigma, p) } \in \sigma_{\text{AODV}}' \Rightarrow vD (rt (\sigma i)) = \{\}
\]

unfolding \( \sigma_{\text{AODV}}'\_def \) \( \text{vD}_\text{def} \) by simp

lemma oaoadv_trans: "trans (opaodv i) = oseqp sos \( \Gamma_{\text{AODV}} \) i"

by simp

declare oseq_invariant_ctermsI [OF aodv_wf oaoadv_control_within aodv_simple_labels oaoadv_trans, cterms_intros]

oseq_step_invariant_ctermsI [OF aodv_wf oaoadv_control_within aodv_simple_labels oaoadv_trans, cterms_intros]

end

1.9 Global invariant proofs over sequential processes

theory A_Global_Invariants

imports A_Seq_Invariants
A_Aodv_Predicates
A_Fresher
A_Quality_Increases
    "../../../AWN/OAWN_Convert"
A_OAodv

begin

lemma other_quality_increases [elim]:

assumes "other quality_increases I \( \sigma \sigma'\)"

shows "\( \forall j. \text{quality_increases } (\sigma j) (\sigma' j)\)"

using assms by (rule, clarsimp) (metis quality_increases_refl)

lemma weaken_otherwith [elim]:

fixes m

assumes *: "otherwith P I \( \text{orecvmsg } Q \) \( \sigma \sigma'\) a"

and weakenP: "\( \\\land \sigma m. P \sigma m \Rightarrow P' \sigma m\)"

and weakenQ: "\( \\\land \sigma m. Q \sigma m \Rightarrow Q' \sigma m\)"

shows "otherwith P' I \( \text{orecvmsg } Q' \) \( \sigma \sigma'\) a"

proof

fix j

assume "j \notin I"

with * have "P (\sigma j) (\sigma' j)" by auto

thus "P' (\sigma j) (\sigma' j)" by (rule weakenP)

next

from * have "\text{orecvmsg } Q \sigma a" by auto

thus "\text{orecvmsg } Q' \sigma a" by rule (erule weakenQ)

qed
lemma oreceived_msg_inv:
assumes other: "∀σ σ' m. \[ P \sigma \equiv P \sigma' \implies P \sigma' m \]
and local: "∀σ m. P σ m \implies P (σ(i := σ i(msg := m))) m"
shows "\( \text{opadv i} \) \( \equiv \) \( \text{otherwith Q \{i\} (\text{orecmsg P}) \), other Q \{i\} \rightarrow \)
\( \text{onl} \Gamma_{\text{AODV}} (\lambda(\sigma, l). l \in \{\text{PAodv-:1}\} \rightarrow P \sigma (msg (\sigma i))) \)"
proof (inv_cterms, intro impI)
fix σ σ'
assume "l = PAodv-:1 \implies P \sigma (msg (\sigma i))"
and "l = PAodv-:1"
and "other Q \{i\} σ σ'"
from this(1-2)
have "P \sigma (msg (\sigma i))" ..

hence "P \sigma' (msg (\sigma i))" using ⟨other Q \{i\} σ σ'⟩
by (rule other)
moreover from ⟨other Q \{i\} σ σ'⟩
have "\(\sigma' i = \sigma i\)" ..

ultimately show "P \sigma' (msg (\sigma' i))" by simp

qed

(Equivalent to) Proposition 7.27

lemma local_quality_increases:
"\( \text{paodv i} \) \( \equiv \) \( \text{recvmsg rreq_rrep_sn} \) \( \rightarrow \) \( \text{onl} \Gamma_{\text{AODV}} (\lambda((\xi, _), _, (\xi', _)). \text{quality_increases } \xi \xi') \)"
proof (rule step_invariantI)
fix s a s'
assume sr: "s \in \text{reachable } (\text{paodv i}) (\text{recvmsg rreq_rrep_sn})"
and tr: "(s, a, s') \in \text{trans } (\text{paodv i})"
and rm: "\text{recvmsg rreq_rrep_sn} a"
from sr have srTT: "s \in \text{reachable } (\text{paodv i}) \text{ TT} ..

from route_tables_fresher sr tr rm
have "\(\text{onl} \Gamma_{\text{AODV}} (\lambda((\xi, _), _, (\xi', _)). \forall dip \in kD \ (\text{rt } \xi). \text{rt } \xi \subseteq dip \rt \xi') (s, a, s') \)"
by (rule step_invariantD)

moreover from known_destinations_increase srTT tr TT_True
have "\(\text{onl} \Gamma_{\text{AODV}} (\lambda((\xi, _), _, (\xi', _)). kD (\text{rt } \xi) \subseteq kD (\text{rt } \xi')) (s, a, s') \)"
by (rule step_invariantD)

moreover from sqns_increase srTT tr TT_True
have "\(\text{onl} \Gamma_{\text{AODV}} (\lambda((\xi, _), _, (\xi', _)). \forall ip. \text{sqn } (\text{rt } \xi) ip \leq \text{sqn } (\text{rt } \xi') ip) (s, a, s') \)"
by (rule step_invariantD)

ultimately show "\(\text{onl} \Gamma_{\text{AODV}} (\lambda((\xi, _), _, (\xi', _)). \text{quality_increases } \xi \xi') (s, a, s') \)"
unfolding onll_def by auto

qed

lemmas olocal_quality_increases =
open_seq_step_invariant [OF local_quality_increases initiali_aodv aodv_trans aodv_trans,
simplified seqll_onll_swap]

lemma oquality_increases:
"\( \text{opadv i} \) \( \equiv \) \( \text{otherwith } \text{quality_increases } \{i\} (\text{orecmsg } (\lambda_. \text{rreq_rrep_sn})) \),
other quality_increases \{i\} \rightarrow \)"
lemma \( \text{rreq\_rrep\_nsqn\_fresh\_any\_step\_invariant} \):

\[
\text{opaodv } i \models_A (\text{act (recvmsg rreq\_rrep\_sn)}, \text{other } A \{i\} \rightarrow)
\]

\[
onll_G^{\text{AODV}} (\lambda((\sigma, \_), a, \_). \text{anycast (msg\_fresh (sigma) a)})
\]

proof (rule ostep\_invariantI, simp del: act\_simp)

\[
\text{fix } p a a' p' l'
\]

assume or: "(\sigma, p) \in \text{oreachable (opaodv } i) ?S (\text{other quality\_increases \{i\})}"

and 11: "l \in \text{labels } \Gamma^{\text{AODV}} p"

and "\text{otherwith}_\Gamma \sigma a"

and tr: "((\sigma, p), a, (\sigma', p')) \in \text{oseq\_sos } \Gamma^{\text{AODV}} i"

and 11': "l' \in \text{labels } \Gamma^{\text{AODV}} p'"

from this(1-3) have "\text{orecvmsg } (\lambda_. \text{rreq\_rrep\_sn}) \sigma a"

by (auto dest!: otherwith\_actionD)

with or have owr: "(\sigma, p) \in \text{oreachable (opaodv } i) (\text{act (recvmsg rreq\_rrep\_sn)})

(\text{other quality\_increases \{i\})}"

by - (erule\_weak\_reachableE, auto)

with tr l l' and \(\text{orecvmsg (\lambda_. \text{rreq\_rrep\_sn}) \sigma a})\) have "\text{quality\_increases } (\sigma i) (\sigma' i)"

by - (drule onll\_ostep\_invariantD [OF olocal\_quality\_increases], auto simp: seqll\_def)

with ?S \sigma \sigma' a show "\text{\forall j. quality\_increases } (\sigma j) (\sigma'( j))"

by (auto dest!: otherwith\_syncD)

qed

lemma \( \text{rreq\_rrep\_nsqn\_fresh\_any\_step\_invariant} \):

"\text{opaodv } i \models_A (\text{act (recvmsg rreq\_rrep\_sn)}, \text{other } A \{i\} \rightarrow)
\]

\[
onll_G^{\text{AODV}} (\lambda((\sigma, \_), a, \_). \text{anycast (msg\_fresh (sigma) a)})
\]

proof (rule ostep\_invariantI, simp del: act\_simp)

\[
\text{fix } p a a' p' l'
\]

assume or: "(\sigma, p) \in \text{oreachable (opaodv } i) (\text{act (recvmsg rreq\_rrep\_sn)}) (\text{other } A \{i\})"

and "((\sigma, p), a, (\sigma', p')) \in \text{oseq\_sos } \Gamma^{\text{AODV}} i"

and \text{recv: } (\text{act (recvmsg rreq\_rrep\_sn)}) (\text{other } A \{i\})

obtain l l' where "\text{\_l\_labels } \Gamma^{\text{AODV}} p" and "\text{\_l\_labels } \Gamma^{\text{AODV}} p'"

by (metis aodv\_ex\_label)

from "((\sigma, p), a, (\sigma', p')) \in \text{oseq\_sos } \Gamma^{\text{AODV}} i"

have tr: "((\sigma, p), a, (\sigma', p')) \in \text{trans (opaodv } i)" by simp

have "\text{anycast } (\text{rreq\_rrep\_fresh } (\text{rt } (\sigma i))) a"

proof -

have "\text{opaodv } i \models_A (\text{act (recvmsg rreq\_rrep\_sn)}, \text{other } A \{i\} \rightarrow)
\]

\[
onll_G^{\text{AODV}} (\text{seqll } l (\lambda((\xi, \_), a, \_). \text{anycast (rreq\_rrep\_fresh (rt } \xi))) a))
\]

by (rule ostep\_invariant\_weak\_E [OF

open\_seq\_step\_invariant [OF rreq\_rrep\_fresh\_any\_step\_invariant\_initiali\_aodv, simplified seqll\_onll\_swap]]) auto

hence "\text{onll } \Gamma^{\text{AODV}} (\text{seqll } l (\lambda((\xi, \_), a, \_). \text{anycast (rreq\_rrep\_fresh (rt } \xi))) a))

((\sigma, p), a, (\sigma', p'))"

using or tr \text{recv} by - (erule(4) ostep\_invariant\_E)

thus \text{?thesis}

using \_l\_labels \Gamma^{\text{AODV}} p and \_l\_labels \Gamma^{\text{AODV}} p' by auto

qed

moreover have "\text{anycast } (\text{rerr\_invalid } (\text{rt } (\sigma i))) a"

proof -

have "\text{opaodv } i \models_A (\text{act (recvmsg rreq\_rrep\_sn)}, \text{other } A \{i\} \rightarrow)
\]

\[
onll_G^{\text{AODV}} (\text{seqll } l (\lambda((\xi, \_), a, \_). \text{anycast (rerr\_invalid (rt } \xi))) a))
\]

by (rule ostep\_invariant\_weak\_E [OF

open\_seq\_step\_invariant [OF rerr\_invalid\_any\_step\_invariant\_initiali\_aodv, simplified seqll\_onll\_swap]]) auto

hence "\text{onll } \Gamma^{\text{AODV}} (\text{seqll } l (\lambda((\xi, \_), a, \_). \text{anycast (rerr\_invalid (rt } \xi))) a))

((\sigma, p), a, (\sigma', p'))"

using or tr \text{recv} by - (erule(4) ostep\_invariant\_E)

thus \text{?thesis}

using \_l\_labels \Gamma^{\text{AODV}} p and \_l\_labels \Gamma^{\text{AODV}} p' by auto

qed

moreover have "\text{anycast } \text{rreq\_rrep\_sn a}"

146
proof -
from or tr recv
have "onll \(\Gamma_{AODV}\) (seqll i (\(\lambda(\_\_\_, a, \_\_\_). ancast rreq_rrep_sn a)) ((\(\sigma\), p), a, (\(\sigma'\), p')))"
  by (rule ostep_invariantE [OF
open_seq_step_invariant [OF rreq_rrep_sn_any_step_invariant initiali_aodv
  oadv_trans adv_trans,
simplified seqll_onll_swap]])
thus ?thesis
using \(\langle l \in \text{labels} \Gamma_{AODV} p \rangle\) and \(\langle l' \in \text{labels} \Gamma_{AODV} p' \rangle\) by auto
qed

moreover have "anycast (\(\lambda m. \text{not}_Pkt m \rightarrow \text{msg_sender} m = i\)) a"
proof -
  have "opaodv i \|=A (act (recvmsg rreq_rrep_sn), other A (\{i\} \rightarrow)
onll \(\Gamma_{AODV}\) (seqll i (\(\lambda((\xi, \_\_), a, \_\_\_). ancast (\(\lambda m. \text{not}_Pkt m \rightarrow \text{msg_sender} m = i\)) a)))"
    by (rule ostep_invariant_weakenE [OF
open_seq_step_invariant [OF sender_ip_valid initiali_aodv,
simplified seqll_onll_swap]]) auto
thus ?thesis using or tr recv \(\langle l \in \text{labels} \Gamma_{AODV} p \rangle\) and \(\langle l' \in \text{labels} \Gamma_{AODV} p' \rangle\)
by - (drule(3) onll_ostep_invariantD, auto)
qed

ultimately have "anycast (msg_fresh \(\sigma\)) a"
by (simp_all add: anycast_def
del: msg_fresh
split: seq_action.split_asm msg.split_asm) simp_all
thus "onll \(\Gamma_{AODV}\) (\(\lambda((\sigma, \_\_), a, \_\_\_). ancast (msg_fresh \(\sigma\)) a)) ((\(\sigma\), p), a, (\(\sigma'\), p'))"
by auto
qed

lemma oreceived_rreq_rrep_nsqn_fresh_inv:
"opaodv i \|= (otherwith quality_increases \(\{i\}\) (orecvmsg msg_fresh),
other quality_increases \(\{i\}\) \rightarrow)
onll \(\Gamma_i\) (\(\lambda(\sigma, 1). l \in \{PAodv:-1\} \rightarrow \text{msg_fresh} \(\sigma\) (msg (\(\sigma\) i))))"
proof (rule oreceived_msg_inv)
fix \(\sigma \sigma' m\)
assume "msg_fresh \(\sigma\) \(m\)"
and "other quality_increases \(\{i\}\) \(\sigma\) \(\sigma'\)"
from this(2) have "\(\forall j. \text{quality_increases} (\(\sigma\) j) (\(\sigma'\) j)\) ..
thus "msg_fresh \(\sigma'\) \(m\)" using * ..
next
fix \(\sigma \ m\)
assume "msg_fresh \(\sigma\) \(m\)"
thus "msg_fresh (\(\sigma\) (i := \(\sigma\) (msg := \(m\))) \(m\))"
proof (cases \(m\))
fix dests sip
assume "m = Rerr dests sip"
with \(\langle \text{msg_fresh} \(\sigma\) \(m\) \rangle\) show ?thesis by auto
qed auto

lemma oquality_increases_nsqn_fresh:
"opaodv i \|= \(otherwith quality_increases \{i\}\) (orecvmsg msg_fresh),
other quality_increases \(\{i\}\) \rightarrow)
onll \(\Gamma_i\) (\(\lambda((\sigma, \_\_), \_\_, (\sigma', \_\_\_)). \forall j. \text{quality_increases} (\(\sigma\) j) (\(\sigma'\) j)))"
by (rule ostep_invariant_weakenE [OF oquality_increases]) auto

lemma oosn_rreq:
"opaodv i \|= \(otherwith quality_increases \{i\}\) (orecvmsg msg_fresh),
other quality_increases \(\{i\}\) \rightarrow)
onll \(\Gamma_i\) (seqll i (\(\lambda(\xi, 1). l \in \{PAodv:-4, PAodv:-5\} \cup \{PRreq:n \in \text{True}\} \rightarrow 1 \leq \text{osn} (\xi)\)))"
by (rule oinvariant_weakenE [OF open_seq_invariant [OF osn_rreq initiali_aodv]])
(auto simp: seqll_onll_swap)
lemma rreq_sip:
"opaodv i \models (otherwith quality_increases \{i\} \{orecmsg msg_fresh\},
\text{other quality_increases \{i\} \rightarrow})
\begin{align*}
onl \Gamma_{AODV} (\lambda (\sigma, i). \\
(\ell \in \{PAodv-:4, PAodv-:5, PRreq-:0, PRreq-:2\} \land \text{sip} (\sigma, i) \neq \text{oip} (\sigma, i)) \\
\quad \rightarrow \text{oip} (\sigma, i) \in kD (rt (\sigma (\text{sip} (\sigma, i)))) \\
\quad \land \text{nsqn} (rt (\sigma (\text{sip} (\sigma, i)))) (\text{oip} (\sigma, i)) \geq \text{osn} (\sigma, i) \\
\quad \land (\text{nsqn} (rt (\sigma (\text{sip} (\sigma, i)))) (\text{oip} (\sigma, i)) = \text{osn} (\sigma, i)) \\
\quad \rightarrow \text{(hops} (\sigma, i) \geq \text{the} (\text{dhops} (rt (\sigma (\text{sip} (\sigma, i)))) (\text{oip} (\sigma, i))) \\
\quad \lor \text{the} (\text{flag} (rt (\sigma (\text{sip} (\sigma, i)))) (\text{oip} (\sigma, i))) = \text{inv}))" \\
\end{align*}
(is \(_\models (?S, ?U \rightarrow \_)
\)
proof (inv_cterms inv add: oseq_step_invariant_sterms \{OF quality_increases_nsqn_fresh
\ aodv_wf oadodv_trans
\ onl_o invariant_sterms \{OF aodv_wf oereceived_rreq_rrep_nsqn_fresh_inv
\ onl_o invariant_sterms \{OF aodv_wf oosn_rreq
\ simp add: seqlsimp
\ simp del: \text{One}\_nat\_def, rule \text{implI})
\fix \sigma \sigma' p l
\assume \"(\sigma, p) \in \text{oreachable} (opaodv i) ?S ?U"
\and \"l \in \text{labels} \Gamma_{AODV} p"
\and \text{pre:
\begin{align*}
(1 = PAodv-:4 \lor 1 = PAodv-:5 \lor 1 = PRreq-:0 \lor 1 = PRreq-:2) \land \text{sip} (\sigma, i) \neq \text{oip} (\sigma, i) \\
\quad \rightarrow \text{oip} (\sigma, i) \in kD (rt (\sigma (\text{sip} (\sigma, i)))) \\
\quad \land \text{osn} (\sigma, i) \leq \text{nsqn} (rt (\sigma (\text{sip} (\sigma, i)))) (\text{oip} (\sigma, i)) \\
\quad \land (\text{nsqn} (rt (\sigma (\text{sip} (\sigma, i)))) (\text{oip} (\sigma, i)) = \text{osn} (\sigma, i)) \\
\quad \rightarrow \text{the} (\text{dhops} (rt (\sigma (\text{sip} (\sigma, i)))) (\text{oip} (\sigma, i))) \leq \text{hops} (\sigma, i) \\
\quad \lor \text{the} (\text{flag} (rt (\sigma (\text{sip} (\sigma, i)))) (\text{oip} (\sigma, i))) = \text{inv})" \\
\end{align*}
and \text{other quality_increases \{i\} \sigma' i"
\and \text{hyp:
\begin{align*}
(1 = PAodv-:4 \lor 1 = PAodv-:5 \lor 1 = PRreq-:0 \lor 1 = PRreq-:2) \land \text{sip} (\sigma', i) \neq \text{oip} (\sigma', i)" \\
(is ?labels \land \text{sip} (\sigma', i) \neq \text{oip} (\sigma', i)"
from (4) have \"\sigma', i = \sigma i ..
with \text{hyp have hyp': ?labels \land \text{sip} (\sigma, i) \neq \text{oip} (\sigma, i)" by simp
\show \text{oip} (\sigma', i) \in kD (rt (\sigma' (\text{sip} (\sigma', i)))) \\
\land (\text{osn} (\sigma', i) \leq \text{nsqn} (rt (\sigma' (\text{sip} (\sigma', i)))) (\text{oip} (\sigma', i))) \\
\land (\text{nsqn} (rt (\sigma' (\text{sip} (\sigma', i)))) (\text{oip} (\sigma', i)) = \text{osn} (\sigma', i)) \\
\rightarrow \text{the} (\text{dhops} (rt (\sigma' (\text{sip} (\sigma', i)))) (\text{oip} (\sigma', i))) \leq \text{hops} (\sigma', i) \\
\lor \text{the} (\text{flag} (rt (\sigma' (\text{sip} (\sigma', i)))) (\text{oip} (\sigma', i))) = \text{inv})" \\
\end{align*}
proof (cases \"\text{sip} (\sigma, i) = i\")
\assume \text{\"\text{sip} (\sigma, i) \neq i\"}
\from (other quality_increases \{i\} \sigma' i;
\have \text{\"quality_increases \{\sigma (\text{sip} (\sigma, i))\} \sigma' (\text{sip} (\sigma', i))\"
\by (rule otherE) (clarsimp simp:
\text{\"\text{sip} (\sigma, i) \neq i\")
\moreover \text{from \{\sigma, p\} \in \text{oreachable} (opaodv i) ?S ?U l \in \text{labels} \Gamma_{AODV} p\} and \text{hyp
\have \text{\"l \leq \text{osn} (\sigma, i)\"}
\by (auto dest!: \text{onl_o invariant_weakenD} \{OF oosn_rreq
\simp add: seqlsimp \text{'}\sigma' i = \sigma i\:
\moreover \text{from \text{sip} (\sigma, i) \neq i\ hyp' and \text{pre
\have \text{\"\text{oip} (\sigma', i) \in kD (rt (\sigma' (\text{sip} (\sigma, i)))) \\
\land (\text{osn} (\sigma', i) \leq \text{nsqn} (rt (\sigma' (\text{sip} (\sigma, i)))) (\text{oip} (\sigma', i))) \\
\land (\text{nsqn} (rt (\sigma' (\text{sip} (\sigma, i)))) (\text{oip} (\sigma', i)) = \text{osn} (\sigma', i)) \\
\rightarrow \text{the} (\text{dhops} (rt (\sigma' (\text{sip} (\sigma, i)))) (\text{oip} (\sigma', i))) \leq \text{hops} (\sigma', i) \\
\lor \text{the} (\text{flag} (rt (\sigma' (\text{sip} (\sigma, i)))) (\text{oip} (\sigma', i))) = \text{inv})" \\
\by (auto simp:
\text{'}\sigma' i = \sigma i\:
\ultimately \text{show \text{'}\text{?thesis
\by (rule quality_increases_rreq_rrep_props
\next
\assume \text{\"\text{sip} (\sigma, i) = i\") thus \text{'}\text{?thesis
\using \text{'}\sigma' i = \sigma i\ hyp and \text{pre by auto
\qed
\qed (auto elim!: \text{quality_increases_rreq_rrep_props'})

lemma odsn_rrep:
"opaodv i \models (otherwith quality_increases \{i\} \{orecmsg msg_fresh\},
\text{other quality_increases \{i\} \rightarrow)
lemma rrep_sip:
"opaodv i |∈ (otherwith quality_increases {i} (orecvmsg msg_fresh),
other quality_increases {i} →)
onl Γ_AODV (λ(σ, 1).
(1 ∈ {PAdv−:6, PAdv−:7, PRep−:0, PRep−:1} ∧ sip (σ i) ≠ dip (σ i))
→ dip (σ i) ∈ kD (rt (σ (sip (σ i))))
∧ dsn (σ i) ≤ nsqn (rt (σ (sip (σ i)))) (dip (σ i))
∧ (nsqn (rt (σ (sip (σ i)))) (dip (σ i)) = dsn (σ i))
→ (dips (σ i) ≥ the (dhops (rt (σ (sip (σ i)))) (dip (σ i)))
∧ the (flag (rt (σ (sip (σ i)))) (dip (σ i))) = inv))"
(is "_ |∈ (?S, ?U → _)")
proof (inv_cterms inv add: oseq_step_invariant_sterms [OF quality_increases_nsqn_fresh aodv_wf
aoadv_trans]
onl_onlvariant_sterms [OF aodv_wf oreceived_rreq_rrep_nsqn_fresh_inv]
onl_onlvariant_sterms [OF aodv_wf ondr_nsq]
and pre:
"(l = PAodv−:6 ∨ l = PAodv−:7 ∨ l = PRep−:0 ∨ l = PRep−:1) ∧ sip (σ i) ≠ dip (σ i))
→ dip (σ i) ∈ kD (rt (σ (sip (σ i))))
∧ dsn (σ i) ≤ nsqn (rt (σ (sip (σ i)))) (dip (σ i))
∧ (nsqn (rt (σ (sip (σ i)))) (dip (σ i)) = dsn (σ i))
→ (dips (σ i) ≥ the (dhops (rt (σ (sip (σ i)))) (dip (σ i)))
∧ the (flag (rt (σ (sip (σ i)))) (dip (σ i))) = inv)"
and "other quality_increases {i} σ σ’"
and hyp: "(l=PA0dv−:6 ∨ l=PA0dv−:7 ∨ l=PRep−:0 ∨ l=PRep−:1) ∧ sip (σ i) ≠ dip (σ’ i)"
(is "?labels ∧ sip (σ i) ≠ dip (σ’ i)")
from this(4) have "σ’ i = σ i" ..
with hyp have hyp’: "?labels ∧ sip (σ i) ≠ dip (σ i)" by simp
show "dip (σ’ i) ∈ kD (rt (σ’ (sip (σ’ i))))
∧ dsn (σ’ i) ≤ nsqn (rt (σ’ (sip (σ’ i)))) (dip (σ’ i))
∧ (nsqn (rt (σ’ (sip (σ’ i)))) (dip (σ’ i)) = dsn (σ’ i))
→ (dips (σ’ i) ≥ the (dhops (rt (σ’ (sip (σ’ i)))) (dip (σ’ i)))
∧ the (flag (rt (σ’ (sip (σ’ i)))) (dip (σ’ i))) = inv)"
proof (cases "sip (σ i) = i")
assume "sip (σ i) ≠ i"
from "other quality_increases {i} σ σ’"
have "quality_increases (σ (sip (σ i))) (σ’ (sip (σ’ i)))"
by (rule otds) (clarsimp simp: sip (σ i) ≠ i).
moreover from ⟨(σ, p) ∈ reachable (opaodv i) ?S ?U ; l ∈ labels Γ_AODV p⟩ and hyp
have "1 ≤ dsn (σ i)"
by (auto dest!: onl_onlvariant_weakenD [OF ondr_nsq]
simp add: seqlsimp ⟨σ i = σ i⟩)
moreover from ⟨sip (σ i) ≠ i⟩ hyp’ and pre
have "dip (σ’ i) ∈ kD (rt (σ’ (sip (σ’ i))))
∧ dsn (σ’ i) ≤ nsqn (rt (σ’ (sip (σ’ i)))) (dip (σ’ i))
∧ (nsqn (rt (σ’ (sip (σ’ i)))) (dip (σ’ i)) = dsn (σ’ i))
→ (dhops (σ i) ≥ the (dhops (rt (σ’ (sip (σ’ i)))) (dip (σ’ i))) ≤ hops (σ i)
∧ the (flag (rt (σ’ (sip (σ’ i)))) (dip (σ’ i))) = inv)"
by (auto simp: ⟨σ i = σ i⟩)
ultimately show ?thesis
by (rule otds) (clarsimp simp: quality_increases_rreq_rrep_props)
next
assume "sip (σ i) = i" thus ?thesis
using ⟨σ i = σ i⟩ hyp and pre by auto
qed
qed (auto simp add: seqlsimp elim!: quality_increases_rreq_rrep_props)"
lemma rerr_sip:
"opaodv \( i \models \) (otherwith quality_increases \( \{ i \} \) (orecvmsg msg_fresh),
other quality_increases \( \{ i \} \) →)
onl \( \Gamma_{OADV} (\lambda(\sigma, 1). \)
\( l \in \{PAodv:-8, PAodv:-9, PRerr:-0, PRerr:-1\}
\rightarrow (\forall ripc\in\text{dom(dests} (\sigma \ i)). ripc\in\text{kD}(rt (\sigma (sip} (\sigma \ i)))) \land
the (\text{dests} (\sigma \ i) ripc) - 1 \leq nsqn (rt (\sigma (sip} (\sigma \ i)))) ripc))"
(is 
- \( (\sigma_i, \sigma_j \rightarrow \) _")
proof -
{ fix dests rip sip rsn and \( \sigma \ \sigma' :: \text{"ip} \Rightarrow \text{state}" \)
assume qinc: "\( \forall \ j. \ \text{quality_increases} (\sigma \ j) (\sigma' \ j)\)"
and *: "\( \forall rip \in \text{dom dests}. \ \text{rip} \in \text{kD}(rt (\sigma \sip)) \land \text{the (dests \ rip)} - 1 \leq \text{nsqn}\ (rt (\sigma \sip)) \ \text{rip}\)"
from this(3) have "\( \text{rip} \in \text{dom dests} \) by auto
with * and (dests \ rip = Some \ rsn) have "\( \text{rip} \in \text{kD}(rt (\sigma \sip))\)"
and "\( \text{rsn} - 1 \leq \text{nsqn}\ (rt (\sigma \sip)) \ \text{rip}\)"
proof from ⟨\( \text{rip} \in \text{kD}(rt (\sigma \sip))\)⟩ and ⟨\( \text{quality_increases} (\sigma \sip) (\sigma' \ sip)\)⟩ show "\( \text{rip} \in \text{kD}(rt (\sigma' \ sip))\)" ..
next from ⟨\( \text{rip} \in \text{kD}(rt (\sigma \sip))\)⟩ and ⟨\( \text{quality_increases} (\sigma \sip) (\sigma' \ sip)\)⟩
have "\( \text{nsqn}\ (rt (\sigma \sip)) \ \text{rip} \leq \text{nsqn}\ (rt (\sigma' \ sip)) \ \text{rip}\)" ..
with \( \text{rsn} - 1 \leq \text{nsqn}\ (rt (\sigma \sip)) \ \text{rip}\) show "\( \text{rsn} - 1 \leq \text{nsqn}\ (rt (\sigma' \ sip)) \ \text{rip}\)"
by (rule le_trans)
qed
} note partial = this

show ?thesis
by (inv_cterms inv add: oseq_step_invariant_sterms [OF \( \text{quality_increases}_\text{nsqn}_\text{fresh} \ \text{aodv}_\text{wf} \ \text{aodv}_\text{trans} \)]
onl \( \text{oaln}_\text{oinvariant}_\text{sterms} [\text{OF} \ \text{aodv}_\text{wf} \ \text{oreceived}_\text{rreq}_\text{rrep}_\text{nsqn}_\text{fresh}_\text{inv}] \)\nother_quality_increases other_localD
simp del: One_nat_def, intro conj)
- (clarsimp simp del: One_nat_def split: split_if_asm option.split_asm, erule(2) partial)+
qed

lemma prerr_guard:
"opaodv \( i \models \) \( (\sigma, 1 = \text{PRerr-1}\)
onl \( \Gamma_{OADV} (\lambda(\xi, l). \ (l = \text{PRerr-1}) \rightarrow (\forall ip\in\text{dom dests} \ \xi) \land \text{ip}\in\text{vD}(rt \ \xi)) \land \text{the (nhop (rt \ \xi) \ ip) = sip} \ \xi \land \text{sqn (rt \ \xi) \ ip < the (dests \ \xi \ ip}))\)"
by (inv_cterms) (clarsimp split: option.split_asm split_if_asm)

lemmas oaddpreRT_welldefined =
open_seq_invariant [OF \( \text{addpreRT}_\text{welldefined} \ \text{initiali}_\text{aodv} \ \text{oaodv}_\text{trans} \ \text{aodv}_\text{trans} \) simplified seql_onl_swap, THEN oinvariant_anyact]

lemmas odests_vD_inc_sqn =
open_seq_invariant [OF \( \text{dests}_\text{vD}_\text{inc}_\text{sqn} \ \text{initiali}_\text{aodv} \ \text{oaodv}_\text{trans} \ \text{aodv}_\text{trans} \) simplified seql_onl_swap, THEN oinvariant_anyact]

lemmas oprerr_guard =
open_seq_invariant [OF \( \text{prerr}_\text{guard} \ \text{initiali}_\text{aodv} \ \text{oaodv}_\text{trans} \ \text{aodv}_\text{trans} \) simplified seql_onl_swap, THEN oinvariant_anyact]

Proposition 7.28

lemma seq_compare_next_hop':
"opadv i |= (otherwith quality_increases {i} (orecmsg msg_fresh),
obother quality_increases {i} → onl l'_{AODV} (λσ, _).

\forall dip. let nhip = the (nhop (rt (σ i))) dip
\in dip ∈ kD(rt (σ i)) ∧ nhip ≠ dip →
dip ∈ kD(rt (σ nhip)) ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ nhip)) dip"
(is "_ |= (?S, ?U → _)"
proof -

{ fix nhop and σ σ' :: "ip ⇒ state"
assume pre: "∀ dip∈kD(rt (σ i)). nhop dip ≠ dip →
dip∈kD(rt (σ (nhop dip))) ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (nhop dip))) dip"
and qinc: "∀ j. quality_increases (σ j) (σ' j)"
have "∀ dip∈kD(rt (σ i)). nhop dip ≠ dip →
dip∈kD(rt (σ' (nhop dip))) ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (nhop dip))) dip"
proof (intro ballI impI)
fix dip
assume "dip∈kD(rt (σ i))" and "nhop dip ≠ dip"
with pre have "dip∈kD(rt (σ (nhop dip)))" and "nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (nhop dip))) dip"
by auto
from qinc have qinc_nhosp: "quality_increases (σ (nhop dip)) (σ' (nhop dip))" ..
with (dip∈kD(rt (σ (nhop dip)))) have "dip∈kD (rt (σ' (nhop dip)))" ..
moreover have "nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (nhop dip))) dip"
proof -
from (dip∈kD(rt (σ (nhop dip)))) qinc_nhosp
have "nsqn (rt (σ (nhop dip))) dip ≤ nsqn (rt (σ' (nhop dip))) dip" ..
with nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (nhop dip))) dip show ?thesis
by simp
qed

ultimately show "dip∈kD(rt (σ' (nhop dip))) ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (nhop dip))) dip" ..
qed
} note basis = this

{ fix nhop and σ σ' :: "ip ⇒ state"
assume pre: "∀ dip∈kD(rt (σ i)). nhop dip ≠ dip → dip∈kD(rt (σ (nhop dip)))
∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (nhop dip))) dip"
and ndest: "∀ ripc∈dom (dests (σ i)). ripc ∈ kD (rt (σ (sip (σ i))))
∧ the (dests (σ i) ripc) - 1 ≤ nsqn (rt (σ (sip (σ i)))) ripc"
and issip: "∀ ip∈dom (dests (σ i)). nhop ip = sip (σ i)"
and qinc: "∀ j. quality_increases (σ j) (σ' j)"
have "∀ dip∈kD(rt (σ i)). nhop dip ≠ dip → dip ∈ kD (rt (σ' (nhop dip)))
∧ nsqn (invalidate (rt (σ i)) (dests (σ i))) dip ≤ nsqn (rt (σ' (nhop dip))) dip"
proof (intro ballI impI)
fix dip
assume "dip∈kD(rt (σ i))" and "nhop dip ≠ dip"
with pre and qinc have "dip∈kD(rt (σ' (nhop dip)))" and "nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (nhop dip))) dip"
by (auto dest!: basic)

have "nsqn (invalidate (rt (σ i)) (dests (σ i))) dip ≤ nsqn (rt (σ' (nhop dip))) dip"
proof (cases "dip∈dom (dests (σ i))")
assume "dip∈dom (dests (σ i))"
with (dip∈kD(rt (σ i))) obtain dsn where "dests (σ i) dip = Some dsn"
by auto
with (dip∈kD(rt (σ i))) have "nsqn (invalidate (rt (σ i)) (dests (σ i))) dip = dsn - 1"
by (rule nsqn_invalidate_eq)
moreover have "dsn - 1 ≤ nsqn (rt (σ' (nhop dip))) dip"
proof -
from (dests (σ i) dip = Some dsn) have "the (dests (σ i) dip) = dsn" by simp
with ndest and \( \text{dip} \in \text{dom} (\text{dests}(\sigma\ i)) \) have "\( \text{dip} \in \text{kD} (\text{rt}(\sigma\ (\text{sip}(\sigma\ i)))) \)"
  "\( \text{dsn} - 1 \leq \text{nsqn}(\text{rt}(\sigma\ (\text{sip}(\sigma\ i)))) \) dip" by auto
moreover from issip and \( \text{dip} \in \text{dom} (\text{dests}(\sigma\ i)) \) have "\( \text{nhop} = \text{sip}(\sigma\ i) \)"
ultimately have "\( \text{dip} \in \text{kD} (\text{rt}(\sigma\ (\text{nhop} \text{ dip}))) \)"
and "\( \text{dsn} - 1 \leq \text{nsqn}(\text{rt}(\sigma\ (\text{nhop} \text{ dip}))) \) dip" by auto
with qinc show "\( \text{dsn} - 1 \leq \text{nsqn}(\text{rt}(\sigma'\ (\text{nhop} \text{ dip}))) \) dip"
by simp (metis kD_kDnsqn_quality_increases_trans)
qed
ultimately show ?thesis by simp
next
assume "\( \text{dip} \notin \text{dom} (\text{dests}(\sigma\ i)) \)"
with \( \text{dip} \in \text{kD}(\text{rt}(\sigma\ i)) \)
  have "\( \text{nsqn} (\text{invalidate}(\text{rt}(\sigma\ i)) (\text{dests}(\sigma\ i))) \) dip = \( \text{nsqn}(\text{rt}(\sigma\ i)) \) dip" by (rule nsqn_invalidate_other)
with \( \text{nsqn}(\text{rt}(\sigma\ i)) \) dip \( \leq \text{nsqn}(\text{rt}(\sigma'\ (\text{nhop} \text{ dip}))) \) dip show ?thesis by simp
qed
with \( \text{dip} \in \text{kD}(\text{rt}(\sigma'\ (\text{nhop} \text{ dip}))) \)
  show "\( \text{dip} \in \text{kD} (\text{rt}(\sigma'\ (\text{nhop} \text{ dip}))) \)"
  "\( \& \text{nsqn}(\text{invalidate}(\text{rt}(\sigma\ i)) (\text{dests}(\sigma\ i))) \) dip \( \leq \text{nsqn}(\text{rt}(\sigma'\ (\text{nhop} \text{ dip}))) \) dip" ..
qed
}

\text{note basic_prerr = this}

\{ fix \( \sigma\ \sigma' \) :: "\( \text{ip} \Rightarrow \text{state} \)"
assume a1: "\( \forall \text{dip}(\text{kD}(\text{rt}(\sigma\ i))).\ \text{the}(\text{nhop}(\text{rt}(\sigma\ i)) \text{ dip}) \neq \text{dip} \Rightarrow \text{dip}(\text{kD}(\text{rt}(\sigma\ (\text{the}(\text{nhop}(\text{rt}(\sigma\ i)) \text{ dip))))) \& \text{nsqn}(\text{rt}(\sigma\ i)) \text{ dip} \leq \text{nsqn}(\text{rt}(\sigma\ (\text{the}(\text{nhop}(\text{rt}(\sigma\ i)) \text{ dip))))) \text{ dip}"\)
  and a2: "\( \forall \text{j}.\ \text{quality}\_\text{increases}(\sigma\ j) (\sigma'\ j) \)"
have "\( \forall \text{dip}(\text{kD}(\text{rt}(\sigma\ i))).\ \text{the}(\text{nhop}(\text{update}(\text{rt}(\sigma\ i)) (\text{sip}(\sigma\ i)) (0, \text{unk}, \text{val}, \text{Suc} 0, \text{sip}(\sigma\ i), \{\})) \text{ dip}) \neq \text{dip} \Rightarrow \text{dip}(\text{kD}(\text{rt}(\sigma'\ (\text{the}(\text{nhop}(\text{rt}(\sigma\ i)) \text{ dip))))) \& \text{nsqn}(\text{update}(\text{rt}(\sigma\ i)) (\text{sip}(\sigma\ i)) (0, \text{unk}, \text{val}, \text{Suc} 0, \text{sip}(\sigma\ i), \{))) \text{ dip} \leq \text{nsqn}(\text{rt}(\sigma'\ (\text{the}(\text{nhop}(\text{rt}(\sigma\ i)) \text{ dip}))) \text{ dip})" (is "\( \forall \text{dip}(\text{kD}(\text{rt}(\sigma\ i))). \text{?P dip}"\)
proof
fix dip
assume "\( \text{dip}(\text{kD}(\text{rt}(\sigma\ i)) \)"
with a1 and a2
  have "\( \text{the}(\text{nhop}(\text{rt}(\sigma\ i)) \text{ dip}) \neq \text{dip} \Rightarrow \text{dip}(\text{kD}(\text{rt}(\sigma'\ (\text{the}(\text{nhop}(\text{rt}(\sigma\ i)) \text{ dip))))) \& \text{nsqn}(\text{rt}(\sigma\ i)) \text{ dip} \leq \text{nsqn}(\text{rt}(\sigma'\ (\text{the}(\text{nhop}(\text{rt}(\sigma\ i)) \text{ dip}))) \text{ dip})" by -(drule(1) basic, auto)
thus "\( \text{?P dip} \)" by (cases "\( \text{dip} = \text{sip}(\sigma\ i) \)") auto
qed
}

\text{note nhop_update_sip = this}

\{ fix \( \sigma\ \sigma'\ \text{oip} \text{ sip osn hops} \)
assume pre: "\( \forall \text{dip}(\text{kD}(\text{rt}(\sigma\ i))).\ \text{the}(\text{nhop}(\text{rt}(\sigma\ i)) \text{ dip}) \neq \text{dip} \Rightarrow \text{dip}(\text{kD}(\text{rt}(\sigma'\ (\text{the}(\text{nhop}(\text{rt}(\sigma\ i)) \text{ dip))))) \& \text{nsqn}(\text{rt}(\sigma\ i)) \text{ dip} \leq \text{nsqn}(\text{rt}(\sigma'\ (\text{the}(\text{nhop}(\text{rt}(\sigma\ i)) \text{ dip}))) \text{ dip})"\)
  and qinc: "\( \forall \text{j}.\ \text{quality}\_\text{increases}(\sigma\ j) (\sigma'\ j) \)"
  and *: "\( \text{sip} \neq \text{oip} \Rightarrow \text{oip}(\text{kD}(\text{rt}(\sigma\ \text{sip})) \& \text{osn} \leq \text{nsqn}(\text{rt}(\sigma\ \text{sip})) \text{ oip} \& (\text{nsqn}(\text{rt}(\sigma\ \text{sip})) \text{ oip} = \text{osn} \Rightarrow \text{the}(\text{dhops}(\text{rt}(\sigma\ \text{sip})) \text{ oip}) \leq \text{hops} \& \text{the}(\text{flag}(\text{rt}(\sigma\ \text{sip})) \text{ oip}) = \text{inv})\)"
from pre and qinc
have pre': "\( \forall \text{dip}(\text{kD}(\text{rt}(\sigma\ i))).\ \text{the}(\text{nhop}(\text{rt}(\sigma\ i)) \text{ dip}) \neq \text{dip} \Rightarrow \text{dip}(\text{kD}(\text{rt}(\sigma'\ (\text{the}(\text{nhop}(\text{rt}(\sigma\ i)) \text{ dip))))) \& \text{nsqn}(\text{rt}(\sigma\ i)) \text{ dip} \leq \text{nsqn}(\text{rt}(\sigma'\ (\text{the}(\text{nhop}(\text{rt}(\sigma\ i)) \text{ dip}))) \text{ dip})" by (rule basic)
have "(the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})) oip) ≠ oip → oip∈kD(rt (σ' (the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})) oip))) ∧ nsqn (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})) oip ≤ nsqn (rt (σ' (the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})) oip))) oip)"
(is "?nhop_not_oip → ?oip_in_kD ∧ ?nsqn_le_nsqn")
proof (rule, split update_rt_split_asm)
assume "rt (σ i) = update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})"
and "the (nhop (rt (σ i)) oip) ≠ oip"
with pre' show "?oip_in_kD ∧ ?nsqn_le_nsqn" by auto
next
assume rtnot: "rt (σ i) ≠ update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})"
and notoip: "?nhop_not_oip" with * qinc
have "?oip_in_kD" by (clarsimp elim!: kD_quality_increases)
moreover with * pre qinc rtnot notoip
have "?nsqn_le_nsqn" by simp (metis kD_nsqn_quality_increases_trans)
ultimately show "?oip_in_kD ∧ ?nsqn_le_nsqn"..
qed

} note update1 = this

{ fix σ σ' oip sip osn hops
  assume pre: "∀ dip∈kD (rt (σ i)). the (nhop (update (rt (σ i)) dip) oip (osn, kno, val, Suc hops, sip, {})) dip ≠ dip
  → dip∈kD(rt (σ (the (nhop (rt (σ i)) dip))) ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (the (nhop (rt (σ i)) dip)))) dip)"
  and qinc: "∀ j. quality_increases (σ j) (σ' j)"
  and*: "sip ≠ oip → oip∈kD(rt (σ sip)) ∧ osn ≤ nsqn (rt (σ sip)) oip ∧ (nsqn (rt (σ sip)) oip = osn
  → the (dhops (rt (σ sip)) oip) ≤ hops ∨ the (flag (rt (σ sip)) oip) = inv)"

  from pre and qinc
  have pre': "∀ dip∈kD (rt (σ i)). the (nhop (update (rt (σ i)) dip) oip (osn, kno, val, Suc hops, sip, {})) dip ≠ dip
  → dip∈kD(rt (σ' (the (nhop (rt (σ i)) dip)))) ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (the (nhop (rt (σ i)) dip)))) dip)"
  by (rule basic)
  have "∀ dip∈kD(rt (σ i)). the (nhop (update (rt (σ i)) dip) oip (osn, kno, val, Suc hops, sip, {})) dip ≠ dip
  → dip∈kD(rt (σ' (the (nhop (update (rt (σ i)) dip))) oip (osn, kno, val, Suc hops, sip, {})) dip)
  ∧ nsqn (update (rt (σ i)) dip oip (osn, kno, val, Suc hops, sip, {})) dip ≤ nsqn (rt (σ' (the (nhop (update (rt (σ i)) dip)) oip
  (osn, kno, val, Suc hops, sip, {})) dip))) dip"
  (is "∀ dip∈kD(rt (σ i)). _ → ?dip_in_kD dip ∧ ?nsqn_le_nsqn dip")
  proof (intro ballI impI, split update_rt_split_asm)
  fix dip
  assume "dip∈kD(rt (σ i))"
  and "the (nhop (rt (σ i)) dip) ≠ dip"
  and "rt (σ i) = update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})"
  with pre' show "?dip_in_kD dip ∧ ?nsqn_le_nsqn dip" by simp
next
  fix dip
  assume "dip∈kD(rt (σ i))"
  and notdip: "the (nhop (update (rt (σ i)) dip) oip (osn, kno, val, Suc hops, sip, {})) dip ≠ dip"
  and rtnot: "rt (σ i) ≠ update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})"
  show "?dip_in_kD dip ∧ ?nsqn_le_nsqn dip"
  proof (cases "dip = oip")
  assume "dip ≠ oip"
  with pre' (dip∈kD(rt (σ i))) notdip
  show ?thesis by clarsimp
next
  assume "dip = oip"
with \( \text{rtnot qinc \& dip} \in \text{kD(rt (\sigma i))} \) notdip *

have "?dip_in_kD dip"
by simp (metis kD_quality_increases)
moreover from \(<\text{dip} = \text{oip}\) \(\text{rtnot qinc \& dip} \in \text{kD(rt (\sigma i))} \) notdip *
have "?nsqn_le_nsqn dip" by simp (metis kD_nsqn_quality_increases_trans)
ultimately show ?thesis ..
qed

Proposition 7.30
lemmas okD_unk_or_atleast_one =
open_seq_invariant [OF kD_unk_or_atleast_one initiali_aodv, simplified seql_onl_swap]
lemmas ozero_seq_unk_hops_one =
open_seq_invariant [OF ozero_seq_unk_hops_one initiali_aodv, simplified seql_onl_swap]

lemma oreachable_fresh_okD_unk_or_atleast_one:
fixes dip
assumes "(\sigma, p) \in \text{oreachable (oapodv i)}"
(otherwith (op=) i) (orecvmsg (\lambda \sigma m. msg_fresh \sigma m 
\wedge msg_zhops m)))
(other quality_increases {i})"
and "\text{dip} \in \text{kD(rt (\sigma i))}"
shows "\pi_3(\text{the (rt (\sigma i) dip)}) = \text{unk} \lor 1 \leq \pi_2(\text{the (rt (\sigma i) dip)})"
(is "?P dip")
proof -
have "\exists l. l \in \text{labels} \Gamma_{\text{AODV}} p" by (metis aodv_ex_label)
with assms(1) have "\forall dip \in \text{kD(rt (\sigma i))}. \text{?P dip}"
by - (drule oinvariant_weakenD [OF okD_unk_or_atleast_one [OF oadv_trans oadv_trans]],
auto dest!: otherwith_actionD onlD simp: seqlssimp)
with (\text{dip} \in \text{kD(rt (\sigma i))}) show ?thesis by simp
qed

lemma oreachable_fresh_ozero_seq_unk_hops_one:
fixes dip
assumes "(\sigma, p) \in \text{oreachable (oapodv i)}"
(otherwith (op=) i) (orecvmsg (\lambda \sigma m. msg_fresh \sigma m 
\wedge msg_zhops m)))
(other quality_increases {i})"
and \( \text{dip} \in kD(\text{rt} (\sigma i)) \)

shows "sqn (rt (\sigma i)) dip = 0 →
\[ \text{sqnf} (\text{rt} (\sigma i)) \text{ dip} = \text{unk} \]
\( \wedge \text{the} (\text{dhops} (\text{rt} (\sigma i)) \text{ dip}) = 1 \)
\( \wedge \text{the} (\text{nhop} (\text{rt} (\sigma i)) \text{ dip}) = \text{dip} \)"

(is "?P dip")

proof -

have \( \exists I. I \in \text{labels } \Gamma_{AODV} \) by (metis aodv_ex_label)

with assms(1) have "\( \forall \text{dip} \in kD (\text{rt} (\sigma i)) \). ?P dip"

by - (drule invariant_weakenD [OF ozero_seq_unk_hops_one [OF oadv_trans aodv_trans]],

auto dest!: onlD otherwith_actionD simp: seqslsimp)

with \( \text{dip} \in kD (\text{rt} (\sigma i)) \) show ?thesis by simp

qed

lemma seq_nhpop_quality_increases':

shows "\( \text{opao}\_d v i \models (\text{otherswith} (\text{op=} \{i\}) (\text{orevmsg} (\lambda\sigma m. \text{msg}_\text{fresh} \sigma m \wedge \text{msg}_\text{hops} m)), \)
\( \text{other} \text{quality}_\text{increases} (i) \rightarrow \)
\( \text{onl} \Gamma_{AODV} (\lambda (\sigma, \_. \). \text{vall} \text{dip} = \text{the} (\text{nhop} (\text{rt} (\sigma i)) \text{ dip}) \)
\( \forall \text{dip} \in vD (\text{rt} (\sigma i)) \cap vD (\text{rt} (\sigma nhip)) \)
\( \wedge \text{nhip} \neq \text{dip} \)
\( \rightarrow (\text{rt} (\sigma i) \sqsubset _\text{dip} (\text{rt} (\sigma \text{nhip})))" \)

(is "\( \_ \models (?)S i, \_ \rightarrow \_" )

proof -

have weaken:
"\( \forall I P R P. \ p \models (\text{otherswith} \text{quality}_\text{increases} I (\text{orevmsg} Q), \text{other} \text{quality}_\text{increases} I \rightarrow P \)
\( \rightarrow p \models (\text{otherswith} (\text{op=} I (\text{orevmsg} (\lambda \sigma m. Q \sigma m \wedge R \sigma m)), \text{other} \text{quality}_\text{increases} I \rightarrow P" by auto

\{ fix i a and \( \sigma \_ \_ : \text{"ip } \rightarrow \text{state"} \)

assume a1: "\( \forall \text{dip. dip} \in vD (\text{rt} (\sigma i)) \)
\( \wedge \text{dip} \in vD (\text{rt} (\sigma i)) \)
\( \text{the} (\text{nhop} (\text{rt} (\sigma i)) \text{ dip}) \)
\( \wedge (\text{the} (\text{nhop} (\text{rt} (\sigma i)) \text{ dip})) \neq \text{dip} \)
\( \rightarrow (\text{rt} (\sigma i) \sqsubset _\text{dip} (\text{rt} (\sigma (\text{nhop} (\text{rt} (\sigma i)) \text{ dip})))" \)

and ow: "\( ?S i, \sigma \_ \_ \_ a\)"

have "\( \forall \text{dip. dip} \in vD (\text{rt} (\sigma i)) \)
\( \wedge \text{dip} \in vD (\text{rt} (\sigma i)) \)
\( \text{the} (\text{nhop} (\text{rt} (\sigma i)) \text{ dip}) \)
\( \wedge (\text{the} (\text{nhop} (\text{rt} (\sigma i)) \text{ dip})) \neq \text{dip} \)
\( \rightarrow (\text{rt} (\sigma i) \sqsubset _\text{dip} (\text{rt} (\sigma (\text{nhop} (\text{rt} (\sigma i)) \text{ dip})))" \)

proof clarify

fix dip

assume a2: "\( \text{dip} \in vD (\text{rt} (\sigma i)) \)"

and a3: "\( \text{dip} \in vD (\text{rt} (\sigma (\text{nhop} (\text{rt} (\sigma i)) \text{ dip})))" \)

and a4: "\( (\text{the} (\text{nhop} (\text{rt} (\sigma i)) \text{ dip})) \neq \text{dip} " \)

from ow have "\( \forall j. j \neq i \rightarrow \sigma j = \sigma' j \)" by auto

show "\( \text{rt} (\sigma i) \sqsubset _\text{dip} (\sigma' (\text{the} (\text{nhop} (\text{rt} (\sigma i)) \text{ dip})))" \)

proof (cases "\( (\text{the} (\text{nhop} (\text{rt} (\sigma i)) \text{ dip}) = i \)"

assume "\( (\text{the} (\text{nhop} (\text{rt} (\sigma i)) \text{ dip})) = i \)"

with \( \text{dip} \in vD (\text{rt} (\sigma i)) \) have "\( \text{dip} \in vD (\text{rt} (\sigma (\text{nhop} (\text{rt} (\sigma i)) \text{ dip})))" by simp

with a1 a2 a4 have "\( \text{rt} (\sigma i) \sqsubset _\text{dip} (\sigma (\text{the} (\text{nhop} (\text{rt} (\sigma i)) \text{ dip})))" by simp

with (\the (\text{nhop} (\text{rt} (\sigma i)) \text{ dip}) = i ) have "\( \text{rt} (\sigma i) \sqsubset _\text{dip} (\sigma i) " by simp

hence False by simp

thus ?thesis ..

next

assume "\( (\text{the} (\text{nhop} (\text{rt} (\sigma i)) \text{ dip}) \neq i \)"

with \( \forall j. j \neq i \rightarrow \sigma j = \sigma' j \)

have *: "\( \sigma (\text{the} (\text{nhop} (\text{rt} (\sigma i)) \text{ dip}) = \sigma' (\text{the} (\text{nhop} (\text{rt} (\sigma i)) \text{ dip})))" by simp

with \( \text{dip} \in vD (\text{rt} (\sigma (\text{nhop} (\text{rt} (\sigma i)) \text{ dip}))) \)

have "\( \text{dip} \in vD (\text{rt} (\sigma (\text{the} (\text{nhop} (\text{rt} (\sigma i)) \text{ dip}))))" by simp

with a1 a2 a4 have "\( \text{rt} (\sigma i) \sqsubset _\text{dip} (\sigma (\text{the} (\text{nhop} (\text{rt} (\sigma i)) \text{ dip})))" by simp

with * show ?thesis by simp

qed

qed

} note basic = this
\{ \text{fix } \sigma \ a \ dip \ sip \ i \} \\
\text{assume a1: } "\text{\forall dip. dip} \in \text{\nuD (rt (} \sigma \ i) \text{)} \\
\quad \land \text{dip} \in \text{\nuD (rt (} \sigma \ (\text{the (nhop (rt (} \sigma \ i) \text{ dip)))) \text{)} } \\
\quad \land \ (\text{the (nhop (rt (} \sigma \ i) \text{ dip) } \neq \text{ dip)} } \\
\quad \rightarrow \text{rt (} \sigma \ i) \sqsubseteq \text{dip} \text{ rt (} \sigma \ (\text{the (nhop (rt (} \sigma \ i) \text{ dip)))) }"
\text{and ow: } "\exists i \ a \ a' a"

\text{have } "\text{\forall dip. dip} \in \text{\nuD (update (rt (} \sigma \ i) \text{)} \text{ sip (0, unk, val, Suc 0, sip, {})) } \\
\quad \land \text{dip} \in \text{\nuD (rt (} \sigma' \ (\text{the (nhop (update (rt (} \sigma \ i) \text{) sip (0, unk, val, Suc 0, sip, {})) dip)))) } \\
\quad \land \ (\text{the (nhop (update (rt (} \sigma \ i) \text{) sip (0, unk, val, Suc 0, sip, {})) dip) } \neq \text{ dip) } \\
\quad \rightarrow \text{update (rt (} \sigma \ i) \text{ sip (0, unk, val, Suc 0, sip, {})) } \\
\quad \sqsubseteq \text{dip} \text{ rt (} \sigma' \ (\text{the (nhop (update (rt (} \sigma \ i) \text{) sip (0, unk, val, Suc 0, sip, {})) dip))))" \text{proof clarify } \\
\text{fix dip } \\
\text{assume a2: } "\text{dip} \in \text{\nuD (update (rt (} \sigma \ i) \text{) sip (0, unk, val, Suc 0, sip, {}))}" \\
\text{and a3: } "\text{dip} \in \text{\nuD (rt (} \sigma' \ (\text{the (update (rt (} \sigma \ i) \text{) sip (0, unk, val, Suc 0, sip, {})) dip}))}" \\
\text{and a4: } "\text{the (update (rt (} \sigma \ i) \text{) sip (0, unk, val, Suc 0, sip, {})) dip) } \neq \text{ dip) } \\
\text{show } "\text{update (rt (} \sigma \ i) \text{) sip (0, unk, val, Suc 0, sip, {})} \\
\text{sqsubseteq dip rt (} \sigma' \ (\text{the (nhop (update (rt (} \sigma \ i) \text{) sip (0, unk, val, Suc 0, sip, {})) dip))))" \text{proof (cases "dip = sip") } \\
\text{assume "dip = sip"} \\
\text{with } (\text{the (nhop (update (rt (} \sigma \ i) \text{) sip (0, unk, val, Suc 0, sip, {})) dip) } \neq \text{ dip) } \text{have } \text{False by simp } \\
\text{thus } ?\text{thesis .. } \\
\text{next } \\
\text{assume [simp]: } "\text{dip } \neq \text{ sip"} \\
\text{from a2 have } "\text{dip} \in \text{\nuD (rt (} \sigma \ i) \text{) } \lor \text{ dip = sip}" \\
\text{by (rule \nuD_update_val) } \\
\text{with } (\text{dip } \neq \text{ sip}) \text{ have } "\text{dip} \in \text{\nuD (rt (} \sigma \ i) \text{)" by simp } \\
\text{moreover from a3 have } "\text{dip} \in \text{\nuD (rt (} \sigma' \ (\text{the (update (rt (} \sigma \ i) \text{) dip))))" by simp } \\
\text{moreover from a4 have } "\text{the (update (rt (} \sigma \ i) \text{) dip) } \neq \text{ dip) by simp } \\
\text{ultimately have } "\text{rt (} \sigma \ i) \sqsubseteq \text{dip rt (} \sigma' \ (\text{the (nhop (rt (} \sigma \ i) \text{) dip)))" by simp } \\
\text{using a1 ow by } \text{ - (erule(1) basic, simp) } \\
\text{with } (\text{dip } \neq \text{ sip) show } ?\text{thesis } \\
\text{by } \text{ - (erule rt_strictly_fresher_update_other, simp) } \\
\text{qed } \\
\text{qed } \\
\} \text{note update_0_unk = this } \\
\{ \text{fix } \sigma \ a \ a' \ \text{nhop} \} \\
\text{assume pre: } "\text{\forall dip. dip} \in \text{\nuD (rt (} \sigma \ i) \text{) } \land dip \in \text{\nuD (rt (} \sigma' \ (\text{nhop dip})) \land dip \neq dip } \\
\quad \rightarrow \text{rt (} \sigma \ i) \sqsubseteq \text{dip rt (} \sigma' \ (\text{nhop dip}))" \\
\text{and ow: } "\exists i \ a \ a' a" \\
\text{have } "\text{\forall dip. dip} \in \text{\nuD (invalidate (rt (} \sigma \ i) \text{) (dests (} \sigma \ i) \text{)) } \\
\quad \land dip \in \text{\nuD (rt (} \sigma' \ (\text{nhop dip})) \land dip \neq dip } \\
\quad \rightarrow \text{rt (} \sigma \ i) \sqsubseteq \text{dip rt (} \sigma' \ (\text{nhop dip}))" \text{proof clarify } \\
\text{fix dip} \\
\text{assume "dip} \in \text{\nuD (invalidate (rt (} \sigma \ i) \text{) (dests (} \sigma \ i) \text{))" } \\
\text{and "dip} \in \text{\nuD (rt (} \sigma' \ (\text{nhop dip}))" } \\
\text{and "nhop dip } \neq \text{ dip" } \\
\text{from this(1) have "dip} \in \text{\nuD (rt (} \sigma \ i) \text{)" by (clarsimp dest!: \nuD_invalidate_vD_not_dests) } \\
\text{moreover from ow have } "\forall j. \ j \neq i \rightarrow \sigma \ j = \sigma' \ j" \text{ by auto } \\
\text{ultimately have } "\text{rt (} \sigma \ i) \sqsubseteq \text{dip rt (} \sigma' \ (\text{nhop dip}))" \text{ by auto } \\
\text{using pre \ dip} \in \text{\nuD (rt (} \sigma' \ (\text{nhop dip})) \land dip \neq dip } \text{by metis } \\
\text{with } (\forall j. \ j \neq i \rightarrow \sigma \ j = \sigma' \ j) \text{ show } "\text{rt (} \sigma \ i) \sqsubseteq \text{dip rt (} \sigma' \ (\text{nhop dip}))" \text{ by (metis rt_strictly_fresher_irefl) } \\
\text{qed } \\
\} \text{note invalidate = this } \\
\{ \text{fix } \sigma \ a \ a' \ \text{dip oip osn sip hops i} \}
assume pre: "∀dip. dip ∈ vD (rt (σ i))
∧ dip ∈ vD (rt (σ (the (nhop (rt (σ i)) dip))))
∧ (the (nhop (rt (σ i)) dip) ≠ dip
→ rt (σ i) ⊏ dip rt (σ (the (nhop (rt (σ i)) dip))))"

and ow: "∀ i ∈ ss (σ a)"
and "Suc 0 ≤ osn"
and a6: "sip ≠ oip → oip ∈ kD (rt (σ sip))
∧ osn ≤ nsqn (rt (σ sip)) oip
∧ (nsqn (rt (σ sip)) oip = osn
→ the (dhops (rt (σ sip)) oip) ≤ hops
∧ the (flag (rt (σ sip)) oip) = inv)"

and after: "'σ i = σ i (rt := update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {}))" have "∀dip. dip ∈ vD (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {}))
∧ dip ∈ vD (rt (σ' (the (nhop (update (rt (σ i)) oip
(osn, kno, val, Suc hops, sip, {}))) dip)))
∧ the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {}))) dip ≠ dip
→ update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})
⊢ dip
rt (σ' (the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {}))) dip)))"

proof clarify
fix dip
assume a2: "dip∈vD(update (rt (σ i)) oip (osn, kno, val, Suc (hops), sip, {}))"
and a3: "dip∈vD(rt (σ' (the (nhop (update (rt (σ i)) oip
(osn, kno, val, Suc hops, sip, {}))) dip)))"
and a4: "the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {}))) dip ≠ dip"
from ow have a5: "∀ j. j ≠ i → σ j = σ' j" by auto
show "update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})
⊢ dip
rt (σ' (the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {}))) dip)))" (is "?rt1 ⊏ dip ?rt2 dip")
proof (cases "?rt1 = rt (σ i)"

assume nochange [simp]:
"update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {}) = rt (σ i)"

from after have "σ' i = σ i" by simp
with a5 have "∀ j. j ≠ i → σ j = σ' j" by metis

from a2 have "dip∈vD (rt (σ i))" by simp
moreover from a3 have "dip∈vD(rt (σ (the (nhop (rt (σ i)) dip))))"
using nochange and "∀ j. j ≠ i → σ j = σ' j" by clarsimp
moreover from a4 have "the (nhop (rt (σ i)) dip) ≠ dip" by simp
ultimately have "rt (σ i) ⊏ dip rt (σ (the (nhop (rt (σ i)) dip)))"
using pre by simp
hence "rt (σ i) ⊏ dip rt (σ' (the (nhop (rt (σ i)) dip)))"
using "∀ j. j ≠ i → σ j = σ' j" by simp
thus "?thesis" by simp

next
assume change: "?rt1 ≠ rt (σ i)"
from after a2 have "dip∈kD(rt (σ' i))" by auto
show ?thesis
proof (cases "dip = oip")
assume "dip ≠ oip"

with a2 have "dip∈vD (rt (σ i))" by auto
moreover with a3 a5 after and "dip ≠ oip"
have "dip∈vD(rt (σ (the (nhop (rt (σ i)) dip))))"
by simp metis
moreover from a4 and "dip ≠ oip" have "the (nhop (rt (σ i)) dip) ≠ dip" by simp
ultimately have "rt (σ i) ⊏ dip rt (σ (the (nhop (rt (σ i)) dip)))"
using pre by simp
with after and a5 and "dip ≠ oip" show ?thesis
by simp (metis rt_strictly_fresher_update_other
next
assume "dip = oip"

with a4 and change have "sip ≠ oip" by simp
with a6 have "oip ∈ kD(rt (σ sip))"
and "osn ≤ nsqn (rt (σ sip)) oip" by auto

from a3 change ⟨dip = oip⟩ have "oip ∈ vD(rt (σ’ sip))" by simp
hence "the (flag (rt (σ’ sip)) oip) = val" by simp

from ⟨oip ∈ kD(rt (σ sip))⟩ have "osn < nsqn (rt (σ’ sip)) oip ∨ (osn = nsqn (rt (σ’ sip)) oip ∧ the (dhops (rt (σ’ sip)) oip) ≤ hops)"
proof
assume "oip ∈ vD(rt (σ sip))"
hence "the (flag (rt (σ sip)) oip) = val" by simp
with a6 ⟨sip ≠ oip⟩ have "nsqn (rt (σ sip)) oip = osn → the (dhops (rt (σ sip)) oip) ≤ hops"
by simp

show ?thesis
proof (cases "sip = i")
assume "sip ≠ i"
with a5 have "σ sip = σ’ sip" by simp
with ⟨osn ≤ nsqn (rt (σ sip)) oip⟩ and ⟨nsqn (rt (σ sip)) oip = osn → the (dhops (rt (σ sip)) oip) ≤ hops⟩
show ?thesis by auto
next
— alternative to using sip_not_ip
assume [simp]: "sip = i"
have "?rt1 = rt (σ i)"
proof (rule update_cases_kD, simp_all)
from ⟨Suc 0 ≤ osn⟩ show "0 < osn" by simp
next
from ⟨oip ∈ kD(rt (σ sip))⟩ and ⟨sip = i⟩ show "oip ∈ kD(rt (σ i))" by simp
next
assume "sqn (rt (σ i)) oip < osn"
also from ⟨osn ≤ nsqn (rt (σ sip)) oip⟩ have "... ≤ nsqn (rt (σ i)) oip" by simp
also have "... ≤ sqn (rt (σ i)) oip" by (rule nsqn_sqn)
finally have "sqn (rt (σ i)) oip < sqn (rt (σ i)) oip" .
hence False by simp
thus "(λa. if a = oip then Some (osn, kno, val, Suc hops, i, π7 (the (rt (σ i) oip))) else rt (σ i) a) = rt (σ i)" ..

next
assume "sqn (rt (σ i)) oip = osn"
and "Suc hops < the (dhops (rt (σ i)) oip)"
from this(1) and ⟨oip ∈ vD(rt (σ sip))⟩ have "nsqn (rt (σ i)) oip = osn" by simp
with ⟨nsqn (rt (σ sip)) oip = osn → the (dhops (rt (σ sip)) oip) ≤ hops⟩ have "the (dhops (rt (σ i)) oip) ≤ hops" by simp
with ⟨Suc hops < the (dhops (rt (σ i)) oip)⟩ have False by simp
thus "(λa. if a = oip then Some (osn, kno, val, Suc hops, i, π7 (the (rt (σ i) oip))) else rt (σ i) a) = rt (σ i)" ..

next
assume "the (flag (rt (σ i)) oip) = inv"
with ⟨the (flag (rt (σ sip)) oip) = val⟩ have False by simp
thus "(λa. if a = oip then Some (osn, kno, val, Suc hops, i, π7 (the (rt (σ i) oip))) else rt (σ i) a) = rt (σ i)" ..

158
next from $oip \in kD(rt (\sigma \ i))$
show "(\lambda a. if a = oip then Some (rt (\sigma \ i) oip) else rt (\sigma \ i) a) = rt (\sigma \ i)"
by (auto dest!: kD_Some)

qed

with change have False ..
thus ?thesis ..

qed

next assume "oip \in iD(rt (\sigma \ i))"

with $\langle$the (flag (rt (\sigma \ i)) oip) = val$\rangle$
by (metis f.distinct(1) iD_flag_is_inv)

from $oip \in iD(rt (\sigma \ i))$
have "the (flag (rt (\sigma \ i)) oip) = val" by simp

thus ?thesis ..

qed

thus ?thesis

proof

assume osnlt: "osn < nsqn (rt (\sigma \ i)) oip"

from $dip \in kD(rt (\sigma \ i))$
and $dip = oip$
have "dip \in kD (?rt1)" by simp

moreover from a3 have "dip \in kD(?rt2 dip)" by simp

moreover have "nsqn ?rt1 dip < nsqn (?rt2 dip) dip"

proof -

have "nsqn ?rt1 oip = osn"

by (simp add: $\langle$dip = oip$\rangle$ nsqn_update_changed_kno_val [OF change [THEN not_sym]])

also have "... < nsqn (rt (\sigma \ i)) oip" using osnlt .

also have "... = nsqn (?rt2 oip) oip" by (simp add: change)

finally show ?thesis

using $\langle$dip = oip$\rangle$ by simp

qed

ultimately show ?thesis

by (rule rt_strictly_fresher_ltI)

next

assume osneq: "osn = nsqn (rt (\sigma \ i)) oip \\
the (dhops (rt (\sigma \ i)) oip) \leq hops"

have "oip \in kD(?rt1)" by simp

moreover from a3 $\langle$dip = oip$\rangle$
have "oip \in kD(?rt2 oip)" by simp

moreover have "nsqn ?rt1 oip = nsqn (?rt2 oip) oip"

proof -

from osneq have "osn = nsqn (rt (\sigma \ i)) oip" ..

also have "osn = nsqn ?rt1 oip"

by (simp add: $\langle$dip = oip$\rangle$ nsqn_update_changed_kno_val [OF change [THEN not_sym]])

also have "nsqn (rt (\sigma \ i)) oip = nsqn (?rt2 oip) oip"

by (simp add: change)

finally show ?thesis .

qed

moreover have "$\pi_5$(the (?rt2 oip oip)) \leq \pi_5$(the (?rt1 oip))"

proof -

from osneq have "the (dhops (rt (\sigma \ i)) oip) \leq hops" ..

moreover from $\langle$oip \in vD (rt (\sigma \ i))$$\rangle$
have "oip\in kD(rt (\sigma \ i))" by auto

ultimately have "$\pi_5$(the (rt (\sigma \ i) oip)) \leq hops"

by (auto simp add: proj5_eq_dhops)

also from change after have "hops < \pi_5$(the (rt (\sigma \ i) oip))"

by (simp add: proj5_eq_dhops) (metis dhops_update_changed_lessI)

finally have "$\pi_5$(the (rt (\sigma \ i) oip)) < \pi_5$(the (rt (\sigma \ i) oip))" .

with change after show ?thesis by simp
ultimately have "?rt1 ⊑_oip ?rt2 oip" by (rule rt_strictly_fresher_eqI)
with (dip = oip) show ?thesis by simp
qed
assume "dip∈D(rt (σ' i))"
and "dip∈D(rt (σ' (the (nhop (rt (σ' i)) dip))))"
and "the (nhop (rt (σ' i)) dip) ≠ dip"

from this(i) and (σ' i = σ i) have "dip∈D(rt (σ i))"
and "dip∈D(rt (σ i))"

by auto

from (the (nhop (rt (σ' i)) dip) ≠ dip) and (σ' i = σ i)
have "the (nhop (rt (σ i)) dip) ≠ dip" (is "?nhip ≠ _") by simp

with \(\text{dip}\in\text{D}(\text{rt} (\sigma \ ?nhip))\) and next_hop
have "\(\text{dip}\in\text{D}(\text{rt} (\sigma \ ?nhip))\)"
and nsqns: "nsqn (\(\text{rt} (\sigma \ i)\)) dip ≤ nsqn (\(\text{rt} (\sigma \ ?nhip)\)) dip"
by (auto simp: Let_def)

have "0 < sqn (\(\text{rt} (\sigma \ i)\)) dip"
proof (rule neq0_conv [THEN iffD1, OF notI])
assume "sqn (\(\text{rt} (\sigma \ i)\)) dip = 0"
with \(\text{dip}\in\text{D}(\text{rt} (\sigma \ i))\) and unk_hops_one
have "?nhip = dip" by simp
with (?nhip ≠ dip) show False ..
qed
also have "... = nsqn (\(\text{rt} (\sigma \ i)\)) dip"
by (rule D_nsqn_sqn [OF \(\text{dip}\in\text{D}(\text{rt} (\sigma \ i))\)], THEN sym)
also have "... ≤ nsqn (\(\text{rt} (\sigma \ ?nhip)\)) dip"
by (rule nsqns)
also have "... ≤ sqn (\(\text{rt} (\sigma \ ?nhip)\)) dip"
by (rule nsqn_sqn)
finally have "0 < sqn (\(\text{rt} (\sigma \ ?nhip)\)) dip" .

have "\(\text{rt} (\sigma \ i) \sqsubseteq \text{dip} \text{rt} (\sigma' \ ?nhip)\)"
proof (cases "\(\text{dip}\in\text{D}(\text{rt} (\sigma \ ?nhip))\)"
assume "\(\text{dip}\in\text{D}(\text{rt} (\sigma \ ?nhip))\)"
with \(\text{pre} \(\text{dip}\in\text{D}(\text{rt} (\sigma \ i))\)\) and (?nhip ≠ dip)
have "\(\text{rt} (\sigma \ i) \sqsubseteq \text{dip} \text{rt} (\sigma \ ?nhip)\)" by auto
moreover from \(\forall j. \text{quality_increases} (\sigma \ j) (\sigma' \ j)\)
have "\(\text{quality_increases} (\sigma \ ?nhip) (\sigma' \ ?nhip)\)" ..
ultimately show \(?\text{thesis}\"
using \(\text{dip}\in\text{D}(\text{rt} (\sigma \ ?nhip))\)
by (rule strictly_fresher_quality_increases_right)

next
assume "\(\text{dip}\in\text{D}(\text{rt} (\sigma \ ?nhip))\)"
with \(\text{dip}\in\text{D}(\text{rt} (\sigma \ ?nhip))\) have "\(\text{dip}\in\text{D}(\text{rt} (\sigma \ ?nhip))\)" ..
hence "the (flag (\(\text{rt} (\sigma \ ?nhip)\)) dip) = inv" by auto
have "\(\text{nsqn} (\(\text{rt} (\sigma \ i)\)) dip ≤ \text{nsqn} (\(\text{rt} (\sigma \ ?nhip)\)) dip\)"
by (rule nsqns)
also from \(\text{dip}\in\text{D}(\text{rt} (\sigma \ ?nhip))\):
have "... = \(\text{sqn} (\(\text{rt} (\sigma \ ?nhip)\)) dip - 1\)" ..
also have "... < \(\text{sqn} (\(\sigma' \ ?nhip\)) dip\)"
proof -
from \(\forall j. \text{quality_increases} (\sigma \ j) (\sigma' \ j)\)
have "\(\text{quality_increases} (\sigma \ ?nhip) (\sigma' \ ?nhip)\)" ..
hence "∀ ip. \(\text{sqn} (\(\text{rt} (\sigma \ ?nhip)\)) ip \leq \text{sqn} (\(\text{rt} (\sigma' \ ?nhip)\)) ip\)" by auto
hence "\(\text{sqn} (\(\text{rt} (\sigma \ ?nhip)\)) dip \leq \text{sqn} (\(\text{rt} (\sigma' \ ?nhip)\)) dip\)" ..
with \(0 < \text{sqn} (\(\text{rt} (\sigma \ ?nhip)\)) dip\) show \(?\text{thesis}\) by auto
qed
also have "... = \(\text{nsqn} (\(\sigma' \ ?nhip\)) dip\)"
proof (rule D_nsqn_sqn [THEN sym])
from \(\text{dip}\in\text{D}(\text{rt} (\sigma' (the (\text{nhop (rt (\sigma' i)) dip))))))\) and (σ' i = σ i)
show "\(\text{dip}\in\text{D}(\text{rt} (\sigma' \ ?nhip))\)" by simp
qed
finally have "\(\text{nsqn} (\(\text{rt} (\sigma \ i)\)) dip < \text{nsqn} (\(\sigma' \ ?nhip\)) dip\)" .

moreover from \(\text{dip}\in\text{D}(\text{rt} (\sigma' (the (\text{nhop (rt (\sigma' i)) dip))))))\) and (σ' i = σ i)
have \(\text{"dip} \in \text{kD}(rt (\sigma' \text{?nhip})\)\}" by auto
ultimately show \(\text{"rt (\sigma i) \sqsubseteq dip rt (\sigma' \text{?nhip})\)\}
using \(\text{\langle dip} \in \text{kD}(rt (\sigma i))\rangle\) by - (rule rt_strictly_fresher_ltI)
qed
with \(\sigma' = \sigma i\) show \(\text{"rt (\sigma' i) \sqsubseteq dip rt (\sigma' (\text{the (nhop (rt (\sigma' i)) dip)))\)\)\}
by simp
qed

thus \(\text{?thesis}\) unfolding Let_def.

qed

lemma seq_compare_next_hop:
fixes \(w\)
shows \(\text{"opaodv i \text{|=} (otherwith (op=} \{i\} (orecvmsg msg_fresh), other quality_increases \{i\} \text{\textarrow})\)\}
global (\(\lambda\sigma. \forall dip. \text{let nhip = the (nhop (rt (\sigma i)) dip)\)\}
in dip \(\in \text{kD}(rt (\sigma i)) \wedge nhip \neq dip \text{\textarrow}\)
dip \(\in \text{kD}(rt (\sigma nhip))\)
\wedge nsqn (rt (\sigma i)) dip \leq nsqn (rt (\sigma nhip)) dip)"
by (rule oinvariant_weakenE [OF seq_compare_next_hop']) (auto dest!: onlD)

lemma seq_nhop_quality_increases:
shows \(\text{"opaodv i \text{|=} (otherwith (op=} \{i\} (orecvmsg (\lambda m. msg_fresh \sigma m \wedge msg_zhops m), other quality_increases \{i\} \text{\textarrow})\)\)\}
global (\(\lambda\sigma. \forall dip. \text{let nhip = the (nhop (rt (\sigma i)) dip)\)\}
in dip \(\in vD (rt (\sigma i)) \cap vD (rt (\sigma nhip)) \wedge nhip \neq dip\)
\text{\textarrow} (rt (\sigma i)) \sqsubseteq dip (rt (\sigma nhip)))"
by (rule oinvariant_weakenE [OF seq_nhop_quality_increases']) (auto dest!: onlD)

end

1.10 Routing graphs and loop freedom

theory A_Loop_Freedom
imports A_Aodv_Predicates A_Fresher
begin

Define the central theorem that relates an invariant over network states to the absence of loops in the associate routing graph.

definition \(\text{rt_graph} :: (ip \Rightarrow state) \Rightarrow ip \Rightarrow ip rel\)
where
\(\text{"rt_graph} \sigma = (\lambda dip. \{\text{ip, ip'} | \text{ip ip'} dsn dsk hops pre. \text{ip} \neq dip \wedge rt (\sigma dip) = \text{Some (dsn, dsk, val, hops, ip', pre)}\)\)"

Given the state of a network \(\sigma\), a routing graph for a given destination ip address \(\text{dip}\) abstracts the details of routing tables into nodes (ip addresses) and vertices (valid routes between ip addresses).

lemma rt_graphE [elim]:
fixes \(n dip ip ip'\)
assumes \(\"(ip, ip') \in rt_graph \sigma dip\)"
shows \(\text{"ip} \neq dip \wedge (\exists r. rt (\sigma ip) = r \wedge (\exists dsn dsk hops pre. r dip = \text{Some (dsn, dsk, val, hops, ip', pre)})\)\)
using asms unfolding rt_graph_def by auto

lemma rt_graph_vD [dest]:
\(\forall dip \sigma dip. (ip, ip') \in rt_graph \sigma dip \Rightarrow dip \in vD(rt (\sigma ip))\)
unfolding rt_graph_def vD_def by auto

lemma rt_graph_vD_trans [dest]:
\(\forall dip \sigma dip. (ip, ip') \in (rt_graph \sigma dip)^+ \Rightarrow dip \in vD(rt (\sigma ip))\)
by (erule converse_tranclE) auto
lemma rt_graph_not_dip [dest]:
"\forall ip ip' \sigma dip. (ip, ip') \in \text{rt_graph} \sigma dip \implies ip \neq dip"
unfolding rt_graph_def by auto

lemma rt_graph_not_dip_trans [dest]:
"\forall ip ip' \sigma dip. (ip, ip') \in (\text{rt_graph} \sigma dip)^+ \implies ip \neq dip"
by (erule converse_tranclE) auto

NB: the property below cannot be lifted to the transitive closure

lemma rt_graph_nhip_is_nhop [dest]:
"\forall ip ip' \sigma dip. (ip, ip') \in \text{rt_graph} \sigma dip \implies ip' = \text{the (nhop (rt (} \sigma \text{ ip)) dip)}"
unfolding rt_graph_def by auto

theorem inv_to_loop_freedom:
assumes "\forall i dip. let nhip = \text{the (nhop (rt (} \sigma \text{ i)) dip)} in dip \in vD (rt (} \sigma \text{ i)) \cap vD (rt (} \sigma \text{ nhip)) \land nhip \neq dip \implies (rt (} \sigma \text{ i)) \sqsubseteq dip (rt (} \sigma \text{ nhip))"
shows "\forall dip. \text{irrefl ((rt_graph} \sigma dip)^+)"
using assms proof (intro allI)
fix \sigma :: "ip \Rightarrow \text{state}" and dip
assume inv: "\forall ip dip. let nhip = \text{the (nhop (rt (} \sigma \text{ ip)) dip)} in dip \in vD(rt (} \sigma \text{ ip)) \cap vD(rt (} \sigma \text{ nhip)) \land nhip \neq dip \implies rt (} \sigma \text{ ip) \sqsubseteq dip rt (} \sigma \text{ nhip)}"
{ fix ip ip'
assume "(ip, ip') \in (rt_graph \sigma dip)^+
and "dip \in vD(rt (} \sigma \text{ ip'}))"
and "ip' \neq dip"
hence "rt (} \sigma \text{ ip) \sqsubseteq dip rt (} \sigma \text{ ip')}"
proof induction
fix nhip
assume "(ip, nhip) \in rt_graph \sigma dip"
and "dip \in vD(rt (} \sigma \text{ nhip))"
and "nhip \neq dip"
from ⟨(ip, nhip) \in rt_graph \sigma dip⟩ have "dip \in vD(rt (} \sigma \text{ ip))"
and "nhip = \text{the (nhop (rt (} \sigma \text{ ip)) dip)}"
by auto
from ⟨dip \in vD(rt (} \sigma \text{ ip))⟩ and ⟨dip \in vD(rt (} \sigma \text{ nhip))⟩
have "dip \in vD(rt (} \sigma \text{ ip)) \cap vD(rt (} \sigma \text{ nhip))" ..
with \text{nhip = the (nhop (rt (} \sigma \text{ ip)) dip)}
and \text{nhip \neq dip}
and inv
show "rt (} \sigma \text{ ip) \sqsubseteq dip rt (} \sigma \text{ nhip)}"
by (clarsimp simp: Let_def)
next
fix nhip nhip'
assume "(ip, nhip) \in rt_graph \sigma dip"
and "nhip, nhip' \in rt_graph \sigma dip"
and IH: "[\dip \in vD(rt (} \sigma \text{ nhip))); nhip \neq dip ] \implies rt (} \sigma \text{ ip) \sqsubseteq dip rt (} \sigma \text{ nhip)}"
and "dip \in vD(rt (} \sigma \text{ nhip'))"
and "nhip' \neq dip"
from ⟨nhip, nhip'⟩ \in rt_graph \sigma dip⟩ have 1: "dip \in vD(rt (} \sigma \text{ nhip))"
and 2: "nhip \neq dip"
and "nhip' = the (nhop (rt (} \sigma \text{ nhip)) dip)"
by auto
from 1 2 have "rt (} \sigma \text{ ip) \sqsubseteq dip rt (} \sigma \text{ nhip)}" by (rule IH)
also have "rt (} \sigma \text{ nhip) \sqsubseteq dip rt (} \sigma \text{ nhip')}"
proof -
from dip \in vD(rt (} \sigma \text{ nhip))⟩ and ⟨dip \in vD(rt (} \sigma \text{ nhip')⟩
have "dip \in vD(rt (} \sigma \text{ nhip)) \cap vD(rt (} \sigma \text{ nhip'))" ..
with \text{nhip' \neq dip}
and \text{nhip' = the (nhop (rt (} \sigma \text{ nhip)) dip)}
and inv
show "rt (σ nhip) ⊏ dip rt (σ nhip)"
by (clarsimp simp: Let_def)
qed

finally show "rt (σ ip) ⊏ dip rt (σ nhip)".
qed }

note fresher = this

show "irrefl ((rt_graph σ dip)+)"
unfolding irrefl_def proof (intro allI notI)
fix ip
assume "(ip, ip) ∈ (rt_graph σ dip)+"
moreover then have "dip ∈ vD (rt (σ ip))"
and "ip ≠ dip"
ultimately have "rt (σ ip) ⊏ dip rt (σ ip)"
by (rule fresher)
thus False by simp
qed

end

1.11 Lift and transfer invariants to show loop freedom

theory A_Aodv_Loop_Freedom
imports ../../../AWN/OClosed_Transfer ../../../AWN/Qmsg_Lifting A_Global_Invariants A_Loop_Freedom
begin

lift to parallel processes with queues

lemma par_step_no_change_on_send_or_receive:
fixes σ s a σ' s'
assumes "((σ, s), a, (σ', s')) ∈ oparp_sos i (oseqp_sos Γ AODV i) (seqp_sos Γ QMSG)"
and "a ≠ τ"
shows "σ' i = σ i"
using assms by (rule qmsg_no_change_on_send_or_receive)

lemma par_nhop_quality_increases:
shows "opaodv i ⟨⟨ i qmsg | ⟨⟩ = A (otherwith (op=) {i} (orecvmsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
other quality_increases {i} →)
global (λσ. ∀ dip. let nhip = the (nhop (rt (σ i)) dip)
in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
→ (rt (σ i)) ⊏ dip (rt (σ nhip))))⟩"
proof (rule lift_into_qmsg [OF seq_nhop_quality_increases])
show "opaodv i | ⟨⟨ i qmsg | ⟨⟩ = A (otherwith (op=) {i} (orecvmsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
other quality_increases {i} →)
globala (λ(σ, _, σ'). quality_increases (σ i) (σ' i)))"
proof (rule ostep_invariant_weakenE [OF oquality_increases], simp_all)
fix t :: "(((nat ⇒ state) × (state, msg, pseqp, pseqp label) seqp), msg seq_action) transition"
assume "onll Γ AODV (λ((σ, _), _, (σ', _)). ∀ j. quality_increases (σ j) (σ' j)) t"
thus "quality_increases (fst (fst t) i) (fst (snd (snd t)) i)"
by (cases t) (clarsimp dest!: onllD, metis aodv_ex_label)
next
fix σ σ' a
assume "otherwith (op=) {i}
(orecvmg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
other quality_increases {i} →)
globala (λ(σ, _, σ'). quality_increases (σ i) (σ' i))"
proof (rule ostep_invariant_weakenE [OF oquality_increases], simp_all)
fix t :: "(((nat ⇒ state) × (state, msg, pseqp, pseqp label) seqp), msg seq_action) transition"
assume "onll Γ AODV (λ((σ, _), _, (σ', _)). ∀ j. quality_increases (σ j) (σ' j)) t"
thus "quality_increases (fst (fst t) i) (fst (snd (snd t)) i)"
by (cases t) (clarsimp dest!: onllD, metis aodv_ex_label)
next
fix σ σ' a
assume "otherwith (op=) {i}
(orecvmg (λσ m. msg_fresh σ m ∧ msg_zhops m)) σ σ' a"
thus "otherwith quality_increases {i} (orecvmg (λ_. rreq_rrep_sn)) σ σ' a"
by - (erule weaken_otherwith, auto)
qed
qed auto

lemma par_rreq_rrep_sn_quality_increases:
"opaodv i ⟨⟨ i qmsg | ⟨⟩ = A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_. _ True) {i} →)
globala (λ(σ, _, σ'). quality_increases (σ i) (σ' i)))"
proof -
have "opaodv i \mid A (\lambda \sigma_. \text{orecvmsg} (\lambda_. \text{rreq\_rrep\_sn}) \sigma, \text{other} (\lambda_. \text{True}) \{i\} \rightarrow)
globala (\lambda(\sigma, _, \sigma'). \text{quality\_increases} (\sigma i) (\sigma' i))"
by (rule ostep\_invariant\_weakenE [OF olocal\_quality\_increases])
\langle\langle
(auto dest!: opaodv_trans aodv\_trans)
ahence "opaodv i \mid (i \text{qmsg} \mid A (\lambda \sigma_. \text{orecvmsg} (\lambda_. \text{rreq\_rrep\_sn}) \sigma, \text{other} (\lambda_. \text{True}) \{i\} \rightarrow)
globala (\lambda(\sigma, a, \sigma'). \text{quality\_increases} (\sigma i) (\sigma' i))"
by (rule lift\_step\_into\_qmsg\_statelessasm) simp\_all
thus \text{?thesis} by rule auto
\text{qed}

\text{lemma \ par\_rreq\_rrep\_nsqn\_fresh\_any\_step:}
shows "opaodv i \mid (i \text{qmsg} \mid A (\lambda \sigma_. \text{orecvmsg} (\lambda_. \text{rreq\_rrep\_sn}) \sigma, \text{other} (\lambda_. \text{True}) \{i\} \rightarrow)
globala (\lambda(\sigma, a, \sigma'). \text{anycast} (\text{msg\_fresh} \sigma) a)"
proof -
have "opaodv i \mid A (\lambda \sigma_. (\text{orecvmsg} (\lambda_. \text{rreq\_rrep\_sn})) \sigma, \text{other} (\lambda_. \text{True}) \{i\} \rightarrow)
globala (\lambda(\sigma, a, \sigma'). \text{anycast} (\text{msg\_fresh} \sigma) a)"
proof (rule ostep\_invariant\_weakenE [OF oreq\_rreq\_rrep\_nsqn\_fresh\_any\_step\_invariant])
fix t
assume "\text{onll} \Gamma_{AODV} (\lambda((\sigma, _), a, _). \text{anycast} (\text{msg\_fresh} \sigma) a) t" thus "globala (\lambda(\sigma, a, \sigma'). \text{anycast} (\text{msg\_fresh} \sigma) a) t"
by (cases t) (clarsimp dest!: onllD, metis aodv\_ex\_label)
\text{qed auto}
\text{hence "opaodv i \mid (i \text{qmsg} \mid A (\lambda \sigma_. (\text{orecvmsg} (\lambda_. \text{rreq\_rrep\_sn})) \sigma, \text{other} (\lambda_. \text{True}) \{i\} \rightarrow)
globala (\lambda(\sigma, a, \sigma'). \text{anycast} (\text{msg\_fresh} \sigma) a)"
by (rule lift\_step\_into\_qmsg\_statelessasm) simp\_all
thus \text{?thesis} by rule auto
\text{qed}

\text{lemma \ par\_anycast\_msg\_zhops:}
shows "opaodv i \mid (i \text{qmsg} \mid A (\lambda \sigma_. \text{orecvmsg} (\lambda_. \text{rreq\_rrep\_sn}) \sigma, \text{other} (\lambda_. \text{True}) \{i\} \rightarrow)
globala (\lambda(\sigma, a, _). \text{anycast} \text{msg\_zhops} a)"
proof -
from anycast\_msg\_zhops\_initial\_aodv initial\_aodv\_trans aodv\_trans\_trans\_aodv\_trans\_aodv\_trans
have "opaodv i \mid A (\act \text{TT}, \text{other} (\lambda_. \text{True}) \{i\} \rightarrow)
\text{seqll} i \mid (\text{onll} \Gamma_{AODV} (\lambda(\_, a, _). \text{anycast} \text{msg\_zhops} a)\}}"
by (rule open\_seq\_step\_invariant)
\text{hence "opaodv i \mid A (\lambda \sigma_. \text{orecvmsg} (\lambda_. \text{rreq\_rrep\_sn}) \sigma, \text{other} (\lambda_. \text{True}) \{i\} \rightarrow)
globala (\lambda(\sigma, a, _). \text{anycast} \text{msg\_zhops} a)"
proof (rule ostep\_invariant\_weakenE)
fix t :: "((\text{nat} \Rightarrow \text{state}) \times (\text{state}, \text{msg}, \text{pseqp}, \text{pseqp label}) \text{seqp}, \text{msg} \text{seq\_action}) \text{transition}"
assume "\text{seqll} i \mid (\text{onll} \Gamma_{AODV} (\lambda(\_, a, _). \text{anycast} \text{msg\_zhops} a) t)" thus "globala (\lambda(\_, a, _). \text{anycast} \text{msg\_zhops} a) t"
by (cases t) (clarsimp dest!: seqllD onllD, metis aodv\_ex\_label)
\text{qed simp\_all}
\text{hence "opaodv i \mid (i \text{qmsg} \mid A (\lambda \sigma_. \text{orecvmsg} (\lambda_. \text{rreq\_rrep\_sn}) \sigma, \text{other} (\lambda_. \text{True}) \{i\} \rightarrow)
globala (\lambda(\_, a, _). \text{anycast} \text{msg\_zhops} a)"
by (rule lift\_step\_into\_qmsg\_statelessasm) simp\_all
thus \text{?thesis} by rule auto
\text{qed}  

\text{1.1.1 Lift to nodes}

\text{lemma \ node\_step\_no\_change\_on\_send\_or\_receive:}
assumes "((\sigma, \text{NodeS i P R}), a, (\sigma', \text{NodeS i' P' R'})) \in \text{node\_sos}
\langle\langle (\text{oparp\_sos} i \mid (\text{oseqp\_sos} \Gamma_{AODV} i) \langle\langle (\text{seqp\_sos} \Gamma_{QMSG})\}}"
and "a \neq \tau"
shows "\sigma' i = \sigma i"
using assms
by (cases a) (auto elim!: par\_step\_no\_change\_on\_send\_or\_receive)

\text{lemma \ node\_n-hop\_quality\_increases:}
shows "\langle i : \text{opaodv} i \mid (i \text{qmsg} \mid R) \mid A \langle\langle (\text{otherwith} (\text{op=}) \{i\})}
\[(\text{oarrivemsg } (\lambda \sigma. \text{msg\_fresh } \sigma \ m \land \text{msg\_zhops } m)), \]
\[\text{other\_quality\_increases } (i) \]
\[\rightarrow \text{global } (\lambda \sigma. \forall \text{dip. let } \text{nhip} = \text{the (nhop (rt (} (\sigma i)) \text{dip) in dip} \in \text{vD (rt (} (\sigma i)) \cap \text{vD (rt (} \text{nhip})) \land \text{nhip} \neq \text{dip} \rightarrow (rt (} (\sigma i)) \subseteq_{\text{dip}} (rt (} (\sigma \text{nhip}))))"\]

by (rule node\_lift [OF par\_nhop\_quality\_increases]) auto

**lemma node\_quality\_increases:**

\["(i : \text{opaodv } i \langle i \text{ qmsg : R } \rangle_o \models_{A} (\lambda _{\ldots}. \text{oarrivemsg } (\lambda _{\ldots}. \text{rreq\_rrep\_sn} ) \sigma), \]
\[\text{other } (\lambda _{\ldots}. \text{True} ) (i) \rightarrow \]
\[\text{global } (\lambda (\sigma, a, \sigma'). \text{quality\_increases } (\sigma i) (\sigma' i))"\]

by (rule node\_lift\_step\_statelessassm [OF par\_rreq\_rrep\_sn\_quality\_increases]) simp

**lemma node\_rreq\_rrep\_nsqn\_fresh\_any\_step:**

shows "(i : \text{opaodv } i \langle i \text{ qmsg : R } \rangle_o \models_{A} (\lambda _{\ldots}. \text{oarrivemsg } (\lambda _{\ldots}. \text{rreq\_rrep\_sn} ) \sigma), \text{other } (\lambda _{\ldots}. \text{True} ) (i) \rightarrow \]
\[\text{global } (\lambda (\sigma, a, \sigma'). \text{castmsg } (\text{msg\_fresh } a))"\]

by (rule node\_lift\_anycast\_statelessassm [OF par\_rreq\_rrep\_nsqn\_fresh\_any\_step])

**lemma node\_anycast\_msg\_zhops:**

shows "(i : \text{opaodv } i \langle i \text{ qmsg : R } \rangle_o \models_{A} (\lambda _{\ldots}. \text{oarrivemsg } (\lambda _{\ldots}. \text{rreq\_rrep\_sn} ) \sigma), \text{other } (\lambda _{\ldots}. \text{True} ) (i) \rightarrow \]
\[\text{global } (\lambda (\sigma, a, \sigma'). \text{castmsg } \text{msg\_zhops } a))"\]

by (rule node\_lift\_anycast\_statelessassm [OF par\_anycast\_msg\_zhops])

**lemma node\_silent\_change\_only:**

shows "(i : \text{opaodv } i \langle i \text{ qmsg : R } \rangle_o \models_{A} (\lambda _{\ldots}. \text{oarrivemsg } (\lambda _{\ldots}. \text{rreq\_rrep\_sn} ) \sigma), \text{other } (\lambda _{\ldots}. \text{True} ) (i) \rightarrow \]
\[\text{global } (\lambda (\sigma, a, \sigma'). a \neq \tau \rightarrow \sigma' i = \sigma i)"\]

proof (rule ostep\_invariantI, simp (no_asm), rule impI)

fix \(\sigma \ ζ \ a \ σ' \ ζ'

assume or: "((\sigma, ζ) \in \text{oreachable } (\langle i : \text{opaodv } i \langle i \text{ qmsg : R } \rangle_o \models_{A} (\lambda _{\ldots}. \text{oarrivemsg } (\lambda _{\ldots}. \text{rreq\_rrep\_sn} ) \sigma), \text{other } (\lambda _{\ldots}. \text{True} ) (i) \rightarrow \]
\[\text{global } (\lambda (\sigma, a, \sigma'). a \neq \tau \rightarrow \sigma' i = \sigma i)"\]

and tr: "((\sigma, ζ, a, (σ', ζ')) \in \text{trans } (\langle i : \text{opaodv } i \langle i \text{ qmsg : R } \rangle_o \models_{A} (\lambda _{\ldots}. \text{oarrivemsg } (\lambda _{\ldots}. \text{rreq\_rrep\_sn} ) \sigma), \text{other } (\lambda _{\ldots}. \text{True} ) (i) \rightarrow \]
\[\text{global } (\lambda (\sigma, a, \sigma'). a \neq \tau \rightarrow \sigma' i = \sigma i)"\]

from or obtain \(p \text{ R where } "(ζ = \text{NodeS } i \ p R)"

by (drule node\_net\_state, metis)

with tr have "((\sigma, \text{NodeS } i \ p R), a, (σ', ζ')) \in \text{onode\_sos } (\text{oparp\_sos } i \ \text{trans } (\text{opaodv } i)) \ (\text{trans } \text{qmsg}))"

by simp

thus "σ' i = σ i" using (a \neq τ_n)

by (cases rule: onode\_sos\_cases)

(auto elim: qmsg\_no\_change\_on\_send\_or\_receive)

qed

1.11.2 Lift to partial networks

**lemma arrive\_rreq\_rrep\_nsqn\_fresh\_inc\_sn [simp]:**

assumes "\text{oarrivemsg } (\lambda \sigma. \text{msg\_fresh } \sigma \ m \land \text{msg\_zhops } m)" \(\sigma m"

shows "\text{oarrivemsg } (\lambda _{\ldots}. \text{rreq\_rrep\_sn} ) \sigma m"

using assms by (cases m) auto

**lemma opnet\_nhop\_quality\_increases:**

shows "opnet \((\lambda i. \text{opaodv } i \langle i \text{ qmsg } \rangle p \models_{A} (\text{otherwith } (\text{op}) = \text{net\_tree\_ips } p) \rightarrow)
\[\text{oarrivemsg } (\lambda \sigma. \text{msg\_fresh } \sigma \ m \land \text{msg\_zhops } m)), \text{other\_quality\_increases } (\text{net\_tree\_ips } p) \rightarrow)\]
\[\text{global } (\lambda \sigma. \forall \text{dip. let } \text{nhip} = \text{the (nhop (rt (} (\sigma i)) \text{dip) in dip} \in \text{vD (rt (} (\sigma i)) \cap \text{vD (rt (} \text{nhip})) \land \text{nhip} \neq \text{dip} \rightarrow (rt (} (\sigma i)) \subseteq_{\text{dip}} (rt (} (\sigma \text{nhip})))))"\]

proof (rule pnet\_lift [OF node\_nhop\_quality\_increases])

fix i R

166
have "(i : opaodv i (∑_ i qmsg : R) |.= (λσ . _ rarrivemsg (λ_. rreq_rrep_sn) σ), (λσ . _ oarrivemsg (λ_. rreq_rrep_sn) σ), other (λ_ . True) {i} |- globala (λ(σ, a, σ').

castmsg (λm. msg_fresh σ m ∧ msg_zhops m) a)"

proof (rule ostep_invariantI, simp (no_asm))

fix σ a σ' a'

assume or: "((σ, s) ∈ oreachable ((i : opaodv i (∑_ i qmsg : R) |.= (λσ . _ oarrivemsg (λ_. rreq_rrep_sn) σ), (λσ . _ oarrivemsg (λ_. rreq_rrep_sn) σ), other (λ_ . True) {i} |- globala (λ(σ, a, σ').

and tr: "((σ, s), a, (σ', s')) ∈ trans ((i : opaodv i (∑_ i qmsg : R) |.= (λσ . _ oarrivemsg (λ_. rreq_rrep_sn) σ), (λσ . _ oarrivemsg (λ_. rreq_rrep_sn) σ), other (λ_ . True) {i} |- globala (λ(σ, a, σ').

and am: "oarrivemsg (λ_. rreq_rrep_sn) σ a"

from or tr am have "castmsg (msg_fresh σ) a"

by (auto dest!: ostep_invariantD [OF node_anycast_msg_zhops]) auto

qed

thus "((σ, s), a, (σ', s')) ∈ trans ((i : opaodv i (∑_ i qmsg : R) |.= (λσ . _ oarrivemsg (λ_. rreq_rrep_sn) σ), (λσ . _ oarrivemsg (λ_. rreq_rrep_sn) σ), other (λ_ . True) {i} |- globala (λ(σ, a, σ').

and am: "oarrivemsg (λ_. rreq_rrep_sn) σ a"

from or tr am have "castmsg (msg_zhops) a"

by (auto dest!: ostep_invariantD [OF node_anycast_msg_zhops]) auto

ultimately show "castmsg (λm. msg_fresh σ m ∧ msg_zhops m) a"

by (case_tac a) auto

qed

1.11.3 Lift to closed networks

lemma onet_nhop_quality_increases:

shows "closed (opnet (λi. opaodv i (∑_ i qmsg : R) |.= (λσ . _ rarrivemsg (λ_. rreq_rrep_sn) σ), (λσ . _ oarrivemsg (λ_. rreq_rrep_sn) σ), other (λ_ . True) {i} |- globala (λ(σ, a, σ').

from opnet_nhop_quality_increases

proof (rule inclosed_closed)

from opnet_nhop_quality_increases

proof (rule inclosed_closed)

proof (rule oinvariant_weakenE)

fix σ σ' :: "ip ⇒ state" and a :: "msg node_action"

assume "otherwith (op=) (net_tree_ips p) inclosed σ σ' a"

thus "oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m) σ σ' a"

proof (rule oinvariant_weakenE)

fix σ :: "ip ⇒ state" and a :: "msg node_action"

assume "inclosed σ a"

thus "oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m) σ a"

proof (cases a)

fix ii ni ms

assume "a = ii ni:arrive(ms)"
moreover with $\text{inoclosed}\, \sigma\, a$ obtain $d\, di$ where "ms = newpkt($d,\, di$)" by (cases ms) auto ultimately show ?thesis by simp
di

1.11.4 Transfer into the standard model

interpretation aodv_openproc: openproc paodv opaodv id
where "aodv_openproc.initmissing = initmissing"
proof -
show "openproc paodv opaodv id"
proof unfold_locales
fix $i :: \text{ip}$
have "$(\sigma, \zeta). (\sigma\, i, \zeta) \in \sigma_{AODV}\, i \land (\forall j. j \neq i \rightarrow \sigma\, j \in \text{fst}'\, \sigma_{AODV}\, j) \subseteq \sigma_{AODV}.$"
unfolding $\sigma_{AODV}\_def$ $\sigma_{AODV}'\_def$
proof (rule equalityD1)
show "$(f, p). ((\sigma, \zeta). (\sigma\, i, \zeta) \in \{f, p\}) \land (\forall j. j \neq i \rightarrow \sigma\, j \in \text{fst}'\, \{f, p\}) \subseteq \{f, p\}" by (rule set_eqI) auto
qed
thus "$\{ (\sigma\, i, \zeta) \mid \sigma\, i, \zeta\, s. s \in \text{init (paodv}\, i) \land (\sigma\, i, \zeta) = \text{id}\, s \land (\forall j. j \neq i \rightarrow \sigma\, j \in \text{fst o id}'\, \text{init (paodv}\, j)\} \subseteq \text{init (opaodv}\, i)" by simp
next
show "$\forall j. \text{init (paodv}\, j) \neq \{}$" unfolding $\sigma_{AODV}\_def$ by simp
next
fix $i\, s\, a\, s'\, \sigma\, \sigma'$
assume "$\sigma\, i = \text{fst (id}\, s)$"
and "$\sigma'\, i = \text{fst (id}\, s')$"
and "$(s, a, s') \in \text{trans (paodv}\, i)" then obtain $q\, q'$ where "$s = (\sigma\, i, q)$"
and "$s' = (\sigma'\, i, q')$"
and "$(\sigma\, i, q), a, (\sigma'\, i, q') \in \text{trans (paodv}\, i)" by (cases s, cases s') auto
from this(3) have "$(\sigma, q, a, (\sigma', q')) \in \text{trans (opaodv}\, i)" by simp (rule open_seqp_action [OF aodv_wf])
with $\vdash s = (\sigma\, i, q)$ and $\vdash s' = (\sigma'\, i, q')$
show "$(\sigma, \text{snd (id}\, s)), a, (\sigma', \text{snd (id}\, s')) \in \text{trans (opaodv}\, i)" by simp
qed
then interpret $\text{op: openproc paodv opaodv id .}$
have [simp]: "$\forall i. (\text{SOME}\, x. x \in \text{fst o id}'\, \text{init (paodv}\, i)) = \text{aodv}\_\text{init}\, i$" unfolding $\sigma_{AODV}\_def$ by simp
hence "$\forall i. \text{openproc.initmissing paodv id i = initmissing i}" unfolding op.initmissing_def op.someinit_def initmissing_def by (auto split: option.split)
thus "openproc.initmissing paodv id = initmissing" ..
qed

interpretation aodv_openproc_par_qmsg: openproc_parq paodv opaodv id qmsg
where "aodv_openproc_par_qmsg.netglobal = netglobal"
and "aodv_openproc_par_qmsg.initmissing = initmissing"
proof -
show "openproc_parq paodv opaodv id qmsg"
by (unfold_locales) simp
then interpret $\text{opq: openproc_parq paodv opaodv id qmsg .}$

have im: "$\forall \sigma. \text{openproc.initmissing (}\lambda i. \text{paodv i (}\langle \text{qmsg}\rangle\, (\lambda(p, q). (\text{fst (id p), snd (id p), q)})\, \sigma$
unfolding opq.initmissing_def opq.someinit_def initmissing_def
unfolding σ_AODV_def σ_MSG_def by (clarsimp cong: option.case_cong)
thus "openproc.initmissing (λ i. paodv i (qmsg) (λ (p, q). (fst (id p), snd (id p), q))) = initmissing" by (rule ext)
have "∀ P σ. openproc.netglobal (λ i. paodv i (qmsg) (λ (p, q). (fst (id p), snd (id p), q))) P σ = netglobal P σ"
unfolding opq.netglobal_def netglobal_def opq.initmissing_def initmissing_def opq.someinit_def
unfolding σ_AODV_def σ_MSG_def by (clarsimp cong: option.case_cong simp del: One_nat_def simp add: fst_initmissing_netgmap_default_aodv_init_netlift [symmetric, unfolded initmissing_def])
thus "openproc.netglobal (λ i. paodv i (qmsg) (λ (p, q). (fst (id p), snd (id p), q))) = netglobal" by auto qed

lemma net_nhop_quality_increases:
assumes "wf_net_tree n"
shows "closed (pnet (λ i. paodv i (qmsg) n) = netglobal (λσ. ∀i. ?inv σ i))"
proof -
from ⟨wf_net_tree n⟩ have proto: "closed (pnet (λ i. paodv i (qmsg) n) = netglobal (λσ. ∀i ∈ net_tree_ips n. ?inv (fst (initmissing (netgmap fst σ)))) i)"
(is "_ = netglobal (λσ. ∀i. ?inv σ i)")
proof (rule, simp only: aodv_openproc_par_qmsg.netglobalsimp fst_initmissing_netgmap_pair_fst, rule allI)
fix σ i
assume sr: "σ ∈ reachable (closed (pnet (λ i. paodv i (qmsg) n)) TT"
hence "∀ i ∈ net_tree_ips n. ?inv (fst (initmissing (netgmap fst σ)))) i" by (drule invariantD [OF proto], simp only: aodv_openproc_par_qmsg.netglobalsimp fst_initmissing_netgmap_pair_fst)
thus "?inv (fst (initmissing (netgmap fst σ)))) i"
proof (cases "i ∈ net_tree_ips n"
  assume "i ∉ net_tree_ips n"
  from sr have "σ ∈ reachable (pnet (λ i. paodv i (qmsg) n)) TT" ..
hence "net_ips σ = net_tree_ips n" ..
  with (i ∉ net_tree_ips n) have "i ∉ net_ips σ" by simp
  hence "(fst (initmissing (netgmap fst σ)))) i = aodv_init i" by simp
thus ?thesis by simp qed metis
qed

1.11.5 Loop freedom of AODV

theorem aodv_loop_freedom:
assumes "wf_net_tree n"
shows "closed (pnet (λ i. paodv i (qmsg) n) = netglobal (λσ. ∀ dip. irrefl ((rt_graph σ dip) +))" using assms by (rule aodv_openproc_par_qmsg.netglobal_weakenE [OF net_nhop_quality_increases inv_to_loop_freedom])

end
Chapter 2

Variant B: Forwarding the Route Reply

Explanation [4, §10.2]: In AODVs route discovery process, a RREP message from the destination node is unicast back along a route towards the originator of the RREQ message. Every intermediate node on the selected route will process the RREP message and, in most cases, forward it towards the originator node. However, there is a possibility that the RREP message is discarded at an intermediate node, which results in the originator node not receiving a reply. The discarding of the RREP message is due to the RFC specification of AODV [6] stating that an intermediate node only forwards the RREP message if it is not the originator node and it has created or updated a routing table entry to the destination node described in the RREP message. The latter requirement means that if a valid routing table entry to the destination node already exists, and is not updated when processing the RREP message, then the intermediate node will not forward the message. A solution to this problem is to require intermediate nodes to forward all RREP messages that they receive.

2.1 Predicates and functions used in the AODV model

theory B_Aodv_Data
imports B_Fwdrreps
begin

2.1.1 Sequence Numbers

Sequence numbers approximate the relative freshness of routing information.

definition inc :: "sqn ⇒ sqn"
  where "inc sn ≡ if sn = 0 then sn else sn + 1"

lemma less_than_inc [simp]: "x ≤ inc x"
  unfolding inc_def by simp

lemma inc_minus_suc_0 [simp]:
  "inc x - Suc 0 = x"
  unfolding inc_def by simp

lemma inc_never_one' [simp, intro]: "inc x ≠ Suc 0"
  unfolding inc_def by simp

lemma inc_never_one [simp, intro]: "inc x ≠ 1"
  by simp

2.1.2 Modelling Routes

A route is a 6-tuple, \((dsn, dsk, flag, hops, nhip, pre)\) where \(dsn\) is the ‘destination sequence number’, \(dsk\) is the ‘destination-sequence-number status’, \(flag\) is the route status, \(hops\) is the number of hops to the destination, \(nhip\) is the next hop toward the destination, and \(pre\) is the set of ‘precursor nodes’ those interested in hearing about changes to the route.

type_synonym r = "sqn × k × f × nat × ip × ip set"

definition proj2 :: "r ⇒ sqn" ("\pi_2")
where \( \pi_2 \equiv \lambda (dsn, _, _, _, _). dsn \)

definition proj3 :: \( 'r \Rightarrow k \) ("\( \pi_3 \)"")
  where \( \pi_3 \equiv \lambda (dsn, dsk, _, _, _). dsk \)

definition proj4 :: \( 'r \Rightarrow f \) ("\( \pi_4 \)"")
  where \( \pi_4 \equiv \lambda (dsn, dsk, flag, _, _). flag \)

definition proj5 :: \( 'r \Rightarrow \text{nat} \) ("\( \pi_5 \)"")
  where \( \pi_5 \equiv \lambda (dsn, dsk, flag, hops, _, _). hops \)

definition proj6 :: \( 'r \Rightarrow ip \) ("\( \pi_6 \)"")
  where \( \pi_6 \equiv \lambda (dsn, dsk, flag, hops, nhip, _). nhip \)

definition proj7 :: \( 'r \Rightarrow \text{ip set} \) ("\( \pi_7 \)"")
  where \( \pi_7 \equiv \lambda (dsn, dsk, flag, hops, nhip, pre). pre \)

lemma projs [simp]:
  \( \pi_2 (dsn, dsk, flag, hops, nhip, pre) = dsn \)
  \( \pi_3 (dsn, dsk, flag, hops, nhip, pre) = dsk \)
  \( \pi_4 (dsn, dsk, flag, hops, nhip, pre) = flag \)
  \( \pi_5 (dsn, dsk, flag, hops, nhip, pre) = hops \)
  \( \pi_6 (dsn, dsk, flag, hops, nhip, pre) = nhip \)
  \( \pi_7 (dsn, dsk, flag, hops, nhip, pre) = pre \)
  by (clarsimp simp: proj2_def proj3_def proj4_def proj5_def proj6_def proj7_def)+

lemma proj3_pred [intro]: \([ P kno; P unk ] \Longrightarrow P \ (\pi_3 \ x)\)
  by (rule k.induct)

lemma proj4_pred [intro]: \([ P val; P inv ] \Longrightarrow P \ (\pi_4 \ x)\)
  by (rule f.induct)

lemma proj6_pair_snd [simp]:
  fixes dsn' r
  shows \( \pi_6 (dsn', \text{snd} (r)) = \pi_6 (r) \)
  by (cases r) simp

2.1.3 Routing Tables

Routing tables map ip addresses to route entries.

type_synonym rt = "ip \rightarrow r"

syntax
  "\( _\Sigma \_ \) :: \( 'rt \Rightarrow \text{ip} \Rightarrow r \) ("\( \sigma_\text{route} (\_, \_') \)"")

translations
  "\( \sigma_\text{route} (rt, dip) \)" \Rightarrow "rt dip"

definition sqn :: \( 'rt \Rightarrow \text{ip} \Rightarrow sqn \)
  where \( sqn \ rt \ dip \equiv \text{case } \sigma_\text{route} (rt, dip) \text{ of } \text{Some } r \Rightarrow \pi_2 (r) \mid \text{None } \Rightarrow 0 \)

definition sqnf :: \( 'rt \Rightarrow \text{ip} \Rightarrow k \)
  where \( sqnf \ rt \ dip \equiv \text{case } \sigma_\text{route} (rt, dip) \text{ of } \text{Some } r \Rightarrow \pi_3 (r) \mid \text{None } \Rightarrow \text{unk} \)

abbreviation flag :: \( 'rt \Rightarrow \text{ip} \Rightarrow f \)
  where \( \text{flag} \ rt \ dip \equiv \text{map_option } \pi_4 (\sigma_\text{route} (rt, dip)) \)

abbreviation dhops :: \( 'rt \Rightarrow \text{ip} \Rightarrow \text{nat} \)
  where \( \text{dhops} \ rt \ dip \equiv \text{map_option } \pi_5 (\sigma_\text{route} (rt, dip)) \)

abbreviation nhop :: \( 'rt \Rightarrow \text{ip} \Rightarrow \text{ip} \)
  where \( \text{nhop} \ rt \ dip \equiv \text{map_option } \pi_6 (\sigma_\text{route} (rt, dip)) \)
abbreviation \texttt{precs :: "rt ⇒ ip → ip set"}
  \begin{itemize}
  \item where \texttt{precs rt dip \equiv map_option π7 (σroute(rt, dip))}
  \end{itemize}

definition \texttt{vD :: "rt ⇒ ip set"}
  \begin{itemize}
  \item where \texttt{vD rt \equiv {dip. flag rt dip = Some val}}
  \end{itemize}

definition \texttt{iD :: "rt ⇒ ip set"}
  \begin{itemize}
  \item where \texttt{iD rt \equiv {dip. flag rt dip = Some inv}}
  \end{itemize}

definition \texttt{kD :: "rt ⇒ ip set"}
  \begin{itemize}
  \item where \texttt{kD rt \equiv {dip. rt dip \neq None}}
  \end{itemize}

lemma \texttt{kD\_is\_vD\_and\_iD: "kD rt = vD rt \cup iD rt"}
  unfolding \texttt{kD\_def vD\_def iD\_def} by auto

lemma \texttt{vD\_iD\_gives\_kD [simp]:
  \"\forall ip rt. ip \in vD rt \implies ip \in kD rt"
  \"\forall ip rt. ip \in iD rt \implies ip \in kD rt"
  unfolding \texttt{kD\_is\_vD\_and\_iD} by simp_all

lemma \texttt{kD\_Some [dest]:
  \begin{itemize}
  \item fixes dip rt
  \item assumes "dip \in kD rt"
  \item shows "∃dsn dsk flag hops nhip pre. 
      σroute(rt, dip) = Some (dsn, dsk, flag, hops, nhip, pre)"
  \item using assms unfolding \texttt{kD\_def} by simp
  \end{itemize}

lemma \texttt{kD\_None [dest]:
  \begin{itemize}
  \item fixes dip rt
  \item assumes "dip \notin kD rt"
  \item shows "σroute(rt, dip) = None"
  \item using assms unfolding \texttt{kD\_def}
  \item by (metis (mono_tags) mem_Collect_eq)
  \end{itemize}

lemma \texttt{vD\_Some [dest]:
  \begin{itemize}
  \item fixes dip rt
  \item assumes "dip \in vD rt"
  \item shows "∃dsn dsk flag hops nhip pre.
      σroute(rt, dip) = Some (dsn, dsk, flag, hops, nhip, pre)"
  \item using assms unfolding \texttt{vD\_def} by simp
  \end{itemize}

lemma \texttt{vD\_empty [simp]: "vD Map.empty = {}"}
  unfolding \texttt{vD\_def} by simp

lemma \texttt{iD\_Some [dest]:
  \begin{itemize}
  \item fixes dip rt
  \item assumes "dip \in iD rt"
  \item shows "∃dsn dsk hops nhip pre.
      σroute(rt, dip) = Some (dsn, dsk, inv, hops, nhip, pre)"
  \item using assms unfolding \texttt{iD\_def} by simp
  \end{itemize}

lemma \texttt{val\_is\_vD [elim]:
  \begin{itemize}
  \item fixes ip rt
  \item assumes "ip ∈ kD(rt)"
  \item and "the (flag rt ip) = val"
  \item shows "ip ∈ vD(rt)"
  \item using assms unfolding \texttt{vD\_def} by auto
  \end{itemize}

lemma \texttt{inv\_is\_iD [elim]:
  \begin{itemize}
  \item fixes ip rt
  \item assumes "ip ∈ kD(rt)"
  \item and "the (flag rt ip) = inv"
  \item shows "ip ∈ iD(rt)"
  \item using assms unfolding \texttt{iD\_def} by auto
  \end{itemize}

172
lemma \texttt{id\_flag\_is\_inv} [elim, simp]:
  fixes \texttt{ip rt}
  assumes "\texttt{ip} \in \texttt{iD(rt)}"
  shows "\texttt{the (flag rt ip)} = \texttt{inv}"
proof -
  from \texttt{(ip} \in \texttt{iD(rt))} have "\texttt{ip} \in \texttt{kD(rt)}" by auto
  with \texttt{assms} show \texttt{?thesis} unfolding \texttt{iD\_def} by auto
qed

lemma \texttt{kD\_but\_not\_vD\_is\_iD} [elim]:
  fixes \texttt{ip rt}
  assumes "\texttt{ip} \in \texttt{kD(rt)}"
  and "\texttt{ip} /\in \texttt{vD(rt)}"
  shows "\texttt{ip} \in \texttt{iD(rt)}"
proof -
  from \texttt{(ip} \in \texttt{kD(rt))} obtain \texttt{dsn dsk f hops nhop pre}
  where \texttt{rtip: "rt ip = Some (dsn, dsk, f, hops, nhop, pre)"}
    by (metis \texttt{kD\_Some})
  from \texttt{(ip} /\in \texttt{vD(rt))} have "f \neq \texttt{val}"
  proof (rule \texttt{contrapos\_nn})
    assume "f = \texttt{val}"
    with \texttt{rtip} have "\texttt{the (flag rt ip)} = \texttt{val}" by simp
    with \texttt{(ip} \in \texttt{kD(rt))} show "\texttt{ip} \in \texttt{vD(rt)}" ..
  qed
  with \texttt{rtip} have "\texttt{the (flag rt ip)} = \texttt{inv}" by simp
  with \texttt{(ip} \in \texttt{kD(rt))} show "\texttt{ip} \in \texttt{iD(rt)}" ..
  qed

lemma \texttt{vD\_or\_iD} [elim]:
  fixes \texttt{ip rt}
  assumes "\texttt{ip} \in \texttt{kD(rt)}"
  and "\texttt{ip} \in \texttt{vD(rt)} = \Rightarrow \texttt{P rt ip}"
  and "\texttt{ip} \in \texttt{iD(rt)} = \Rightarrow \texttt{P rt ip}"
  shows "\texttt{P rt ip}"
proof -
  from \texttt{(ip} \in \texttt{kD(rt))} have "\texttt{ip} \in \texttt{vD(rt)} \cup \texttt{iD(rt)}"
    by (simp add: \texttt{kD\_is\_vD\_and\_iD})
  thus \texttt{?thesis} by (auto elim: \texttt{assms(2-3)})
  qed

lemma \texttt{proj5\_eq\_dhops}: 
  \texttt{\(\forall dip rt. dip \in kD(rt) \Rightarrow \pi_5(\text{the (rt dip)}) = \text{the (dhops rt dip)})\)}
  unfolding \texttt{sqn\_def} by (drule \texttt{kD\_Some}) clarsimp

lemma \texttt{proj4\_eq\_flag}: 
  \texttt{\(\forall dip rt. dip \in kD(rt) \Rightarrow \pi_4(\text{the (rt dip)}) = \text{the (flag rt dip)})\)}
  unfolding \texttt{sqn\_def} by (drule \texttt{kD\_Some}) clarsimp

lemma \texttt{proj2\_eq\_sqn}: 
  \texttt{\(\forall dip rt. dip \in kD(rt) \Rightarrow \pi_2(\text{the (rt dip)}) = sqn rt dip\)}
  unfolding \texttt{sqn\_def} by (drule \texttt{kD\_Some}) clarsimp

lemma \texttt{kD\_sqnf\_is\_proj3} [simp]:
  \texttt{\(\forall ip rt. ip \in kD(rt) \Rightarrow \pi_3(\text{the (rt ip)})\)}
  unfolding \texttt{sqnf\_def} by auto

lemma \texttt{vD\_flag\_val} [simp]:
  \texttt{\(\forall dip rt. dip \in vD (rt) \Rightarrow \text{the (flag rt dip)} = \text{val}\)}
  unfolding \texttt{vD\_def} by clarsimp

lemma \texttt{kD\_update} [simp]:
  \texttt{\(\forall rt nip v. kD (rt(nip \mapsto v)) = insert nip (kD rt)\)}
  unfolding \texttt{kD\_def} by auto

lemma \texttt{kD\_empty} [simp]: "kD Map.empty = {}"
  unfolding \texttt{kD\_def} by simp

---

173
lemma ip_equal_or_known [elim]:
  fixes rt ip ip'
  assumes "ip = ip' ∨ ip ∈ kD(rt)"
  and "ip = ip' ⟹ P rt ip ip'"
  and "[ ip ≠ ip'; ip ∈ kD(rt)] ⟹ P rt ip ip'"
  shows "P rt ip ip'"
  using assms by auto

2.1.4 Updating Routing Tables

Routing table entries are modified through explicit functions. The properties of these functions are important in invariant proofs.

Updating Precursor Lists

definition addpre :: "r ⇒ ip set ⇒ r"
  where "addpre r npre ≡ let (dsn, dsk, flag, hops, nhip, pre) = r in
          (dsn, dsk, flag, hops, nhip, pre ∪ npre)"

lemma proj2_addpre:
  fixes v pre
  shows "π_2(addpre v pre) = π_2(v)"
  unfolding addpre_def by (cases v) simp

lemma proj3_addpre:
  fixes v pre
  shows "π_3(addpre v pre) = π_3(v)"
  unfolding addpre_def by (cases v) simp

lemma proj4_addpre:
  fixes v pre
  shows "π_4(addpre v pre) = π_4(v)"
  unfolding addpre_def by (cases v) simp

lemma proj5_addpre:
  fixes v pre
  shows "π_5(addpre v pre) = π_5(v)"
  unfolding addpre_def by (cases v) simp

lemma proj6_addpre:
  fixes dsn dsk flag hops nhip pre npre
  shows "π_6(addpre v npre) = π_6(v)"
  unfolding addpre_def by (cases v) simp

lemma proj7_addpre:
  fixes dsn dsk flag hops nhip pre npre
  shows "π_7(addpre v npre) = π_7(v) ∪ npre"
  unfolding addpre_def by (cases v) simp

lemma addpre_empty: "addpre r {} = r"
  unfolding addpre_def by simp

lemma addpre_r:
  "addpre (dsn, dsk, fl, hops, nhip, pre) npre = (dsn, dsk, fl, hops, nhip, pre ∪ npre)"
  unfolding addpre_def by simp

lemmas addpre_simps [simp] = proj2_addpre proj3_addpre proj4_addpre proj5_addpre proj6_addpre proj7_addpre addpre_empty addpre_r

definition addpreRT :: "rt ⇒ ip set ⇒ rt"
  where "addpreRT rt dip npre ≡ map_option (λs. rt (dip ↦→ addpre s npre)) (σ_route(rt, dip))"
lemma snd_addpre [simp]:
"\(dsn \ dsn' \ v \ pre. (dsn, snd(addpre (dsn', v) pre)) = addpre (dsn, v) pre\"
unfolding addpre_def by clarsimp

lemma proj2_addpreRT [simp]:
  fixes ip rt ip' npre
  assumes "ip \in kD rt"
  and "ip' \in kD rt"
  shows "\(\pi_2(\text{the}(\text{the}(\text{addpreRT rt ip' npre) ip)) = \pi_2(\text{the}(rt ip)))\)"
  using assms [THEN kD_Some] unfolding addpreRT_def by clarsimp

lemma proj3_addpreRT [simp]:
  fixes ip rt ip' npre
  assumes "ip \in kD rt"
  and "ip' \in kD rt"
  shows "\(\pi_3(\text{the}(\text{the}(\text{addpreRT rt ip' npre) ip)) = \pi_3(\text{the}(rt ip)))\)"
  using assms [THEN kD_Some] unfolding addpreRT_def by clarsimp

lemma proj5_addpreRT [simp]:
"\(\forall rt dip ip npre. dip \in kD(rt) \Rightarrow \pi_5(\text{the}(\text{addpreRT rt dip npre) ip)) = \pi_5(\text{the}(rt ip)))\"
unfolding addpreRT_def by auto

lemma flag_addpreRT [simp]:
  fixes rt pre ip dip
  assumes "dip \in kD rt"
  shows "\(flag(\text{the}(\text{addpreRT rt dip npre}) ip) = flag rt ip\)"
  unfolding addpreRT_def
  using assms [THEN kD_Some] by (clarsimp)

lemma kD_addpreRT [simp]:
  fixes rt dip npre
  assumes "dip \in kD rt"
  shows "\(kD(\text{the}(\text{addpreRT rt dip npre}) = kD rt)\)"
  unfolding kD_def addpreRT_def
  using assms [THEN kD_Some] by clarsimp blast

lemma vD_addpreRT [simp]:
  fixes rt dip npre
  assumes "dip \in kD rt"
  shows "\(vD(\text{the}(\text{addpreRT rt dip npre}) = vD rt)\)"
  unfolding vD_def addpreRT_def
  using assms [THEN kD_Some] by clarsimp auto

lemma iD_addpreRT [simp]:
  fixes rt dip npre
  assumes "dip \in kD rt"
  shows "\(iD(\text{the}(\text{addpreRT rt dip npre}) = iD rt)\)"
  unfolding iD_def addpreRT_def
  using assms [THEN kD_Some] by clarsimp auto

lemma nhop_addpreRT [simp]:
  fixes rt pre ip dip
  assumes "dip \in kD rt"
  shows "\(nhop(\text{the}(\text{addpreRT rt dip npre}) ip) = nhop rt ip\)"
  unfolding sqn_def addpreRT_def
  using assms [THEN kD_Some] by (clarsimp)

lemma sqn_addpreRT [simp]:
  fixes rt pre ip dip
  assumes "dip \in kD rt"
  shows "\(sqn(\text{the}(\text{addpreRT rt dip npre}) ip) = sqn rt ip\)"
  unfolding sqn_def addpreRT_def
  using assms [THEN kD_Some] by (clarsimp)
lemma dhops_addpreRT [simp]:
  fixes rt pre ip dip
  assumes "dip ∈ kD rt"
  shows "dhops (the (addpreRT rt dip pre)) ip = dhops rt ip"
  unfolding addpreRT_def
  using assms [THEN kD_Some] by (clarsimp)

lemma sqnf_addpreRT [simp]:
  "∀ip dip. ip∈kD(rt ξ) ⇒ sqnf (the (addpreRT (rt ξ) ip npre)) dip = sqnf (rt ξ) dip"
  unfolding sqnf_def addpreRT_def by auto

Updating route entries

lemma in_kD_case [simp]:
  fixes dip rt
  assumes "dip ∈ kD(rt)"
  shows "(case rt dip of None ⇒ en | Some r ⇒ es r) = es (the (rt dip))"
  using assms [THEN kD_Some] by auto

lemma not_in_kD_case [simp]:
  fixes dip rt
  assumes "dip /∈ kD(rt)"
  shows "(case rt dip of None ⇒ en | Some r ⇒ es r) = en"
  using assms [THEN kD_None] by auto

lemma rt_Some_sqn [dest]:
  fixes rt and ip dsn dsk flag hops nhip pre
  assumes "rt ip = Some (dsn, dsk, flag, hops, nhip, pre)"
  shows "sqn rt ip = dsn"
  unfolding sqn_def using assms by simp

lemma not_kD_sqn [simp]:
  fixes dip rt
  assumes "dip /∈ kD(rt)"
  shows "sqn rt dip = 0"
  using assms unfolding sqn_def by simp

definition update_arg_wf :: "r ⇒ bool"
where "update_arg_wf r ≡ π₄(r) = val ∧
  (π₂(r) = 0) = (π₃(r) = unk) ∧
  (π₃(r) = unk → π₅(r) = 1)"

lemma update_arg_wf_gives_cases:
  "∀r. update_arg_wf r ⇒ (π₂(r) = 0) = (π₃(r) = unk)"
  unfolding update_arg_wf_def by simp

lemma update_arg_wf_tuples [simp]:
  "∀nhip pre. update_arg_wf (0, unk, val, Suc 0, nhip, pre)"
  "∀hops nhip pre. update_arg_wf (Suc n, kno, val, hops, nhip, pre)"
  unfolding update_arg_wf_def by auto

lemma update_arg_wf_tuples' [elim]:
  "∀hops nhip pre. Suc 0 ≤ n ⇒ update_arg_wf (n, kno, val, hops, nhip, pre)"
  unfolding update_arg_wf_def by auto

lemma wf_r_cases [intro]:
  fixes P r
  assumes "update_arg_wf r"
  and c1: "∀nhip pre. P (0, unk, val, Suc 0, nhip, pre)"
  and c2: "∀dsn hops nhip pre. dsn > 0 ⇒ P (dsn, kno, val, hops, nhip, pre)"
  shows "P r"
  proof -
obtain dsn dsk flag hops nhip pre
where *: "r = (dsn, dsk, flag, hops, nhip, pre)" by (cases r)
with update_arg_wf r have wf1: "flag = val"
and wf2: "(dsn = 0) = (dsk = unk)"
and wf3: "dsk = unk --> (hops = 1)"

unfolding update_arg_wf_def by auto
have "P (dsn, dsk, flag, hops, nhip, pre)"
proof (cases r)
assume "dsk = unk"
moreover with wf2 wf3 have "dsn = 0" by simp
ultimately show "P r" using ⟨flag = val⟩ by simp (rule c1)
next
assume "dsk = kno"
moreover with wf2 have "dsn > 0" by simp
ultimately show "P r" using ⟨flag = val⟩ by simp (rule c2)
qed

definition update :: "rt \Rightarrow ip \Rightarrow r \Rightarrow rt"
where "update rt ip r \equiv case σ_route (rt, ip) of
None \Rightarrow rt (ip \mapsto r)
| Some s \Rightarrow if \pi_2 (s) < \pi_2 (r) then rt (ip \mapsto addpre r (\pi_7 (s)))
else if \pi_2 (s) = \pi_2 (r) \land (\pi_5 (s) > \pi_5 (r) \lor \pi_4 (s) = inv)
then rt (ip \mapsto addpre r (\pi_7 (s)))
else if \pi_3 (r) = unk
then rt (ip \mapsto (\pi_3 (s), snd (addpre r (\pi_7 (s)))))
else rt (ip \mapsto addpre s (\pi_7 (r))))"

lemma update_simps [simp]:
fixes r s nrt nr nr' ns rt ip
defines "s \equiv the σ_route (rt, ip)"
and "nr \equiv addpre r (\pi_7 (s))"
and "nr' \equiv (\pi_3 (s), \pi_3 (nr), \pi_4 (nr), \pi_5 (nr), \pi_6 (nr), \pi_7 (nr))"
and "ns \equiv addpre s (\pi_7 (r))"
shows
"[ip \not\in kD (rt)] \Rightarrow update rt ip r = rt (ip \mapsto r)"
"[ip \in kD (rt); sqn rt ip < \pi_2 (r)] \Rightarrow update rt ip r = rt (ip \mapsto nr)"
"[ip \in kD (rt); \pi_2 (r) \equiv \pi_2 (r); sqn rt ip = \pi_2 (r);
the (dhops rt ip) > \pi_5 (r)] \Rightarrow update rt ip r = rt (ip \mapsto nr)"
"[ip \in kD (rt); \pi_2 (r) \equiv \pi_2 (r);
flag rt ip = Some inv] \Rightarrow update rt ip r = rt (ip \mapsto nr)"
"[ip \in kD (rt); \pi_3 (r) = unk; (\pi_2 (r) = 0) = (\pi_3 (r) = unk)] \Rightarrow update rt ip r = rt (ip \mapsto nr')"
"[ip \in kD (rt); sqn rt ip \geq \pi_2 (r); \pi_3 (r) = kno;
snd rt ip = \pi_2 (r) \Rightarrow the (dhops rt ip) \leq \pi_5 (r) \land the (flag rt ip) = val ]\Rightarrow update rt ip r = rt (ip \mapsto ns)"

proof -
assume "ip \not\in kD (rt)"
hence "σ_route (rt, ip) = None" ..
thus "update rt ip r = rt (ip \mapsto r)"
unfolding update_def by simp
next
assume "ip \in kD (rt)"
and "sqn rt ip < \pi_2 (r)"
from this(1) obtain dsn dsk fl hops nhip pre
where "rt ip = Some (dsn, dsk, fl, hops, nhip, pre)"
by (metis kD_Some)
with (sqn rt ip < \pi_2 (r)) show "update rt ip r = rt (ip \mapsto nr)"
unfolding update_def nr_def s_def by auto
next
assume "ip \in kD (rt)"

and "sqn rt ip = π₂(r)"
and "the (dhops rt ip) > π₅(r)"
from this(1) obtain dsn dsk fl hops nhip pre
  where "rt ip = Some (dsn, dsk, fl, hops, nhip, pre)"
  by (metis kD_Some)
with ⟨sqn rt ip = π₂(r)⟩ and ⟨the (dhops rt ip) > π₅(r)⟩
show "update rt ip r = rt (ip ↦ nr)"
  unfolding update_def nr_def s_def by auto

next
assume "ip ∈ kD(rt)"
and "sqn rt ip = π₃(r)"
and "flag rt ip = Some inv"
from this(1) obtain dsn dsk fl hops nhip pre
  where "rt ip = Some (dsn, dsk, fl, hops, nhip, pre)"
  by (metis kD_Some)
with ⟨sqn rt ip = π₂(r)⟩ and ⟨flag rt ip = Some inv⟩
show "update rt ip r = rt (ip ↦ nr)"
  unfolding update_def nr_def s_def by auto

next
assume "ip ∈ kD(rt)"
and "π₃(r) = unk"
and "(π₂(r) = 0) = (π₃(r) = unk)"
from this(1) obtain dsn dsk fl hops nhip pre
  where "rt ip = Some (dsn, dsk, fl, hops, nhip, pre)"
  by (metis kD_Some)
with ⟨π₂(r) = 0⟩ and ⟨π₃(r) = unk⟩
show "update rt ip r = rt (ip ↦ nr')"
  unfolding update_def nr'_def nr_def s_def by (cases r) simp

next
assume "ip ∈ kD(rt)"
and otherassms: "sqn rt ip ≥ π₂(r)"
  "π₃(r) = kno"
  "sqn rt ip = π₂(r) ⟹ the (dhops rt ip) ≤ π₅(r) ∧ the (flag rt ip) = val"
from this(1) obtain dsn dsk fl hops nhip pre
  where "rt ip = Some (dsn, dsk, fl, hops, nhip, pre)"
  by (metis kD_Some)
with otherassms show "update rt ip r = rt (ip ↦ ns)"
  unfolding update_def ns_def s_def by auto

qed

lemma update_cases [elim]:
assumes "(π₂(r) = 0) = (π₃(r) = unk)"
and c1: "[\[ ip \notin kD(rt) \] \implies P (rt (ip ↦ r))"
and c2: "[\[ ip ∈ kD(rt); sqn rt ip < π₂(r) \] \implies P (rt (ip ↦ addpre r (π₇(the σroute(rt, ip))))))"
and c3: "[\[ ip ∈ kD(rt); sqn rt ip = π₂(r); the (dhops rt ip) > π₅(r) \] \implies P (rt (ip ↦ addpre r (π₇(the σroute(rt, ip))))))"
and c4: "[\[ ip ∈ kD(rt); sqn rt ip = π₂(r); the (flag rt ip) = inv \] \implies P (rt (ip ↦ addpre r (π₇(the σroute(rt, ip))))))"
and c5: "[\[ ip ∈ kD(rt); π₃(r) = unk \] \implies P (rt (ip ↦ (π₇(the σroute(rt, ip)), π₃(r), π₄(r), π₅(r), π₆(r), π₇(addpre r (π₇(the σroute(rt, ip))))))))"
and c6: "[\[ ip ∈ kD(rt); sqn rt ip ≥ π₂(r); π₃(r) = kno; sqn rt ip = π₂(r) ⟹ the (dhops rt ip) ≤ π₅(r) ∧ the (flag rt ip) = val \] \implies P (rt (ip ↦ addpre (the σroute(rt, ip)) (π₇(r))))]"
shows "(P (update rt ip r))"
proof (cases "ip ∈ kD(rt)"
assume "ip \notin kD(rt)"
with c1 show ?thesis
  by simp
next
assume "ip ∈ kD(rt)"
moreover then obtain dsn dsk fl hops nhip pre
where rteq: "rt ip = Some (dsn, dsk, fl, hops, nhip, pre)"
by (metis kD_Some)
movere obtain dsn' dsk' fl' hops' nhip' pre'
where req: "r = (dsn', dsk', fl', hops', nhip', pre')"
by (cases r) metis
ultimately show "thesis"
using ⟨
(π 2 (r) = 0) = (π 3 (r) = unk)
⟩ c2 [OF ⟨ip ∈ kD (rt)⟩]
c3 [OF ⟨ip ∈ kD (rt)⟩]
c4 [OF ⟨ip ∈ kD (rt)⟩]
c5 [OF ⟨ip ∈ kD (rt)⟩]
c6 [OF ⟨ip ∈ kD (rt)⟩]
unfolding update_def sqn_def by auto
qed

lemma update_cases_kD:
assumes "(π 2 (r) = 0) = (π 3 (r) = unk)"
and "ip ∈ kD (rt)"
shows "(P (update rt ip r))"
using assms(1) proof (rule update_cases)
  assume "sqn rt ip < π 2 (r)"
  thus "P (rt (ip ↦ addpre r (π 7 (the (σ route (rt, ip))))))" by (rule c2)
next
  assume "sqn rt ip = π 2 (r)"
  and "(the (dhops rt ip) > π 5 (r))"
  thus "P (rt (ip ↦ addpre r (π 7 (the (σ route (rt, ip))))))" by (rule c3)
next
  assume "sqn rt ip = π 2 (r)"
  and "(the (flag rt ip) = inv)"
  thus "P (rt (ip ↦ addpre r (π 7 (the (σ route (rt, ip))))))" by (rule c4)
next
  assume "π 3 (r) = unk"
  thus "P (rt (ip ↦ (π 2 (the (σ route (rt, ip))))) (π 7 (r))))" by (rule c5)
next
  assume "sqn rt ip ≥ π 2 (r)"
  and "π 3 (r) = kno; sqn rt ip = π 2 (r) ⇒ the (dhops rt ip) ≤ π 5 (r) ∧ the (flag rt ip) = val"
  thus "P (rt (ip ↦ addpre (the (σ route (rt, ip))) (π 7 (r))))" by (rule c6)
qed (simp add: ⟨ip ∈ kD (rt)⟩)

lemma in_kD_after_update [simp]:
fixes rt nip dsn dsk flag hops nhip pre
shows "kD (update rt nip (dsn, dsk, flag, hops, nhip, pre)) = insert nip (kD rt)"
unfolding update_def
by (cases "rt nip") auto

lemma nhop_of_update [simp]:
fixes rt dip dsn dsk flag hops nhip
assumes "rt ≠ update rt dip (dsn, dsk, flag, hops, nhip, {})"
shows "the (nhop (update rt dip (dsn, dsk, flag, hops, nhip, {})) dip) = nhip"
proof -
from assms
have update_neq: "∀ v. rt dip = Some v ⇒ update rt dip (dsn, dsk, flag, hops, nhip, {}) ≠ rt(dip ↦→ addpre (the (rt dip)) (π₇ (dsn, dsk, flag, hops, nhip, {})))"
  by auto
show ?thesis
proof (cases "rt dip = None")
  assume "rt dip = None"
  thus ?thesis
next
  assume "rt dip ≠ None"
  then obtain v where "rt dip = Some v" by (metis not_None_eq)
  with update_neq [OF this]
  show ?thesis
    unfolding update_def
    by auto
qed
qed

lemma sqn_if_updated:
  fixes rip v rt ip
  shows "sqn (λ x. if x = rip then Some v else rt x) ip
           = (if ip = rip then π₂(v) else sqn rt ip)"
  unfolding sqn_def
  by simp

lemma update_sqn [simp]:
  fixes rt dip rip dsn dsk hops nhip pre
  assumes "(dsn = 0) = (dsk = unk)"
  shows "sqn rt dip ⪯ sqn (update rt rip (dsn, dsk, val, hops, nhip, pre)) dip"
  proof (rule update_cases)
    show "(π₂ (dsn, dsk, val, hops, nhip, pre) = 0) = (π₃ (dsn, dsk, val, hops, nhip, pre) = unk)"
      by simp (rule assms)
  qed (clarsimp simp: sqn_if_updated sqn_def)+

lemma sqn_update_bigger [simp]:
  fixes rt ip ip' dsn dsk flag hops nhip pre
  assumes "1 ≤ hops"
  shows "sqn rt ip ⪯ sqn (update rt ip' (dsn, dsk, flag, hops, nhip, pre)) ip"
  using assms unfolding update_def sqn_def
  by (clarsimp split: option.split) auto

lemma dhops_update [intro]:
  fixes rt dsn dsk flag hops ip rip nhip pre
  assumes ex: "∀ ip∈kD rt. the (dhops rt ip) ≥ 1"
  and ip: "(ip = rip ∧ Suc 0 ≤ hops) ∨ (ip ≠ rip ∧ ip∈kD rt)"
  shows "Suc 0 ≤ the (dhops (update rt rip (dsn, dsk, flag, hops, nhip, pre)) ip)"
  using ip proof
    assume "ip = rip ∧ Suc 0 ≤ hops" thus ?thesis
    unfolding update_def using ex
    by (cases "rip∈kD rt") (drule(1) bspec, auto)
next
  assume "ip ≠ rip ∧ ip∈kD rt" thus ?thesis
  using ex unfolding update_def
  by (cases "rip∈kD rt") auto
qed

lemma update_another [simp]:
  fixes dip ip rt dsn dsk flag hops nhip pre
  assumes "ip ≠ dip"
  shows "(update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = rt ip"
  using assms unfolding update_def
  by (clarsimp split: option.split)
lemma nhop_update_another [simp]:
  fixes dip ip rt dan dsk flag hops nhip pre
  assumes "ip ≠ dip"
  shows "nhop (update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = nhop rt ip"
  using assms unfolding update_def
  by (clarsimp split: option.split)

lemma dhops_update_another [simp]:
  fixes dip ip rt dan dsk flag hops nhip pre
  assumes "ip ≠ dip"
  shows "dhops (update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = dhops rt ip"
  using assms unfolding update_def
  by (clarsimp split: option.split)

lemma sqn_update_same [simp]:
"\( rt \mapsto \pi_2(v) \)" \( \rt \mapsto \pi_2(v) \) =⇒ sqn (rt (ip ↦ v)) ip = π₂(v)"
  unfolding sqn_def
  by simp

lemma dhops_update_changed [simp]:
  fixes rt dip dsn dsk flg hops nip
  assumes "rt ≠ update rt dip (dsn, dsk, flg, hops, nip, {})")"
  shows "the (dhops (update rt dip (dsn, dsk, flg, hops, nip, {})) dip) = hops"
  using assms unfolding update_def
  by (clarsimp split: option.splits split_if_asm) auto

lemma nhop_update_unk_val [simp]:
"\( rt \mapsto \pi_2(v) \) =⇒ the (nhop (update rt dip (dsn, unk, val, hops, ip, npre)) dip) = ip"
  unfolding update_def
  by (clarsimp split: option.split)

lemma nhop_update_changed [simp]:
  fixes rt dip dsn dsk flg hops sip
  assumes "update rt dip (dsn, dsk, flg, hops, sip, {} ≠ rt"
  shows "the (nhop (update rt dip (dsn, dsk, flg, hops, sip, {})) dip) = sip"
  using assms unfolding update_def
  by (clarsimp split: option.split: option.split split_if_asm) auto

lemma update_rt_split_asm:
"\( rt \mapsto \pi_2(v) \) =⇒ P (update rt dip (dsn, dsk, flag, hops, sip, {}))
  = (∀(rt = update rt dip (dsn, dsk, flag, hops, sip, {})) \land \neg P rt
      \lor rt ≠ update rt dip (dsn, dsk, flag, hops, sip, {})
      \land \neg P (update rt dip (dsn, dsk, flag, hops, sip, {}))))"
  by auto

lemma sqn_update [simp]: "\( rt \mapsto \pi_2(v) \) =⇒ sqn (update rt dip (dsn, kno, flg, hops, sip, {})) dip = dsn"
  unfolding update_def
  by (clarsimp split: option.split split_if_asm) auto

lemma sqnf_update [simp]: "\( rt \mapsto \pi_2(v) \) =⇒ sqnf (update rt dip (dsn, kno, flg, hops, sip, {})) dip = dsk"
  unfolding update_def sqnf_def
  by (clarsimp split: option.splits split_if_asm) auto

lemma update_kno_dsn_greater_zero:
"\( rt \mapsto \pi_2(v) \) =⇒ 1 ≤ dsn \Rightarrow 1 ≤ (sqn (update rt dip (dsn, kno, val, hops, ip, npre)) dip)"
  unfolding update_def
  by (clarsimp split: option.splits)

lemma proj3_update [simp]: "\( rt \mapsto \pi_2(v) \)"
\[ rt \neq \text{update rt dip (dsn, dsk, flg, hops, sip, {})} \]
\[ \implies \pi_3(\text{the (update rt dip (dsn, dsk, flg, hops, sip, {}) dip)}) = dsk \]

unfolding update_def sqnf_def
by (clarsimp split: option.splits split_if_asm) auto

lemma nhop_update_changed_kno_val [simp]: "\( \forall rt \ ip \ dsn \ dsk \ hops \ nhip. \)
\[ rt \neq \text{update rt dip (dsn, kno, val, hops, nhip, {})} \]
\[ \implies \text{the (nhop (update rt dip (dsn, kno, val, hops, nhip, {}) dip)) = nhip} \]

using assms unfolding update_def
by (clarsimp split: option.splits split_if_asm) auto

lemma flag_update [simp]: "\( \forall rt \ dip \ dsn \ flg \ hops \ sip. \)
\[ rt \neq \text{update rt dip (dsn, kno, flg, hops, sip, {})} \]
\[ \implies \text{the (flag (update rt dip (dsn, kno, flg, hops, sip, {}) dip)) = flg} \]

unfolding update_def
by (clarsimp split: option.splits split_if_asm) auto

lemma the_flag_Some [dest!]:
fixes ip rt
assumes "\( \text{the (flag rt ip)} = x \)"
and "\( ip \in kD rt \)"
shows "\( \text{flag rt ip = Some x} \)"
using assms by auto

lemma kD_update_unchanged [dest]:
fixes rt dip dsn dsk flag hops nhip pre
assumes "\( rt = \text{update rt dip (dsn, dsk, flag, hops, nhip, pre)} \)"
shows "\( dip \in kD(rt) \)"
proof -
  have "\( dip \in kD(\text{update rt dip (dsn, dsk, flag, hops, nhip, pre)}) \)" by simp
  with assms show ?thesis by simp
qed

lemma nhop_update [simp]: "\( \forall rt \ dip \ dsn \ dsk \ flg \ hops \ sip. \)
\[ rt \neq \text{update rt dip (dsn, dsk, flg, hops, sip, {})} \]
\[ \implies \text{the (nhop (update rt dip (dsn, dsk, flg, hops, sip, {}) dip)) = sip} \]

unfolding update_def sqnf_def
by (clarsimp split: option.splits split_if_asm) auto

lemma sqn_update_another [simp]:
fixes dip ip rt dsn dsk flag hops nhip pre
assumes "\( ip \neq dip \)"
shows "\( \text{sqn (update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = sqn rt ip} \)"
using assms unfolding update_def sqnf_def
by (clarsimp split: option.splits) auto

lemma sqnf_update_another [simp]:
fixes dip ip rt dsn dsk flag hops nhip pre
assumes "\( ip \neq dip \)"
shows "\( \text{sqnf (update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = sqnf rt ip} \)"
using assms unfolding update_def sqnf_def
by (clarsimp split: option.splits) auto

lemma vD_update_val [dest]:
"\( \forall dip \ dip' \ dsn \ dsk \ hops \ nhip \ pre. \)
\[ dip \in vD(\text{update rt dip'} (dsn, dsk, val, hops, nhip, pre)) \implies (dip \in vD(rt) \lor dip=dip') \]"

unfolding update_def vD_def by (clarsimp split: option.split_asm split_if_asm)

Invalidating route entries

definition invalidate :: "\( rt \Rightarrow (ip \Rightarrow sqn) \Rightarrow rt \)"
where "\( \text{invalidate rt dests} \equiv \lambda ip. \ \text{case (rt ip, dests ip) of} \)
\( (\text{None, _}) \Rightarrow \text{None} \)"
lemma proj3_invalidate [simp]:
"∀ dip. π₃(the ((invalidate rt dests) dip)) = π₃(the (rt dip))"
unfolding invalidate_def by (clarsimp split: option.split)

lemma proj5_invalidate [simp]:
"∀ dip. π₅(the ((invalidate rt dests) dip)) = π₅(the (rt dip))"
unfolding invalidate_def by (clarsimp split: option.split)

lemma proj6_invalidate [simp]:
"∀ dip. π₆(the ((invalidate rt dests) dip)) = π₆(the (rt dip))"
unfolding invalidate_def by (clarsimp split: option.split)

lemma proj7_invalidate [simp]:
"∀ dip. π₇(the ((invalidate rt dests) dip)) = π₇(the (rt dip))"
unfolding invalidate_def by (clarsimp split: option.split)

lemma invalidate_kD_inv [simp]:
"∀ rt dests. kD (invalidate rt dests) = kD rt"
unfolding invalidate_def kD_def
by (simp split: option.split)

lemma sqn_invalidate_in_dests [simp]:
fixes rt dip dests
assumes "∀ rsn. dests dip = Some rsn → sqn rt dip ≤ rsn"
shows "sqn rt dip ≤ sqn (invalidate rt dests) dip"
proof (cases "dip /∈ kD(rt)")
  assume "¬ dip /∈ kD(rt)"
  hence "dip∈kD(rt)" by simp
  then obtain dsn dsk flag hops nhip pre where "rt dip = Some (dsn, dsk, flag, hops, nhip, pre)"
    by (metis kD_Some)
  with assms show "sqn rt dip ≤ sqn (invalidate rt dests) dip"
    by (cases "dests dip") (auto simp add: invalidate_def sqn_def)
qed simp

lemma sqn_invalidate_in_dests [simp]:
fixes dests ipa rsn rt
assumes "dests ipa = Some rsn"
  and "ipa∈kD(rt)"
shows "sqn (invalidate rt dests) ipa = rsn"
unfolding invalidate_def sqn_def
using assms(1) assms(2) [THEN kD_Some]
by clarsimp

lemma dhops_invalidate [simp]:
"∀ dip. the (dhops (invalidate rt dests) dip) = the (dhops rt dip)"
unfolding invalidate_def by (clarsimp split: option.split)

lemma sqnf_invalidate [simp]:
"∀ dip. sqnf (invalidate (rt ξ) (dests ξ)) dip = sqnf (rt ξ) dip"
unfolding sqnf_def invalidate_def by (clarsimp split: option.split)

lemma nhop_invalidate [simp]:
"∀ dip. the (nhop (invalidate (rt ξ) (dests ξ)) dip) = the (nhop (rt ξ) dip)"
unfolding invalidate_def by (clarsimp split: option.split)

lemma invalidate_other [simp]:
fixes rt dests dip
assumes "dip/∈dom(dests)"
shows "invalidate rt dests dip = rt dip"
using assms unfolding invalidate_def
by (clarsimp split: option.split_asm)

lemma invalidate_none [simp]:
  fixes rt dests dip
  assumes "dip \in kD(rt)"
  shows "invalidate rt dests dip = None"
  using assms unfolding invalidate_def by clarsimp

lemma vD_invalidate_vD_not_dests:
  "\forall dip rt dests. dip \in vD(invalidate rt dests) \Longrightarrow dip \in vD(rt) \land dests dip = None"
  unfolding invalidate_def vD_def
  by (clarsimp split: option.split_asm)

lemma sqn_invalidate_not_in_dests [simp]:
  fixes dests dip rt
  assumes "dip \not\in \text{dom(dests)}"
  shows "sqn (invalidate rt dests) dip = sqn rt dip"
  using assms unfolding sqn_def
  by (clarsimp, cases "dests dip") auto

lemma invalidate_changes:
  fixes rt dests dip dsn dsk flag hops nhip pre
  assumes "invalidate rt dests dip = Some (dsn, dsk, flag, hops, nhip, pre)"
  shows "dsn = (case dests dip of None \Rightarrow \pi_2 (the (rt dip)) | Some rsn \Rightarrow rsn)
  \land dsk = \pi_3 (the (rt dip))
  \land flag = (if dests dip = None then \pi_4 (the (rt dip)) else inv)
  \land hops = \pi_5 (the (rt dip))
  \land nhip = \pi_6 (the (rt dip))
  \land pre = \pi_7 (the (rt dip))"
  using assms unfolding invalidate_def
  by (cases "rt dip", clarsimp, cases "dests dip") auto

lemma proj3_inv: "\forall dip rt dests. dip \in kD (rt) \Rightarrow \pi_3 (the (invalidate rt dests dip)) = \pi_3 (the (rt dip))"
  by (clarsimp simp: invalidate_def kD_def split: option.split)

lemma dests_iD_invalidate [simp]:
  assumes "dests ip = Some rsn" and "ip \in kD(rt)"
  shows "ip \in iD(invalidate rt dests)"
  using assms(1) assms(2) [THEN kD_Some] unfolding invalidate_def iD_def
  by (clarsimp split: option.split)

2.1.5 Route Requests

Generate a fresh route request identifier.

definition nrreqid :: "(ip \times rreqid) set \Rightarrow ip \Rightarrow rreqid"
  where "nrreqid rreqs ip = Max \{n. (ip, n) \in rreqs\} \cup \{0\} + 1"

2.1.6 Queued Packets

Functions for sending data packets.

type_synonym store = "ip \rightarrow (p \times data list)"

definition sigma_queue :: "store \Rightarrow ip \Rightarrow data list" ("\sigma_queue'(., _')")
  where "\sigma_queue(store, dip) \equiv case store dip of None \Rightarrow [] \mid Some (p, q) \Rightarrow q"

definition qD :: "store \Rightarrow ip set"
  where "qD \equiv \text{dom}"

definition add :: "data \Rightarrow ip \Rightarrow store \Rightarrow store"
  where "add d dip store \equiv case store dip of
  None \Rightarrow store (dip \mapsto (req, [d]))"
lemma qD_add [simp]:
  fixes d dip store
  shows "qD(add d dip store) = insert dip (qD store)"
  unfolding add_def Let_def qD_def
  by (clarsimp split: option.split)

definition drop :: "ip ⇒ store → store"
  where "drop dip store ≡
    map_option (λ(p, q). if tl q = [] then store (dip := None)
      else store (dip ⇒ (p, tl q))) (store dip)"

definition sigma_p_flag :: "store ⇒ ip ⇒ p" ("σ_p-flag(_, _)")
  where "σ_p-flag(store, dip) ≡ map_option fst (store dip)"

definition unsetRRF :: "store ⇒ ip ⇒ store"
  where "unsetRRF store dip ≡
    case store dip of
      None ⇒ store
    | Some (p, q) ⇒ store (dip ⇒ (noreq, q))"

definition setRRF :: "store ⇒ (ip ⇒ sqn) ⇒ store"
  where "setRRF store dests ≡
    λ dip. if dests dip = None then store dip
      else map_option (λ(_, q). (req, q)) (store dip)"

2.1.7 Comparison with the original technical report

The major differences with the AODV technical report of Fehnker et al are:

1. nhop is partial, thus a ‘the’ is needed, similarly for dhops and addpreRT.
2. precs is partial.
3. σ_p-flag(store, dip) is partial.
4. The routing table (rt) is modelled as a map (ip ⇒ r option) rather than a set of 7-tuples, likewise, the r
   is a 6-tuple rather than a 7-tuple, i.e., the destination ip-address (dip) is taken from the argument to the function, rather than a part of the result. Well-definedness then follows from the structure of the type and more related facts are available automatically, rather than having to be acquired through tedious proofs.
5. Similar remarks hold for the dests mapping passed to invalidate, and store.
The msg type models the different messages used within AODV. The instantiation as a msg is a technicality due to the special treatment of newpkt messages in the AWN SOS rules. This use of classes allows a clean separation of the AWN-specific definitions and these AODV-specific definitions.

```
definition rreq :: "nat × rreqid × ip × sqn × k × ip × sqn × ip ⇒ msg"
  where "rreq ≡ λ(hops, rreqid, dip, dsn, dsk, oip, osn, sip).
  Rreq hops rreqid dip dsn dsk oip osn sip"

lemma rreq_simp [simp]:
  "rreq(hops, rreqid, dip, dsn, dsk, oip, osn, sip) = Rreq hops rreqid dip dsn dsk oip osn sip"
  unfolding rreq_def by simp

definition rrep :: "nat × ip × sqn × ip × ip ⇒ msg"
  where "rrep ≡ λ(hops, dip, dsn, oip, sip). Rrep hops dip dsn oip sip"

lemma rrep_simp [simp]:
  "rrep(hops, dip, dsn, oip, sip) = Rrep hops dip dsn oip sip"
  unfolding rrep_def by simp

definition rerr :: "(ip ⇀ sqn) × ip ⇒ msg"
  where "rerr ≡ λ(dests, sip). Rerr dests sip"

lemma rerr_simp [simp]:
  "rerr(dests, sip) = Rerr dests sip"
  unfolding rerr_def by simp

lemma not_eq_newpkt_rreq [simp]: "¬ eq_newpkt (Rreq hops rreqid dip dsn dsk oip osn sip)"
  unfolding eq_newpkt_def by simp

lemma not_eq_newpkt_rrep [simp]: "¬ eq_newpkt (Rrep hops dip dsn oip sip)"
  unfolding eq_newpkt_def by simp

lemma not_eq_newpkt_rerr [simp]: "¬ eq_newpkt (Rerr dests sip)"
  unfolding eq_newpkt_def by simp

lemma not_eq_newpkt_pkt [simp]: "¬ eq_newpkt (Pkt d dip sip)"
  unfolding eq_newpkt_def by simp

definition pkt :: "data × ip × ip ⇒ msg"
  where "pkt ≡ λ(d, dip, sip). Pkt d dip sip"

lemma pkt_simp [simp]:
  "pkt(d, dip, sip) = Pkt d dip sip"
  unfolding pkt_def by simp
```

2.3 The AODV protocol

theory B_Aodv
imports B_Aodv_Data B_Aodv_Message
  "../../../AWN/AWN_SOS_Labels" "../../../AWN/AWN_Invariants"
begin

2.3.1 Data state

record state =
  ip :: "ip"
  sn :: "sqn"
  rt :: "rt"
  rreqs :: "(ip × rreqid) set"
  store :: "store"

end
abbreviation aodv_init :: "ip ⇒ state"
where "aodv_init i ≡ |
ip = i,
  sn = 1,
  rt = empty,
  rreqs = {},
  store = empty,
msg = (SOME x. True),
data = (SOME x. True),
dests = (SOME x. True),
pre = (SOME x. True),
rreqid = (SOME x. True),
dip = (SOME x. True),
oip = (SOME x. True),
hops = (SOME x. True),
dsn = (SOME x. True),
dsk = (SOME x. True),
osn = (SOME x. True),
sip = (SOME x. x ≠ i)
|"
2.3.2 Auxilliary message handling definitions

definition is_newpkt
where "is_newpkt ξ ≡ case msg ξ of
         Newpkt data' dip' ⇒ \{ ξ[|data := data', dip := dip'] \}
      | _ ⇒ {}"

definition is_pkt
where "is_pkt ξ ≡ case msg ξ of
         Pkt data' dip' oip' ⇒ \{ ξ[|data := data', dip := dip', oip := oip'] \}
      | _ ⇒ {}"

definition is_rreq
where "is_rreq ξ ≡ case msg ξ of
         Rreq hops' rreqid' dip' dsn' dsk' oip' osn' sip' ⇒ 
         \{ ξ[|hops := hops', rreqid := rreqid', dip := dip', dsn := dsn',
                dsk := dsk', oip := oip', osn := osn', sip := sip'] \}
      | _ ⇒ {}"

lemma is_rreq_asm [dest!]:
  assumes "ξ' ∈ is_rreq ξ"
  shows "(∀ hops' rreqid' dip' dsn' dsk' oip' osn' sip'.
             msg ξ = Rreq hops' rreqid' dip' dsn' dsk' oip' osn' sip' ∧
                 dsk := dsk', oip := oip', osn := osn', sip := sip'] )"
  using assms unfolding is_rreq_def
  by (cases "msg ξ") simp_all

definition is_rrep
where "is_rrep ξ ≡ case msg ξ of
         Rrep hops' dip' dsn' oip' sip' ⇒ 
      | _ ⇒ {}"

lemma is_rrep_asm [dest!]:
  assumes "ξ' ∈ is_rrep ξ"
  shows "(∃ hops' dip' dsn' oip' sip'.
             msg ξ = Rrep hops' dip' dsn' oip' sip' ∧
  using assms unfolding is_rrep_def
  by (cases "msg ξ") simp_all

definition is_rerr
where "is_rerr ξ ≡ case msg ξ of
         Rerr dests' sip' ⇒ \{ ξ[|dests := dests', sip := sip'] \}
      | _ ⇒ {}"

lemma is_rerr_asm [dest!]:
  assumes "ξ' ∈ is_rerr ξ"
  shows "(∃ dests' sip'.
             msg ξ = Rerr dests' sip' ∧
             ξ' = ξ[|dests := dests', sip := sip'] )"
  using assms unfolding is_rerr_def
  by (cases "msg ξ") simp_all

lemmas is_msg_defs =
is_rerr_def is_rrep_def is_rreq_def is_pkt_def is_newpkt_def

lemma is_msg_inv_ip [simp]:
    "ξ' ∈ is_rerr ξ    \implies ip ξ' = ip ξ"
    "ξ' ∈ is_rrep ξ    \implies ip ξ' = ip ξ"
    "ξ' ∈ is_rreq ξ    \implies ip ξ' = ip ξ"
unfolding is_msg_defs by (cases "msg \( \xi \)", clarsimp+)+

lemma is_msg_inv_sn [simp]:
  "\( \xi' \in is_rerr \( \xi \) \) \( \Rightarrow \) sn \( \xi' \) = sn \( \xi \)"
  "\( \xi' \in is_rrep \( \xi \) \) \( \Rightarrow \) sn \( \xi' \) = sn \( \xi \)"
  "\( \xi' \in is_rreq \( \xi \) \) \( \Rightarrow \) sn \( \xi' \) = sn \( \xi \)"
  "\( \xi' \in is_pkt \( \xi \) \) \( \Rightarrow \) sn \( \xi' \) = sn \( \xi \)"
  "\( \xi' \in is_newpkt \( \xi \) \) \( \Rightarrow \) sn \( \xi' \) = sn \( \xi \)"
unfolding is_msg_defs
by (cases "msg \( \xi \)", clarsimp+)+

lemma is_msg_inv_rt [simp]:
  "\( \xi' \in is_rerr \( \xi \) \) \( \Rightarrow \) rt \( \xi' \) = rt \( \xi \)"
  "\( \xi' \in is_rrep \( \xi \) \) \( \Rightarrow \) rt \( \xi' \) = rt \( \xi \)"
  "\( \xi' \in is_rreq \( \xi \) \) \( \Rightarrow \) rt \( \xi' \) = rt \( \xi \)"
  "\( \xi' \in is_pkt \( \xi \) \) \( \Rightarrow \) rt \( \xi' \) = rt \( \xi \)"
  "\( \xi' \in is_newpkt \( \xi \) \) \( \Rightarrow \) rt \( \xi' \) = rt \( \xi \)"
unfolding is_msg_defs
by (cases "msg \( \xi \)", clarsimp+)+

lemma is_msg_inv_rreqs [simp]:
  "\( \xi' \in is_rerr \( \xi \) \) \( \Rightarrow \) rreqs \( \xi' \) = rreqs \( \xi \)"
  "\( \xi' \in is_rrep \( \xi \) \) \( \Rightarrow \) rreqs \( \xi' \) = rreqs \( \xi \)"
  "\( \xi' \in is_rreq \( \xi \) \) \( \Rightarrow \) rreqs \( \xi' \) = rreqs \( \xi \)"
  "\( \xi' \in is_pkt \( \xi \) \) \( \Rightarrow \) rreqs \( \xi' \) = rreqs \( \xi \)"
  "\( \xi' \in is_newpkt \( \xi \) \) \( \Rightarrow \) rreqs \( \xi' \) = rreqs \( \xi \)"
unfolding is_msg_defs
by (cases "msg \( \xi \)", clarsimp+)+

lemma is_msg_inv_store [simp]:
  "\( \xi' \in is_rerr \( \xi \) \) \( \Rightarrow \) store \( \xi' \) = store \( \xi \)"
  "\( \xi' \in is_rrep \( \xi \) \) \( \Rightarrow \) store \( \xi' \) = store \( \xi \)"
  "\( \xi' \in is_rreq \( \xi \) \) \( \Rightarrow \) store \( \xi' \) = store \( \xi \)"
  "\( \xi' \in is_pkt \( \xi \) \) \( \Rightarrow \) store \( \xi' \) = store \( \xi \)"
  "\( \xi' \in is_newpkt \( \xi \) \) \( \Rightarrow \) store \( \xi' \) = store \( \xi \)"
unfolding is_msg_defs
by (cases "msg \( \xi \)", clarsimp+)+

lemma is_msg_inv_sip [simp]:
  "\( \xi' \in is_pkt \( \xi \) \) \( \Rightarrow \) sip \( \xi' \) = sip \( \xi \)"
  "\( \xi' \in is_newpkt \( \xi \) \) \( \Rightarrow \) sip \( \xi' \) = sip \( \xi \)"
unfolding is_msg_defs
by (cases "msg \( \xi \)", clarsimp+)+

2.3.3 The protocol process

datatype pseqp =
  PAodv |
  PNewPkt |
  PPkt |
  PRreq |
  PRrep |
  PRerr

fun nat_of_seqp :: "pseqp \Rightarrow nat"
where
  "nat_of_seqp PAodv = 1"
| "nat_of_seqp PNewPkt = 2"
| "nat_of_seqp PPkt = 3"
| "nat_of_seqp PRreq = 4"
| "nat_of_seqp PRrep = 5"

189
instantiation "pseqp" :: ord
begin
  definition less_eq_seqp [iff]: "l1 ≤ l2 = (nat_of_seqp l1 ≤ nat_of_seqp l2)"
  definition less_seqp [iff]: "l1 < l2 = (nat_of_seqp l1 < nat_of_seqp l2)"
instance ..
end

abbreviation AODV
where
  "AODV ≡ λ_. [clear_locals] call(PAodv)"

abbreviation PKT
where
  "PKT args ≡
  [ξ. let (data, dip, oip) = args ξ in
   (clear_locals ξ) [] data := data, dip := dip, oip := oip ]]
call(PPkt)"

abbreviation NEWPKT
where
  "NEWPKT args ≡
  [ξ. let (data, dip) = args ξ in
   (clear_locals ξ) [] data := data, dip := dip ]]
call(PNewPkt)"

abbreviation RREQ
where
  "RREQ args ≡
  [ξ. let (hops, rreqid, dip, dsn, dsk, oip, osn, sip) = args ξ in
   (clear_locals ξ) [] hops := hops, rreqid := rreqid, dip := dip,
   dsn := dsn, dsk := dsk, oip := oip,
   osn := osn, sip := sip ]]
call(PRreq)"

abbreviation RREP
where
  "RREP args ≡
  [ξ. let (hops, dip, dsn, oip, sip) = args ξ in
   (clear_locals ξ) [] hops := hops, dip := dip, dsn := dsn,
   oip := oip, sip := sip ]]
call(PRrep)"

abbreviation RERR
where
  "RERR args ≡
  [ξ. let (dests, sip) = args ξ in
   (clear_locals ξ) [] dests := dests, sip := sip ]]
call(PRerr)"

fun Γ_AODV :: "(state, msg, pseqp, pseqp label) seqp_env"
where
  "Γ_AODV PAodv = labelled PAodv (
    receive(λmsg' ξ. ξ [] msg := msg' )).
    (is_newpkt) NEWPKT(λξ. (data ξ, ip ξ))
    (is_pkt) PKT(λξ. (data ξ, dip ξ, oip ξ))
    (is_rreq)
      [ξ. ξ [] rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ, {}) ]]
    RREQ(λξ. (hops ξ, rreqid ξ, dip ξ, dsn ξ, dsk ξ, oip ξ, osn ξ, sip ξ))
    (is_rrep)
      [ξ. ξ [] rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ, {}) ]]
    RREP(λξ. (hops ξ, dip ξ, dsn ξ, oip ξ, sip ξ))
    (is_rerr)
AODV

191
(ξ. dip ξ = ip ξ)
(ξ. ξ ∈ vD (rt ξ) \wedge sn := max (sn ξ) (dsn ξ) \} ]
unicast(λξ. the (nhop (rt ξ) (oip ξ)), λξ. rrep(0, dip ξ, sn ξ, oip ξ, ip ξ)).AODV()
)

(ξ. dip ξ \notin vD (rt ξ) \wedge rreq(hops ξ + 1, rreqid ξ, dip ξ, max (sqn (rt ξ) (oip ξ)) (dsn ξ)),
dsk ξ, oip ξ, sn ξ, ip ξ)).AODV()
)

)

"Γ_{AODV} PRrep = labelled PRrep (}
(ξ. ξ \notin rt := update (rt ξ) (dip ξ) (dsn ξ, kno, val, hops ξ + 1, sip ξ, \} ] ]
(ξ. oip ξ \notin ip ξ)
(ξ. oip ξ ∈ vD (rt ξ) \wedge dip ξ ∈ vD (rt ξ))
(ξ. ξ \notin rt := the (addpreRT (rt ξ) (dip ξ))
\{ (the (nhop (rt ξ) (oip ξ)))) \} ] ]
unicast(λξ. the (nhop (rt ξ) (oip ξ)), λξ. rrep(0, dip ξ, sn ξ, oip ξ, ip ξ)).AODV()
)

(ξ. dip ξ \notin vD (rt ξ) \vee dip ξ \notin vD (rt ξ))

(ξ. oip ξ \notin ip ξ)

(ξ. dip ξ \notin vD (rt ξ) \wedge nhop (rt ξ) rip = nhop (rt ξ) (oip ξ))
then Some (inc (sqn (rt ξ) (rip)) else None) ] ]

(ξ. dip ξ \notin vD (rt ξ) \wedge dip ξ \notin vD (rt ξ) )
\[
\begin{align*}
\text{groupcast}(\lambda \xi. \text{pre} (\xi), \lambda \xi. \text{rerr} (\xi)) \\
\text{PAodv} = \text{seqp_skeleton} (\Gamma_{\text{AODV}} \text{PAodv}) \\
\text{PNewPkt} = \text{seqp_skeleton} (\Gamma_{\text{AODV}} \text{PNewPkt}) \\
\text{PPkt} = \text{seqp_skeleton} (\Gamma_{\text{AODV}} \text{PPkt}) \\
\text{PRreq} = \text{seqp_skeleton} (\Gamma_{\text{AODV}} \text{PRreq}) \\
\text{PRrep} = \text{seqp_skeleton} (\Gamma_{\text{AODV}} \text{PRrep}) \\
\text{PRerr} = \text{seqp_skeleton} (\Gamma_{\text{AODV}} \text{PRerr})
\end{align*}
\]

\text{lemma } \Gamma_{\text{AODV}} \text{_skeleton_wf [simp]:} \\
"\text{wellformed } \Gamma_{\text{AODV}} \text{_skeleton}"

\text{proof (rule, intro allI)}
\begin{itemize}
\item fix \(pn\) \(pn'\)
\item show "\(\text{call}(pn') \notin \text{ctermsl} (\Gamma_{\text{AODV}} \text{skeleton} pn)\)"
\item by (cases pn) simp_all
\end{itemize}
\text{qed}

\text{declare } \Gamma_{\text{AODV}} \text{_skeleton.simps [simp del, code del]}
\text{lemmas } \Gamma_{\text{AODV}} \text{_skeleton_simps [simp, code]} = \Gamma_{\text{AODV}} \text{.simps [simplified]}

\text{fun } \Gamma_{\text{AODV}} \text{_skeleton}
\text{where}
\begin{align*}
\text{"}\Gamma_{\text{AODV}} \text{_skeleton} \text{PAodv} &= \text{seqp_skeleton} (\Gamma_{\text{AODV}} \text{PAodv}) \\
\text{PAodv} &= \text{seqp_skeleton} (\Gamma_{\text{AODV}} \text{PNewPkt}) \\
\text{PPkt} &= \text{seqp_skeleton} (\Gamma_{\text{AODV}} \text{PPkt}) \\
\text{PRreq} &= \text{seqp_skeleton} (\Gamma_{\text{AODV}} \text{PRreq}) \\
\text{PRrep} &= \text{seqp_skeleton} (\Gamma_{\text{AODV}} \text{PRrep}) \\
\text{PRerr} &= \text{seqp_skeleton} (\Gamma_{\text{AODV}} \text{PRerr}) \text{"
}\end{align*}

\text{lemma } \sigma_{\text{AODV}} :: "ip \Rightarrow (\text{state} \times (\text{state, msg, pseqp, pseqp label}) \text{ seq}) \text{ set}"
\text{where } "\sigma_{\text{AODV}} \ i \equiv \{(\text{aodv_init} \ i, \ \Gamma_{\text{AODV}} \text{PAodv})\}"

\text{abbreviation } \text{paodv} :: "ip \Rightarrow (\text{state} \times (\text{state, msg, pseqp, pseqp label}) \text{ seq, msg seq_action}) \text{ automaton}"
\text{where }
"\text{paodv} \ i \equiv \{(\text{init} = \sigma_{\text{AODV}} \ i, \ \text{trans} = \text{seqp_sos} \ \Gamma_{\text{AODV}} \ \})\"

\text{lemma } \text{aodv_trans: } "\text{trans} (\text{paodv} \ i) = \text{seqp_sos} \ \Gamma_{\text{AODV}} \text{"
\text{by simp}

\text{lemma } \text{aodv_control_within [simp]: } "\text{control_within} \ \Gamma_{\text{AODV}} \ (\text{init} (\text{paodv} \ i))"
\text{unfolding } \sigma_{\text{AODV}} \text{def by (rule control_withinI)} (\text{auto simp del: } \Gamma_{\text{AODV.simps}})
lemma aodv_wf [simp]:
  "wellformed \( \Gamma_{AODV} \)
proof (rule, intro allI)
  fix pn pn'
  show "call(pn') \notin \text{stermsl}(\Gamma_{AODV} \ pn)"
    by (cases pn) simp_all
qed

lemmas aodv_labels_not_empty [simp] = labels_not_empty [OF aodv_wf]

lemma aodv_ex_label [intro]: \"\exists l. l \in \text{labels} \ \Gamma_{AODV} p\"
by (metis aodv_labels_not_empty all_not_in_conv)

lemma aodv_ex_labelE [elim]:
  assumes \"\forall l \in \text{labels} \ \Gamma_{AODV} p. P l p\"
  and \"\exists l. P l p \Rightarrow Q\"
  shows "Q"
using assms by (metis aodv_ex_label)

lemma aodv_simple_labels [simp]: "simple_labels \( \Gamma_{AODV} \)
proof
  fix pn p
  assume "p \in \text{subterms}(\Gamma_{AODV} \ pn)"
  thus "\exists ! l. \text{labels} \ \Gamma_{AODV} p = \{l\}"
    by (cases pn) (simp_all cong: seqp_congs | elim disjE)+
qed

lemma \( \sigma_{AODV} \)_labels [simp]: "(\xi, p) \in \sigma_{AODV} i = \Rightarrow \text{labels} \ \Gamma_{AODV} p = \{PAodv-:0\}"
unfolding \( \sigma_{AODV} \)_def by simp

lemma aodv_init_kD_empty [simp]:
  "(\xi, p) \in \sigma_{AODV} i = \Rightarrow \ kD (rt \xi) = {}"
unfolding \( \sigma_{AODV} \)_def kD_def by simp

lemma aodv_init_sip_not_ip [simp]: "$\neg (\text{sip} (\text{aodv\_init} i) = i)\" by simp

lemma aodv_init_sip_not_ip' [simp]:
  assumes "$\neg (\text{sip} i = i)\"
  shows "$\neg (\text{sip} (\text{clear\_locals} \xi) = i)\"
  using assms unfolding \( \sigma_{AODV} \)_def by simp

lemma clear_locals_sip_not_ip':
  assumes "$\neg (\text{sip} i = i)\"
  shows "$\neg (\text{sip} (\text{clear\_locals} \xi) = i)\"
  using assms by auto

Stop the simplifier from descending into process terms.

declare seqp_congs [cong]

Configure the main invariant tactic for AODV.

declare \( \Gamma_{AODV} \)_simps [cterms_env]
aodv_proc_cases [ctermsml_cases]
seq_invariant_ctermsI [OF aodv_wf aodv_control_within aodv_simple_labels aodv_trans, cterms_intros]
seq_step_invariant_ctermsI [OF aodv_wf aodv_control_within aodv_simple_labels aodv_trans, cterms_intros]

end
2.4 Invariant assumptions and properties

theory B_Aodv_Predicates
imports B_Aodv
begin

Definitions for expression assumptions on incoming messages and properties of outgoing messages.

abbreviation not_Pkt :: "msg ⇒ bool"
where "not_Pkt m ≡ case m of Pkt _ _ _ ⇒ False | _ ⇒ True"

definition msg_sender :: "msg ⇒ ip"
where "msg_sender m ≡ case m of Rreq _ _ _ _ _ _ _ ipc ⇒ ipc
| Rrep _ _ _ _ ipc ⇒ ipc
| Rerr _ _ ipc ⇒ ipc
| Pkt _ _ _ ipc ⇒ ipc"

lemma msg_sender_simps [simp]:
"⋀ hops rreqid dip dsn dsk oip osn sip.
  msg_sender (Rreq hops rreqid dip dsn dsk oip osn sip) = sip"
"⋀ hops dip dsn oip sip. msg_sender (Rrep hops dip dsn oip sip) = sip"
"⋀ dests sip. msg_sender (Rerr dests sip) = sip"
"⋀ d dip sip. msg_sender (Pkt d dip sip) = sip"
unfolding msg_sender_def by simp_all

definition msg_zhops :: "msg ⇒ bool"
where "msg_zhops m ≡ case m of
  Rreq hopsc _ _ _ _ oipc osnc ipcc ⇒ hopsc = 0 −→ oipc = sipc
  | Rrep hopsc _ _ _ _ sipc ⇒ hopsc = 0 −→ dipc = sipc
  | _ ⇒ True"

lemma msg_zhops_simps [simp]:
"⋀ hops rreqid dip dsn dsk oip osn sip.
  msg_zhops (Rreq hops rreqid dip dsn dsk oip osn sip) = (hops = 0 −→ oip = sip)"
"⋀ hops dip dsn oip sip. msg_zhops (Rrep hops dip dsn oip sip) = (hops = 0 −→ dip = sip)"
"⋀ dests sip. msg_zhops (Rerr dests sip) = True"
"⋀ d dip. msg_zhops (Newpkt d dip) = True"
"⋀ d dip sip. msg_zhops (Pkt d dip sip) = True"
unfolding msg_zhops_def by simp_all

definition rreq_rrep_sn :: "msg ⇒ bool"
where "rreq_rrep_sn m ≡ case m of
  Rreq hopsc _ _ _ _ _ _ _ osnc _ ⇒ osnc ≥ 1
  | Rrep _ _ dsnc _ _ ⇒ dsnc ≥ 1
  | _ ⇒ True"

lemma rreq_rrep_sn_simps [simp]:
"⋀ hops rreqid dip dsn dsk oip osn sip.
  rreq_rrep_sn (Rreq hops rreqid dip dsn dsk oip osn sip) = (osn ≥ 1)"
"⋀ hops dip dsn oip sip. rreq_rrep_sn (Rrep hops dip dsn oip sip) = (dsn ≥ 1)"
"⋀ dests sip. rreq_rrep_sn (Rerr dests sip) = True"
"⋀ d dip. rreq_rrep_sn (Newpkt d dip) = True"
"⋀ d dip sip. rreq_rrep_sn (Pkt d dip sip) = True"
unfolding rreq_rrep_sn_def by simp_all

definition rreq_rrep_fresh :: "rt ⇒ msg ⇒ bool"
where "rreq_rrep_fresh crt m ≡ case m of
  Rreq hopsc _ _ _ _ _ _ _ _ oipc osnc ipcc ⇒ (ipcc ≠ oipc −→
  oipc∈kD(crt) ∧ (sqn crt oip = osnc
  ∨ the (dhops crt oip) ≤ hopsc
  ∧ the (flag crt oip) = val))
  | Rrep hopsc _ _ _ _ dipc _ _ _ ipcc ⇒ (ipcc ≠ dipc −→
    dipc∈kD(crt)
  ∧ sqn crt dipc = dsnc
  ∧ the (dhops crt dipc) = hopsc
  ∧ the (flag crt dipc) = val)

lemma rreq_rrep_fresh [simp]:
"∀ hops rreqid dip dsn dsk oip osn sip.
   rreq_rrep_fresh crt (Rreq hops rreqid dip dsn dsk oip osn sip) =
   (sip ≠ oip → oip ∈ kD(crt)
   ∧ (sqn crt oip > osn
   ∨ (sqn crt oip = osn
   ∧ the (dhops crt oip) ≤ hops
   ∧ the (flag crt oip) = val)))"
"∀ hops dip dsn oip sip.  rreq_rrep_fresh crt (Rrep hops dip dsn oip sip) =
   (sip ≠ dip → dip ∈ kD(crt)
   ∧ sqn crt dip = dsn
   ∧ the (dhops crt dip) = hops
   ∧ the (flag crt dip) = val)
"∀ dests sip.  rreq_rrep_fresh crt (Rerr dests sip) = True
"∀ dip.  rreq_rrep_fresh crt (Newpkt dip) = True
"∀ dip sip.  rreq_rrep_fresh crt (Pkt dip sip) = True
unfolding rreq_rrep_fresh_def by simp_all

definition rerr_invalid :: "rt ⇒ msg ⇒ bool"
where "rerr_invalid crt m = (case m of
   Rreq _ _ _ _ _ _ _ _ _ ⇒ False
   Rrep _ _ _ _ _ _ _ _ _ ⇒ True
   Rerr _ _ _ _ _ _ _ _ _ ⇒ False
   Newpkt _ _ _ _ _ _ _ _ _ ⇒ False
   Pkt _ _ _ _ _ _ _ _ _ ⇒ False)
unfolding rerr_invalid_def by simp_all

lemma not_in_net_ips_fst_init_missing [simp]:
assumes "i /∈ net_ips σ"
shows "fst (initmissing (netgmap fst σ)) i = aodv_init i"
using assms unfolding initmissing_def by simp

lemma fst_initmissing_netmap_pair_fst [simp]:
"fst (initmissing (netgmap (λ (p, q). (fst (id p), snd (id p), q)) s)) =
   fst (initmissing (netgmap fst s))"
unfolding initmissing_def by auto

We introduce a streamlined alternative to initmissing with netgmap to simplify invariant statements and thus facilitate their comprehension and presentation.

lemma fst_initmissing_netmap_default_aodv_init_netlift:
"fst (initmissing (netgmap fst s)) = default aodv_init (netlift fst s)
unfolding initmissing_def default_def by (simp add: fst_netmap_netlift_def: One_nat_def)

definition netglobal :: "((nat ⇒ state) ⇒ bool) ⇒ ((state × 'b) × 'c) net_state ⇒ bool"
where "netglobal P = (λs. P (default aodv_init (netlift fst s)))"
2.5 Quality relations between routes

theory B_Fresher
imports B_Aodv_Data
begin

2.5.1 Net sequence numbers

On individual routes

definition
  \textit{nsqn} :: \textit{r} \Rightarrow \textit{sqn}
where
  \textit{nsqn} \ (	extit{r}) \equiv \text{if } \pi_4(\textit{r}) = \text{val} \lor \pi_2(\textit{r}) = 0 \text{ then } \pi_2(\textit{r}) - 1 \text{ else } \pi_2(\textit{r})

lemma \textit{nsqn}\textunderscore\textit{def}':
  \textit{nsqn} \textit{r} = (\text{if } \pi_4(\textit{r}) = \text{inv} \text{ then } \pi_2(\textit{r}) - 1 \text{ else } \pi_2(\textit{r}))

unfolding \textit{nsqn}\textunderscore\textit{def} by simp

lemma \textit{nsqn}\textunderscore\textit{zero} [simp]:
  \"\textit{\A dsn dsk flag hops nhip pre. nsqn} \textit{(0, dsk, flag, hops, nhip, pre)} = 0\"

unfolding \textit{nsqn}\textunderscore\textit{def} by clarsimp

lemma \textit{nsqn}\textunderscore\textit{val} [simp]:
  \"\textit{\A dsn dsk hops nhip pre. nsqn} \textit{(dsn, dsk, val, hops, nhip, pre)} = dsn\"

unfolding \textit{nsqn}\textunderscore\textit{def} by clarsimp

lemma \textit{nsqn}\textunderscore\textit{inv} [simp]:
  \"\textit{\A dsn dsk hops nhip pre. nsqn} \textit{(dsn, dsk, inv, hops, nhip, pre)} = dsn - 1\"

unfolding \textit{nsqn}\textunderscore\textit{def} by clarsimp

lemma \textit{nsqn}\textunderscore\textit{lte_dsn} [simp]:
  \"\textit{\A dsn dsk flag hops nhip pre. nsqn} \textit{(dsn, dsk, flag, hops, nhip, pre)} \leq dsn\"

unfolding \textit{nsqn}\textunderscore\textit{def} by clarsimp

On routes in routing tables

definition
  \textit{nsqn} :: \textit{rt} \Rightarrow \textit{ip} \Rightarrow \textit{sqn}
where
  \textit{nsqn} \equiv \text{lamb\textit{rt dip. case }\sigma_{\text{route}}(\textit{rt, dip})\text{ of None }\Rightarrow 0 \mid \text{Some } \textit{r} \Rightarrow \textit{nsqn}_r(\textit{r})}"

lemma \textit{nsqn}\textunderscore\textit{sqn}\textunderscore\textit{def}:
  \"\textit{\A rt dip. nsqn} \textit{rt dip} = (\text{if } \text{flag rt dip} = \text{Some} \text{ val} \lor \text{sqn rt dip} = 0 \text{ then } \text{sqn rt dip} \text{ else } \text{sqn rt dip - 1})\"

unfolding \textit{nsqn}\textunderscore\textit{def} by (clarsimp split: option.split) auto

lemma \textit{not_in_kD}\textunderscore\textit{nsqn} [simp]:
  \text{assumes } \"\textit{dip} \notin \textit{kD}(\textit{rt})\"\n  \text{shows } \"\textit{nsqn} \textit{rt dip} = 0\"

using \textit{assms} unfolding \textit{nsqn}\textunderscore\textit{def} by simp

lemma \textit{kD}\textunderscore\textit{nsqn}:
  \text{assumes } \"\textit{dip} \in \textit{kD}(\textit{rt})\"\n  \text{shows } \"\textit{nsqn} \textit{rt dip} = \textit{nsqn}_r(\text{the } (\sigma_{\text{route}}(\textit{rt, dip})))\"

using \textit{assms} [THEN \textit{kD_Some}] unfolding \textit{nsqn}\textunderscore\textit{def} by clarsimp

lemma \textit{nsqn}\textunderscore\textit{r_flag}\textunderscore\textit{pred} [simp, intro]:
  \text{fixes} \textit{dsn dsk flag hops nhip pre}\n  \text{assumes } \"\textit{P} (\textit{nsqn}, (\textit{dsn, dsk, val, hops, nhip, pre}))\"
  \text{and } \"\textit{P} (\textit{nsqn}, (\textit{dsn, dsk, inv, hops, nhip, pre}))\"
  \text{shows } \"\textit{P} (\textit{nsqn}, (\textit{dsn, dsk, flag, hops, nhip, pre}))\"

using \textit{assms} by (cases flag) auto

lemma \textit{nsqn}\textunderscore\textit{add}\textunderscore\textit{preRT}\textunderscore\textit{inv} [simp]:
\[ \forall \text{rt dip npre dip}'. \text{dip} \in kD(\text{rt}) \implies \] 
\[ \text{nsqn}, (\text{the (the (addpreRT rt dip npre) dip')} ) = \text{nsqn}, (\text{the (rt dip')} ) \]

unfolding addpreRT_def nsqn杜兰def

by (frule kD_Some) (clarsimp split: option.split)

lemma sqn_nsqn:
"\[ \forall \text{rt dip. sqn rt dip - 1} \leq \text{nsqn rt dip} \]

unfolding sqn杜兰def nsqn杜兰def by (clarsimp split: option.split)

lemma nsqn_sqn: "\[ \text{nsqn rt dip} \leq \text{sqn rt dip} \]

unfolding sqn杜兰def nsqn杜兰def by (cases "rt dip") auto

lemma val_nsqn_sqn [elim, simp]:
assumes "\[ \text{ip} \in kD(\text{rt}) \]
and "\[ \text{the (flag rt ip)} = \text{val} \]
shows "\[ \text{nsqn rt ip} = \text{sqn rt ip} \]

using assms unfolding nsqn杜兰sqn杜兰def by auto

lemma vD_nsqn_sqn [elim, simp]:
assumes "\[ \text{ip} \in vD(\text{rt}) \]
shows "\[ \text{nsqn rt ip} = \text{sqn rt ip} \]

proof -
  from \[ \text{ip} \in vD(\text{rt}) \]
  have "\[ \text{ip} \in kD(\text{rt}) \]
  and "\[ \text{the (flag rt ip)} = \text{val} \]
  by auto
  thus ?thesis ..

qed

lemma inv_nsqn_sqn [elim, simp]:
assumes "\[ \text{ip} \in kD(\text{rt}) \]
and "\[ \text{the (flag rt ip)} = \text{inv} \]
shows "\[ \text{nsqn rt ip} = \text{sqn rt ip} - 1 \]

using assms unfolding nsqn杜兰sqn杜兰def by auto

lemma iD_nsqn_sqn [elim, simp]:
assumes "\[ \text{ip} \in iD(\text{rt}) \]
shows "\[ \text{nsqn rt ip} = \text{sqn rt ip} - 1 \]

proof -
  from \[ \text{ip} \in iD(\text{rt}) \]
  have "\[ \text{ip} \in kD(\text{rt}) \]
  and "\[ \text{the (flag rt ip)} = \text{inv} \]
  by auto
  thus ?thesis ..

qed

lemma nsqn_update_changed_kno_val [simp]: "\[ \forall \text{rt ip dsn dsk hops nhip. rt} \neq \text{update rt ip (dsn, dsk, hops, nhip, {})} \implies \text{nsqn (update rt ip (dsn, dsk, hops, nhip, {})) ip} = \text{dsn} \]

using assms unfolding nsqn杜兰def update杜兰def

by (clarsimp simp: kD_nsqn杜兰split: option杜兰split杜兰asm option杜兰split杜兰split杜兰if杜兰asm)

  (metis fun杜兰upd杜兰triv)

lemma nsqn_addpreRT_inv [simp]:
"\[ \forall \text{rt dip npre dip}'. \text{dip} \in kD(\text{rt}) \implies \] 
\[ \text{nsqn (the (addpreRT rt dip npre)) dip'} = \text{nsqn rt dip'} \]

unfolding addpreRT杜兰def nsqn杜兰杜兰def

by (frule kD杜兰Some) (clarsimp split: option杜兰split)

lemma nsqn_update_other [simp]:
fixes dsn dsk flag hops nhip pre rt ip
assumes "\[ \text{dip} \neq \text{ip} \]
shows "\[ \text{nsqn (update rt ip (dsn, dsk, flag, hops, nhip, pre)) dip} = \text{nsqn rt dip} \]

using assms unfolding nsqn杜兰def

by (clarsimp split: option杜兰split)

lemma nsqnDispose_eq:
assumes "\[ \text{dip} \in kD(\text{rt}) \]

198
and "dests dip = Some rsn"
shows "nsqn (invalidate rt dests) dip = rsn - 1"
using asms
proof -
  from asms obtain dsk hops nhip pre
  where "invalidate rt dests dip = Some (rsn, dsk, inv, hops, nhip, pre)"
  unfolding invalidate_def
  by auto
  moreover from ⟨dip ∈ kD(rt)⟩ have "dip ∈ kD(invalidate rt dests)" by simp
  ultimately show ?thesis
  using ⟨dests dip = Some rsn⟩ by simp
qed

lemma nsqn_invalidate_other [simp]:
  assumes "dip ∈ kD(rt)"
  and "dip /∈ dom dests"
  shows "nsqn (invalidate rt dests) dip = nsqn rt dip"
  using asms
  by (clarsimp simp add: kD_nsqn)

2.5.2 Comparing routes
definition fresher :: "r ⇒ r ⇒ bool" ("(_/ ⊑ _)" [51, 51] 50)
  where "fresher r r' ≡ ((nsqn, r < nsqn, r') ∨ (nsqn, r = nsqn, r' ∧ π₅(r) ≥ π₅(r')))"

lemma fresherI1 [intro]:
  assumes "nsqn, r < nsqn, r'"
  shows "r ⊑ r'"
  unfolding fresher_def using asms by simp

lemma fresherI2 [intro]:
  assumes "nsqn, r = nsqn, r'" and "π₅(r) ≥ π₅(r')"
  shows "r ⊑ r'"
  unfolding fresher_def using asms by simp

lemma fresherI [intro]:
  assumes "(nsqn, r < nsqn, r') ∨ (nsqn, r = nsqn, r' ∧ π₅(r) ≥ π₅(r'))"
  shows "r ⊑ r'"
  unfolding fresher_def using asms .

lemma fresherE [elim]:
  assumes "r ⊑ r'" and "nsqn, r < nsqn, r'"Longrightarrow P r r'"
  and "nsqn, r = nsqn, r' ∧ π₅(r) ≥ π₅(r')"Longrightarrow P r r'"
  shows "P r r'"
  using asms unfolding fresher_def by auto

lemma fresher_refl [simp]: "r ⊑ r"
  unfolding fresher_def by simp

lemma fresher_trans [elim, trans]:
  "[ x ⊑ y; y ⊑ z ] ⇒ x ⊑ z"
  unfolding fresher_def by auto

lemma not_fresher_trans [elim, trans]:
  "[ ¬(x ⊑ y); ¬(z ⊑ x) ] ⇒ ¬(z ⊑ y)"
  unfolding fresher_def by auto

lemma fresher_dsn_flag_hops_const [simp]:
  fixes dsn dsk dsk' flag hops nhip nhip' pre pre'
  shows "(dsn, dsk, flag, hops, nhip, pre) ⊑ (dsn, dsk', flag, hops, nhip', pre')"
  unfolding fresher_def by (cases flag) simp_all
2.5.3 Comparing routing tables

definition
  \textit{rt\_fresher} :: "ip \Rightarrow rt \Rightarrow rt \Rightarrow bool"
where
  \textit{rt\_fresher} \equiv \lambda dip rt rt'. \textup{ the } (\sigma_{route}(rt, dip))) \sqsubseteq \textup{ (the } (\sigma_{route}(rt', dip)))"

abbreviation
  \textit{rt\_fresher\_syn} :: "rt \Rightarrow ip \Rightarrow rt \Rightarrow bool" ("(_/ \sqsubseteq _)"
where
  "rt1 \sqsubseteq rt2 \equiv rt\_fresher\_syn i rt1 rt2"

lemma \textit{rt\_fresher\_def'}:
  "\((rt1 \sqsubseteq rt2) = (nsqn_{r} (the (rt1 i)) < nsqn_{r} (the (rt2 i)) \lor
  nsqn_{r} (the (rt1 i)) = nsqn_{r} (the (rt2 i)) \land \pi_{5} (the (rt2 i)) \leq \pi_{5} (the (rt1 i)))"
unfolding \textit{rt\_fresher\_def} \textit{fresher\_def} by \textit{rule refl}

lemma \textit{single\_rt\_fresher} [intro]:
  assumes "the (rt1 ip) \sqsubseteq the (rt2 ip)"
  shows "rt1 \sqsubseteq ip rt2"
  using assms unfolding \textit{rt\_fresher\_def}.

lemma \textit{rt\_fresher\_single} [intro]:
  assumes "rt1 \sqsubseteq ip rt2"
  shows "the (rt1 ip) \sqsubseteq the (rt2 ip)"
  using assms unfolding \textit{rt\_fresher\_def}.

lemma \textit{rt\_fresher\_def2}:
  assumes "dip \in kD(rt1)"
  and "dip \in kD(rt2)"
  shows "(rt1 \sqsubseteq dip rt2) = (nsqn_{r} rt1 dip < nsqn rt2 dip
  \lor (nsqn rt1 dip = nsqn rt2 dip
  \land the (dhops rt1 dip) \geq the (dhops rt2 dip)))"
  using assms unfolding \textit{rt\_fresher\_def} \textit{fresher\_def} by \textit{simp add: kD_nsqn proj5_eq_dhops}

lemma \textit{rt\_fresher\_I1} [intro]:
  assumes "dip \in kD(rt1)"
  and "dip \in kD(rt2)"
  and "nsqn rt1 dip < nsqn rt2 dip"
  shows "rt1 \sqsubseteq dip rt2"
unfolding \textit{rt\_fresher\_def2} [OF assms(1-2)] using assms(3) by \textit{simp}

lemma \textit{rt\_fresher\_I2} [intro]:
  assumes "dip \in kD(rt1)"
  and "dip \in kD(rt2)"
  and "nsqn rt1 dip = nsqn rt2 dip"
  and "the (dhops rt1 dip) \geq the (dhops rt2 dip)"
  shows "rt1 \sqsubseteq dip rt2"
unfolding \textit{rt\_fresher\_def2} [OF assms(1-2)] using assms(3-4) by \textit{simp}

lemma \textit{rt\_fresher\_E} [elim]:
  assumes "rt1 \sqsubseteq dip rt2"
  and "dip \in kD(rt1)"
  and "dip \in kD(rt2)"
  and "[ nsqn rt1 dip < nsqn rt2 dip ] \Longrightarrow P rt1 rt2 dip"
  and "[ nsqn rt1 dip = nsqn rt2 dip; the (dhops rt1 dip) \geq the (dhops rt2 dip) ] \Longrightarrow P rt1 rt2 dip"
  shows "P rt1 rt2 dip"
using assms(1) unfolding \textit{rt\_fresher\_def2} [OF assms(2-3)] using assms(4-5) by \textit{auto}
lemma \texttt{rt\_fresher\_refl} [simp]: "rt \sqsubseteq dip rt"
  unfolding \texttt{rt\_fresher\_def} by simp

lemma \texttt{rt\_fresher\_trans} [elim, trans]:
  assumes "rt1 \sqsubseteq dip rt2"
  and "rt2 \sqsubseteq dip rt3"
  shows "rt1 \sqsubseteq dip rt3"
  using assms unfolding \texttt{rt\_fresher\_def} by auto

lemma \texttt{rt\_fresher\_if\_Some} [intro!]:
  assumes "the (rt dip) \sqsubseteq r"
  shows "rt \sqsubseteq dip (\lambda ip. if ip = dip then Some r else rt ip)"
  using assms unfolding \texttt{rt\_fresher\_def} by simp

definition \texttt{rt\_fresh\_as} :: "ip \Rightarrow rt \Rightarrow rt \Rightarrow bool"
where
"rt\_fresh\_as \equiv \lambda dip rt1 rt2. (rt1 \sqsubseteq dip rt2) \land (rt2 \sqsubseteq dip rt1)"

abbreviation \texttt{rt\_fresh\_as\_syn} :: "rt \Rightarrow ip \Rightarrow rt \Rightarrow bool" ("(_/ \approx / _)"
  [51, 999, 51] 50)
where
"rt1 \approx i rt2 \equiv rt\_fresh\_as i rt1 rt2"

lemma \texttt{rt\_fresh\_as\_refl} [simp]: "\forall rt dip. rt \approx dip rt"
  unfolding \texttt{rt\_fresh\_as\_def} by simp

lemma \texttt{rt\_fresh\_as\_trans} [simp, intro, trans]:
"\forall rt1 rt2 rt3 dip. \[ rt1 \approx dip rt2; rt2 \approx dip rt3 \] \Rightarrow rt1 \approx dip rt3"
  unfolding \texttt{rt\_fresh\_as\_def} \texttt{rt\_fresher\_def}
  by (metis (mono_tags) fresher\_trans)

lemma \texttt{rt\_fresh\_as\_I} [intro!]:
  assumes "rt1 \sqsubseteq dip rt2"
  and "rt2 \sqsubseteq dip rt1"
  shows assms unfolding \texttt{rt\_fresh\_as\_def} by simp

lemma \texttt{rt\_fresh\_as\_fresher\_I} [intro]:
  assumes "dip \in kD(rt1)"
  and "dip \in kD(rt2)"
  and "the (rt1 dip) \sqsubseteq the (rt2 dip)"
  and "the (rt2 dip) \sqsubseteq the (rt1 dip)"
  shows "rt1 \approx dip rt2"
  using assms unfolding \texttt{rt\_fresh\_as\_def}
  by (clarsimp dest!: single\_rt\_fresher)

lemma \texttt{nsqn\_rt\_fresh\_as\_I}:
  assumes "dip \in kD(rt)"
  and "dip \in kD(rt')"
  and "\texttt{nsqn rt dip} = \texttt{nsqn rt'} dip" and "\pi_5\texttt{(the (rt dip))} = \pi_5\texttt{(the (rt'} dip))"
  shows "rt \approx dip rt'"
  proof
    from assms(1-2,4) have dhops': "the (dhops rt'} dip) \leq the (dhops rt dip)"
      by (simp add: proj5\_eq\_dhops)
    with assms(1-3) show "rt \sqsubseteq dip rt'"
      by (rule rt\_fresher2)
  next
    from assms(1-2,4) have dhops: "the (dhops rt dip) \leq the (dhops rt'} dip)"
      by (simp add: proj5\_eq\_dhops)
    with assms(2,1) assms(3) [symmetric] show "rt'} \sqsubseteq dip rt" by (rule rt\_fresher2)
  qed

201
lemma \textit{rt\_fresh\_asE} [elim]:
\begin{itemize}
\item \textbf{assumes} "\textit{rt1} \equiv \textit{dip} \textit{rt2}"
\item \textbf{and} "[ \textit{rt1} \sqsubseteq_{\textit{dip}} \textit{rt2}; \textit{rt2} \sqsubseteq_{\textit{dip}} \textit{rt1} ] \Rightarrow \textit{P} \textit{rt1} \textit{rt2} \textit{dip}"
\item \textbf{shows} "\textit{P} \textit{rt1} \textit{rt2} \textit{dip}"
\end{itemize}
\textbf{using \textit{assms unfolding rt\_fresh\_as_def by simp}}

lemma \textit{rt\_fresh\_asD1} [dest]:
\begin{itemize}
\item \textbf{assumes} "\textit{rt1} \equiv \textit{dip} \textit{rt2}"
\item \textbf{shows} "\textit{rt1} \sqsubseteq_{\textit{dip}} \textit{rt2}"
\end{itemize}
\textbf{using \textit{assms unfolding rt\_fresh\_as_def by simp}}

lemma \textit{rt\_fresh\_asD2} [dest]:
\begin{itemize}
\item \textbf{assumes} "\textit{rt1} \equiv \textit{dip} \textit{rt2}"
\item \textbf{shows} "\textit{rt2} \sqsubseteq_{\textit{dip}} \textit{rt1}"
\end{itemize}
\textbf{using \textit{assms unfolding rt\_fresh\_as_def by simp}}

lemma \textit{rt\_fresh\_as\_sym}: 
\begin{itemize}
\item \textbf{assumes} "\textit{rt1} \equiv \textit{dip} \textit{rt2}"
\item \textbf{shows} "\textit{rt2} \equiv_{\textit{dip}} \textit{rt1}"
\end{itemize}
\textbf{using \textit{assms unfolding rt\_fresh\_as_def by simp}}

lemma \textit{not\_rt\_fresh\_asI1} [intro]:
\begin{itemize}
\item \textbf{assumes} "\neg( \textit{the (rt1 ip)} \sqsubseteq_{\textit{ip}} \textit{the (rt2 ip)})"
\item \textbf{shows} "\neg( \textit{rt1} \equiv_{\textit{ip}} \textit{rt2})"
\end{itemize}
\textbf{proof}
\begin{itemize}
\item \textbf{assume} "\textit{rt1} \equiv_{\textit{ip}} \textit{rt2}"
\item \textbf{hence} "\textit{rt1} \sqsubseteq_{\textit{ip}} \textit{rt2}" ..
\item \textbf{with} "\neg( \textit{rt1} \equiv_{\textit{ip}} \textit{rt2})" \textbf{show False ..}
\end{itemize}
\textbf{qed}

lemma \textit{not\_rt\_fresh\_asI2} [intro]:
\begin{itemize}
\item \textbf{assumes} "\neg( \textit{rt2} \sqsubseteq_{\textit{ip}} \textit{rt1})"
\item \textbf{shows} "\neg( \textit{rt1} \equiv_{\textit{ip}} \textit{rt2})"
\end{itemize}
\textbf{proof}
\begin{itemize}
\item \textbf{assume} "\textit{rt1} \equiv_{\textit{ip}} \textit{rt2}"
\item \textbf{hence} "\textit{rt1} \sqsubseteq_{\textit{ip}} \textit{rt1}" ..
\item \textbf{with} "\neg( \textit{rt2} \equiv_{\textit{ip}} \textit{rt1})" \textbf{show False ..}
\end{itemize}
\textbf{qed}

lemma \textit{not\_single\_rt\_fresher} [elim]:
\begin{itemize}
\item \textbf{assumes} "\neg( \textit{the (rt1 ip)} \sqsubseteq_{\textit{ip}} \textit{the (rt2 ip)})"
\item \textbf{shows} "\neg( \textit{rt1} \sqsubseteq_{\textit{ip}} \textit{rt2})"
\end{itemize}
\textbf{proof}
\begin{itemize}
\item \textbf{assume} "\textit{rt1} \sqsubseteq_{\textit{ip}} \textit{rt2}"
\item \textbf{hence} "\textit{the (rt1 ip)} \sqsubseteq_{\textit{ip}} \textit{the (rt2 ip)}" ..
\item \textbf{with} "\neg( \textit{the (rt1 ip)} \sqsubseteq_{\textit{ip}} \textit{the (rt2 ip)})" \textbf{show False ..}
\end{itemize}
\textbf{qed}

lemmas \textit{not\_single\_rt\_fresh\_asI1} [intro] = \textit{not\_rt\_fresh\_asI1} [OF \textit{not\_single\_rt\_fresher}]
lemmas \textit{not\_single\_rt\_fresh\_asI2} [intro] = \textit{not\_rt\_fresh\_asI2} [OF \textit{not\_single\_rt\_fresher}]

lemma \textit{not\_rt\_fresher\_single} [elim]:
\begin{itemize}
\item \textbf{assumes} "\neg( \textit{the (rt1 ip)} \sqsubseteq_{\textit{ip}} \textit{the (rt2 ip)})"
\item \textbf{shows} "\neg( \textit{rt1} \sqsubseteq_{\textit{ip}} \textit{rt2})"
\end{itemize}
\textbf{proof}
\begin{itemize}
\item \textbf{assume} "\textit{the (rt1 ip)} \sqsubseteq_{\textit{ip}} \textit{the (rt2 ip)}"
\item \textbf{hence} "\textit{the (rt1 ip)} \sqsubseteq_{\textit{ip}} \textit{rt2}" ..
\item \textbf{with} "\neg( \textit{the (rt1 ip)} \sqsubseteq_{\textit{ip}} \textit{rt2})" \textbf{show False ..}
\end{itemize}
\textbf{qed}

lemma \textit{rt\_fresh\_as\_nsqnr}:
\begin{itemize}
\item \textbf{assumes} "\textit{dip} \in \textit{kD(rt1)}"
\item \textbf{and} "\textit{dip} \in \textit{kD(rt2)}"
\end{itemize}
lemma rt_fresher_mapupd [intro!]:
  assumes "dip ∈ kD(rt)"
  and "the (rt dip) ⊑ r"
  shows "rt ⊑ dip rt (dip ↦ r)"
  using assms unfolding rt_fresher_def
  by simp

lemma rt_fresher_map_update_other [intro!]:
  assumes "dip ∈ kD(rt)"
  and "dip ≠ ip"
  shows "rt ⊑ dip rt (ip ↦ r)"
  using assms unfolding rt_fresher_def
  by simp

lemma rt_fresher_update_other [simp]:
  assumes inkD: "dip ∈ kD(rt)"
  and "dip ≠ ip"
  shows "rt ⊑ dip update rt ip r"
  using assms unfolding update_def
  by (clarsimp split: option.split) (fastforce)

theorem rt_fresher_update [simp]:
  assumes "dip ∈ kD(rt)"
  and "the (dhops rt dip) ≥ 1"
  and "update_arg_wf r"
  shows "rt ⊑ dip update rt ip r"
  proof
    (cases "dip = ip")
    assume "dip ≠ ip"
    with ⟨dip ∈ kD(rt)⟩
    have "hops ≥ 1"
      by (metis proj5_eq_dhops projs(4))
    from ⟨update_arg_wf r⟩
    have "the (update rt dip (dsn, unk, val, Suc 0, nhip, pre) dip)" | "the ((update rt dip r) dip)"
      by (rule update_cases_kD, simp_all)
  next
    assume "dip = ip"
    from ⟨dip ∈ kD(rt)⟩ obtain dsn, dsk, fn, hops, nhip, pre
      where rtn [simp]: "the (rt dip) = (dsn, dsk, fn, hops, nhip, pre)"
      by (metis prod_cases6)
    with ⟨the (dhops rt dip) ≥ 1⟩ and ⟨dip ∈ kD(rt)⟩
    have "hops ≥ 1"
      by (metis proj5_eq_dhops projs(4))
    from ⟨dip ∈ kD(rt)⟩
    have "sqn rt dip = dsn" and "the (dhops rt dip) = hops"
      and "the (flag rt dip) = fn"
      by (simp add: sqn_def proj5_eq_dhops [symmetric]
        proj4_eq_flag [symmetric])
    from ⟨update_arg_wf r⟩
    have "((dsn, dsk, fn, hops, nhip, pre) dip)" | "the (update rt dip (dsn, unk, val, Suc 0, nhip, pre) dip)"
      by (rule update_cases_kD, simp_all)
    next
    fix dsn :: sqn and hops nhip pre
    assume "0 < dsn"
    show "((dsn, dsk, fn, hops, nhip, pre) dip)" | "the (update rt dip (dsn, kno, val, hops, nhip, pre) dip)"
proof (rule update_cases_kD [OF _ (dip∈kD(rt))], simp_all add: 0 < dsn)
  assume "dsn_n < dsn"
  thus "([], dip, f_n, hops_n, nhp_n, pre_n)
       ⊑ (dsn, kno, val, hops, nhp, pre ∪ pre_n)"
  unfolding fresher_def by auto
next
  assume "dsn_n = dsn"
  and "hops < hops_n"
  thus "([], dip, f_n, hops_n, nhp_n, pre_n)
       ⊑ (dsn, kno, val, hops, nhp, pre ∪ pre_n)"
  unfolding fresher_def nsqn_r_def by simp
next
  assume "dsn_n = dsn"
  with ⟨0 < dsn⟩
  show "([], dip, inv, hops_n, nhp_n, pre_n)
       ⊑ (dsn, kno, val, hops, nhp, pre ∪ pre_n)"
  unfolding fresher_def by simp
  qed
  qed
  hence "rt ⊑ dip update rt dip r"
  by (rule single_rt_fresher, simp)
  with ⟨dip = ip⟩
  show ?thesis by simp
  qed

theorem rt_fresher_invalidate [simp]:
  assumes "dip∈kD(rt)"
  and indests: "∀ rip∈dom(dests). rip∈vD(rt) ∧ sqn rt rip < the (dests rip)"
  shows "rt ⊑ dip invalidate rt dests"
proof (cases "dip∈dom(dests)"
  assume "dip∈dom(dests)"
  thus ?thesis using ⟨dip∈kD(rt)⟩
  by (rule single_rt_fresher, simp)
next
  assume "dip∈dom(dests)"
  moreover with indests
  have "dip∈vD(rt)"
    and "the (dests dip) = inc (sqn rt dip)"
    by auto
  ultimately show ?thesis
  unfolding invalidate_def sqn_def
  by (rule single_rt_fresher, auto simp: fresher_def)
  qed

lemma nsqn_r.invalidate [simp]:
  assumes "dip∈kD(rt)"
  and "dip∈dom(dests)"
  shows "nsqn r (the (invalidate rt dests dip)) = the (dests dip) - 1"
using assms unfolding invalidate_def by auto

lemma rt_fresh_as_inc_invalidate [simp]:
  assumes "dip∈kD(rt)"
  and "∀ rip∈dom(dests). rip∈vD(rt) ∧ the (dests rip) = inc (sqn rt rip)"
  shows "rt ≈ dip invalidate rt dests"
proof (cases "dip∈dom(dests)"
  assume "dip∈dom(dests)"
  with ⟨dip∈kD(rt)⟩
  have "dip∈kD(invalidate rt dests)"
  by simp
  with ⟨dip∈kD(rt)⟩
  show ?thesis
  by rule (simp_all add: ⟨dip∈dom(dests)⟩)
next
  assume "dip∈dom(dests)"
  with assms(2) have "dip∈vD(rt)"
    and "the (dests dip) = inc (sqn rt dip)" by auto
  from ⟨dip∈vD(rt)⟩
  have "dip∈kD(rt)" by simp
  moreover then have "dip∈kD(invalidate rt dests)" by simp
ultimately show \( \text{thesis} \)

proof (rule nsqn.rt.fresh_asI)
  from \( \langle \text{dip} \in \text{vD(rt)} \rangle \) have "nsqn rt dip = sqn rt dip" by simp
  also have "sqn rt dip = nsqn r (the (invalidate rt dests dip))"
    proof -
      from \( \langle \text{dip} \in \text{kD(rt)} \rangle \) have "nsqn r (the (invalidate rt dests dip)) = \text{the (dests dip)} - 1"
        using \( \langle \text{dip} \in \text{dom(dests)} \rangle \) by (rule nsqn_r_invalidate)
      with \( \langle \text{the (dests dip)} = \text{inc (sqn rt dip)} \rangle \)
        show "sqn rt dip = nsqn r (the (invalidate rt dests dip))" by simp
    qed
  also have "nsqn rt dip = nsqn (invalidate rt dests) dip"
    proof -
      from \( \langle \text{dip} \in \text{kD(invalidate rt dests)} \rangle \) have "nsqn (invalidate rt dests) dip = nsqn rt dip" by (simp add: kD_nsqn)
      finally show "nsqn rt dip = nsqn (invalidate rt dests) dip" .
    qed
  qed

lemmas rt_fresher_inc_invalidate [simp] = rt_fresh_as_inc_invalidate [THEN rt_fresh_asD1]

lemma rt_fresh_as_addpreRT [simp]:
  assumes "ip \in \text{kD(rt)}"
  shows "rt \approx dip the (addpreRT rt ip npre)"
    using assms [THEN kD_Some] by (auto simp: addpreRT_def)

lemmas rt_fresher_addpreRT [simp] = rt_fresh_as_addpreRT [THEN rt_fresh_asD1]

2.5.4 Strictly comparing routing tables

definition rt_strictly_fresher :: "ip \Rightarrow rt \Rightarrow rt \Rightarrow bool"
  where "rt_strictly_fresher \equiv \lambda dip rt1 rt2. (rt1 \sqsubseteq dip rt2) \land \neg(rt1 \approx dip rt2)"

abbreviation
  rt_strictly_fresher_syn :: "rt \Rightarrow ip \Rightarrow rt \Rightarrow bool" ("(\_ / \_ / \_)" [51, 999, 51] 50)
  where "rt1 \sqsubseteq rt2 \equiv rt_strictly_fresher i rt1 rt2"

lemma rt_strictly_fresher_def':
  "rt1 \sqsubseteq rt2 = (\neg(rt1 \sqsubseteq rt2) \land \neg(rt2 \sqsubseteq rt1))"
    unfolding rt_strictly_fresher_def rt_fresh_as_def by auto

lemma rt_strictly_fresherI' [intro]:
  assumes "rt1 \sqsubseteq rt2"
  and "\neg(rt2 \sqsubseteq rt1)"
  shows "rt1 \sqsubseteq rt2"
    using assms unfolding rt_strictly_fresher_def' by simp

lemma rt_strictly_fresherE' [elim]:
  assumes "rt1 \sqsubseteq rt2"
  and "[ rt1 \sqsubseteq rt2; \neg(rt2 \sqsubseteq rt1) ] \Longrightarrow P rt1 rt2 i"
  shows "P rt1 rt2 i"
    using assms unfolding rt_strictly_fresher_def' by simp

lemma rt_strictly_fresherI [intro]:
  assumes "rt1 \sqsubseteq rt2"
  and "\neg(rt1 \approx_i rt2)"
  shows "rt1 \sqsubseteq_i rt2"
    unfolding rt_strictly_fresher_def using assms ..

lemmas rt_strictly_fresher_singleI [elim] = rt_strictly_fresherI [OF single_rt_fresher]

lemma rt_strictly_fresherE [elim]:
  assumes "rt1 \sqsubseteq rt2"
  and "[ rt1 \sqsubseteq rt2; \neg(rt1 \approx_i rt2) ] \Longrightarrow P rt1 rt2 i"
shows "P rt1 rt2 i"
using assms(1) unfolding rt_strictly_fresher_def
by rule (erule(1) assms(2))

lemma rt_strictly_fresher_def':
"rt1 ⊏_⊥ rt2 =
(nsqr_r (the (rt1 i)) < nsqr_r (the (rt2 i))
 ∨ (nsqr_r (the (rt1 i)) = nsqr_r (the (rt2 i)) ∧ π5(the (rt1 i)) > π5(the (rt2 i))))"
unfolding rt_strictly_fresher_def'' rt_fresher_def fresher_def by auto

lemma rt_strictly_fresher_fresherD [dest]:
assumes "rt1 ⊏ dip rt2"
shows "the (rt1 dip) ⊑ the (rt2 dip)"
using assms unfolding rt_strictly_fresher_def rt_fresher_def by auto

lemma rt_strictly_fresher_not_fresh_asD [dest]:
assumes "rt1 ⊏ dip rt2"
shows "¬ rt1 ≈ dip rt2"
using assms unfolding rt_strictly_fresher_def by auto

lemma rt_strictly_fresher_trans [elim, trans]:
assumes "rt1 ⊏ dip rt2"
and "rt2 ⊑ dip rt3"
shows "rt1 ⊏ dip rt3"
proof -
  from ⟨rt1 ⊏ dip rt2⟩ obtain "the (rt1 dip) ⊑ the (rt2 dip)" by auto
  also from ⟨rt2 ⊑ dip rt3⟩ obtain "the (rt2 dip) ⊑ the (rt3 dip)" by auto
  finally have "the (rt1 dip) ⊑ the (rt3 dip)".
  moreover have "¬(rt1 ≈ dip rt3)"
  proof -
    from ⟨rt1 ⊏ dip rt2⟩ obtain "¬(the (rt2 dip) ⊑ the (rt1 dip))" by auto
    also from ⟨rt2 ⊑ dip rt3⟩ obtain "¬(the (rt3 dip) ⊑ the (rt2 dip))" by auto
    finally have "¬(the (rt3 dip) ⊑ the (rt1 dip))".
    thus ?thesis ..
  qed
  ultimately show "rt1 ⊏ dip rt3" ..
qed

lemma rt_strictly_fresher_irefl [simp]: "¬ (rt ⊏ dip rt)"
using assms unfolding rt_strictly_fresher_def by clarsimp

lemma rt_fresher_trans_rt_strictly_fresher [elim, trans]:
assumes "rt1 ⊏ dip rt2"
and "rt2 ⊏ dip rt3"
shows "rt1 ⊏ dip rt3"
proof -
  from ⟨rt1 ⊏ dip rt2⟩ have "rt1 ⊏ dip rt2"
    and "¬(rt2 ⊏ dip rt1)"
    unfolding rt_strictly_fresher_def'' by auto
  from this(1) and ⟨rt2 ⊏ dip rt3⟩ have "rt1 ⊏ dip rt3" ..
  moreover from "¬(rt2 ⊏ dip rt1)" have "¬(rt3 ⊏ dip rt1)"
  proof (rule contrapos_nm)
    assume "rt3 ⊏ dip rt1"
    with ⟨rt2 ⊏ dip rt3⟩ show "rt2 ⊏ dip rt1" ..
    qed
  ultimately show "rt1 ⊏ dip rt3"
    unfolding rt_strictly_fresher_def'' by auto
  qed

lemma rt_fresher_trans_rt_strictly_fresher' [elim, trans]:
assumes "rt1 ⊑ dip rt2"
  and "rt2 ⊑ dip rt3"
shows "rt1 ⊑ dip rt3"
proof -
  from rt2 ⊑ dip rt3: have "rt2 ⊑ dip rt3"
    and "¬ (rt3 ⊑ dip rt2)"
  unfolding rt_strictly_fresher_def'' by auto
  from rt1 ⊑ dip rt2: and this(1) have "rt1 ⊑ dip rt3" ..
moreover from (¬rt3 ⊑ dip rt2) have "¬(rt3 ⊑ dip rt1)"
proof (rule contrapos_nn)
  assume "rt3 ⊑ dip rt1"
  thus "rt3 ⊑ dip rt2" using (rt1 ⊑ dip rt2) ..
  qed
ultimately show "rt1 ⊑ dip rt3"
  unfolding rt_strictly_fresher_def'' by auto
 qed

lemma rt_fresher_imp_nsqn_le:
  assumes "rt1 ⊑ ip rt2"
  and "ip ∈ kD rt1"
  and "ip ∈ kD rt2"
shows "nsqn rt1 ip ≤ nsqn rt2 ip"
using assms(1)
by (auto simp add: rt_fresher_def2 [OF assms(2-3)])

lemma rt_strictly_fresher_ltI [intro]:
  assumes "dip ∈ kD(rt1)"
  and "dip ∈ kD(rt2)"
  and "nsqn rt1 dip < nsqn rt2 dip"
shows "rt1 ⊏ dip rt2"
proof
  from assms show "rt1 ⊑ dip rt2" ..
  next
  show "¬(rt1 ≈ dip rt2)"
    proof
      assume "rt1 ≈ dip rt2"
      hence "rt2 ⊑ dip rt1" ..
      hence "nsqn rt2 dip ≤ nsqn rt1 dip"
      using (dip ∈ kD(rt2)) (dip ∈ kD(rt1))
        by (rule rt_fresher_imp_nsqn_le)
      with ⟨nsqn rt1 dip < nsqn rt2 dip⟩ show "False"
        by simp
      qed
    qed

lemma rt_strictly_fresher_eqI [intro]:
  assumes "i ∈ kD(rt1)"
  and "i ∈ kD(rt2)"
  and "nsqn rt1 i = nsqn rt2 i"
  and "π_5(the (rt2 i)) < π_5(the (rt1 i))"
shows "rt1 ⊏ i rt2"
  using assms unfolding rt_strictly_fresher_def' by (auto simp add: kD_nsqn)

lemma invalidate_rtsf_left [simp]:
"dests dip rt rt'. dests dip = None ⟹ (invalidate rt dests ⊑ dip rt') = (rt ⊑ dip rt')""unfolding invalidate_def rt_strictly_fresher_def'
by (rule iffI) (auto split: option.split_asm)

lemma vD_invalidate_rt_strictly_fresher [simp]:
  assumes "dip ∈ vD(invalidate rt1 dests)"
shows "(invalidate rt1 dests ⊑ dip rt2) = (rt1 ⊑ dip rt2)"
proof (cases "dip ∈ dom(dests)")
assume "dip ∈ dom(dests)"
hence "dip ∉ vD(invalidate rt1 dests)"
  unfolding invalidate_def vD_def
  by clarsimp (metis assms option.simps(3) vD_invalidate_vD_not_dests)
with ⟨dip ∈ vD(invalidate rt1 dests)⟩ show ?thesis by simp

next
assume "dip ∉ dom(dests)"
hence "dests dip = None" by auto
moreover with ⟨dip ∈ vD(invalidate rt1 dests)⟩
  have "dip ∈ vD(rt1)"
  unfolding invalidate_def vD_def
  by clarsimp (metis (hide_lams, no_types) assms vD_Some vD_invalidate_vD_not_dests)
ultimately show ?thesis
  unfolding invalidate_def rt_strictly_fresher_def' by auto

qed

lemma rt_strictly_fresher_update_other [elim!]:
"∀ dip ip rt rt'. [ dip ≠ ip; rt ⊏ dip rt' ] ⇒ update rt ip r ⊏ dip rt'"
unfolding rt_strictly_fresher_def' by clarsimp

lemma addpreRT_strictly_fresher [simp]:
assumes "dip ∈ kD(rt)"
shows "(the (addpreRT rt dip npre) ⊏ ip rt2) = (rt ⊏ ip rt2)"
using assms unfolding rt_strictly_fresher_def' by clarsimp

lemma lt_sqn_imp_update_strictly_fresher:
assumes "dip ∈ vD (rt2 nhip)"
and*: "osn < sqn (rt2 nhip) dip"
and**: "rt ≠ update rt dip (osn, kno, val, hops, nhip, {})"
shows "update rt dip (osn, kno, val, hops, nhip, {}) ⊏ dip rt2 nhip"
unfolding rt_strictly_fresher_def'
proof (rule disjI1)
  from ** have "nsqn (update rt dip (osn, kno, val, hops, nhip, {})) dip = osn"
    by (rule nsqn_update_changed_kno_val)
with ⟨dip ∈ vD(rt2 nhip)⟩
  have "nsqn, (the (update rt dip (osn, kno, val, hops, nhip, {})) dip)) = osn"
    by (simp add: kD_nsqn)
also have "osn < sqn (rt2 nhip) dip" by (rule *)
also have "sqn (rt2 nhip) dip = nsqn, (the (rt2 nhip dip))"
  unfolding nsqn_def using ⟨dip ∈ vD(rt2 nhip)⟩
  by (metis vD_flag_val proj2_eq_sqn proj4_eq_flag vD_iD_gives_kD(1))
finally show "nsqn, (the (update rt dip (osn, kno, val, hops, nhip, {})) dip)) < nsqn, (the (rt2 nhip dip))".

qed

lemma dhops_le_hops_imp_update_strictly_fresher:
assumes "dip ∈ vD (rt2 nhip)"
and sqn: "sqn (rt2 nhip) dip = osn"
and hop: "the (dhops (rt2 nhip) dip) ≤ hops"
and**: "rt ≠ update rt dip (osn, kno, val, Suc hops, nhip, {})"
shows "update rt dip (osn, kno, val, Suc hops, nhip, {}) ⊏ dip rt2 nhip"
unfolding rt_strictly_fresher_def'
proof (rule disjI2, rule conjI)
  from ** have "nsqn, (the (update rt dip (osn, kno, val, Suc hops, nhip, {})) dip)) dip = osn"
    by (rule nsqn_update_changed_kno_val)
with ⟨dip ∈ vD(rt2 nhip)⟩
  have "nsqn, (the (update rt dip (osn, kno, val, Suc hops, nhip, {})) dip)) = osn"
    by (simp add: kD_nsqn)
also have "osn = sqn (rt2 nhip) dip" by (rule sqn [symmetric])
also have "sqn (rt2 nhip) dip = nsqn, (the (rt2 nhip dip))"
  unfolding nsqn_def using ⟨dip ∈ vD(rt2 nhip)⟩
  by (metis vD_flag_val proj2_eq_sqn proj4_eq_flag vD_iD_gives_kD(1))
finally show "nsqn, (the (update rt dip (osn, kno, val, Suc hops, nhip, {})) dip)) = nsqn, (the (rt2 nhip dip))".

next
have 

\((dhops \ (rt2 \ nhip) \ dip) \leq \ hops\) 
by (rule hop)

also have 

\(hops < hops + 1\) 
by simp

also have 

\(hops + 1 = (dhops \ (update \ rt \ dip \ (osn, \ kno, \ val, \ Suc \ hops, \ nhip, \ {})) \ dip)\)

using ** by simp

finally have 

\(the \ (dhops \ (rt2 \ nhip) \ dip) < the \ (dhops \ (update \ rt \ dip \ (osn, \ kno, \ val, \ Suc \ hops, \ nhip, \ {})) \ dip)\).

thus 

\(\pi_5 \ (the \ (rt2 \ nhip \ dip)) < \pi_5 \ (the \ (update \ rt \ dip \ (osn, \ kno, \ val, \ Suc \ hops, \ nhip, \ {})) \ dip)\)

using \langle \ dip \in vD(rt2 nhip) \rangle by (simp add: proj5_eq_dhops)

qed

lemma nsqn_invalidate:

assumes

\(dip \in kD(rt)\)

and

\(\forall \ ip \in dom(dests). \ ip \in vD(rt) \wedge the \ (dests \ ip) = inc \ (sqn \ rt \ ip)\)

shows 

\(nsqn \ (invalidate \ rt \ dests) \ dip = nsqn \ rt \ dip\)

proof -

from \langle \ dip \in kD(rt) \rangle have "dip \in kD(invalidate rt dests)" by simp

from assms have "rt \approx dip \ invalidate \ rt \ dests"

by (rule rt_fresh_as_inc_invalidate)

with \langle \ dip \in kD(rt) \rangle \langle \ dip \in kD(invalidate \ rt \ dests) \rangle show ?thesis

by (simp add: kD_nsqn del: invalidate_kD_inv)

(erule(2) rt_fresh_as_nsqnr)

qed

end

2.6 Invariant proofs on individual processes

theory B_Seq_Invariants

imports "../../../AWN/Invariants" B_Aodv B_Aodv_Data B_Aodv_Predicates B_Fresher

begin

The proposition numbers are taken from the December 2013 version of the Fehnker et al technical report.

Proposition 7.2

lemma sequence_number_increases:

"paodv \ i\|=\ A\ onll\ \Gamma_{AODV} \ (\lambda((\xi, \ _), \ _), \ (\xi', \ _)). \ sn \ \xi \leq \ sn \ \xi')"

by inv_cterms

lemma sequence_number_one_or_bigger:

"paodv \ i\|=\ onl\ \Gamma_{AODV} \ (\lambda(\xi, \ _). \ 1 \leq \ sn \ \xi)"

by (rule onll_step_to_invariantI [OF sequence_number_increases])

(auto simp: \sigma_{AODV-def})

We can get rid of the onl/onll if desired...

lemma sequence_number_increases':

"paodv \ i\|=\ (\lambda((\xi, \ _), \ _), \ (\xi', \ _)). \ sn \ \xi \leq \ sn \ \xi')"

by (rule step_invariant_weakenE [OF sequence_number_increases]) (auto dest!: onllD)

lemma sequence_number_one_or_bigger':

"paodv \ i\|=\ (\lambda(\xi, \ _). \ 1 \leq \ sn \ \xi)"

by (rule invariant_weakenE [OF sequence_number_one_or_bigger]) auto

lemma sip_in_kD:

"paodv \ i\|=\ onl\ \Gamma_{AODV} \ (\lambda(\xi, \ 1). \ 1 \in \ {\{PAodv-:7\} \cup\ {PAodv-:5\} \cup\ {PRrep-:0..PRrep-:4}\}

\cup \ {PRreq-:0..PRreq-:3}) \rightarrow \ sip \ \xi \in kD \ (rt \ \xi))"

by inv_cterms

lemma addpreRT_partly_welldefined:

"paodv \ i\|=\ onl\ \Gamma_{AODV} \ (\lambda(\xi, \ 1). \ 1 \in \ {PRreq-:16..PRreq-:18} \cup \ {PRrep-:1..PRrep-:5} \rightarrow \ dip \ \xi \in kD \ (rt \ \xi))"
Proposition 7.38
lemma includes_nhip:
"paodv i \models onl \Gamma_{AODV} (\lambda(\xi, l). \forall dip \in kD(rt \xi). the (nhop (rt \xi) dip) \in kD(rt \xi))"

proof -
\{ fix ip and \xi' :: state 
assume "\forall dip \in kD(rt \xi). the (nhop (rt \xi) dip) \in kD(rt \xi)"
and "\xi' = \xi[rt := update (rt \xi) ip (0, unk, val, Suc 0, ip, {})]"
hence "\forall dip \in kD(rt \xi). the (nhop (update (rt \xi) ip (0, unk, val, Suc 0, ip, {})) dip) \in kD(rt \xi)"
by clarsimp (metis nhop_update_unk_val update_another)
\}

note onehop = this
\{ fix sip sn hops and \xi' :: state
assume "\forall dip \in kD(rt \xi). the (nhop (rt \xi) dip) \in kD(rt \xi)"
and "\xi' = \xi[rt := update (rt \xi) ip (sn, kno, val, Suc hops, sip, {})]"
and "sip \in kD(rt \xi)"
hence "(the (nhop (update (rt \xi) ip (sn, kno, val, Suc hops, sip, {})) ip) = ip 
\lor the (nhop (update (rt \xi) ip (sn, kno, val, Suc hops, sip, {})) ip) \in kD(rt \xi))"
\}

note nhipis_sip = this
show ?thesis
by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf sip_in_kD]
onl_invariant_sterms [OF aodv_wf addpreRT_partly_welldefined]
solve: onehop nhipis_sip)
qed

Proposition 7.22: needed in Proposition 7.4
lemma addpreRT_welldefined:
"paodv i \models onl \Gamma_{AODV} (\lambda(\xi, l). (l \in \{PRreq-:16..PRreq-:18\} \longrightarrow dip \xi \in kD(rt \xi)) 
\land (l = PRreq-:17 \longrightarrow oip \xi \in kD(rt \xi)) 
\land (l = PRrep-:4 \longrightarrow dip \xi \in kD(rt \xi)) 
\land (l = PRrep-:5 \longrightarrow (the (nhop (rt \xi) dip) \in kD(rt \xi)))"
(is "-\models onl \Gamma_{AODV} ?P")

unfolding invariant_def

proof
\{ fix s 
assume "s \in reachable (paodv i) TT"
then obtain \xi p where "s = (\xi, p)"
and "(\xi, p) \in reachable (paodv i) TT"
by (metis PairE)
have "onl \Gamma_{AODV} ?P (\xi, p)"
proof (rule onlI)
\{ fix l 
assume "l \in labels \Gamma_{AODV} p"
with (\xi, p) \in reachable (paodv i) TT
have I1: "l \in \{PRreq-:16..PRreq-:18\} \longrightarrow dip \xi \in kD(rt \xi)
\land (l = PRreq-:17 \longrightarrow oip \xi \in kD(rt \xi)) 
\land (l = PRrep-:4 \longrightarrow dip \xi \in kD(rt \xi)) 
\land (l = PRrep-:5 \longrightarrow (the (nhop (rt \xi) dip) \in kD(rt \xi)))"
by (auto dest!: invariantD [OF addpreRT_partly_welldefined])
moreover from (\xi, p) \in reachable (paodv i) TT (l \in labels \Gamma_{AODV} p) and I3
have "l = PRrep-:5 \longrightarrow (the (nhop (rt \xi) dip) \in kD(rt \xi))"
by (auto dest!: invariantD [OF includes_nhip])
ultimately show "?P (\xi, l)"
by simp
\}
\}
with s = (\xi, p) show "onl \Gamma_{AODV} ?P s"
by simp
qed
Proposition 7.4

lemma known_destinations_increase:
"paodv i |=A onll Γ_AODV (λ(ξ, _, (ξ', _)). kD (rt ξ) ⊆ kD (rt ξ'))" by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf addpreRT_welldefined] simp add: subset_insertI)

Proposition 7.5

lemma rreqs_increase:
"paodv i |=A onll Γ_AODV (λ((ξ, _), _, (ξ', _)). rreqs ξ ⊆ rreqs ξ')" by (inv_cterms simp add: subset_insertI)

lemma dests_bigger_than_sqn:
"paodv i |= onll Γ_AODV (λ(ξ, l). l ∈ {PAodv-:15..PAodv-:19} ∪ {PPkt-:7..PPkt-:11} ∪ {PRreq-:9..PRreq-:13} ∪ {PRreq-:21..PRreq-:25} ∪ {PRrep-:9..PRrep-:13} ∪ {PRerr-:1..PRerr-:5} −→ (∀ ip ∈ dom(dests ξ). ip ∈ kD(rt ξ) ∧ sqn (rt ξ) ip ≤ the (dests ξ ip)))" proof -
  have sqninv:
    "∀ dests rt rsn ip. (∀ ip ∈ dom(dests). ip ∈ kD(rt) ∧ sqn rt ip ≤ (dests ip); dests ip = Some rsn) −→ sqn (invalidate rt dests) ip ≤ rsn"
    by (rule sqn_invalidate_in_dests [THEN eq_imp_le], assumption) auto
  have indests:
    "∀ dests rt rsn ip. (∀ ip ∈ dom(dests). ip ∈ kD(rt) ∧ sqn rt ip ≤ (dests ip); dests ip = Some rsn) −→ ip ∈ kD(rt) ∧ sqn rt ip ≤ rsn"
    by (metis domI option.sel)
  show ?thesis
    by inv_cterms
      (clarsimp split: split_if_asm option.split_asm elim!: sqninv indests)+
qed

Proposition 7.6

lemma sqns_increase:
"paodv i |= onll Γ_AODV (λ((ξ, _), (ξ', _)). ∀ ip. sqn (rt ξ) ip ≤ sqn (rt ξ') ip)"
proof -
  { fix ξ :: state
    assume *: "∀ ip ∈ dom(dests ξ). ip ∈ kD (rt ξ) ∧ sqn (rt ξ) ip ≤ (dests ξ ip)" have "∀ ip. sqn (rt ξ) ip ≤ sqn (invalidate (rt ξ) (dests ξ ip))" proof
      fix ip
      from * have "ip ∈ dom(dests ξ) ∨ sqn (rt ξ) ip ≤ (dests ξ ip)" by simp thus "sqn (rt ξ) ip ≤ sqn (invalidate (rt ξ) (dests ξ ip))" by (metis domI invalidate_sqn option.sel)
    qed
  } note solve_invalidate = this
  show ?thesis
qed

Proposition 7.7

lemma ip_constant:
"paodv i |= onll Γ_AODV (λ(ξ, _). ip ξ = i)"
by (inv_cterms simp add: σ_AODV_def)

Proposition 7.8

lemma sender_ip_valid'::
"paodv i \models A \onl \Gamma_{ADOV} (\lambda((\xi, _), a, _). \text{anycast (}\lambda m. \not\text{Pkt } m \rightarrow \msg_{sender} m = \ip (\xi) a)"
by \text{inv_cterms}

\text{lemma sender_ip_valid:}
"paodv i \models A \onl \Gamma_{ADOV} (\lambda((\xi, _), a, _). \text{anycast (}\lambda m. \not\text{Pkt } m \rightarrow \msg_{sender} m = \i) a)"
by (rule step_invariant_weaken_with_invariantE [OF ip_constant sender_ip_valid'])
(auto dest: \onlD onlID)

\text{lemma received_msg_inv:}
"paodv i \models (\text{recvmsg } P \rightarrow) \onl \Gamma_{ADOV} (\lambda(\xi, 1). 1 \in \{PAodv:-1\} \rightarrow P (\msg (\xi)))"
by \text{inv_cterms}

\text{Proposition 7.9}

\text{lemma sip_not_ip':}
"paodv i \models (\text{recvmsg (}\lambda m. \not\text{Pkt } m \rightarrow \msg_{sender} m \neq \i) \rightarrow) \onl \Gamma_{ADOV} (\lambda(\xi, _). \text{sip } \xi \neq \ip (\xi))"
by (\text{inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]
  onl_invariant_sterms [OF aodv_wf ip_constant [THEN invariant_restrict_inD]]
  simp add: clear_locals_sip_not_ip') clarsimp+

\text{lemma sip_not_ip:}
"paodv i \models (\text{recvmsg (}\lambda m. \not\text{Pkt } m \rightarrow \msg_{sender} m \neq \i) \rightarrow) \onl \Gamma_{ADOV} (\lambda(\xi, _). \text{sip } \xi \neq \i)"
by (\text{inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]
  onl_invariant_sterms [OF aodv_wf ip_constant [THEN invariant_restrict_inD]]
  simp add: clear_locals_sip_not_ip') clarsimp+

\text{Neither sip_not_ip' nor sip_not_ip is needed to show loop freedom.}

\text{Proposition 7.10}

\text{lemma hop_count_positive:}
"paodv i \models \onl \Gamma_{ADOV} (\lambda(\xi, _). \\forall \ip \in kD (\rt (\rt (\ip) \ip) \geq 1)"
by (\text{inv_cterms inv add: onl_invariant_sterms [OF aodv_wf addpreRT_welldefined]} auto

\text{lemma rreq_dip_in_vD_dip_eq_ip:}
"paodv i \models \onl \Gamma_{ADOV} (\lambda(\xi, 1). (1 \in \{PRreq-:16..PRreq-:18\} \rightarrow \dip (\xi) \in \vd (\rt (\rt) ))
  \land (1 \in \{PRreq-:5, PRreq-:6\} \rightarrow \dip (\xi) = \ip (\xi))
  \land (1 \in \{PRreq-:15..PRreq-:18\} \rightarrow \dip (\xi) \neq \ip (\xi)))"

\text{proof (inv_cterms, elim conjE)}
\text{fix } l \xi pp p'
\text{assume "}(\xi, pp) \in \text{reachable (paodv i) TT"}
  and "\{PRreq-:17\}[\lambda \xi. \xi((\rt := \text{the (addpreRT (\rt (\xi) (\oip (\xi)) (the (nhop (\rt (\xi) (\dip (\xi)))))))))]] p'
  \in \text{sterms } \Gamma_{ADOV} pp"
  and "1 = PRreq-:17"
  and "dip (\xi) \in \vd (\rt (\rt) )"
\text{from this(1-3) have "oip (\xi) \in kD (\rt (\rt) )"}
by (\text{auto dest: onl_invariant_sterms [OF aodv_wf addpreRT_welldefined, where \( l=\text{PRreq-:17} \)])
\text{with } dip (\xi) \in \vd (\rt (\rt) )
\text{show "dip (\xi) \in \vd (\text{the (addpreRT (\rt (\xi) (\oip (\xi)) (the (nhop (\rt (\xi) (\dip (\xi)))))))")" by simp}
\text{qed}

\text{lemma rrep_dip_in_vD:}
"paodv i \models \onl \Gamma_{ADOV} (\lambda(\xi, 1). (1 \in \{PRrep-:4..PRrep-:6\} \rightarrow \dip (\xi) \in \vd (\rt (\rt) )))"
\text{proof inv_cterms}
\text{fix } l \xi pp p'
\text{assume "}(\xi, pp) \in \text{reachable (paodv i) TT"}
  and "\{PRrep-:5\}[\lambda \xi. \xi((\rt := \text{the (addpreRT (\rt (\xi) (\oip (\xi)) (the (nhop (\rt (\xi) (\dip (\xi)))))))))]] p'
  \in \text{sterms } \Gamma_{ADOV} pp"
  and "1 = PRrep-:5"
  and "dip (\xi) \in \vd (\rt (\rt) )"
\text{from this(1-3) have "the (nhop (\rt (\xi) (\dip (\xi))) \in kD (\rt (\rt) )"}
by (\text{auto dest: onl_invariant_sterms [OF aodv_wf addpreRT_welldefined, where \( l=\text{PRrep-:5} \)])
\text{with } dip (\xi) \in \vd (\rt (\rt) )
\text{show "dip (\xi) \in \vd (\text{the (addpreRT (\rt (\xi) (\oip (\xi)) (the (nhop (\rt (\xi) (\dip (\xi))))))))")" by simp}
Proposition 7.11

lemma anycast_msg_zhops:
  "∀rreqid dip dsn desk oip osn sip.
poaov i ⊩ₐ onll Γ_{AODV} (λ(_, a, _). anycast msg_zhops a)"

proof (inv_cterms inv add:
  onl_invariant_sterms [OF aodv_wf rreq_dip_in_vD dip_eq_ip]
onl_invariant_sterms [OF aodv_wf rrep_dip_in_vD]
onl_invariant_sterms [OF aodv_wf hop_count_positive],
  elim conjE)
fix l ξ a pp p' p''
assume "((ξ, pp) ∈ reachable (poaov i) TT" and "(PRrep-:18)unicast(λξ. the (nhop (rt ξ) (oip ξ)),
                                           λξ. Rrep (the (dhops (rt ξ) (dip ξ))) (dip ξ) (sqn (rt ξ) (dip ξ)) (oip ξ) (ip ξ)).
p' > pp' ∈ sterms Γ_{AODV} pp"
and "1 = PRreq-:18" and "a = unicast (the (nhop (rt ξ) (oip ξ)))
                      (Rrep (the (dhops (rt ξ) (dip ξ))) (dip ξ) (sqn (rt ξ) (dip ξ)) (oip ξ) (ip ξ))" and *: "∀ip∈kD (rt ξ). Suc 0 ≤ the (dhops (rt ξ) ip)"
and "dip ξ ∈ vD (rt ξ)" from dip ξ ∈ vD (rt ξ) have "dip ξ ∈ kD (rt ξ)"
by (rule vD_iD_gives_kD(1)) with * have "Suc 0 ≤ the (dhops (rt ξ) (dip ξ))" ..
thus "0 < the (dhops (rt ξ) (dip ξ))" by simp
next
fix l ξ a pp p' p''
assume "((ξ, pp) ∈ reachable (poaov i) TT" and "(PRrep-:6)unicast(λξ. the (nhop (rt ξ) (oip ξ)),
                                           λξ. Rrep (the (dhops (rt ξ) (dip ξ))) (dip ξ) (sqn (rt ξ) (dip ξ)) (oip ξ) (ip ξ)).
p' > pp' ∈ sterms Γ_{AODV} pp"
and "1 = PRrep-:6" and "a = unicast (the (nhop (rt ξ) (oip ξ)))
                      (Rrep (the (dhops (rt ξ) (dip ξ))) (dip ξ) (sqn (rt ξ) (dip ξ)) (oip ξ) (ip ξ))" and *: "∀ip∈kD (rt ξ). Suc 0 ≤ the (dhops (rt ξ) ip)"
and "dip ξ ∈ vD (rt ξ)" from dip ξ ∈ vD (rt ξ) have "dip ξ ∈ kD (rt ξ)"
by (rule vD_iD_gives_kD(1)) with * have "Suc 0 ≤ the (dhops (rt ξ) (dip ξ))" ..
thus "the (dhops (rt ξ) (dip ξ)) = 0 → dip ξ = ip ξ" by auto
qed

lemma hop_count_zero_oip_dip_sip:
  "poaov i ⊩ (recvmsg msg_zhops →) onll Γ_{AODV} (λξ, l).
   (l∈{Paoov-:4, Paoov-:5} ∪ {PRreq-:n|n. True} → 
    (hops ξ = 0 → oip ξ = sip ξ))
   ∧ 
   ((l∈{Paoov-:6, Paoov-:7} ∪ {PRrep-:n|n. True} → 
    (hops ξ = 0 → dip ξ = sip ξ))))"
by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]) auto

lemma osn_rreq:
  "poaov i ⊩ (recvmsg rreq_rrep_sn →) onll Γ_{AODV} (λξ, l).
   l ∈ {Paoov-:4, Paoov-:5} ∪ {PRreq-:n|n. True} → 1 ≤ osn ξ)"
by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]) clarsimp

lemma osn_rreq':
  "poaov i ⊩ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) →) onll Γ_{AODV} (λξ, l).
   l ∈ {Paoov-:4, Paoov-:5} ∪ {PRreq-:n|n. True} → 1 ≤ osn ξ)"
proof (rule invariant_weakenE [OF osn_rreq])
fix a
assume "recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) a"
thus "recvmsg rreq_rrep_sn a"
by (cases a) simp_all
qed

lemma dsn_rrep:
"paodv i |= (recvmsg rreq_rrep_sn \rightarrow) onl Γ_{AODV} (λ(ξ, 1).
1 ∈ (PAdv⁻:6, PAdv⁻:7) \cup \{PRrep⁻:n|n. True} \rightarrow 1 ≤ dsn ξ)"
by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]) clarsimp

lemma dsn_rrep':
"paodv i ||= (recvmsg (λm. rreq_rrep_sn m \& msg_zhops m) \rightarrow) onl Γ_{AODV} (λ(ξ, l).
l ∈ \{PAodv⁻:6, PAodv⁻:7\} \cup \{PRrep⁻:n|n. True\} \rightarrow 1 ≤ dsn ξ)"
proof (rule invariant_weakenE [OF dsn_rrep])
fix a
assume "recvmsg (λm. rreq_rrep_sn m \& msg_zhops m) a"
thus "recvmsg rreq_rrep_sn a" by (cases a) simp_all
qed

lemma hop_count_zero_oip_dip_sip':
"paodv i ||= (recvmsg (λm. rreq_rrep_sn m \& msg_zhops m) \rightarrow) onl Γ_{AODV} (λ(ξ, l).
l ∈ \{PAodv⁻:4..PAodv⁻:5\} \cup \{PRreq⁻:n|n. True\} \rightarrow (hops ξ = 0 \rightarrow oip ξ = sip ξ) ∧ (hops ξ = 0 \rightarrow dip ξ = sip ξ))"
proof (rule invariant_weakenE [OF hop_count_zero_oip_dip_sip])
fix a
assume "recvmsg (λm. rreq_rrep_sn m \& msg_zhops m) a"
thus "recvmsg msg_zhops a" by (cases a) simp_all
qed

Proposition 7.12

lemma zero_seq_unk_hops_one':
"paodv i ||= (recvmsg (λm. rreq_rrep_sn m \& msg_zhops m) \rightarrow) onl Γ_{AODV} (λ(ξ, _).
∀ dip ∈ kD(rt ξ). (sqn (rt ξ) dip = 0 \rightarrow sqnf (rt ξ) dip = unk) ∧ (sqnf (rt ξ) dip = unk \rightarrow the (dhops (rt ξ) dip) = 1) ∧ (the (dhops (rt ξ) dip) = 1 \rightarrow the (nhop (rt ξ) dip) = dip))"
proof -
{ fix dip and ξ :: state and P
  assume "sqn (invalidate (rt ξ) (dests ξ)) dip = 0"
  and all: "∀ip. sqn (rt ξ) ip ≤ sqn (invalidate (rt ξ) (dests ξ)) ip"
  and *: "sqn (rt ξ) dip = 0 \implies P ξ dip"
  have "P ξ dip"
  proof -
    from all have "sqn (rt ξ) dip ≤ sqn (invalidate (rt ξ) (dests ξ)) dip" ..
    with (sqn (invalidate (rt ξ) (dests ξ)) dip = 0) have "sqn (rt ξ) dip = 0" by simp
    thus "P ξ dip" by (rule *)
  qed
} note sqn_invalidate_zero [elim!] = this

{ fix dsn hops :: nat and sip oip rt and ip dip :: ip
  assume "∀ dip ∈ kD(rt).
    (sqn rt dip = 0 \rightarrow π_3(the (rt dip)) = unk) ∧
    (π_3(the (rt dip)) = unk \rightarrow the (dhops rt dip) = Suc 0) ∧
    (the (dhops rt dip) = Suc 0 \rightarrow the (nhop rt dip) = dip))"
  and "hops = 0 \rightarrow sip = dip"
  and "Suc 0 ≤ dsn"
  and "ip ≠ dip \rightarrow ip ∈ kD(rt)"
  hence "the (dhops (update rt dip (dsn, kno, val, Suc hops, sip, {})) ip) = Suc 0 \rightarrow
    the (nhop (update rt dip (dsn, kno, val, Suc hops, sip, {})) ip) = ip"
  by (rule update_cases, auto simp add: sqn_def dest!: bspec)
} note prreq_ok1 [simp] = this
\{ fix ip dsn hops sip oip rt dip \\
  assume \("\forall dip \in kD(rt) . \\
  (sqn rt dip = 0 \rightarrow \pi_3(\text{the } (rt dip)) = unk) \land \\
  (\pi_3(\text{the } (rt dip)) = unk \rightarrow \text{the } (dhops rt dip) = \text{Suc } 0) \land \\
  (\text{the } (dhops rt dip) = \text{Suc } 0 \rightarrow \text{the } (nhop rt dip) = dip)"
  \\
  and \("Suc 0 \leq dsn"
  \\
  and \("ip \neq dip \rightarrow ip \in kD(rt)"
  \\
  hence \("\pi_3(\text{the } (update rt dip (dsn, kno, val, Suc hops, sip, {})) ip)) = unk \rightarrow \\
  \text{the } (dhops (update rt dip (dsn, kno, val, Suc hops, sip, {})) ip) = \text{Suc } 0"
  \\
  by - (rule update_cases, auto simp add: sqn_def sqnf_def dest!: bspec)
\} note prreq_ok2 [simp] = this

\{ fix ip dsn hops sip oip rt dip \\
  assume \("\forall dip \in kD(rt) . \\
  (sqn rt dip = 0 \rightarrow \pi_3(\text{the } (rt dip)) = unk) \land \\
  (\pi_3(\text{the } (rt dip)) = unk \rightarrow \text{the } (dhops rt dip) = \text{Suc } 0) \land \\
  (\text{the } (dhops rt dip) = \text{Suc } 0 \rightarrow \text{the } (nhop rt dip) = dip)"
  \\
  and \("Suc 0 \leq dsn"
  \\
  and \("ip \neq dip \rightarrow ip \in kD(rt)"
  \\
  hence \("\pi_3(\text{the } (update rt dip (dsn, kno, val, Suc hops, sip, {})) ip)) = unk \rightarrow \\
  \pi_3(\text{the } (update rt dip (dsn, kno, val, Suc hops, sip, {}) ip)) = unk"
  \\
  by - (rule update_cases, auto simp add: sqn_def sqnf_def dest!: bspec)
\} note prreq_ok3 [simp] = this

\{ fix rt sip \\
  assume \("\forall dip \in kD rt . \\
  (sqn rt dip = 0 \rightarrow \pi_3(\text{the } (rt dip)) = unk) \land \\
  (\pi_3(\text{the } (rt dip)) = unk \rightarrow \text{the } (dhops rt dip) = \text{Suc } 0) \land \\
  (\text{the } (dhops rt dip) = \text{Suc } 0 \rightarrow \text{the } (nhop rt dip) = dip)"
  \\
  hence \("\forall dip \in kD rt . \\
  (sqn (update rt sip (0, unk, val, Suc 0, sip, {})) dip = 0 \rightarrow \\
  \pi_3(\text{the } (update rt sip (0, unk, val, Suc 0, sip, {}) dip)) = unk) \land \\
  (\pi_3(\text{the } (update rt sip (0, unk, val, Suc 0, sip, {})) dip)) = unk \rightarrow \\
  \text{the } (dhops (update rt sip (0, unk, val, Suc 0, sip, {})) dip) = \text{Suc } 0) \land \\
  (\text{the } (dhops (update rt sip (0, unk, val, Suc 0, sip, {})) dip) = \text{Suc } 0 \rightarrow \\
  \text{the } (nhop (update rt sip (0, unk, val, Suc 0, sip, {})) dip) = dip)"
  \\
  by - (rule update_cases) simp_all
\} note prreq_ok4 [simp] = this

have prreq_ok5 [simp]: "\(\forall dip . \\
  \pi_3(\text{the } (update rt sip (0, unk, val, Suc 0, sip, {}) sip)) = unk \rightarrow \\
  \text{the } (dhops (update rt sip (0, unk, val, Suc 0, sip, {}) sip)) = \text{Suc } 0"
  \\
  by (rule update_cases) simp_all

have prreq_ok6 [simp]: "\(\forall dip . \\
  \text{sqn } (\text{update rt sip (0, unk, val, Suc 0, sip, {})) sip = 0 \rightarrow \\
  \pi_3(\text{the } (update rt sip (0, unk, val, Suc 0, sip, {}) sip)) = unk"
  \\
  by (rule update_cases) simp_all

show \(\text{thesis}
  \\
  by (inv_cterms inv add: onl_invariant_sterms_TT [OF aodv_wf addpreRT_welldefined] \\
  onl_invariant_sterms [OF aodv_wf hop_count_zero_oip_dip_sip'] \\
  seq_step_invariant_sterms_TT [OF sqns_increase aodv_wf aodv_trans] \\
  onl_invariant_sterms [OF aodv_wf osn_rreq'] \\
  onl_invariant_sterms [OF aodv_wf dsn_rrep']) clarsimp+
qed

lemma zero_seq_unk_hops_one:
  \("paodv i \models (recvmsg (\lambda m. rreq_rrep_sn m \land msg_zhops m) \rightarrow onl \Gamma_{AODV} (\lambda (\xi, \_). \\
  \forall dip \in kD rt . (sqn (rt \xi) dip = 0 \rightarrow (sqnf (rt \xi) dip = unk \\
  \land \text{the } (dhops (rt \xi) dip) = 1 \\
  \land \text{the } (nhop (rt \xi) dip) = dip))))"
  \\
  by (rule invariant_weakeneE [OF zero_seq_unk_hops_one']) auto
\}
lemma kD_unk_or_atleast_one:
"paodv i |= (recvmsg rreq rrep_sn \rightarrow) onl \Gamma_{AODV} (\lambda(\xi, 1).
\forall dip \in kD(rt \xi). \pi_3((rt \xi dip)) = unk \lor 1 \leq \pi_2((rt \xi dip)))"

proof -
{ fix sip rt dsn1 dsn2 dsk1 dsk2 flag1 flag2 hops1 hops2 nhip1 nhip2 pre1 pre2
assume "dsk1 = unk \lor Suc 0 \leq dsn2"
hence "\pi_3((update rt sip (dsn1, dsk1, flag1, hops1, nhip1, pre1) sip)) = unk
\lor Suc 0 \leq sqn ((update rt sip (dsn2, dsk2, flag2, hops2, nhip2, pre2)) sip)"
unfolding update_def by (cases "dsk1 = unk") (clarsimp split: option.split)+
} note fromsip [simp] = this

{ fix dip sip rt dsn1 dsn2 dsk1 dsk2 flag1 flag2 hops1 hops2 nhip1 nhip2 pre1 pre2
assume allkd: "\forall dip \in kD(rt). \pi_3((rt dip)) = unk \lor Suc 0 \leq sqn rt dip"
and **: "dsk1 = unk \lor Suc 0 \leq dsn2"
have "\forall dip \in kD(rt). \pi_3((update rt sip (dsn1, dsk1, flag1, hops1, nhip1, pre1) dip)) = unk
\lor Suc 0 \leq sqn ((update rt sip (dsn2, dsk2, flag2, hops2, nhip2, pre2)) dip)"
is "\forall dip \in kD(rt). ?prop dip"
proof
fix dip
assume "dip \in kD(rt)"
thus "?prop dip"
proof (cases "dip = sip")
assume "dip = sip"
with ** show ?thesis
by simp
next
assume "dip \neq sip"
with \langle dip \in kD(rt) \rangle allkd show ?thesis
by simp
qed
qed
}

note solve_update [simp] = this

{ fix dip rt dests
assume *: "\forall ip \in dom(dests). ip \in kD(rt) \land sqn rt ip \leq the (dests ip)"
and **: "\forall ip \in kD(rt). \pi_3((rt ip)) = unk \lor Suc 0 \leq sqn rt ip"
have "\forall dip \in kD(rt). \pi_3((rt dip)) = unk \lor Suc 0 \leq sqn (invalidate rt dests dip)"
proof
fix dip
assume "dip \in kD(rt)"
with ** have "\pi_3((rt dip)) = unk \lor Suc 0 \leq sqn rt dip" ..
thus "\pi_3 ((rt dip)) = unk \lor Suc 0 \leq sqn (invalidate rt dests dip)" proof
assume "\pi_3 ((rt dip)) = unk" thus ?thesis ..
next
assume "Suc 0 \leq sqn rt dip"
have "Suc 0 \leq sqn (invalidate rt dests dip)"
proof (cases "dip \in dom(dests)"
assume "dip \in dom(dests)"
with * have "sqn rt dip \leq the (dests dip)" by simp
with Suc 0 \leq sqn rt dip have "Suc 0 \leq the (dests dip)" by simp
with \langle dip \in dom(dests) \rangle \langle dip \in kD(rt) \rangle [THEN kD_Some] show ?thesis
unfolding invalidate_def sqn_def by auto
next
assume "dip \in dom(dests)"
with Suc 0 \leq sqn rt dip \langle dip \in kD(rt) \rangle [THEN kD_Some] show ?thesis
unfolding invalidate_def sqn_def by auto
next
thus ?thesis by (rule disjI2)
qed
qed
}

note solve_invalidate [simp] = this

show ?thesis
Proposition 7.13

**Lemma rreq_rrep_sn_any_step_invariant:**

\[ \text{paodv } i \vdash_A (\text{recvmsg } \text{rreq_rrep_sn} \rightarrow \text{onll } \Gamma_{\text{AODV}} (\lambda(\_, a, \_). \text{anycast } \text{rreq_rrep_sn a}) \]

**Proof:**

- **have sqn_f_kno:** 
  \[ \text{paodv } i \vdash \text{onl } \Gamma_{\text{AODV}} (\lambda(\xi, 1).
  1 \in \{\text{PRreq-:16..PRreq-:18}\} \rightarrow \text{sqnf } (\text{rt } \xi) (\text{dip } \xi) = \text{kno})" \]
  by (inv_cterms inv add: onl_invariant_sterms_TT [OF aodv_wf addpreRT_welldefined])

- **have rrep_sqn_greater_dsn:** 
  \[ \text{paodv } i \vdash (\text{recvmsg } \text{rreq_rrep_sn} \rightarrow \text{onl } \Gamma_{\text{AODV}} (\lambda(\xi, l).
  l \in \{\text{PRrep-:1 .. PRrep-:6}\} \rightarrow 1 \leq \text{sqn } (\text{rt } \xi) (\text{dip } \xi)))" \]
  by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv] onl_invariant_sterms_TT [OF aodv_wf addpreRT_welldefined])

- **show ?thesis**

qed

Proposition 7.14

**Lemma rreq_rrep_fresh_any_step_invariant:**

\[ \text{paodv } i \vdash \text{onll } \Gamma_{\text{AODV}} (\lambda((\xi, \_), a, \_). \text{anycast } \text{rreq_rrep_fresh } (\text{rt } \xi) a) \]

**Proof:**

- **have rreq_oip:** 
  \[ \text{paodv } i \vdash \text{onl } \Gamma_{\text{AODV}} (\lambda(\xi, 1).
  1 \in \{\text{PRreq-:3, PRreq-:4, PRreq-:15, PRreq-:27}\} \rightarrow \text{oip } \xi \in \text{kD}(\text{rt } \xi)
  \land (\text{sqn } (\text{rt } \xi) (\text{oip } \xi)) > (\text{osn } \xi)
  \lor (\text{sqn } (\text{rt } \xi) (\text{oip } \xi)) = (\text{osn } \xi)
  \land \text{the } (\text{dhops } (\text{rt } \xi) (\text{oip } \xi)) \leq \text{Suc } (\text{hops } \xi)
  \land \text{the } (\text{flag } (\text{rt } \xi) (\text{oip } \xi)) = \text{val}))" \]
  proof inv_cterms
  fix l l' pp p'
  assume "((\xi, pp) \in \text{reachable } (\text{paodv } i) \text{TT}"
  and "(\text{PRreq-:2})[\lambda \xi. \xi[\text{rt} :=
  \text{update } (\text{rt } \xi) (\text{oip } \xi) (\text{osn } \xi, \text{kno}, \text{val}, \text{Suc } (\text{hops } \xi), \text{sip } \xi, \{\})][\text{p'} \in \text{sterms } \Gamma_{\text{AODV}} pp"
  and "l' = \text{PRreq-:3}"
  show "\text{osn } \xi < \text{sqn } (\text{update } (\text{rt } \xi) (\text{oip } \xi) (\text{osn } \xi, \text{kno}, \text{val}, \text{Suc } (\text{hops } \xi), \text{sip } \xi, \{\})) (\text{oip } \xi)
  \lor (\text{sqn } (\text{update } (\text{rt } \xi) (\text{oip } \xi) (\text{osn } \xi, \text{kno}, \text{val}, \text{Suc } (\text{hops } \xi), \text{sip } \xi, \{\})) (\text{oip } \xi) = \text{osn } \xi
  \land \text{the } (\text{dhops } (\text{update } (\text{rt } \xi) (\text{oip } \xi) (\text{osn } \xi, \text{kno}, \text{val}, \text{Suc } (\text{hops } \xi), \text{sip } \xi, \{\})) (\text{oip } \xi)) \leq \text{Suc } (\text{hops } \xi)
  \land \text{the } (\text{flag } (\text{update } (\text{rt } \xi) (\text{oip } \xi) (\text{osn } \xi, \text{kno}, \text{val}, \text{Suc } (\text{hops } \xi), \text{sip } \xi, \{\})) (\text{oip } \xi)) = \text{val})" \]
  unfolding update_def by (clarsimp split: option.split)
  (metis linorder_neqE_nat not_less)

qed

have rrep_prrep: 
  \[ \text{paodv } i \vdash \text{onl } \Gamma_{\text{AODV}} (\lambda(\xi, 1).
  1 \in \{\text{PRreq-:4..PRreq-:6}\} \rightarrow (\text{dip } \xi \in \text{kD}(\text{rt } \xi)
  \land \text{the } (\text{flag } (\text{rt } \xi) (\text{dip } \xi)) = \text{val})))" \]
by (inv_cterms inv add: onl_invariant_sterms_TA [OF aodv_wf adpreRT_welldefined]
onl_invariant_sterms [OF aodv_wf sip_in_kD])

show ?thesis
by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf rreq_oip]
onl_invariant_sterms [OF aodv_wf rreq_dip_in_vD_dip_eq_ip]
onl_invariant_sterms [OF aodv_wf rrep_prrep])

qed

Proposition 7.15

lemma rerr_invalid_any_step_invariant:
"paodv i ||= onll Γ AODV (λ((ξ, _, a, _). anycast (rerr_invalid (rt ξ)) a)"
proof -
have dests_inv: "paodv i ||= onll Γ AODV (λ((ξ, _), l). (l ∈ {PAodv-:15, PPkt-:7, PRreq-:9, PRreq-:21, PRrep-:9, PRerr-:1})
→ (∀ip∈dom(dests ξ). ip∈vD(rt ξ)))
∧ (l ∈ {PAodv-:16..PAodv-:19})
∪ {PPkt-:8..PPkt-:11}
∪ {PRreq:-10..PRreq:-13}
∪ {PRreq:-22..PRreq:-25}
∪ {PRrep:-10..PRrep:-13}
∪ {PRerr:-2..PRerr:-5} → (∀ip∈dom(dests ξ). ip∈iD(rt ξ)
∧ the (dests ξ ip) = sqn (rt ξ ip))
∧ (l = PPkt-:14 → dip ξ∈iD(rt ξ))"
by inv_cterms (clarsimp split: split_if_asm option.split_asm)+
show ?thesis
by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf dests_inv])

qed

Proposition 7.16

Some well-definedness obligations are irrelevant for the Isabelle development:

1. In each routing table there is at most one entry for each destination: guaranteed by type.
2. In each store of queued data packets there is at most one data queue for each destination: guaranteed by structure.
3. Whenever a set of pairs (rip, rsn) is assigned to the variable dests of type ip ↦ sqn, or to the first argument of the function rerr, this set is a partial function, i.e., there is at most one entry (rip, rsn) for each destination rip: guaranteed by type.

lemma dests_vD_inc_sqn:
"paodv i ||= onll Γ AODV (λ((ξ, _, l). (l ∈ {PAodv-:15, PPkt-:7, PRreq:-9, PRreq:-21, PRrep:-9} → (∀ip∈dom(dests ξ). ip∈vD(rt ξ) ∧ the (dests ξ ip) = inc (sqn (rt ξ ip)))
∧ (l = PPkt-:14 → dip ξ∈iD(rt ξ))
∧ (l = PRerr:-1 → (∀ip∈dom(dests ξ). ip∈vD(rt ξ) ∧ the (dests ξ ip) > sqn (rt ξ ip))))"
by inv_cterms (clarsimp split: split_if_asm option.split_asm)+

Proposition 7.27

lemma route_tables_fresher:
"paodv i ||= onll Γ AODV (λ((ξ, _, _, (ξ′, _)).
∀dip∈kD(rt ξ). rt ξ ⊑ dip rt ξ′))"
proof (inv_cterms inv add:
onl_invariant_sterms [OF aodv_wf dests_vD_inc_sqn [THEN invariant_restrict_inD]]
onl_invariant_sterms [OF aodv_wf hop_count_positive [THEN invariant_restrict_inD]]
onl_invariant_sterms [OF aodv_wf osn_rreq]
onl_invariant_sterms [OF aodv_wf dsn_rreq]
onl_invariant_sterms [OF aodv_wf adpreRT_welldefined [THEN invariant_restrict_inD]])
fix ξ pp p'
assume "((ξ, pp) ∈ reachable (paodv i) (recvmsg rreq_rrep_sn)"
and "{PRreq:-2} |ξ. (rt := update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ, {}))"

218
\[ p' \in \text{terms} \Gamma_{\text{AODV}} pp \]
and \( \text{Suc} \ 0 \leq \text{osn} \ \xi \)
and \(*: \forall \text{ip} \in kD (rt \ \xi). \text{Suc} \ 0 \leq \text{the} (\text{dhops} (rt \ \xi) \ \text{ip})\)

\text{show} \ "\forall \text{ip} \in kD (rt \ \xi). \ \text{rt} \ \xi \sqsubseteq_{ip} \text{update} (rt \ \xi) (\text{oip} \ \xi) (\text{osn} \ \xi, \ \kno, \ \val, \ \text{Suc} (\text{hops} \ \xi), \ \text{sip} \ \xi, \ \{}")''

\text{proof}

\text{fix} \ \text{ip}
\text{assume} \ "\text{ip} \in kD (rt \ \xi)"
\text{moreover with} * \ \text{have} \ "1 \leq \text{the} (\text{dhops} (rt \ \xi) \ \text{ip})" \ \text{by simp}
\text{moreover from} \ \text{Suc} \ 0 \leq \text{dsn} \ \xi
\text{have} \ "\text{update} _{\text{arg_wf}} (\text{dsn} \ \xi, \ \kno, \ \val, \ \text{Suc} (\text{hops} \ \xi), \ \text{sip} \ \xi, \ \{}")'' ..
\text{ultimately show} \ "\text{rt} \ \xi \sqsubseteq_{ip} \text{update} (rt \ \xi) (\text{dip} \ \xi) (\text{dsn} \ \xi, \ \kno, \ \val, \ \text{Suc} (\text{hops} \ \xi), \ \text{sip} \ \xi, \ \{}")"

by (rule rt\_fresher\_update)

\text{qed}

\text{next}

\text{fix} \ \xi \ \text{pp} \ p' \text{ assume} \ "(\xi, \text{pp}) \in \text{reachable} (\text{paodv} \ i) (\text{recvmsg} \ \text{rreq_rrep_sn})"
\text{and} \ "\text{Suc} \ 0 \leq \text{dsn} \ \xi"
\text{and} \ *: \ "\forall \text{ip} \in kD (rt \ \xi). \ \text{Suc} \ 0 \leq \text{the} (\text{dhops} (rt \ \xi) \ \text{ip})"
\text{show} \ "\forall \text{ip} \in kD (rt \ \xi). \ \text{rt} \ \xi \sqsubseteq_{ip} \text{update} (rt \ \xi) (\text{dip} \ \xi) (\text{dsn} \ \xi, \ \kno, \ \val, \ \text{Suc} (\text{hops} \ \xi), \ \text{sip} \ \xi, \ \{}")"

\text{proof}

\text{fix} \ \text{ip}
\text{assume} \ "\text{ip} \in kD (rt \ \xi)"
\text{moreover with} * \ \text{have} \ "1 \leq \text{the} (\text{dhops} (rt \ \xi) \ \text{ip})" \ \text{by simp}
\text{moreover from} \ \text{Suc} \ 0 \leq \text{dsn} \ \xi
\text{have} \ "\text{update} _{\text{arg_wf}} (\text{dsn} \ \xi, \ \kno, \ \val, \ \text{Suc} (\text{hops} \ \xi), \ \text{sip} \ \xi, \ \{}")'' ..
\text{ultimately show} \ "\text{rt} \ \xi \sqsubseteq_{ip} \text{update} (rt \ \xi) (\text{dip} \ \xi) (\text{dsn} \ \xi, \ \kno, \ \val, \ \text{Suc} (\text{hops} \ \xi), \ \text{sip} \ \xi, \ \{}")"

by (rule rt\_fresher\_update)

\text{qed}

\text{end}

2.7 The quality increases predicate

\text{theory} \ B\_Quality\_Increases
\text{imports} B\_Aodv\_Predicates B\_Fresher
\text{begin}

\text{definition} quality_increases :: "state \Rightarrow state \Rightarrow bool"
where "quality_increases \ \xi \ \xi' \equiv \ (\forall \text{dip} \in kD (rt \ \xi). \ \text{dip} \in kD (rt \ \xi') \land \text{rt} \ \xi \sqsubseteq \text{dip} \ \text{rt} \ \xi')
\land (\forall \text{dip}. \ \text{sqn} (rt \ \xi) \ \text{dip} \leq \text{sqn} (rt \ \xi') \ \text{dip})"

\text{lemma} quality_increasesI [intro!]:
\text{assumes} "(\forall \text{dip}. \ \\text{dip} \in kD (rt \ \xi) = \Rightarrow \ \text{dip} \in kD (rt \ \xi'))"
\text{and} "(\forall \text{dip}. \ \text{dip} \in kD (rt \ \xi); \ \text{dip} \in kD (rt \ \xi') \ \Rightarrow \text{rt} \ \xi \sqsubseteq_{\text{dip}} \text{rt} \ \xi')"
\text{and} "(\forall \text{dip}. \ \text{sqn} (rt \ \xi) \ \text{dip} \leq \text{sqn} (rt \ \xi') \ \text{dip})"\n\text{shows} "quality_increases \ \xi \ \xi'"
\text{unfolding} quality_increases_def using \text{assms} by \text{clarsimp}

\text{lemma} quality_increasesE [elim]:
\text{fixes} \ \text{dip}
\text{assumes} "quality_increases \ \xi \ \xi'"
\text{and} "\text{dip} \in kD (rt \ \xi)"
\text{and} "(\text{dip} \in kD (rt \ \xi'); \ \text{rt} \ \xi \sqsubseteq_{\text{dip}} \text{rt} \ \xi'; \ \text{sqn} (rt \ \xi) \ \text{dip} \leq \text{sqn} (rt \ \xi') \ \text{dip}) \ \Rightarrow \ \text{R} \ \text{dip} \ \xi'"
\text{shows} "\text{R} \ \text{dip} \ \xi'"
\text{using} \text{assms unfolding} quality_increases_def by \text{clarsimp}

\text{lemma} quality_increases_rt_fresherD [dest]:
\text{fixes} \ \text{ip}
\text{assumes} "quality_increases \ \xi \ \xi'"
\text{and} "\text{ip} \in kD (rt \ \xi)"
\text{shows} "\text{rt} \ \xi \sqsubseteq_{ip} \text{rt} \ \xi'"

\text{end}
using assms by auto

lemma quality_increases_sqnE [elim]:
  fixes dip
  assumes "quality_increases ξ ξ′"
  and "sqn (rt ξ) dip ≤ sqn (rt ξ′) dip ⇒ R dip ξ ξ′"
  shows "R dip ξ ξ′"
using assms unfolding quality_increases_def by clarsimp

lemma quality_increases_refl [intro, simp]: "quality_increases ξ ξ"
  by rule simp_all

lemma strictly_fresher_quality_increases_right [elim]:
  fixes σ σ′ dip
  assumes "rt (σ i) ⊏ dip rt (σ′ nhip)"
  and qinc: "quality_increases (σ nhip) (σ′ nhip)"
  and "dip ∈ kD(rt (σ nhip))"
  shows "rt (σ i) ⊏ dip rt (σ′ nhip)"
proof -
  from qinc have "rt (σ nhip) ⊑ dip rt (σ′ nhip)" using ⟨dip ∈ kD(rt (σ nhip))⟩
  by auto
  with ⟨rt (σ i) ⊏ dip rt (σ′ nhip)⟩ show ?thesis ..
qed

lemma kD_quality_increases [elim]:
  assumes "i ∈ kD(rt ξ)"
  and "quality_increases ξ ξ′"
  shows "i ∈ kD(rt ξ′)"
using assms by auto

lemma kD_nsqn_quality_increases [elim]:
  assumes "i ∈ kD(rt ξ)"
  and "quality_increases ξ ξ′"
  shows "i ∈ kD(rt ξ′) ∧ nsqn (rt ξ) i ≤ nsqn (rt ξ′) i"
proof -
  from assms have "i ∈ kD(rt ξ′)" ..
  moreover with assms have "rt ξ ⊑_i rt ξ′" by auto
  ultimately have "nsqn (rt ξ) i ≤ nsqn (rt ξ′) i" using ⟨i ∈ kD(rt ξ)⟩ by - (erule(2) rt_fresher_imp_nsqn_le)
  with ⟨i ∈ kD(rt ξ′)⟩ show ?thesis ..
qed

lemma nsqn_quality_increases [elim]:
  assumes "i ∈ kD(rt ξ)"
  and "quality_increases ξ ξ′"
  shows "nsqn (rt ξ) i ≤ nsqn (rt ξ′) i"
using assms by (rule kD_nsqn_quality_increases [THEN conjunct2])

lemma kD_nsqn_quality_increases_trans [elim]:
  assumes "i ∈ kD(rt ξ)"
  and "s ≤ nsqn (rt ξ) i"
  and "quality_increases ξ ξ′"
  shows "i ∈ kD(rt ξ′) ∧ s ≤ nsqn (rt ξ′) i"
proof
  from ⟨i ∈ kD(rt ξ)⟩ and ⟨quality_increases ξ ξ′⟩ show "i ∈ kD(rt ξ′)" ..
  next
  from ⟨i ∈ kD(rt ξ)⟩ and ⟨quality_increases ξ ξ′⟩ have "nsqn (rt ξ) i ≤ nsqn (rt ξ′) i" ..
  with ⟨s ≤ nsqn (rt ξ) i⟩ show "s ≤ nsqn (rt ξ′) i" by (rule le_trans)
qed

lemma nsqn_quality_increases_nsqn_lt_lt [elim]:
  assumes "i ∈ kD(rt ξ)"
  and "quality_increases ξ ξ′"
  and "s < nsqn (rt ξ) i"
  qed
shows "s < nsqn (rt ξ') i"

proof -
from assms(1-2) have "nsqn (rt ξ) i ≤ nsqn (rt ξ') i" ..
with "s < nsqn (rt ξ) i" show "s < nsqn (rt ξ') i" by simp
qed

lemma nsqn_quality_increases_dhops [elim]:
assumes "i ∈ kD(rt ξ)"
and "quality_increases ξ ξ'"
and "nsqn (rt ξ) i = nsqn (rt ξ') i"
shows "the (dhops (rt ξ) i) ≥ the (dhops (rt ξ') i)"
using assms unfolding quality_increases_def
by (clarsimp) (drule(1) bspec, clarsimp simp: rt_fresher_def2)

lemma nsqn_quality_increases_nsqn_eq_le [elim]:
assumes "i ∈ kD(rt ξ)"
and "quality_increases ξ ξ'"
and "s = nsqn (rt ξ) i"
shows "s < nsqn (rt ξ') i ∨ (s = nsqn (rt ξ') i ∧ the (dhops (rt ξ) i) ≥ the (dhops (rt ξ') i))"
using assms by (metis nat_less_le nsqn_quality_increases nsqn_quality_increases_dhops)

lemma quality_increases_rreq_rrep_props [elim]:
fixes sn ip hops sip
assumes qinc: "quality_increases (σ sip) (σ' sip)"
and "1 ≤ sn"
and *: "ip ∈ kD(rt (σ sip)) ∧ sn ≤ nsqn (rt (σ sip)) ip
∧ (nsqn (rt (σ sip)) ip = sn
→ (the (dhops (rt (σ sip)) ip) ≤ hops
∧ the (flag (rt (σ sip)) ip) = inv))"
shows "ip ∈ kD(rt (σ' sip)) ∧ sn ≤ nsqn (rt (σ' sip)) ip
∧ (nsqn (rt (σ' sip)) ip = sn
→ (the (dhops (rt (σ' sip)) ip) ≤ hops
∧ the (flag (rt (σ' sip)) ip) = inv))"
(is "_ ∧ ?nsqnafter")
proof -
from * obtain "ip ∈ kD(rt (σ sip))" and "sn ≤ nsqn (rt (σ sip)) ip" by auto
from quality_increases (σ sip) (σ' sip)
have "sqn (rt (σ sip)) ip ≤ sqn (rt (σ' sip)) ip" ..
from quality_increases (σ sip) (σ' sip) and ip ∈ kD (rt (σ sip))
have "ip ∈ kD (rt (σ' sip))" ..
from (sn ≤ nsqn (rt (σ sip)) ip) have ?nsqnafter
proof
assume "sn < nsqn (rt (σ sip)) ip"
also from (ip ∈ kD(rt (σ sip))) and (quality_increases (σ sip) (σ' sip))
have "... ≤ nsqn (rt (σ', sip)) ip" ..
finally have "sn < nsqn (rt (σ' sip)) ip" .
thus ?thesis by simp
next
assume "sn = nsqn (rt (σ sip)) ip"
with (ip ∈ kD(rt (σ sip))) and (quality_increases (σ sip) (σ' sip))
have "sn < nsqn (rt (σ') sip)" ip
∨ (sn = nsqn (rt (σ' sip)) ip
∧ the (dhops (rt (σ' sip)) ip) ≤ the (dhops (rt (σ sip)) ip))" ..
hence "sn < nsqn (rt (σ' sip)) ip
∨ (nsqn (rt (σ' sip)) ip = sn ∧ (the (dhops (rt (σ' sip)) ip) ≤ hops
∧ the (flag (rt (σ' sip)) ip) = inv))"
proof
assume "sn < nsqn (rt (σ' sip)) ip" thus ?thesis ..
next
assume "sn = nsqn (rt (σ' sip)) ip
∧ the (dhops (rt (σ sip)) ip) ≥ the (dhops (rt (σ' sip)) ip)"
hence "sn = nsqn (rt (σ' sip)) ip"
and \( \text{the (dhops (rt (\sigma' sip)) ip) \leq the (dhops (rt (\sigma sip)) ip) } \) by auto

from \* and \( \langle sn = nsqn (rt (\sigma sip)) ip \rangle \) have \( \text{the (dhops (rt (\sigma sip)) ip) \leq hops} \)
\( \lor \text{the (flag (rt (\sigma sip)) ip) = inv} \)

by simp

thus \?thesis

proof

assume \( \text{the (dhops (rt (\sigma sip)) ip) \leq hops} \)

with \( \text{the (dhops (rt (\sigma sip)) ip) \leq the (dhops (rt (\sigma sip)) ip)} \)

have \( \text{the (dhops (rt (\sigma sip)) ip) \leq hops} \) by simp

with \( \langle sn = nsqn (rt (\sigma sip)) ip \rangle \) show \?thesis by simp

next

assume \( \text{the (flag (rt (\sigma sip)) ip) = inv} \)

with \( \langle ip \in kD(rt (\sigma sip)) \rangle \) have \( nsqn (rt (\sigma sip)) ip =_sqn (rt (\sigma sip)) ip - 1 \) ..

with \( \langle ip \in kD(rt (\sigma sip)) \rangle \) show \?thesis

proof (rule vD_or_iD)

assume \( \langle ip \in vD(rt (\sigma sip)) \rangle \)

hence \( \text{nsqn (rt (\sigma sip)) ip = sqn (rt (\sigma sip)) ip} \) ..

with \( \langle sqn (rt (\sigma sip)) ip \leq sqn (rt (\sigma sip)) ip \rangle \) have \( \text{nsqn (rt (\sigma sip)) ip \geq sqn (rt (\sigma sip)) ip} \) by simp

with \( \langle \text{nsqn (rt (\sigma sip)) ip} \rangle \) have \( \text{nsqn (rt (\sigma sip)) ip \geq sqn (rt (\sigma sip)) ip} \) by simp

thus \?thesis ..

qed

qed

thus \?thesis by (metis (mono_tags) le_cases not_le)

qed

with \( \langle ip \in kD (rt (\sigma sip)) \rangle \) show \( \langle ip \in kD (rt (\sigma sip)) \rangle \land nsqnafter \) ..

qed

lemma quality_increases_rreq_rrep_props':

fixes sn ip hops sip

assumes "\( \forall j. \text{quality_increases (\sigma j) (\sigma' j)} \)"

and "\( 1 \leq sn \)"

and \*: "\( \langle ip \in kD(rt (\sigma sip)) \rangle \land sn \leq nsqn (rt (\sigma sip)) ip \land (nsqn (rt (\sigma sip)) ip = sn \to (\text{the (dhops (rt (\sigma sip)) ip) \leq hops} \lor \text{the (flag (rt (\sigma sip)) ip) = inv})) \)"

shows "\( \langle ip \in kD(rt (\sigma sip)) \rangle \land sn \leq nsqn (rt (\sigma sip)) ip \land (nsqn (rt (\sigma sip)) ip = sn \to (\text{the (dhops (rt (\sigma sip)) ip) \leq hops} \lor \text{the (flag (rt (\sigma sip)) ip) = inv})) \)"

proof -

from assms(1) have "\( \text{quality_increases (\sigma sip) (\sigma' sip)} \) ..

thus \?thesis using assms(2-3) by (rule quality_increases_rreq_rrep_props)

qed

lemma rreq_quality_increases:

222
assumes "∀ j. j ≠ i → quality_increases (σ j) (σ' j)"
and "rt (σ' i) = rt (σ i)"
shows "∀ j. quality_increases (σ j) (σ' j)"
using assms by clarsimp (metis order_refl quality_increasesI rt_fresher_refl)

definition msg_fresh :: "(ip ⇒ state) ⇒ msg ⇒ bool"
where "msg_fresh σ m ≡
  case m of Rreq hopsc _ _ _ _ oipc osnc sipc ⇒
    osnc ≥ 1 ∧ (sipc ≠ oipc →
      oipc ∈ kD(rt (σ sipc)) ∧ nsqn (rt (σ sipc)) oipc ≥ osnc
      ∧ (nsqn (rt (σ sipc)) oipc = osnc
        → (hopsc ≥ the (dhops (rt (σ sipc)) oipc)
          ∨ the (flag (rt (σ sipc)) oipc) = inv))))
  | Rrep hopsc dipc dsnc _ sipc ⇒
    dsnc ≥ 1 ∧ (sipc ≠ dipc →
      dipc ∈ kD(rt (σ sipc)) ∧ nsqn (rt (σ sipc)) dipc ≥ dsnc
      ∧ (nsqn (rt (σ sipc)) dipc = dsnc
        → (hopsc ≥ the (dhops (rt (σ sipc)) dipc)
          ∨ the (flag (rt (σ sipc)) dipc) = inv))))
  | Rerr destsc sipc ⇒
    (∀ ripc ∈ dom(destsc). (ripc ∈ kD(rt (σ sipc))
      ∧ the (destsc ripc) - 1 ≤ nsqn (rt (σ sipc)) ripc)
  | _ ⇒ True"

lemma msg_fresh [simp]:
"∀ hops rreq dip dsn dsk oip osn sip.
  msg_fresh σ (Rreq hops rreqid dip dsn dsk oip osn sip) =
    (osn ≥ 1 ∧ (sipc ≠ oipc → oipc ∈ kD(rt (σ sipc))
      ∧ nsqn (rt (σ sipc)) oipc ≥ osnc
      ∧ (nsqn (rt (σ sipc)) oipc = osnc
        → (hopsc ≥ the (dhops (rt (σ sipc)) oipc)
          ∨ the (flag (rt (σ sipc)) oipc) = inv))))"

"∀ hops dip dsn oip sip. msg_fresh σ (Rreq hops dip dsn oip sip) =
  (dsn ≥ 1 ∧ (sipc ≠ dipc → dipc ∈ kD(rt (σ sipc))
    ∧ nsqn (rt (σ sipc)) dipc ≥ dsn
    ∧ (nsqn (rt (σ sipc)) dipc = dsn
      → (hopsc ≥ the (dhops (rt (σ sipc)) dipc)
        ∨ the (flag (rt (σ sipc)) dipc) = inv))))"

"∀ dests sip. msg_fresh σ (Rerr dests sip) =
  (∀ ripc ∈ dom(dests). (ripc ∈ kD(rt (σ sipc))
    ∧ the (dests ripc) - 1 ≤ nsqn (rt (σ sipc)) ripc))"

"∀ d dip. msg_fresh σ (Newpkt d dip) = True"
"∀ d dip sip. msg_fresh σ (Pkt d dip sip) = True"

unfolding msg_fresh_def by simp_all

lemma msg_fresh_inc_sn [simp, elim]:
"msg_fresh σ m =⇒ rreq_rrep_sn m"
by (cases m) simp_all

lemma recv_msg_fresh_inc_sn [simp, elim]:
"orecvmsg (msg_fresh) σ m =⇒ recvmsg rreq_rrep_sn m"
by (cases m) simp_all

lemma rreq_nsqn_is_fresh [simp]:
fixes σ msg hops rreqid dip dsn dsk oip osn sip
assumes "rreq_rrep_fresh (rt (σ sipc)) (Rreq hops rreqid dip dsn dsk oip osn sip)"
and "rreq_rrep_sn (Rreq hops rreqid dip dsn dsk oip osn sip)"
shows "msg_fresh σ (Rreq hops rreqid dip dsn dsk oip osn sip)"
(is "msg_fresh σ ?msg")

proof -
  let ?rt = "rt (σ sipc)"
  from assms(2) have "1 ≤ osn" by simp
  thus ?thesis
  unfolding msg_fresh_def
  proof (simp only: msg.case, intro conjI impI)
    assume "sipc ≠ oipc"
    with assms(1) show "oipc ∈ kD(?rt)" by simp
next
assume "sip ≠ oip"
and "nsqn ?rt oip = osn"
show "the (dhops ?rt oip) ≤ hops ∨ the (flag ?rt oip) = inv"
proof (cases "oip∈vD(?rt)"
  assume "oip∈vD(?rt)"
  hence "nsqn ?rt oip = sqn ?rt oip" ..
  with ⟨nsqn ?rt oip = osn⟩ have "sqn ?rt oip = osn" by simp
with assms(1) and ⟨sip ≠ oip⟩ have "the (dhops ?rt oip) ≤ hops" by simp
  thus ?thesis ..
next
assume "oip ∈ vD(?rt)"
moreover from assms(1) and ⟨sip ≠ oip⟩ have "oip∈kD(?rt)" by simp
ultimately have "oip∈iD(?rt)" by auto
hence "the (flag ?rt oip) = inv" ..
thus ?thesis ..
qed
next
assume "sip ≠ oip"
with assms(1) have "osn ≤ sqn ?rt oip" by auto
thus "osn ≤ nsqn ?rt oip" by simp
proof (rule nat_le_eq_or_lt)
  assume "osn < sqn ?rt oip"
  hence "osn ≤ sqn ?rt oip - 1" by simp
  also have "... ≤ nsqn ?rt oip" by (rule sqn_nsqn)
  finally show "osn ≤ nsqn ?rt oip" .
next
assume "osn = sqn ?rt oip"
with ⟨osn = sqn ?rt oip⟩ have "oip∈kD(?rt)" and "the (flag ?rt oip) = val" by auto
hence "nsqn ?rt oip = sqn ?rt oip" ..
with assms(1) and ⟨sip ≠ oip⟩ have "nsqn ?rt oip = osn" by simp
thus "osn ≤ nsqn ?rt oip" by simp
qed
qed simp

lemma rrep_nsqn_is_fresh [simp]:
fixes σ msg hops dip dsn oip sip
assumes "rreq_rrep_fresh (rt (σ sip)) (Rrep hops dip dsn oip sip)"
and "rreq_rrep_sn (Rrep hops dip dsn oip sip)"
shows "msg_fresh σ (Rrep hops dip dsn oip sip)"
(is "msg_fresh σ ?msg")
proof -
  let ?rt = "rt (σ sip)"
  from assms have "sip ≠ dip → dip∈kD(?rt) ∧ sqn ?rt dip = dsn ∧ the (flag ?rt dip) = val"
    by simp
  hence "sip ≠ dip → dip∈kD(?rt) ∧ nsqn ?rt dip ≥ dsn"
    by clarsimp
  with assms show "msg_fresh σ ?msg"
    by clarsimp
qed

lemma rerr_nsqn_is_fresh [simp]:
fixes σ msg dests sip
assumes "rerr_invalid (rt (σ sip)) (Rerr dests sip)"
shows "msg_fresh σ (Rerr dests sip)"
(is "msg_fresh σ ?msg")
proof -
  let ?rt = "rt (σ sip)"
  from assms have "*: "(∀ rip∈dom(dests). (rip∈iD(rt (σ sip))
    ∧ the (dests rip) = sqn (rt (σ sip)) rip))"
by clarsimp
have "(\forall \text{rip} \in \text{dom(dests)}). \text{rip} \in \text{kD(rt (\sigma sip))} \\
    \land \text{the (dests rip) - 1} \leq \text{nsqn (rt (\sigma sip)) rip})"

proof
  fix rip
  assume "\text{rip} \in \text{dom dests}"
  with \* have "\text{rip} \in \text{iD(rt (\sigma sip))}" and "\text{the (dests rip) = sqn (rt (\sigma sip)) rip}
    \land \text{the (dests rip) - 1} \leq \text{nsqn (rt (\sigma sip)) rip}" by auto
  from this(2) have "\text{the (dests rip) - 1} = \text{sqn (rt (\sigma sip)) rip - 1}"
    by simp
  also have "...
    \leq \text{nsqn (rt (\sigma sip)) rip}" by (rule sqn_nsqn)
  finally have "\text{the (dests rip) - 1} \leq \text{nsqn (rt (\sigma sip)) rip}"
    with \langle \text{rip} \in \text{iD(rt (\sigma sip))} \rangle
    show "\text{rip} \in \text{kD(rt (\sigma sip))} \\
    \land \text{the (dests rip) - 1} \leq \text{nsqn (rt (\sigma sip)) rip}"
    by clarsimp
deq
thus "\text{msg_fresh } \sigma ?msg"
  by simp
deq
lemma quality_increases_msg_fresh [elim]:
  assumes qinc: "\forall j. \text{quality_increases (\sigma j) (\sigma' j)}"
    and "\text{msg_fresh } \sigma m"
  shows "\text{msg_fresh } \sigma' m"
using assms(2)
proof (cases m)
  fix hops rreqid dip dsn dsk oip osn sip
  assume [simp]: "m = Rreq hops rreqid dip dsn dsk oip osn sip"
  and "\text{msg_fresh } \sigma m"
  then have "\text{osn} \geq 1" and "\text{sip} = \text{oip} \lor (\text{oip} \in \text{kD(rt (\sigma sip))} \land \text{osn} \leq \text{nsqn (rt (\sigma sip)) oip}
    \land (\text{nsqn (rt (\sigma sip)) oip} = \text{osn}
    \land (\text{the (dhops (rt (\sigma sip)) oip)} \leq \text{hops}
    \lor (\text{the (flag (rt (\sigma sip)) oip}) = \text{inv})))"
    by auto
  from this(2) show ?thesis
  proof
    assume "\text{sip} = \text{oip}" with \langle \text{osn} \geq 1 \rangle show ?thesis by simp
  next
    assume "\text{oip} \in \text{kD(rt (\sigma sip))} \land \text{osn} \leq \text{nsqn (rt (\sigma sip)) oip}
    \land (\text{nsqn (rt (\sigma sip)) oip} = \text{osn}
    \rightarrow (\text{the (dhops (rt (\sigma sip)) oip)} \leq \text{hops}
    \lor (\text{the (flag (rt (\sigma sip)) oip}) = \text{inv})))"
    moreover from qinc have "\text{quality_increases (\sigma sip) (\sigma' sip)}" ..
    ultimately have "\text{oip} \in \text{kD(rt (\sigma sip))} \land \text{osn} \leq \text{nsqn (rt (\sigma sip)) oip}
    \land (\text{nsqn (rt (\sigma sip)) oip} = \text{osn}
    \rightarrow (\text{the (dhops (rt (\sigma sip)) oip)} \leq \text{hops}
    \lor (\text{the (flag (rt (\sigma sip)) oip}) = \text{inv})))"
    using \langle \text{osn} \geq 1 \rangle by (rule quality_increases_rreq_rrep_props [rotated 2])
    with \langle \text{osn} \geq 1 \rangle show "\text{msg_fresh } \sigma' m"
      by (clarsimp)
deq
next
  fix hops dip dsn oip sip
  assume [simp]: "m = Rrep hops dip dsn oip sip"
  and "\text{msg_fresh } \sigma m"
  then have "\text{dsn} \geq 1" and "\text{sip} = \text{dip} \lor (\text{dip} \in \text{kD(rt (\sigma sip))} \land \text{dsn} \leq \text{nsqn (rt (\sigma sip)) dip}
    \land (\text{nsqn (rt (\sigma sip)) dip} = \text{dsn}
    \rightarrow (\text{the (dhops (rt (\sigma sip)) dip)} \leq \text{hops}
    \lor (\text{the (flag (rt (\sigma sip)) dip}) = \text{inv})))"
    by auto
  from this(2) show ?thesis
  proof
    assume "\text{sip} = \text{dip}" with \langle \text{dsn} \geq 1 \rangle show ?thesis by simp
next
assume "dip ∈ kD(rt (σ sip)) \land dsn ≤ nsqn (rt (σ sip)) dip
\land (nsqn (rt (σ sip)) dip = dsn
\rightarrow (the (dhops (rt (σ sip)) dip) ≤ hops
\lor the (flag (rt (σ sip)) dip) = inv))"
moreover from qinc have "quality_increases (σ sip) (σ' sip)"
ultimately have "dip ∈ kD(rt (σ' sip)) \land dsn ≤ nsqn (rt (σ' sip)) dip
\land (nsqn (rt (σ' sip)) dip = dsn
\rightarrow (the (dhops (rt (σ' sip)) dip) ≤ hops
\lor the (flag (rt (σ' sip)) dip) = inv))"
using (dsn ≥ 1) by (rule quality_increases_rreq_rrep_props [rotated 2])
with (dsn ≥ 1) show "msg_fresh σ' m"
by clarsimp
qed
next
fix dests sip
assume [simp]: "m = Rerr dests sip"
and "msg_fresh σ m"
then have *: "∀ rip ∈ dom(dests). rip ∈ kD(rt (σ sip))
\land the (dests rip) - 1 ≤ nsqn (rt (σ sip)) rip"
by simp
proof
fix rip
assume "rip ∈ dom(dests)"
with * have "rip ∈ kD(rt (σ sip))" and "the (dests rip) - 1 ≤ nsqn (rt (σ sip)) rip"
by (drule(1) bspec, clarsimp)+
moreover from qinc have "quality_increases (σ sip) (σ' sip)" by simp
ultimately show "rip ∈ kD(rt (σ' sip)) \land the (dests rip) - 1 ≤ nsqn (rt (σ' sip)) rip" ..
qed thus ?thesis by simp
qed simp_all
end

2.8 The ‘open’ AODV model

theory B_OAodv
imports B_Aodv ../../../AWN/OAWN_SOS_Labels ../../../AWN/OAWN_Convert
begin
Definitions for stating and proving global network properties over individual processes.
definition σ_AODV' :: "((ip ⇒ state) × ((state, msg, pseqp, pseqp label) seqp)) set"
where "σ_AODV' ≡ {λi. aodv_init i, Γ_AODV PAodv}"
abbreviation opaodv :: "ip ⇒ ((ip ⇒ state) × (state, msg, pseqp, pseqp label) seqp, msg seq_action) automaton"
where "opaodv i ≡ (|init = σ_AODV', trans = oseqp_sos Γ_AODV i |)"
lemma initiali_aodv [intro!, simp]: "initiali i (init (opaodv i)) (init (paodv i))"
unfolding σ_AODV_def σ_AODV'_def by rule simp_all
lemma oaoovd_control_within [simp]: "control_within Γ_AODV (init (opaodv i))"
unfolding σ_AODV'_def by (rule control_withinI) (auto simp del: Γ_AODV_simps)
lemma σ_AODV'_labels [simp]: "(σ, p) ∈ σ_AODV' \implies labels Γ_AODV p = {PAodv−:0}"
unfolding σ_AODV'_def by simp
lemma oaoovd_init_kD_empty [simp]: "(σ, p) ∈ σ_AODV' \implies kD (rt (σ i)) = {}"
unfolding σ_AODV'_def kD_def by simp
lemma oaodv_init_vD_empty [simp]:
"(σ, p) ∈ σ AODV' =⇒ vD (rt (σ i)) = {}"
unfolding σ AODV'_def vD_def by simp

lemma oaodv_trans: "trans (opaodv i) = oseqp_sos Γ AODV i"
  by simp

declare oseq_invariant_ctermsI [OF aodv_wf oaodv_control_within aodv_simple_labels oaodv_trans, cterms_intros]
oseq_step_invariant_ctermsI [OF aodv_wf oaodv_control_within aodv_simple_labels oaodv_trans, cterms_intros] end

2.9 Global invariant proofs over sequential processes

theory B_Global_Invariants
imports B_Seq_Invariants
B_Aodv_Predicates
B_Fresher
B_Quality_Increases
"../../../AWN/OAWN_Convert"
B_OAodv
begin

lemma other_quality_increases [elim]:
assumes "other quality_increases I σ σ'"
shows "∀ j. quality_increases (σ j) (σ' j)"
using assms by (rule, clarsimp) (metis quality_increases_refl)

lemma weaken_otherwith [elim]:
fixes m
assumes *: "otherwith P I (orecvmq Q) σ σ' a"
and weakenP: "∀ σ m. P σ m =⇒ P' σ m"
and weakenQ: "∀ σ m. Q σ m =⇒ Q' σ m"
shows "otherwith P' I (orecvmq Q') σ σ' a"
proof
  fix j
  assume "j ∉ I"
  with * have "P (σ j) (σ' j)" by auto
  thus "P' (σ j) (σ' j)" by (rule weakenP)
next
  from * have "orecvmq Q σ a" by auto
  thus "orecvmq Q' σ a" by rule (erule weakenQ)
qed

lemma orecceived_msg_inv:
assumes other: "∀ σ σ' m. [ [ P σ m; other Q {i} σ σ' ] ] =⇒ P σ' m"
  and local: "∀ σ m. P σ m =⇒ P (σ(i := σ i(msg := m))) m"
shows "opaodv i = (otherwith Q {i} (orecvmq P), other Q {i} →)
onl Γ AODV (λ(σ, l). l = PAodv-:1 → P σ (msg (σ i)))"
proof (inv_cterms, intro impI)
  fix σ σ'
  assume l = PAodv-:1 =⇒ P σ (msg (σ i))
  and l = PAodv-:1
  and "other Q {i} σ σ'"
  from this(1-2) have "P σ (msg (σ i))" ..
hence "P σ' (msg (σ i))" using "other Q {i} σ σ'"
    by (rule other)
moreover from "other Q {i} σ σ'" have "σ' i = σ i" ..
ultimately show "P σ' (msg (σ' i))" by simp
next
  fix σ σ' msg
  assume "otherwith Q {i} (orecvmq P) σ σ' (receive msg)"
and \("\sigma' \cdot i = \sigma \cdot i([\text{msg} := \text{msg}])\)"

from this(1) have \("P\ \sigma \cdot \text{msg}\"
and \("\forall j. j \neq i \rightarrow Q (\sigma \cdot j) (\sigma' \cdot j)\)" by auto

from this(1) have \("P (\sigma \cdot i := \sigma \cdot i([\text{msg} := \text{msg}])) \cdot \text{msg}\)" by (rule local)
thus \("P \cdot \sigma' \cdot \text{msg}\"

proof (rule other)
from \(\sigma' \cdot i = \sigma \cdot i([\text{msg} := \text{msg}])\) and \(\forall j. j \neq i \rightarrow Q (\sigma \cdot j) (\sigma' \cdot j)\)
show \("\text{other Q (}\ i \ (\sigma \cdot i := \sigma \cdot i([\text{msg} := \text{msg}])) \ \sigma'\ ""
by \(-(\text{rule otherI, auto)}\)

qed

qed

(Equivalent to) Proposition 7.27

lemma local_quality_increases:
\("\text{ppadv i \rightarrow onll} \Gamma_{AODV} (\lambda(\xi, \_), \_ (\xi', \_)). \text{quality_increases } (\xi'\)"

proof (rule step_invariantI)
fix \(s \ \ a \ s'\)
assume sr: "s \in \text{reachable } (\text{ppadv i}) (\text{recvmsg } \text{rreq\_rrep\_sn})"
and tr: "(s, a, s') \in \text{trans } (\text{ppadv i})"
and rm: "\text{recvmsg } \text{rreq\_rrep\_sn } a"
from sr have srTT: "s \in \text{reachable } (\text{ppadv i}) \ \text{TT} " ..

from route_tables_fresher sr tr rm
have "onll \Gamma_{AODV} (\lambda(\xi, \_), \_ (\xi', \_)). \ \text{all dip } \in \text{kD } (rt \xi). \ \text{rt } \xi \subseteq \text{dip } rt \xi' (s, a, s')"
by (rule step_invariantD)

moreover from known_destinations_increase srTT tr TT_True
have "onll \Gamma_{AODV} (\lambda(\xi, \_), \_ (\xi', \_)). \ \text{kD } (rt \xi) \subseteq \text{kD } (rt \xi') (s, a, s')"
by (rule step_invariantD)

moreover from sqns_increase srTT tr TT_True
have "onll \Gamma_{AODV} (\lambda(\xi, \_), \_ (\xi', \_)). \ \text{all ip, sqn } (rt \xi) \ \text{ip} \leq \text{sqn } (rt \xi') \ \text{ip} (s, a, s')"
by (rule step_invariantD)

ultimately show "onll \Gamma_{AODV} (\lambda(\xi, \_), \_ (\xi', \_)). \ \text{quality_increases } (\xi') (s, a, s')"

unfolding onll_def by auto

qed

lemmas olocal_quality_increases =
open_seq_step_invariant [OF local_quality_increases initiali_aodv aodv_trans aodv_trans, simplified seqll_onll_swap]

lemma quality_increases:
\("\text{opaodv i } \equiv_{A} (\text{otherswith quality_increases } \{i\}) (\text{orecvmsg } (\lambda_. \text{rreq\_rrep\_sn}))\),\n\text{other_quality_increases } \{i\} \rightarrow\)
\(\text{onll} \Gamma_{AODV} (\lambda(\sigma, \_), \_ (\sigma', \_)). \ \text{all j. quality_increases } (\sigma \cdot j) (\sigma' \cdot j)"
(is "\_ =_{A} (\_? S, \_ \rightarrow \_)")

proof (rule onll_ostep_invariantI, simp)
fix \(\sigma \ \ p \ \ \ a \ \ \ \sigma' \ \ \ p' \ \ \ \)
assume or: "((\sigma, p) \in \text{reachable } (\text{opaodv i}) \ ?S (\text{other_quality_increases } \{i\}))"
and ll: "\text{l } \in \ \text{labels } \Gamma_{AODV} p"
and "?S \ \sigma' \ a"  
and tr: "((\sigma, p), a, (\sigma', p')) \in \text{oseqp sos } \Gamma_{AODV} i"
and ll': "\text{l' } \in \ \text{labels } \Gamma_{AODV} p'"
from this(1-3) have "\text{orecvmsg } (\lambda_. \text{rreq\_rrep\_sn }) \sigma \ a"
by (auto dest!: oreachable_weakenE [where QS="\text{act } (\text{recvmsg } \text{rreq\_rrep\_sn})"
and QU="\text{other_quality_increases } \{i\}]"
otherwith_actionD)
with or have orw: "((\sigma, p) \in \text{reachable } (\text{opaodv i}) \ \text{act } (\text{recvmsg } \text{rreq\_rrep\_sn}))"
(other_quality_increases \{i\})"
by \(-(\text{erule oreachable_weakenE, auto)}\)
with tr ll ll' and (orecvmsg (\lambda_. \text{rreq\_rrep\_sn }) \sigma \ a) have "\text{quality_increases } (\sigma \cdot i) (\sigma' \cdot i)"
by \-(\text{drule onll_ostep_invariantD [OF olocal_quality_increases], auto simp: seqll_def})
with \(?S \ \sigma' \ a\) show "\text{other Q } (\ i \ (\sigma \cdot i := \sigma \cdot i([\text{msg} := \text{msg}])) \ \sigma'""
lemma rreq_rrep_nsqn_fresh_any_step_invariant:
"opaodv i \models_A (act (recvmsg rreq_rrep_sn), other A \{i\} \rightarrow)
onll \Gamma_{AODV} (\lambda((\sigma, \_), a, \_). anycast (msg_fresh \sigma) a)"

proof (rule ostep_invariantI, simp del: act_simp)
fix \sigma \, p \, a \, \sigma' \, p'
assume or: "((\sigma, p) \in oreachable (opaodv i) (act (recvmsg rreq_rrep_sn)) (other A \{i\})"
and "((\sigma, p), a, (\sigma', p')) \in osseqp_sos \Gamma_{AODV} i"
and recv: "act (recvmsg rreq_rrep_sn) \sigma \sigma' a"
obtain l l' where "l \in labels \Gamma_{AODV} p" and "l' \in labels \Gamma_{AODV} p'" by (metis aodv_ex_label)
from "((\sigma, p), a, (\sigma', p')) \in trans (opaodv i)" have tr: "((\sigma, p), a, (\sigma', p')) \in trans (opaodv i)" by simp
have "anycast (rreq_rrep_fresh (rt (\sigma i))) a"
proof -
have "opaodv i \models_A (act (recvmsg rreq_rrep_sn), other A \{i\} \rightarrow)
onll \Gamma_{AODV} (seqll i (\lambda((\xi, \_), a, \_). anycast (rreq_rrep_fresh (rt \xi)) a))"
by (rule ostep_invariant_weakenE [OF open_seq_step_invariant [OF rreq_rrep_fresh_any_step_invariant initiali_aodv, simplified seqll_onll_swap]]) auto
hence "onll \Gamma_{AODV} (seqll i (\lambda((\xi, \_), a, \_). anycast (rreq_rrep_fresh (rt \xi)) a)) ((\sigma, p), a, (\sigma', p'))" using or tr recv by - (erule(4) ostep_invariantE)
thus ?thesis using "l \in labels \Gamma_{AODV} p" and "l' \in labels \Gamma_{AODV} p'" by auto
qed

moreover have "anycast (rerr_invalid (rt (\sigma i))) a"
proof -
have "opaodv i \models_A (act (recvmsg rreq_rrep_sn), other A \{i\} \rightarrow)
onll \Gamma_{AODV} (seqll i (\lambda((\xi, \_), a, \_). anycast (rerr_invalid (rt \xi)) a))"
by (rule ostep_invariant_weakenE [OF open_seq_step_invariant [OF rerr_invalid_any_step_invariant initiali_aodv, simplified seqll_onll_swap]]) auto
hence "onll \Gamma_{AODV} (seqll i (\lambda((\xi, \_), a, \_). anycast (rerr_invalid (rt \xi)) a)) ((\sigma, p), a, (\sigma', p'))" using or tr recv by - (erule(4) ostep_invariantE)
thus ?thesis using "l \in labels \Gamma_{AODV} p" and "l' \in labels \Gamma_{AODV} p'" by auto
qed

moreover have "anycast rreq_rrep_sn a"
proof -
from or tr recv have "onll \Gamma_{AODV} (seqll i (\lambda(_, a, \_). anycast rreq_rrep_sn a)) ((\sigma, p), a, (\sigma', p'))" by (rule ostep_invariantE [OF open_seq_step_invariant [OF rreq_rrep_sn_any_step_invariant initiali_aodv, oadv_trans aodv_trans, simplified seqll_onll_swap]])
thus ?thesis using "l \in labels \Gamma_{AODV} p" and "l' \in labels \Gamma_{AODV} p'" by auto
qed

moreover have "anycast (\lambda m. not_Pkt m \rightarrow msg_sender m = i) a"
proof -
have "opaodv i \models_A (act (recvmsg rreq_rrep_sn), other A \{i\} \rightarrow)
onll \Gamma_{AODV} (seqll i (\lambda((\xi, \_), a, \_). anycast (\lambda m. not_Pkt m \rightarrow msg_sender m = i) a))"
by (rule ostep_invariant_weakenE [OF open_seq_step_invariant [OF sender_ip_valid initiali_aodv, simplified seqll_onll_swap]]) auto
thus ?thesis using or tr recv "l \in labels \Gamma_{AODV} p" and "l' \in labels \Gamma_{AODV} p'" by auto
qed
ultimately have "anycast (msg_fresh σ) a"
by (simp_all add: anycast_def
del: msg_fresh
split: seq.action.split_asm msg.split_asm) simp_all
thus "onll Γ_{AODV} (λ((σ, _), a, _). anycast (msg_fresh σ) a) (((σ, p), a, (σ', p')))"
by auto
qed

lemma oreceived_rreq_rrep_nsqn_fresh_inv:
"opaodv i |=( otherwith quality_increases {i} (orecvmsg msg_fresh),
other quality_increases {i} →)
onll Γ_{AODV} (λ((σ, _), a, _), a, _). anycast (msg_fresh σ) a)
((σ, p), a, (σ', p'))"
proof (rule oreceived_msg_inv)
fix σ σ σ' m
assume *: "msg_fresh σ m"
and "other quality_increases {i} σ σ'"
from this(2) have "∀ j. quality_increases (σ j) (σ' j)" ..
thus "msg_fresh σ' m" using *..
next
fix σ m
assume "msg_fresh σ m"
thus "msg_fresh (σ(i := σ i(\mid msg := m\mid))) m"
proof (cases m)
fix dests sip
assume "m = Rerr dests sip"
with "msg_fresh σ m" show ?thesis by auto
qed auto
qed

lemma oquality_increases_nsqn_fresh:
"opaodv i |=( otherwith quality_increases {i} (orecvmsg msg_fresh),
other quality_increases {i} →)
onl Γ_{AODV} (λ((σ, _), _, (_), (σ', _)). ∀ j. quality_increases (σ j) (σ' j))"
by (rule ostep_invariant_weakenE [OF oquality_increases]) auto

lemma oosn_rreq:
"opaodv i |=( otherwith quality_increases {i} (orecvmsg msg_fresh),
other quality_increases {i} →)
onl Γ_{AODV} (seql i (λ(ξ, l). l ∈ {PAodv-:4, PAodv-:5} ∪ {PRreq-:n |n. True} → 1 ≤ osn ξ))"
by (rule oinvariant_weakenE [OF open_seq_invariant [OF osn_rreq initiali_aodv]])
(auto simp: seql_onl_swap)

lemma rreq_sip:
"opaodv i |=( otherwith quality_increases {i} (orecvmsg msg_fresh),
other quality_increases {i} →)"
onl Γ_{AODV} (seql i (λ(ξ, l). l ∈ {PAodv-:4, PAodv-:5, PRreq-:0, PRreq-:2} ∧ sip ξ ≠ oip ξ)
∧ nsqn (rt ξ (σ (sip ξ))) (oip σ i) ≥ osn σ i)
∧ (nsqn (rt ξ (σ (sip ξ))) (oip σ i) = osn σ i)
→ (hops ξ ≥ \text{the}(dhops (rt ξ (σ (sip ξ))) (oip σ i)) \text{the}(flag (rt ξ (σ (sip ξ))) (oip σ i)) \text{inv}))")
(is "_ |=( ?S, ?U → _")
proof (inv_cterms inv add: oseq_step_invariant_sterms [OF oquality_increases_nsqn_fresh]
aodv_wf oadv_trans)
onl oinvariant_sterms [OF aodv_wf oreceived_rreq_rrep_nsqn_fresh_inv]
onl oinvariant_sterms [OF aodv_wf oosn_rreq]
simp add: seqlsimp
simp del: One_nat_def, rule impI)
fix σ σ' p l
assume "(σ, p) ∈ oreachable (opaodv i) ?S ?U"
and "1 ∈ labels $\Gamma_{AODV}$ p"
and pre:

\[(I = PAdv-:4 \lor I = PAdv-:5 \lor I = PRreq-:0 \lor I = PRreq-:2) \land \text{sip } (\sigma \ i) \neq \text{oip } (\sigma \ i)\]

\[\rightarrow \text{oip } (\sigma \ i) \in kD \ (rt \ (\sigma \ (\text{sip } (\sigma \ i))))\]

\[\land \ \text{osn } (\sigma \ i) \leq \text{nsqn } (rt \ (\sigma \ (\text{sip } (\sigma \ i)))) \ (\text{oip } (\sigma \ i))\]

\[\land \ (\text{nsqn } (rt \ (\sigma \ (\text{sip } (\sigma \ i)))) \ (\text{oip } (\sigma \ i)) = \text{osn } (\sigma \ i)\]

\[\rightarrow \ \text{the } (\text{dhops } (rt \ (\sigma \ (\text{sip } (\sigma \ i)))) \ (\text{oip } (\sigma \ i))) \leq \text{hops } (\sigma \ i)\]

\[\lor \ \text{the } (\text{flag } (rt \ (\sigma \ (\text{sip } (\sigma \ i)))) \ (\text{oip } (\sigma \ i))) = \text{inv}"

and "other_quality_increases \{i\} σ σ′"
and hyp: "(I = PAdv-:4 \lor I = PAdv-:5 \lor I = PRreq-:0 \lor I = PRreq-:2) \land \text{sip } (\sigma \ i) \neq \text{oip } (\sigma \ i)"

(is "?labels \land \text{sip } (\sigma \ i) \neq \text{oip } (\sigma \ i)"
from (4) have "σ′ \ i = σ \ i"
with hyp have hyp': "?labels \land \text{sip } (\sigma \ i) \neq \text{oip } (\sigma \ i)" by simp
show "\text{sip } (\sigma \ i) \in kD \ (rt \ (\sigma \ (\text{sip } (\sigma \ i))))
\land \ \text{osn } (\sigma \ i) \leq \text{nsqn } (rt \ (\sigma \ (\text{sip } (\sigma \ i)))) \ (\text{oip } (\sigma \ i))\]

\[\land \ (\text{nsqn } (rt \ (\sigma \ (\text{sip } (\sigma \ i)))) \ (\text{oip } (\sigma \ i)) = \text{osn } (\sigma \ i)\]

\[\rightarrow \ \text{the } (\text{dhops } (rt \ (\sigma \ (\text{sip } (\sigma \ i)))) \ (\text{oip } (\sigma \ i))) \leq \text{hops } (\sigma \ i)\]

\[\lor \ \text{the } (\text{flag } (rt \ (\sigma \ (\text{sip } (\sigma \ i)))) \ (\text{oip } (\sigma \ i))) = \text{inv}"

proof (cases "\text{sip } (\sigma \ i) = i")
assume "\text{sip } (\sigma \ i) \neq i"
from other_quality_increases \{i\} σ σ′,
    have "quality_increases \{\text{sip } (\sigma \ i)\} \ (\sigma \ i) \ (\text{sip } (\sigma \ i))" by (rule otherE) simp
moreover from (σ, p) ∈ orachable (opaodv i) ?S ?U \ 1 ∈ labels $\Gamma_{AODV}$ p and hyp
    have "1 ≤ \text{osn } (\sigma \ i)"
    by (auto dest!: onl_openinvariant_weakenD OF oosn_rreq simp add: seqlsimp simp smp i = σ i)
moreover from (\text{sip } (\sigma \ i) \neq i) hyp' and pre
    have "\text{oip } (\sigma \ i) \in kD \ (rt \ (\sigma \ (\text{sip } (\sigma \ i))))
\land \ \text{osn } (\sigma \ i) \leq \text{nsqn } (rt \ (\sigma \ (\text{sip } (\sigma \ i)))) \ (\text{oip } (\sigma \ i))\]

\[\land \ (\text{nsqn } (rt \ (\sigma \ (\text{sip } (\sigma \ i)))) \ (\text{oip } (\sigma \ i)) = \text{osn } (\sigma \ i)\]

\[\rightarrow \ \text{the } (\text{dhops } (rt \ (\sigma \ (\text{sip } (\sigma \ i)))) \ (\text{oip } (\sigma \ i))) \leq \text{hops } (\sigma \ i)\]

\[\lor \ \text{the } (\text{flag } (rt \ (\sigma \ (\text{sip } (\sigma \ i)))) \ (\text{oip } (\sigma \ i))) = \text{inv}"

by (auto simp: σ′ i = σ i)
ultimately show ?thesis
by (rule quality_increases_rreq_rrep_props)
next
assume "\text{sip } (\sigma \ i) = i" thus ?thesis
using smp i = σ i hyp and pre by auto
qed
qed (auto elim!: quality_increases_rreq_rrep_props')

lemma odsn_rrep:
"opaodv i \models \text{(otherwith quality_increases \{i\} \ \text{orecvmsg msg_fresh), other_quality_increases \{i\} \rightarrow)}\]

\text{onl } \Gamma_{AODV} \ \text{(seql } (\lambda(\xi, 1). \ l ∈ \{PAdv-:6, PAdv-:7\} \cup \{PRrep-:n|n. True \ \rightarrow \ 1 ≤ \text{dsn } \xi\})"

by (rule onlvariant_weakenE OF open_seq_invariant [OF dsn_rrep initiali_aodv])
(auto simp: seql_onl_swap)

lemma rrep_Sip:
"opaodv i \models \text{(otherwith quality_increases \{i\} \ \text{orecvmsg msg_fresh), other_quality_increases \{i\} \rightarrow)}\]

\text{onl } \Gamma_{AODV} \ \text{(\lambda(\sigma, 1).}
\text{ (l ∈ \{PAdv-:6, PAdv-:7, PRreq-:0, PRreq-:1\} \land \text{sip } (\sigma \ i) \neq \text{dip } (\sigma \ i)\]

\text{ \rightarrow \ \text{dip } (\sigma \ i) \in kD (rt \ (\sigma \ (\text{sip } (\sigma \ i))))\]

\text{ \land \ \text{nsqn } (rt \ (\sigma \ (\text{sip } (\sigma \ i)))) \ (\text{dip } (\sigma \ i)) ≥ \text{dsn } (\sigma \ i)\]

\text{ \land \ (\text{nsqn } (rt \ (\sigma \ (\text{sip } (\sigma \ i)))) \ (\text{dip } (\sigma \ i)) = \text{dsn } (\sigma \ i)\]

\text{ \rightarrow \ (\text{hops } (\sigma \ i) ≥ \text{the } (\text{dhops } (rt \ (\sigma \ (\text{sip } (\sigma \ i)))) \ (\text{dip } (\sigma \ i))\]

\text{ \ \lor \ \text{the } (\text{flag } (rt \ (\sigma \ (\text{sip } (\sigma \ i)))) \ (\text{dip } (\sigma \ i))) = \text{inv}))"}

(is "\_ \models \text{(?S, ?U \rightarrow \_)}")
proof (inv_cterms inv add: oseq_step_invariant_stems [OF equality_increases_nsqn_fresh aodv_wf oadtrans]
\text{onlvariant_stems [OF aodv_wf orceived_rreq_rrep nsqn_fresh_inv]
\text{onlvariant_stems [OF aodv_wf odsn_rrep]}

231
simp del: One_nat_def, rule impI)

fix σ σ' p l
assume "((σ, p) ∈ oreachable (opaodv i) ?S ?U"
and "l ∈ labels Γ_AODV p"
and pre:
  "(l = PAodv->:6 ∨ l = PAodv->:7 ∨ l = PRrep->:0 ∨ l = PRrep->:1) ∧ sip (σ i) ≠ dip (σ i)
  → dip (σ i) ∈ kD (rt (σ (sip (σ i))))
  ∧dsn (σ i) ≤ nsqn (rt (σ (sip (σ i)))) (dip (σ i))
  ∧ (nsqn (rt (σ (sip (σ i)))) (dip (σ i))) = dsn (σ i)
  → the (dhops (rt (σ (sip (σ i)))) (dip (σ i))) ≤ hops (σ i)
  ∨ the (flag (rt (σ (sip (σ i)))) (dip (σ i))) = inv"
and "other quality_increases {i} σ σ'"
and hyp: "(l=PAodv->:6 ∨ l=PAodv->:7 ∨ l=PRrep->:0 ∨ l=PRrep->:1) ∧ sip (σ i) ≠ dip (σ i)"
(is {?labels ∧ sip (σ i) ≠ dip (σ i))"
from(4) have "σ i = σ i"
with hyp have hyp': "?labels ∧ sip (σ i) ≠ dip (σ i)" by simp
show "dip (σ i) ∈ kD (rt (σ (sip (σ i))))
∧dsn (σ i) ≤ nsqn (rt (σ (sip (σ i)))) (dip (σ i))
∧ (nsqn (rt (σ (sip (σ i)))) (dip (σ i))) = dsn (σ i)
→ the (dhops (rt (σ (sip (σ i)))) (dip (σ i))) ≤ hops (σ i)
∧ the (flag (rt (σ (sip (σ i)))) (dip (σ i))) = inv"
proof (cases "sip (σ i) = i")
assume "sip (σ i) ≠ i"
from "other quality_increases {i} σ σ'"
have "quality_increases (σ (sip (σ i))) (σ' (sip (σ i)))"
by (rule otherE) (clarsimp simp: is)
moreover from (σ, p) ∈ oreachable (opaodv i) ?S ?U l ∈ labels Γ_AODV p and hyp
have "1 ≤ dsn (σ i)"
by (auto dest!: onl_o invariant_weakenD [OF odsn_rrep]
simp add: seqlsimp σ' i = σ i)
moreover from (sip (σ i) ≠ i) hyp' and pre
have "dip (σ i) ∈ kD (rt (σ (sip (σ i))))
∧dsn (σ i) ≤ nsqn (rt (σ (sip (σ i)))) (dip (σ i))
∧ (nsqn (rt (σ (sip (σ i)))) (dip (σ i))) = dsn (σ i)
→ the (dhops (rt (σ (sip (σ i)))) (dip (σ i))) ≤ hops (σ i)
∧ the (flag (rt (σ (sip (σ i)))) (dip (σ i))) = inv"
by (auto simp: σ' i = σ i)
ultimately show thesis
by (rule quality_increases_rreq_rrep_props)
next
assume "sip (σ i) = i" thus thesis
using σ' i = σ i hyp and pre by auto
qed

lemma rerr_sip:
"opaodv i = (otherwith quality_increases {i} (orecmsg msg_fresh),
  other_quality_increases {i} →)
onl Γ_AODV (λ(σ, i).
  l ∈ {PAodv->:8, PAodv->:9, PRrep->:0, PRrep->:1} → (∀ ripc∈dom(dests (σ i)). ripc∈kD(rt (σ (sip (σ i)))) ∧
  the (dests (σ i) ripc) - 1 ≤ nsqn (rt (σ (sip (σ i))))))")
(is "_ = (?S, ?U → _)")
proof -
{ fix dests rip sip rsn and σ σ' :: "ip ⇒ state"
  assume qinc: "∀ j. quality_increases (σ j) (σ' j)"
  and *: "∀ ripc∈dom dests. rip ∈ kD (rt (σ sip))
  ∧ the (dests rip) - 1 ≤ nsqn (rt (σ sip)) rip"
  and "dests rip = Some rsn"
from this(3) have "rip∈dom dests" by auto
with * and "dests rip = Some rsn" have "rip∈kD(rt (σ sip))"
  and "rsn - 1 ≤ nsqn (rt (σ sip)) rip"
  by (auto dest!: bspec)
from qinc have "quality_increases (σ sip) (σ' sip)" ..
have "rip ∈ kD((σ i) rip) ∧ rsn - 1 ≤ nsqn (rt (σ i) rip)" ..
proof from ⟨rip ∈ kD(rt (σ i))⟩ and ⟨quality_increases (σ i) (σ i) (σ i)⟩
show "rip ∈ kD(rt (σ i) rip)" ..
next from ⟨rip ∈ kD(rt (σ i))⟩ and ⟨quality_increases (σ i) (σ i) (σ i)⟩
have "nsqn (rt (σ i) rip) ≤ nsqn (rt (σ i) rip)" ..
with ⟨rsn - 1 ≤ nsqn (rt (σ i) rip)⟩ show "rsn - 1 ≤ nsqn (rt (σ i) rip)" by (rule le_trans)
qed

} note partial = this

proof

show ?thesis
by (inv_cterms inv add: oseq_step_invariant_sterms [OF equality_increases_nsqn_fresh aodv_wf aodv_trans]
onl_oivariable_invariant_sterms [OF aodv_wf oreceived_rreq_rrep_nsqn_fresh_inv]
other_quality_increases other_localD
simp del: One_nat_def, intro conjI)
(clarsimp simp del: One_nat_def split: split_if_asm option.split_asm, erule(2) partial)+
qed

lemma prerr_guard: "paodv i ||=
onl Γ AODV (λ(ξ, l). (l = PReerr-:1
→ (∀ip ∈ dom(dests ξ). ip ∈ vD(rt ξ)
∧ the (nhop (rt ξ) ip) = sip ξ
∧ sqn (rt ξ) ip < the (dests ξ ip))))"

by (inv_cterms) (clarsimp split: option.split_asm split_if_asm)

lemmas oaddpreRT_welldefined =
open_seq_invariant [OF addpreRT_welldefined initiali_aodv aodv_trans aodv_trans,
simplified seql_onl_swap,
THEN oinvariant_anyact]

lemmas odests_vD_inc_sqn =
open_seq_invariant [OF dests_vD_inc_sqn initiali_aodv aodv_trans aodv_trans,
simplified seql_onl_swap,
THEN oinvariant_anyact]

lemmas oprerr_guard =
open_seq_invariant [OF prerr_guard initiali_aodv aodv_trans aodv_trans,
simplified seql_onl_swap,
THEN oinvariant_anyact]

Proposition 7.28

lemma seq_compare_next_hop':
"opaodv i ||= (otherwith quality_increases {i} (orecvmsg msg_fresh),
other_quality_increases {i} → onl Γ AODV (λ(σ, _).
∀dip. let nhip = the (nhop (rt (σ i)) dip)
in dip ∈ kD(rt (σ i)) ∧ nhip ≠ dip →
dip ∈ kD(rt (σ nhip)) ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ nhip)) dip"
(is "_ |= (?S, ?U → _)")
proof -

{ fix nhip and σ σ' :: "ip ⇒ state"
assume pre: "∀dip ∈ kD(rt (σ i)). nhip dip ≠ dip →
dip∈kD(rt (σ (nhop dip))) ∧ sqn (rt (σ i)) dip ≤ sqn (rt (σ (nhop dip))) dip"
and qinc: "∀j. quality_increases (σ j) (σ' j)"
have "∀dip ∈ kD(rt (σ i)). nhip dip ≠ dip →
dip∈kD(rt (σ' (nhop dip))) ∧ sqn (rt (σ i)) dip ≤ sqn (rt (σ' (nhop dip))) dip"
proof (intro ballI impI)
fix dip
assume "dip∈kD(rt (σ i))"
and "nhip dip ≠ dip"
with pre have "dip∈kD(rt (σ (nhop dip)))"
and "nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (nhop dip))) dip"
by auto
from qinc have qinc_nhop: "quality_increases (σ (nhop dip)) (σ' (nhop dip))" ..
with ⟨dip∈kD(rt (σ (nhop dip)))⟩ have "dip∈kD (rt (σ' (nhop dip)))" ..
moreover have "nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (nhop dip))) dip"
proof -
from ⟨dip∈kD(rt (σ (nhop dip)))⟩ qinc_nhop
have "nsqn (rt (σ (nhop dip))) dip ≤ nsqn (rt (σ' (nhop dip))) dip" ..
with nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (nhop dip))) dip: show ?thesis
by simp
qed
ultimately show "dip∈kD(rt (σ' (nhop dip)))
∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (nhop dip))) dip" ..
qed
}
ote basic = this

{ fix nhop and σ σ' :: "ip ⇒ state"
assume pre: "∀dip∈kD(rt (σ i)). nhop dip ≠ dip ⟷ dip∈kD(rt (σ (nhop dip)))
∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (nhop dip))) dip"
and ndest: "∀rip∈dom (dests (σ i)). ripc ∈ kD (rt (σ (sip (σ i))))
∧ the (dests (σ i) ripc) - 1 ≤ nsqn (rt (σ (sip (σ i)))) ripc"
and issip: "∀ip∈dom (dests (σ i)). nhop ip = sip (σ i)"
and qinc: "∀j. quality_increases (σ j) (σ' j)"
have "∀dip∈kD(rt (σ i)). nhop dip ≠ dip ⟷ dip∈kD (rt (σ' (nhop dip)))
∧ nsqn (invalidate (rt (σ i)) (dests (σ i))) dip ≤ nsqn (rt (σ' (nhop dip))) dip"
proof (intro ballI impI)
fix dip
assume "dip∈kD (rt (σ i))"
and "nhop dip ≠ dip"
with pre and qinc have "dip∈kD (rt (σ' (nhop dip)))"
and "nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (nhop dip))) dip"
by (auto dest!: basic)
have "nsqn (invalidate (rt (σ i)) (dests (σ i))) dip ≤ nsqn (rt (σ' (nhop dip))) dip"
proof (cases "dip∈dom (dests (σ i))")
assume "dip∈dom (dests (σ i))"
with ⟨dip∈kD(rt (σ i)))⟩ obtain dsn where "dests (σ i) dip = Some dsn"
by auto
with ⟨dip∈kD(rt (σ i)))⟩ have "nsqn (invalidate (rt (σ i)) (dests (σ i))) dip = dsn - 1"
by (rule nsqn_invalidate_eq)
moreover have "dsn - 1 ≤ nsqn (rt (σ' (nhop dip))) dip"
proof -
from ⟨dests (σ i) dip = Some dsn⟩ have "the (dests (σ i) dip) = dsn" by simp
with ndest and ⟨dip∈dom (dests (σ i))⟩ have "dip∈kD (rt (σ (sip (σ i))))"
"dsn - 1 ≤ nsqn (rt (σ (sip (σ i)))) dip"
by auto
moreover from issip and ⟨dip∈dom (dests (σ i))⟩ have "nhop dip = sip (σ i)" ..
ultimately have "dip∈kD (rt (σ (nhop dip)))"
and "dsn - 1 ≤ nsqn (rt (σ (nhop dip))) dip" by auto
with qinc show "dsn - 1 ≤ nsqn (rt (σ' (nhop dip))) dip"
by simp (metis kD_nsqn_quality_increases_trans)
qed
ultimately show ?thesis by simp
next
assume "dip ∉ dom (dests (σ i))"
with ⟨dip∈kD(rt (σ i)))⟩
have "nsqn (invalidate (rt (σ i)) (dests (σ i))) dip = nsqn (rt (σ i)) dip"
by (rule nsqn_invalidate_other)
with nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (nhop dip))) dip: show ?thesis by simp
qed
with ⟨dip∈kD(rt (σ' (nhop dip)))⟩
show "dip∈kD (rt (σ' (nhop dip)))"
\[ (\text{invalidate} (rt (\sigma i)) \text{ (dests} (\sigma i))) \text{ dip} \leq \text{nsqn} (rt (\sigma \text{ (nhop dip)})) \text{ dip} \] 

\( \text{qed} \) 

\section*{basic_prerr = this}

\{ 
\text{fix} \ \sigma, \ \sigma' :: "ip \Rightarrow \text{state}"
\begin{enumerate}
\item \text{assume a1:} "\forall \text{dip} \in kD(rt (\sigma i)). \ (\text{nhop} (rt (\sigma i)) \text{ dip} \neq \text{ dip} \rightarrow \text{dip} \in kD(rt (\sigma \text{ (nhop} (rt (\sigma i)) \text{ dip})))) \\
\land \text{nsqn} (rt (\sigma i)) \text{ dip} \leq \text{nsqn} (rt (\sigma \text{ (nhop} (rt (\sigma i)) \text{ dip})))) \text{ dip}"
\item \text{and a2:} "\forall j. \text{quality Increases} (\sigma j) (\sigma' j)"
\end{enumerate}

\text{have "\forall \text{dip} \in kD(rt (\sigma i)).} \\
\text{the (nhop (update (rt (\sigma i)) (sip (\sigma i)) (0, unk, val, Suc 0, sip (\sigma i), {})) dip) \neq \text{ dip} \rightarrow \text{dip} \in kD(rt (\sigma' (\text{the (nhop (rt (\sigma i)) dip})))) \\
\land \text{nsqn} (rt (\sigma i)) \text{ dip} \leq \text{nsqn} (rt (\sigma' (\text{the (nhop (rt (\sigma i)) dip})))) \text{ dip}"

\text{by \ (- (drule(1) basic, auto) \ )}
\text{thus "\text{?P dip}" by \ (cases "\text{dip = sip} (\sigma i)"") auto}

\text{qed}
\} \text{ note nhop_update_sip = this}

\{ 
\text{fix} \ \sigma, \ \sigma' \ oip \ sip \ osn \ hops 
\begin{enumerate}
\item \text{assume pre:} "\forall \text{dip} \in kD(rt (\sigma i)). \ (\text{nhop} (rt (\sigma i)) \text{ dip} \neq \text{ dip} \rightarrow \text{dip} \in kD(rt (\sigma \text{ (the (nhop (rt (\sigma i)) dip)))) \\
\land \text{nsqn} (rt (\sigma i)) \text{ dip} \leq \text{nsqn} (rt (\sigma \text{ (the (nhop (rt (\sigma i)) dip))})) \text{ dip}"
\item \text{and qinc:} "\forall j. \text{quality Increases} (\sigma j) (\sigma' j)"
\item \text{and \*:} "\text{sip} \neq \text{oip} \rightarrow \text{oip} \in kD(rt (\sigma \text{ sip})) \\
\land \text{osn} \leq \text{nsqn} (rt (\sigma \text{ sip})) \text{ oip} \\
\land (\text{nsqn} (rt (\sigma \text{ sip})) \text{ oip} = \text{osn} \\
\rightarrow \text{the (dhops} (rt (\sigma \text{ sip})) \text{ oip} \leq \text{hops} \\
\lor \text{the (flag} (rt (\sigma \text{ sip})) \text{ oip} = \text{inv})")
\end{enumerate}

\text{from pre and qinc}
\begin{enumerate}
\item \text{have pre':} "\forall \text{dip} \in kD (rt (\sigma i)). \ (\text{nhop} (rt (\sigma i)) \text{ dip} \neq \text{ dip} \rightarrow \text{dip} \in kD(rt (\sigma' (\text{the (nhop} (rt (\sigma i)) \text{ dip)))) \\
\land \text{nsqn} (rt (\sigma i)) \text{ dip} \leq \text{nsqn} (rt (\sigma' (\text{the (nhop} (rt (\sigma i)) \text{ dip)))) \text{ dip}"
\item \text{by \ (rule basic) \ )}
\item \text{have "(the (nhop (update (rt (\sigma i)) oip (osn, kno, val, Suc hops, sip, {})) oip) \neq \text{ oip} \rightarrow \text{oip} \in kD(rt (\sigma' (\text{the (nhop} (update (rt (\sigma i)) \text{ oip} \\
(osn, kno, val, Suc hops, sip, {})) \text{ oip})))) \\
\land \text{nsqn} (update (rt (\sigma i)) oip (osn, kno, val, Suc hops, sip, {})) oip \\
\leq \text{nsqn} (rt (\sigma' (\text{the (nhop} (update (rt (\sigma i)) oip \\
(osn, kno, val, Suc hops, sip, {})) \text{ oip})))) oip"
\item \text{(is "\text{?nhop_not_oip} \rightarrow \text{?oip_in_kD} \land \text{?nsqn_le_nsqn}"")}
\item \text{proof \ (rule, split update rt_split_asm) \ )}
\item \text{assume "rt (\sigma i) = update (rt (\sigma i)) oip (osn, kno, val, Suc hops, sip, {})"
\item \text{and "the (nhop (rt (\sigma i)) oip) \neq \text{ oip}"
\item \text{with pre' show "\text{?oip_in_kD} \land \text{?nsqn_le_nsqn}" by auto}
\item \text{next}
\item \text{assume rtnot: "rt (\sigma i) \neq update (rt (\sigma i)) oip (osn, kno, val, Suc hops, sip, {})"
\item \text{and notoip: ?nhop_not_oip
\item \text{with \* qinc have ?oip_in_kD
\item \text{by \ (clar simp elim! : kD_quality_increases)\ )}
\item \text{moreover with \* pre qinc rtnot notoip have ?nsqn_le_nsqn
\item \text{by simp (metis kD_nsqn_quality_increases_trans)
\item \text{ultimately show "?oip_in_kD \land ?nsqn_le_nsqn" ..}
qed 

} note update1 = this 

{ 
  fix σ σ' oip sip osn hops 
  assume pre: "∀ dip ∈ kD (rt (σ i)). the (nhop (rt (σ i)) dip) ≠ dip 
          → dip ∈ kD (rt (σ (the (nhop (rt (σ i)) dip)))) 
          ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (the (nhop (rt (σ i)) dip)))) dip" 
  and qinc: "∀ j. quality_increases (σ j) (σ' j)" 
  and *: "sip ≠ oip → oip ∈ kD (rt (σ sip)) 
          ∧ osn ≤ nsqn (rt (σ sip)) oip 
          ∧ (nsqn (rt (σ sip)) oip = osn 
              → the (dhops (rt (σ sip)) oip) ≤ hops 
              ∨ the (flag (rt (σ sip)) oip) = inv)" 
  from pre and qinc 
  have pre': "∀ dip ∈ kD (rt (σ i)). the (nhop (rt (σ i)) dip) ≠ dip 
            → dip ∈ kD (rt (σ' (the (nhop (rt (σ i)) dip)))) 
            ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (the (nhop (rt (σ i)) dip)))) dip" 
  by (rule basic) 
  have "∀ dip ∈ kD (rt (σ i)). the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})) dip) ≠ dip 
       → dip ∈ kD (rt (σ' (the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})) dip)))) 
       ∧ nsqn (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})) dip 
       ≤ nsqn (rt (σ' (the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})) dip)))) dip" 
  (is "∀ dip ∈ kD (rt (σ i)). _ → ?dip_in_kD dip ∧ ?nsqn_le_nsqn dip") 
  proof (intro ballI impI, split update_rt_split_asm) 
  fix dip 
  assume "dip ∈ kD (rt (σ i))" 
  and "the (nhop (rt (σ i)) dip) ≠ dip" 
  and "rt (σ i) = update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})" 
  with pre' show "?dip_in_kD dip ∧ ?nsqn_le_nsqn dip" by simp 
  next 
  fix dip 
  assume "dip ∈ kD (rt (σ i))" 
  and notdip: "the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})) dip) ≠ dip" 
  and rtnot: "rt (σ i) ≠ update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})" 
  show "?dip_in_kD dip ∧ ?nsqn_le_nsqn dip" 
  proof (cases "dip = oip") 
  assume "dip ≠ oip" 
  with pre' (dip ∈ kD (rt (σ i))) notdip * 
  show ?thesis by clarsimp 
  next 
  assume "dip = oip" 
  with rtnot qinc (dip ∈ kD (rt (σ i))) notdip * 
  have "?dip_in_kD dip" 
  by simp (metis kD_quality_increases) 
  moreover from (dip = oip) rtnot qinc (dip ∈ kD (rt (σ i))) notdip * 
  have "?nsqn_le_nsqn dip" by simp (metis kD_nsqn_quality_increases_trans) 
  ultimately show ?thesis . . 
  qed 
  qed 
} note update2 = this 

have "opaodv i ⪰ (S, U) onl Γ_AODV (λ(σ, _). 
  ∀ dip ∈ kD (rt (σ i)). the (nhop (rt (σ i)) dip) ≠ dip 
  → dip ∈ kD (rt (σ (the (nhop (rt (σ i)) dip)))) 
  ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (the (nhop (rt (σ i)) dip)))) dip)" 
  by (inv_cterms inv add: oseq_step_invariant_sterms [OF oquality_increases_nsqn_fresh aodv_wf oaoadv_trans] 
      onloinvariant_sterms [OF aodv_wf oaddpreRT_welldefined] 
      onloinvariant_sterms [OF aodv_wf odests_vD_inc_sqn] 
      onloinvariant_sterms [OF aodv_wf oprerr_guard] 
236
lemmas okD_unk_or_atleast_one =
  open_seq_invariant [OF kD_unk_or_atleast_one initiali_aodv,
  simplified seql_onl_swap]

lemmas ozero_seq_unk_hops_one =
  open_seq_invariant [OF zero_seq_unk_hops_one initiali_aodv,
  simplified seql_onl_swap]

lemma oreachable_fresh_okD_unk_or_atleast_one:
  fixes dip
  assumes "(σ, p) ∈ oreachable (opaodv i)
    (otherwith (op=) {i} (orecvmsg (λσ m. msg_fresh σ m
    ∧ msg_zhops m)))
    (other quality_increases {i})"
  and "dip∈kD(rt (σ i))"
  shows "π3(the (rt (σ i) dip)) = unk ∨ 1 ≤ π2(the (rt (σ i) dip))"
  (is "?P dip")
proof -
  have "∃l. l∈labels Γ AODV p" by (metis aodv_ex_label)
  with assms(1) have "∀dip∈kD (rt (σ i)). ?P dip"
    by - (drule oinvariant_weakenD [OF okD_unk_or_atleast_one [OF oaodv_trans aodv_trans]],
      auto dest!: otherwith_actionD onlD simp: seqlsimp)
  with ⟨dip∈kD(rt (σ i))⟩ show ?thesis by simp
qed

lemma oreachable_fresh_ozero_seq_unk_hops_one:
  fixes dip
  assumes "(σ, p) ∈ oreachable (opaodv i)
    (otherwith (op=) {i} (orecvmsg (λσ m. msg_fresh σ m
    ∧ msg_zhops m)))
    (other quality_increases {i})"
  and "dip∈kD(rt (σ i))"
  shows "sqn (rt (σ i) dip) = 0 −→
    sqnf (rt (σ i) dip) = unk
    ∧ the (dhops (rt (σ i) dip)) = 1
    ∧ the (nhop (rt (σ i) dip)) = dip"
  (is "?P dip")
proof -
  have "∃l. l∈labels Γ AODV p" by (metis aodv_ex_label)
  with assms(1) have "∀dip∈kD (rt (σ i)). ?P dip"
    by - (drule oinvariant_weakenD [OF ozero_seq_unk_hops_one [OF oaodv_trans aodv_trans]],
      auto dest!: onlD otherwith_actionD simp: seqlsimp)
  with ⟨dip∈kD(rt (σ i))⟩ show ?thesis by simp
qed

lemma seq_nhcop quality_increases':
  shows "opaodv i =⇒ (otherwith (op=) {i}
    (orecvmsg (λσ m. msg_fresh σ m ∧ msg_zhops m))),
    other quality_increases {i} −→)
    onl Γ AODV (λ(σ, _). ∀dip. let nhop = the (nhop (rt (σ i) dip))

\[ \text{in dip} \in vD (rt (\sigma i)) \cap vD (rt (\sigma nhip)) \]
\[ \land \ nhip \neq \text{dip} \]
\[ \rightarrow (rt (\sigma i)) \Box_{\text{dip}} (rt (\sigma nhip)) \]

(is "\_ \models (?S i, \_ \rightarrow) \_")

**proof**

have weaken:

"\[ \forall I \ Q \ R \ P. \ p \models (\text{otherwith} \ quality\text{-increases} I \ (\text{orecvmsg} Q), \text{other} quality\text{-increases} I \rightarrow P) \]
\[ \rightarrow p \models (\text{otherwith} \ (op=) I \ (\text{orecvmsg} (\lambda \sigma. Q \ \sigma m \ \land R \ \sigma m)), \text{other} quality\text{-increases} I \rightarrow P) \]
by auto
\]

\{
fix i a and \( \sigma \sigma' \::\: \text{"ip} \Rightarrow \text{state"} \)

assume a1: "\forall dip. dip \in vD(rt (\sigma i))
\land dip \in vD(rt (\sigma (\text{the} (\text{nhop} (rt (\sigma i)) dip)))
\land (\text{the} (\text{nhop} (rt (\sigma i)) dip)) \neq dip
\rightarrow rt (\sigma i) \Box_{\text{dip}} rt (\sigma (\text{the} (\text{nhop} (rt (\sigma i)) dip)))"

and ow: "?S i \ σ σ' a"

have "\forall dip. dip \in vD(rt (\sigma i))
\land dip \in vD rt (\sigma (\text{the} (\text{nhop} (rt (\sigma i)) dip)))
\land (\text{the} (\text{nhop} (rt (\sigma i)) dip)) \neq dip
\rightarrow rt (\sigma i) \Box_{\text{dip}} rt (\sigma (\text{the} (\text{nhop} (rt (\sigma i)) dip)))"

proof clarify

fix dip

assume a2: "dip \in vD(rt (\sigma i))"

and a3: "dip \in vD rt (\sigma' (\text{the} (\text{nhop} (rt (\sigma i)) dip)))"

and a4: "(\text{the} (\text{nhop} (rt (\sigma i)) dip)) \neq dip"

from ow have "\forall j. j \neq i \rightarrow j \neq \sigma' j" by auto

show "rt (\sigma i) \Box_{\text{dip}} rt (\sigma' (\text{the} (\text{nhop} (rt (\sigma i)) dip)))"

proof (cases "(\text{the} (\text{nhop} (rt (\sigma i)) dip)) = i")

assume "(\text{the} (\text{nhop} (rt (\sigma i)) dip)) = i"

with \( \text{dip} \in vD(rt (\sigma i))\) have "dip \in vD rt (\sigma (\text{the} (\text{nhop} (rt (\sigma i)) dip)))" by simp

with a1 a2 a4 have "rt (\sigma i) \Box_{\text{dip}} rt (\sigma (\text{the} (\text{nhop} (rt (\sigma i)) dip)))" by simp

with (\text{the} (\text{nhop} (rt (\sigma i)) dip)) = i have "rt (\sigma i) \Box_{\text{dip}} rt (\sigma i)" by simp

hence False by simp

thus ?thesis ..

next

assume "(\text{the} (\text{nhop} (rt (\sigma i)) dip)) \neq i"

with \( \forall j. j \neq i \rightarrow j \neq \sigma' j \)

have "\sigma (\text{the} (\text{nhop} (rt (\sigma i)) dip)) = \sigma' (\text{the} (\text{nhop} (rt (\sigma i)) dip))" by simp

with \( \text{dip} \in vD(rt (\sigma i))\) have "dip \in vD rt (\sigma' (\text{the} (\text{nhop} (rt (\sigma i)) dip)))" by simp

with a1 a2 a4 have "rt (\sigma i) \Box_{\text{dip}} rt (\sigma (\text{the} (\text{nhop} (rt (\sigma i)) dip)))" by simp

with * show ?thesis by simp

qed

qed

\}

note basic = this

\{
fix \sigma \ σ' a dip sip i

assume a1: "\forall dip. dip \in vD(rt (\sigma i))
\land dip \in vD(rt (\sigma (\text{the} (\text{nhop} (rt (\sigma i)) dip)))
\land (\text{the} (\text{nhop} (rt (\sigma i)) dip)) \neq dip
\rightarrow rt (\sigma i) \Box_{\text{dip}} rt (\sigma (\text{the} (\text{nhop} (rt (\sigma i)) dip)))"

and ow: "?S i \ σ \ σ' a"

have "\forall dip. dip \in vD(update (rt (\sigma i)) sip (0, unk, val, Suc 0, sip, \{\}))
\land dip \in vD(rt (\sigma' (\text{the} (\text{nhop} (update (rt (\sigma i)) sip (0, unk, val, Suc 0, sip, \{\})) dip)))
\land (\text{the} (\text{nhop} (update (rt (\sigma i)) sip (0, unk, val, Suc 0, sip, \{\})) dip)) \neq dip
\rightarrow update (rt (\sigma i)) sip (0, unk, val, Suc 0, sip, \{\}) \Box_{\text{dip}} rt (\sigma' (\text{the} (\text{nhop} (update (rt (\sigma i)) sip (0, unk, val, Suc 0, sip, \{\})) dip)))"

proof clarify

fix dip

assume a2: "dip \in vD (update (rt (\sigma i)) sip (0, unk, val, Suc 0, sip, \{\}))"

and a3: "dip \in vD (rt (\sigma' (\text{the} (\text{nhop} (update (rt (\sigma i)) sip (0, unk, val, Suc 0, sip, \{\})) dip))))"

and a4: "(\text{the} (\text{nhop} (update (rt (\sigma i)) sip (0, unk, val, Suc 0, sip, \{\})) dip) \neq dip"

show "update (rt (\sigma i)) sip (0, unk, val, Suc 0, sip, \{\})"
\[ \square_{rt} (\sigma' (\text{the (nhop (update (rt (\sigma i)) sip (0, unk, val, Suc 0, sip, {})) dip))) \]

proof (cases "dip = sip")
assume "dip = sip"
with \(\langle \text{the (nhop (update (rt (\sigma i)) sip (0, unk, val, Suc 0, sip, {})) dip) \neq dip} \)
have False by simp
thus \(?thesis .. .
next
assume [simp]: "dip \neq sip"
from a2 have "dip \in_vd (rt (\sigma i)) \lor dip = sip"
  by (rule \(\text{vd_update_val} \))
with \(\langle dip \neq dip \rangle \) have "dip \in_vd (rt (\sigma i))" by simp
moreover from a3 have "dip \in_vd (rt (\sigma' (\text{the (nhop (rt (\sigma i)) dip))))" by simp
moreover from a4 have "the (nhop (rt (\sigma i)) dip) \neq dip" by simp
ultimately have "rt (\sigma i) \in_dip rt (\sigma' (\text{the (nhop (rt (\sigma i)) dip))"
  using a1 ow by - (drule(1) basic, simp)
with \(\langle dip \neq dip \rangle \) show \(?thesis
  by - (erule rt_strictly_fresher_update_other, simp)
qed
qed

} note update_0_unk = this

{ fix \(\sigma a \sigma' \text{ nhop}\)
assume pre: "\(\forall dip. dip \in_vd (rt (\sigma i)) \land dip \in_vd (rt (\sigma' (\text{nhop dip}))) \land \text{nhop dip} \neq dip
\to rt (\sigma i) \in_dip rt (\sigma' (\text{nhop dip}))"\)
  and ow: "?S i \sigma \sigma' a"
have "\(\forall dip. dip \in_vd (\text{invalidate (rt (\sigma i)) (dests (\sigma i))) \land dip \in_vd (rt (\sigma' (\text{nhop dip}))) \land \text{nhop dip} \neq dip
\to rt (\sigma i) \in_dip rt (\sigma' (\text{nhop dip}))"\)
proof clarify
fix dip
assume "dip \in_vd (\text{invalidate (rt (\sigma i)) (dests (\sigma i)))"
and "dip \in_vd (rt (\sigma' (\text{nhop dip})))"
and "\(\text{nhop dip} \neq dip""
from this(1) have "dip \in_vd (rt (\sigma i))"
  by (clarsimp dest!: vD_invalidate_vD_not_dests)
moreover from ow have "\(\forall j. j \neq i \to \sigma j = \sigma' j" by auto
ultimately have "rt (\sigma i) \in_dip rt (\sigma' (\text{nhop dip}))"
  using pre \(\langle dip \in_vd (rt (\sigma' (\text{nhop dip}))) \land \text{nhop dip} \neq dip \rangle \)
  by metis
with \(\forall j. j \neq i \to \sigma j = \sigma' j\) show "rt (\sigma i) \in_dip rt (\sigma' (\text{nhop dip}))"
  by (metis rt_strictly_fresher_irefl)
qed

} note invalidate = this

{ fix \(\sigma a \sigma' \text{ dip oip osn sip hops i}\)
assume pre: "\(\forall dip. dip \in_vd (rt (\sigma i)) \land dip \in_vd (rt (\sigma (\text{the (nhop (rt (\sigma i)) dip)))) \land (\text{the (nhop (rt (\sigma i)) dip) \neq dip
\to rt (\sigma i) \in_dip rt (\sigma (\text{the (nhop (rt (\sigma i)) dip))))"\)
  and ow: "?S i \sigma \sigma' a"
and "Suc 0 \leq osn"
and a6: "\(\text{sip} \neq oip \to oip \in_kd (rt (\sigma sip))\)
\(\land dip \in_vd (rt (\sigma (\text{the (nhop (rt (\sigma i)) dip)))) \land (\text{the (nhop (rt (\sigma i)) dip) \neq dip
\to rt (\sigma i) \in_dip rt (\sigma (\text{the (nhop (rt (\sigma i)) dip))))"\)
and after: "\(\sigma' i = i || rt := \text{update (rt (\sigma i)) oip (osn, kno, val, Suc hops, sip, {}))}\)"
have "\(\forall dip. dip \in_vd (\text{update (rt (\sigma i)) oip (osn, kno, val, Suc hops, sip, {})))\)
\(\land dip \in_vd (rt (\sigma' (\text{the (nhop (update (rt (\sigma i)) oip
(\text{osn, kno, val, Suc hops, sip, {})) dip)))) \land (\text{the (nhop (update (rt (\sigma i)) oip
(\text{osn, kno, val, Suc hops, sip, {})) dip) \neq dip
\to update (rt (\sigma i)) oip (osn, kno, val, Suc hops, sip, {})) dip) \neq dip
\to update (rt (\sigma i)) oip (osn, kno, val, Suc hops, sip, {})) dip)" \)
\(\land dip \in_dip rt (\sigma' (\text{the (nhop (update (rt (\sigma i)) oip (osn, kno, val, Suc hops, sip, {})) dip))))"
proof clarify

fix dip
assume a2: "dip∈vD(update rt σ j) oip (osn, kno, val, Suc hops, sip, {})
and a3: "dip∈vD(rt σ i (the (nhop update rt σ i) oip
(osn, kno, val, Suc hops, sip, {})) dip)))"
and a4: "the (nhop update rt σ i) oip (osn, kno, val, Suc hops, sip, {})) dip) ≠ dip"
from ow have a5: "∀ j. j ≠ i → σ j = σ’ j" by auto
show "update rt σ i) oip (osn, kno, val, Suc hops, sip, {})
dip rt (σ’ (the (nhop update rt σ i) oip (osn, kno, val, Suc hops, sip, {})))" (is "?rt1 ⊏ dip ?rt2 dip")
proof (cases "?rt1 ≠ rt σ i")
assume nochange [simp]:
"update rt σ i) oip (osn, kno, val, Suc hops, sip, {}) = rt σ i)"
from after have "σ’ i = σ i" by simp
with a5 have "∀ j. j ≠ i → σ j = σ’ j" by metis
from a2 have "dip∈vD (rt σ i))" by simp
moreover from a3 have "dip∈vD(rt (σ (the (nhop rt σ i) dip))))"
using nochange and "∀ j. j = σ’ j" by clarsimp
moreover from a4 have "the (nhop (rt σ i) dip) ≠ dip" by simp
ultimately have "rt (σ i) ⊏ dip rt (σ (the (nhop rt σ i) dip)))" by simp
with a2 have "dip∈vD (rt (σ i))" by simp
moreover from a3 have "dip∈vD(rt (σ (the (nhop rt σ i) dip))))"
using nochange and "∀ j. j = σ’ j" by clarsimp
moreover from a4 have "the (nhop (rt σ i) dip) ≠ dip" by simp
ultimately have "rt (σ i) ⊏ dip rt (σ (the (nhop rt σ i) dip)))" by simp
with after have "σ’ i = σ i" by simp
from after have "∀ j. j ≠ i → σ j = σ’ j" by metis
hence "rt (σ i) ⊏ dip rt (σ’ (the (nhop rt σ i) dip)))" by simp
with a5 have "∀ j. j ≠ i → σ j = σ’ j" by simp
thus "?thesis" by simp
next
assume change: "?rt1 ≠ rt σ i)"
from after a2 have "dip∈vD (rt σ i))" by simp
moreover from a3 have "dip∈vD(rt (σ (the (nhop rt σ i) dip))))"
using nochange and "∀ j. j = σ’ j" by clarsimp
moreover from a4 have "the (nhop (rt σ i) dip) ≠ dip" by simp
ultimately have "rt (σ i) ⊏ dip rt (σ (the (nhop rt σ i) dip)))" by simp
with a2 have "dip∈vD (rt σ i))" by simp
moreover from a3 have "dip∈vD(rt σ i))" by simp
moreover from a4 have "the (nhop (rt σ i) dip) ≠ dip" by simp
ultimately have "rt (σ i) ⊏ dip rt (σ (the (nhop rt σ i) dip)))" by simp
with after and a5 have "dip∉vD (rt σ i))" by simp
moreover from a3 have "dip∉vD(rt σ i))" by simp
moreover from a4 have "the (nhop (rt σ i) dip) ≠ dip" by simp
ultimately have "rt (σ i) ⊏ dip rt (σ (the (nhop rt σ i) dip)))" by simp
with after and a5 have "dip∉vD (rt σ i))" by simp
moreover from a3 have "dip∉vD(rt σ i))" by simp
moreover from a4 have "the (nhop (rt σ i) dip) ≠ dip" by simp
ultimately have "rt (σ i) ⊏ dip rt (σ (the (nhop rt σ i) dip)))" by simp
with after have "σ’ i = σ i" by simp
from after have "∀ j. j ≠ i → σ j = σ’ j" by metis
hence "rt (σ i) ⊏ dip rt (σ’ (the (nhop rt σ i) dip)))" by simp
with a5 have "∀ j. j ≠ i → σ j = σ’ j" by simp
thus "?thesis" by simp (metis rt_strictly_fresher_update_other
rt_strictly_fresher_irefl)
next
assume "dip = oip"
with a2 have "dip∈vD (rt σ i))" by simp
moreover from a3 have "dip∈vD(rt σ i))" by simp
moreover from a4 have "the (nhop (rt σ i) dip) ≠ dip" by simp
ultimately have "rt (σ i) ⊏ dip rt (σ (the (nhop rt σ i) dip)))" by simp
with after and a5 have "dip∈vD (rt σ i))" by simp
moreover from a3 have "dip∈vD(rt σ i))" by simp
moreover from a4 have "the (nhop (rt σ i) dip) ≠ dip" by simp
ultimately have "rt (σ i) ⊏ dip rt (σ (the (nhop rt σ i) dip)))" by simp
with after have "σ’ i = σ i" by simp
from after have "∀ j. j ≠ i → σ j = σ’ j" by metis
hence "the (flag (rt (σ’ sip)) oip) = val" by simp
from "oip∈kD(rt σ sip))" have "oip∈vD(rt (σ’ sip))" by simp
hence "the (flag (rt (σ’ sip)) oip) = val" by simp
with a6 "sip ≠ oip) have "osn ≤ nsqn (rt (σ’ sip)) oip" by auto
from a3 change have "osn ≤ nsqn (rt (σ’ sip)) oip" by simp
hence "the (dhops (rt (σ’ sip)) oip) = val" by simp
with a6 have "osn < nsqn (rt (σ’ sip)) oip ∨ (osn = nsqn (rt (σ’ sip)) oip
∧ the (dhops (rt (σ’ sip)) oip) ≤ hops)"
proof
assume "oip∈vD(rt σ sip))" hence "the (flag (rt (σ sip)) oip) = val" by simp
with a6 "sip ≠ oip) have "nsqn (rt (σ sip)) oip = osn →
the (dhops (rt (σ sip)) oip) ≤ hops"
by simp
show ?thesis

proof (cases "sip = i")
  assume "sip ≠ i"
  with a5 have "σ sip = σ’ sip" by simp
  with \(\text{osn} \leq \text{nsqn (rt (σ sip)) oip}\)
  and \(\text{nsqn (rt (σ sip)) oip} = \text{osn \rightarrow the (dhops (rt (σ sip)) oip)} \leq \text{hops}\)
  show ?thesis by auto

next
  -- alternative to using sip_not_ip
  assume [simp]: "sip = i"
  have "(?rt1 = rt (σ i))"
  proof (rule update_cases_kD, simp_all)
    from \(\text{Suc 0} \leq \text{osn}\) show "0 < osn" by simp
  next
    from \(\text{oip} \in vD (rt (σ sip))\) and \(\text{osp = i}\) show "oip \in kD (rt (σ i))"
    by simp
  next
    assume "sqn (rt (σ i)) oip < osn"
    also from \(\text{osn} \leq \text{nsqn (rt (σ sip)) oip}\)
    have "... \leq \text{nsqn (rt (σ i)) oip}" by simp
    also have "... \leq \text{sqn (rt (σ i)) oip}" by simp
    finally have "sqn (rt (σ i)) oip < sqn (rt (σ i)) oip" .
    hence False by simp
    thus "(λa. if a = oip then Some (osn, kno, val, Suc hops, i, π7 (the (rt (σ i) oip)))
    else rt (σ i) a) = rt (σ i) " ..
  next
    assume "the (flag (rt (σ i)) oip) = inv"
    with \(\text{the (flag (rt (σ sip)) oip)} = \text{val}\) have False by simp
    thus "(λa. if a = oip then Some (osn, kno, val, Suc hops, i, π7 (the (rt (σ i) oip)))
    else rt (σ i) a) = rt (σ i) " ..
  next
    from \(\text{oip} \in kD (rt (σ sip))\)
    show "(λa. if a = oip then Some (the (rt (σ i) oip)) else rt (σ i) a) = rt (σ i)"
    by (auto dest!: kD_Some)
  qed

with change have False ..
thus ?thesis ..
qed

next
  assume "oip \in iD (rt (σ sip))"
  with \(\text{the (flag (rt (σ’ sip)) oip)} = \text{val}\) and a5 have "sip = i"
  by (metis f.distinct(1) iD_flag_is_inv)
  from \(\text{oip} \in iD (rt (σ sip))\) have "the (flag (rt (σ sip)) oip) = inv" by auto
  with \(\text{osp = i}\) (Suc 0 \leq osn) change after \(\text{oip} \in kD (rt (σ sip))\)
  have "nsqn (rt (σ sip)) oip < nsqn (rt (σ’ sip)) oip"
  unfolding update_def
  by (clarsimp split: option.split_asm split_if_asm)
  (auto simp: sqn_def)
  with \(\text{osn} \leq \text{nsqn (rt (σ sip)) oip}\) have "osn < nsqn (rt (σ’ sip)) oip"
by simp 
thus \( ?\)thesis ..
qed
thus \( ?\)thesis proof
  assume osnlt: "osn < nsqn (rt (σ' sip)) oip"
  from \( \langle \text{dip} \in kD(\text{rt (σ' i)}) \rangle \) and \( \langle \text{dip} = oip \rangle \) have "dip \( \in kD (?rt1)" by simp
  moreover from a3 have "dip \( \in kD(?rt2 dip)" by simp
  moreover have "nsqn ?rt1 dip < nsqn (?rt2 dip) dip" proof -
    have "nsqn ?rt1 oip = osn"
      by (simp add: \( \langle \text{dip} = oip \rangle \) nsqn_update_changed_kno_val [OF change [THEN not_sym]])
    also have "... < nsqn (rt (σ' sip)) oip" using osnlt .
    also have "... = nsqn (?rt2 oip) oip" by (simp add: change)
    finally show \( ?\)thesis
      using \( \langle \text{dip} = oip \rangle \) by simp
  qed
  ultimately show \( ?\)thesis
    by (rule rt_strictly_fresher_ltI)
next
  assume osneq: "osn = nsqn (rt (σ' sip)) oip ∧ the (dhops (rt (σ' sip)) oip) ≤ hops"
  have "oip\( \in kD(?rt1)" by simp
  moreover from a3 \( \langle \text{dip} = oip \rangle \) have "oip \( \in kD(?rt2 dip)" by simp
  moreover have "nsqn ?rt1 oip = nsqn (?rt2 dip) oip" proof -
    from osneq have "osn = nsqn (rt (σ' sip)) oip" ..
    also have "osn = nsqn ?rt1 oip"
      by (simp add: \( \langle \text{dip} = oip \rangle \) nsqn_update_changed_kno_val [OF change [THEN not_sym]])
    also have "nsqn (rt (σ' sip)) oip = nsqn (?rt2 dip) oip"
      by (simp add: change)
    finally show \( ?\)thesis .
  qed
  moreover have "\( π5(\text{the (?rt2 oip oip)}) < π5(\text{the (?rt1 oip)})\)"
  proof -
    from osneq have "the (dhops (rt (σ' sip)) oip) ≤ hops" ..
    moreover from \( oip \in vD (\text{rt (σ' sip)}) \) have "oip\( \in kD(\text{rt (σ' sip)})" by auto
    ultimately have "\( π5(\text{the (rt (σ' sip) oip)}) ≤ hops\"
      by (auto simp add: proj5_eq_dhops)
    also from change after have "hops < \( π5(\text{the (rt (σ' i) oip)})\)"
      by (simp add: proj5_eq_dhops) (metis dhops_update_changed lessI)
    finally have "\( π5(\text{the (rt (σ' sip) oip)}) < π5(\text{the (rt (σ' i) oip)})\)" .
    with change after show \( ?\)thesis by simp
  qed
  ultimately have "?rt1 ⊑oip ?rt2 oip"
    by (rule rt_strictly_fresher_eqI)
  with \( \langle \text{dip} = oip \rangle \) show \( ?\)thesis by simp
  qed
  qed
  qed
\}

note rreq_rrep_update = this

have "opaodv i \models (otherwith (op\(=\)) \{i\}) (orecvmsg (λσ m. msg_fresh σ m ∧ msg_zhops m)), other quality_increases \{i\} \rightarrow)
onl \( Γ_AODV (\lambda(σ, _). ∀\text{dip. dip} ∈ vD (\text{rt (σ i)}) ∩ vD (\text{rt (σ (the (nhop (rt (σ i)) dip))})) \∧ the (nhop (rt (σ i)) dip) \neq dip → rt (σ i) ⊑dip rt (σ (the (nhop (rt (σ i)) dip))))" proof (inv_cterms inv add: onl_oinvariant_sterms [OF aodv_wf rreq_sip [THEN weaken]]
fix \sigma \sigma' p l
assume or: "(\sigma, p) \in \text{oreachable (opaodv i) (?S i) (other quality_increases \{i\})}"
and "other quality_increases \{i\} \sigma \sigma'"
and ll: "l \in \text{labels } \Gamma_{AODV}^p" and pre: "\forall \text{dip. dip} \in \text{vD (rt (\sigma i))} \land \text{nhip} \neq \text{dip} \\
\rightarrow \text{dip} \in \text{kD (rt (\sigma nhip))} \\
\land \text{nsqn (rt (\sigma i)) dip} \leq \text{nsqn (rt (\sigma nhip)) dip}"
from this(1-2) have or': "(\sigma', p) \in \text{foreachable (opaodv i) (?S i) (other quality_increases \{i\})}
by (rule oreachable_other')
from or and ll have next_hop: "\forall \text{dip. let nhip = the (nhop (rt (\sigma i)))} \\
\text{dip} \in \text{kD (rt (\sigma nhip))} \\
\land \text{nsqn (rt (\sigma i)) dip} \leq \text{nsqn (rt (\sigma nhip)) dip}"
by (auto dest!: onl_oinvariant_weakenD [OF seq_compare_next_hop'])
from or and ll have unk_hops_one: "\forall \text{dip} \in \text{kD (rt (\sigma i)). sqn (rt (\sigma i)) dip} = 0 \\
\rightarrow \text{sqnf (rt (\sigma i)) dip} = \text{unk} \\
\land \text{the (dhops (rt (\sigma i))) dip} = 1 \\
\land \text{the (nhop (rt (\sigma i)) dip} = \text{dip}"
by (auto dest!: onl_oinvariant_weakenD [OF ozero_seq_unk_hops_one [OF oaodv_trans aodv_trans]
otherwith_actionD simp: seqlsimp)
from ⟨other quality_increases \{i\} \sigma \sigma'⟩ have "\sigma' i = \sigma i" by auto
hence "quality_increases (\sigma i) (\sigma' i)" by auto
with ⟨other quality_increases \{i\} \sigma \sigma'⟩ have "\forall j. quality_increases (\sigma j) (\sigma' j)"
by (erule otherE, metis singleton_iff)
show "\forall dip. dip \in \text{vD (rt (\sigma i))} \\
\land dip \in \text{vD (rt (\sigma' (the (nhop (rt (\sigma' i) dip)))})} \\
\land the (nhop (rt (\sigma' i)) dip) \neq dip \\
\rightarrow rt (\sigma' i) \sqsubseteq dip rt (\sigma' (the (nhop (rt (\sigma' i) dip)))"
proof clarify
fix dip
assume "dip \in \text{vD (rt (\sigma i))}"
and "dip \in \text{vD (rt (\sigma' (the (nhop (rt (\sigma' i) dip)))})}"
and "the (nhop (rt (\sigma' i)) dip) \neq dip"
from this(1) and \sigma' i = \sigma i have "dip \in \text{vD (rt (\sigma i))}"
and "dip \in \text{kD (rt (\sigma i))}"
by auto
from ⟨the (nhop (rt (\sigma' i)) dip) \neq dip⟩ and \sigma' i = \sigma i
have "the (nhop (rt (\sigma i)) dip) \neq dip" (is "\?nhip \neq _") by simp
with ⟨dip \in \text{kD (rt (\sigma i))}⟩ and next_hop
have "dip \in \text{kD (rt (\sigma (?nhip)))}"
and sqns: "sqn (rt (\sigma i)) dip \leq sqn (rt (\sigma ?nhip)) dip" 
by (auto simp: Let_def)
have "0 < sqn (rt (\sigma i)) dip"
proof (rule neq0_conv [THEN iffD1, OF notI])
assume "sqn (rt (\sigma i)) dip = 0"
with ⟨dip \in \text{kD (rt (\sigma i))}⟩ and unk_hops_one
have "?nhip = dip" by simp
with (?nhip ≠ dip) show False ..

qed

also have "... = nsqn (rt (σ i)) dip"
  by (rule vD_nsqn_sqn [OF dip∈vD(rt (σ i)), THEN sym])

also have "... ≤ nsqn (rt (σ ?nhip)) dip"
  by (rule nsqns)

also have "... ≤ sqn (rt (σ ?nhip)) dip"
  by (rule nsqn_sqn)

finally have "0 < sqn (rt (σ ?nhip)) dip".

have "rt (σ i) ⊑ dip rt (σ' ?nhip)"
proof (cases "dip∈vD(rt (σ ?nhip))")
  assume "dip∈vD(rt (σ ?nhip))"
  with pre "dip∈vD(rt (σ i))" and ?nhip ≠ dip
  have "rt (σ i) ⊑ dip rt (σ ?nhip)" by auto
  moreover from ∀. quality_increases (σ j) (σ' j)
    have "quality_increases (σ ?nhip) (σ' ?nhip)" ..
  ultimately show ?thesis
    using ⟨dip∈kD(rt (σ ?nhip))⟩ by (rule strictly_fresher_quality_increases_right)
next
  assume "dip∈kD(rt (σ ?nhip))"
  have "nsqn (rt (σ i)) dip ≤ nsqn (rt (σ ?nhip)) dip"
    by auto
  also from ⟨dip∈iD(rt (σ ?nhip))⟩
    have "... = sqn (rt (σ ?nhip)) dip - 1" ..
  also have "... < sqn (rt (σ' ?nhip)) dip"
    proof -
      from ∀. quality_increases (σ j) (σ' j)
        have "quality_increases (σ ?nhip) (σ' ?nhip)" ..
      hence "∀. sqn (rt (σ ?nhip)) ip ≤ sqn (rt (σ' ?nhip)) ip" by auto
      hence "sqn (rt (σ ?nhip)) dip ≤ sqn (rt (σ' ?nhip)) dip" ..
      with 0 < sqn (rt (σ ?nhip)) dip
        show ?thesis by auto
    qed
  also have "... = nsqn (rt (σ' ?nhip)) dip"
    proof (rule vD_nsqn_sqn [THEN sym])
      from ⟨dip∈vD(rt (σ' the (nhop (rt (σ' i)) dip)))⟩ and (σ' i = σ i)
        show "dip∈vD(rt (σ' ?nhip))" by simp
    qed
  finally have "nsqn (rt (σ i)) dip < nsqn (rt (σ' ?nhip)) dip" .

moreover from ⟨dip∈vD(rt (σ' (the (nhop (rt (σ' i)) dip))))⟩
  and (σ' i = σ i)
  have "dip∈kD(rt (σ' ?nhip))" by auto
ultimately show "rt (σ i) ⊑ dip rt (σ' ?nhip)"
  using ⟨dip∈kD(rt (σ i))⟩ by (rule rt_strictly_fresher_ltI)

thus ?thesis unfolding Let_def .

qed

lemma seq_compare_next_hop:
  fixes w
  shows "opaodv i = (otherwith (op=) {i} (orecmsg msg_fresh),
    other quality_increases {i} ⟷)
      global (λσ. ∀ dip. let nhip = the (nhop (rt (σ i)) dip)
        in dip ∈ kD(rt (σ i)) ∧ nhip ≠ dip" →
        dip ∈ kD(rt (σ nhip))
      ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ nhip)) dip)"
by (rule oinvariant_weakenE [OF seq_compare_next_hop']) (auto dest!: onlD)

lemma seq_nhop_quality_increases:
  shows "opaodv i \|= (otherwith (op=) {i} (orecvmsg (\σ m. msg_fresh σ m \& msg_zhops m)),
  other quality_increases {i} \→)
global (\σ. \∀ dip. let nhip = the (nhop (rt σ i)) dip
  in dip \∈ vD (rt σ i) \& vD (rt σ nhip) \& nhip \≠ dip
  \→ (rt σ i) \sqsubseteq dip (rt σ nhip)))"
by (rule oinvariant_weakenE [OF seq_nhop_quality_increases']) (auto dest!: onlD)
end

2.10 Routing graphs and loop freedom

theory B_Loop_Freedom
imports B_Aodv_Predicates B_Fresher
begin

Define the central theorem that relates an invariant over network states to the absence of loops in the associate routing graph.

definition
  rt_graph :: "(ip ⇒ state) ⇒ ip ⇒ ip rel"
where
"rt_graph σ = (\dip. {(ip, ip') | ip ip' dsn dsk hops pre.
  ip \≠ dip \& rt σ ip = Some (dsn, dsk, val, hops, ip', pre)})"

Given the state of a network σ, a routing graph for a given destination ip address dip abstracts the details of routing tables into nodes (ip addresses) and vertices (valid routes between ip addresses).

lemma rt_graphE [elim]:
  fixes n dip ip ip'
  assumes "(ip, ip') ∈ rt_graph σ dip"
  shows "ip \≠ dip \& (\∃ rt σ ip = r
  \& (\∃ dsn dsk hops pre. r dip = Some (dsn, dsk, val, hops, ip', pre)))"
using assms unfolding rt_graph_def by auto

lemma rt_graph_vD [dest]:
"\∀ ip ip' dip. (ip, ip') ∈ rt_graph σ dip \→ dip \∈ vD(rt σ ip)"
unfolding rt_graph_def vD_def by auto

lemma rt_graph_vD_trans [dest]:
"\∀ ip ip' dip. (ip, ip') ∈ (rt_graph σ dip) + \→ dip \∈ vD(rt σ ip)"
by (erule converse_tranclE) auto

lemma rt_graph_not_dip [dest]:
"\∀ ip ip' dip. (ip, ip') ∈ rt_graph σ dip \→ ip \≠ dip"
unfolding rt_graph_def by auto

lemma rt_graph_not_dip_trans [dest]:
"\∀ ip ip' dip. (ip, ip') ∈ (rt_graph σ dip) + \→ ip \≠ dip"
by (erule converse_tranclE) auto

NB: the property below cannot be lifted to the transitive closure

lemma rt_graph_nhip_is_nhop [dest]:
"\∀ ip ip' dip. (ip, ip') ∈ rt_graph σ dip \→ ip' = the (nhop (rt σ ip) dip)"
unfolding rt_graph_def by auto

theorem inv_to_loop_freedom:
  assumes "\∀ dip. irrefl ((rt_graph σ dip) +)"
  shows "\∀ dip. irrefl ((rt_graph σ dip) +)"

245
using assms proof (intro allI)
fix σ :: "ip ⇒ state" and dip
assume inv: "∀ip dip. let nhip = the (nhop (rt (σ ip)) dip) in dip ∈ vD(rt (σ ip)) ∩ vD(rt (σ nhip)) ∧ nhip ≠ dip → rt (σ ip) ⊑ dip rt (σ nhip)"
{ fix ip ip' assume "(ip, ip') ∈ (rt_graph σ dip)⁺" and "dip ∈ vD(rt (σ ip))" and "ip' ≠ dip" hence "rt (σ ip) ⊑ dip rt (σ ip')" proof induction
fix nhip
assume "(ip, nhip) ∈ rt_graph σ dip" and "dip ∈ vD(rt (σ nhip))" and "nhip ≠ dip"
from ⟨(ip, nhip) ∈ rt_graph σ dip⟩ have "dip ∈ vD(rt (σ ip))" and "nhip = the (nhop (rt (σ ip)) dip)" by auto
from ⟨dip ∈ vD(rt (σ ip))⟩ and ⟨dip ∈ vD(rt (σ nhip))⟩ have "dip ∈ vD(rt (σ nhip)) ∩ vD(rt (σ nhip'))" ..
with ⟨nhip = the (nhop (rt (σ ip)) dip)⟩ and ⟨nhip' = the (nhop (rt (σ ip')) dip)⟩ and inv show "rt (σ ip) ⊑ dip rt (σ nhip)" by (clarsimp simp: Let_def)
fix nhip nhip'
assume "(ip, nhip) ∈ (rt_graph σ dip)⁺" and "(nhip, nhip') ∈ rt_graph σ dip" and IH: "(dip ∈ vD(rt (σ nhip))); nhip ≠ dip ] ⇒ rt (σ ip) ⊑ dip rt (σ nhip)"
and "dip ∈ vD(rt (σ nhip'))" and "nhip' ≠ dip"
from ⟨(nhip, nhip') ∈ rt_graph σ dip⟩ have 1: "dip ∈ vD(rt (σ nhip))" and 2: "nhip ≠ dip" and "nhip' = the (nhop (rt (σ nhip)) dip)" by auto
from 1 2 have "rt (σ ip) ⊑ dip rt (σ nhip)" by (rule IH)
also have "rt (σ nhip) ⊑ dip rt (σ nhip')" proof -
from ⟨dip ∈ vD(rt (σ nhip))⟩ and ⟨dip ∈ vD(rt (σ nhip'))⟩ have "dip ∈ vD(rt (σ nhip)) ∩ vD(rt (σ nhip'))" ..
with ⟨nhip' ≠ dip⟩ and ⟨nhip' = the (nhop (rt (σ nhip)) dip)⟩ and inv show "rt (σ nhip) ⊑ dip rt (σ nhip')" by (clarsimp simp: Let_def)
qed finally show "rt (σ ip) ⊑ dip rt (σ nhip')" . note fresher = this
}
show "irrefl ((rt_graph σ dip)⁺)" unfolding irrefl_def proof (intro allI notI)
fix ip
assume "(ip, ip) ∈ (rt_graph σ dip)⁺" moreover then have "dip ∈ vD(rt (σ ip))" and "ip ≠ dip" by auto
ultimately have "rt (σ ip) ⊑ dip rt (σ ip)" by (rule fresher)
thus False by simp
qed
qed

end
2.11 Lift and transfer invariants to show loop freedom

theory B_Aodv_Loop_Freedom
imports "../../../AWN/OClosed_Transfer" "../../../AWN/Qmsg_Lifting" B_Global_Invariants B_Loop_Freedom
begin

2.11.1 Lift to parallel processes with queues

lemma par_step_no_change_on_send_or_receive: fixes σ s a σ' s' assumes "((σ, s), a, (σ', s')) ∈ oparp_sos i (oseqp_sos Γ_AODV i) (seqp_sos Γ_QMSG)
and "a ≠ τ"
shows "σ' = σ"
using assms by (rule qmsg_no_change_on_send_or_receive)

lemma par_nhop_quality_increases: shows "opaodv i |⟨⟨ i qmsg | otherwith (op=) {i} (orecvmsg (λσ. msg_fresh σ ∧ msg_zhops m)),
other quality_increases {i} →
global (λσ. ∀ dip. let nhip = the (nhop (rt (σ i)) dip)
in dip ∈ VD (rt (σ i)) ∩ VD (rt (σ nhip)) ∧ nhip ≠ dip
→ (rt (σ i)) ⊏ dip (rt (σ nhip)))"
proof (rule lift_into_qmsg [OF seq_nhop_quality_increases])
show "opaodv i |⟨⟨ i qmsg | otherwith (op=) {i} (orecvmsg (λ_. rreq_rrep_sn)),
other quality_increases {i} →
globala (λ(σ, a, σ'). quality_increases (σ i) (σ' i))"
proof (rule ostep_invariant_weakenE [OF olocal_quality_increases], simp_all)
fix t :: "(((nat ⇒ state) × (state, msg, pseqp, pseqp label) seqp), msg seq_action) transition"
assume "onll Γ_AODV (λ((σ, _, σ'), quality_increases (σ i) (σ' i))) t"
thus "quality_increases (fst (fst t) i) (fst (snd (snd t)) i)"
by (cases t) (clarsimp dest!: onllD, metis aodv_ex_label)
next
fix σ σ' a
assume "otherwith (op=) {i} (orecvmsg (λ_. rreq_rrep_sn)) σ σ' a"
thus "otherwith quality_increases (i) (orecvmsg (λ_. rreq_rrep_sn)) σ σ' a"
by (erule weaken_otherwith, auto)
qed

lemma par_rreq_rrep_nsqn_fresh_any_step:
shows "opaodv i |⟨⟨ i qmsg | otherwith (op=) {i} (orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ True) {i} →
globala (λ(σ, a, σ'). anycast (msg_fresh σ) a))"
proof -
have "opaodv i |⟨⟨ i qmsg | otherwith (op=) {i} (orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ True) {i} →
globala (λ(σ, a, σ'). anycast (msg_fresh σ) a))"
by (rule ostep_invariant_weakenE [OF olocal_quality_increases])
(auto dest!: onllD seq1D elim!: aodv_ex_labelE)
therefore "opaodv i |⟨⟨ i qmsg | otherwith (op=) {i} (orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ True) {i} →
globala (λ(σ, a, σ'). anycast (msg_fresh σ) a))"
by (rule lift_step_into_qmsg_statelessassm) simp_all
thus ?thesis by rule auto
qed

lemma rreq_rrep_nsqn_fresh_any_step:
shows "opaodv i |⟨⟨ i qmsg | otherwith (op=) {i} (orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ True) {i} →
globala (λ(σ, a, σ'). anycast (msg_fresh σ) a))"
proof -
have "opaodv i |⟨⟨ i qmsg | otherwith (op=) {i} (orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ True) {i} →
globala (λ(σ, a, σ'). anycast (msg_fresh σ) a))"
by (rule ostep_invariant_weakenE [OF rreq_nhop_quality_increases], simp_all)
fix t
assume "onll Γ AODV (λ(σ, a, _). ancast (msg_fresh σ) a) t"
thus "globala (λ(σ, a, _). ancast (msg_fresh σ) a) t"
  by (cases t) (clarsimp dest!: onllD, metis aodv_ex_label)
qed

hence "opaodv i \langle\langle i qmsg \rangle \rangle A (\lambda _ . \text{orecvmsg} (\lambda . \text{rreq}_rrep_{sn}) \sigma, \text{other} (\lambda _ . \text{True}) \{i\} \rightarrow \
globala (\lambda (\sigma, a, _). \text{ancast} (\text{msg}_\text{fresh} \sigma) a)"
  by (rule lift_step_into_qmsg_statelessassm) simp_all
thus ?thesis by rule auto
qed

2.11.2 Lift to nodes

lemma node_step_no_change_on_send_or_receive: assumes "((σ, NodeS i P R), a, (σ', NodeS i' P' R')) ∈ onode sos
(oarp sos i (oseqp sos Γ AODV i) (seqp sos Γ QMSG))"
and "a ≠ τ"
shows "σ' i = σ i"
using assms
by (cases a) (auto elim!: par_step_no_change_on_send_or_receive)

lemma node_nhops_quality_increases: shows "{ i : opaodv i \langle\langle i qmsg : R \rangle \rangle \rightarrow 
(oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
other quality_increases (i)} \rightarrow 
global (λσ. ∀ dip. let nhop = the (nhop (rt (σ i)) dip)
in dip ∈ vD (rt (σ i)) \cap vD (rt (σ nhop)) ∧ nhop ≠ dip
\rightarrow (rt (σ i)) ⊏ dip (rt (σ nhop)))"
by (rule node_lift [OF par_nhops_quality_increases]) auto

lemma node_quality_increases: "i : opaodv i \langle\langle i qmsg : R \rangle \rangle \rightarrow 
(λσ . oarrivemsg (λ . rreq_rrep_{sn}) σ, \text{other} (\lambda _ . \text{True}) \{i\} \rightarrow 
\globala (\lambda σ . (\text{castmsg} (\text{msg}_\text{fresh} \sigma) a))"
by (rule node_lift_step_statelessassm [OF par_rreq_rrep_sn_quality_increases]) simp

lemma node_rreq_rrep_nsqn_fresh_any_step: shows "i : opaodv i \langle\langle i qmsg : R \rangle \rangle \rightarrow 
(λσ . oarrivemsg (λ . rreq_rrep_{sn}) σ, \text{other} (\lambda _ . \text{True}) \{i\} \rightarrow 
\globala (λσ, a, _): \text{castmsg} (\text{msg}_\text{fresh} σ) a)"
by (rule node_lift_anycast_statelessassm [OF par_rreq_rrep_nsqn_fresh_any_step])
lemma node_anycast_msg_zhops:
shows "(i : opaodv i ⟨⟨i qmsg : R⟩⟩o ⊆ A)
(λσ _, oarrivemsg (λσ msg_zhops a)" by (rule node_lift_anycast_statelessasm [OF par_anycast_msg_zhops])

lemma node_silent_change_only:
shows "(i : opaodv i ⟨⟨i qmsg : R⟩⟩o ⊆ A)
globala (λσ _, a, σ'). a ≠ τ i = σ i"
proof (rule ostep_invariantI, simp (no_asm), rule impI)
fix σ ζ a σ' ζ'
assume or: "(σ, ζ) ∈ orreachable ((i : opaodv i ⟨⟨i qmsg : R⟩⟩o)
(λσ _, oarrivemsg (λσ msg_zhops a)"
and tr: "((σ, ζ), a, (σ', ζ')) ∈ trans ((i : opaodv i ⟨⟨i qmsg : R⟩⟩o)" and "a ≠ τ n" from or obtain p R where "ζ = NodeS i p R"
by - (drule node_net_state, metis)
with tr have "((σ, i = σ i) ∈ onode_sos (oparp_sos i (trans (opaodv i)) (trans qmsg)))" by simp thus "σ' = σ i" using (a ≠ τ n)
by (cases rule: onode_sos_cases)
(auto elim: qmsg_no_change_on_send_or_receive)
qed

2.11.3 Lift to partial networks

lemma arrive_rreq_rrep_msg_zhops [simp]:
assumes "oarrivemsg (λσ m. msg_fresh σ m ∧ P σ m) σ m"
shows "oarrivemsg (λσ msg_zhops m) σ m"
using assms by (cases (m m) auto)

lemma opnet_nhop_quality_increases:
shows "opnet (λi. opaodv i ⟨⟨i qmsg : R⟩⟩o ⊆ A)
(otherwise (opm) (net_tree_ips p)
(oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
other_quality_increases (net_tree_ips p) →)
global (λσ. ∀ i∈net_tree_ips p. ∀ dip.
let nhip = the (nhop (rt (σ i)) dip)
in dip ∈ vD (rt (σ i)) \ vD (rt (σ nhip)) ∧ nhip ≠ dip
→ (rt (σ i)) ⊑ dip (rt (σ nhip)))"
proof (rule pnet_lift [OF node_nhop_quality_increases])
fix i R
have "(i : opaodv i ⟨⟨i qmsg : R⟩⟩o ⊆ A)
(λσ _, oarrivemsg (λσ msg_zhops m) a)" proof (rule ostep_invariantI, simp (no_asm))
fix σ s a σ' s'
assume or: "(σ, s) ∈ orreachable ((i : opaodv i ⟨⟨i qmsg : R⟩⟩o)
(λσ _, oarrivemsg (λσ msg_zhops m) σ)"
and tr: "((σ, s), a, (σ', s')) ∈ trans ((i : opaodv i ⟨⟨i qmsg : R⟩⟩o)" and am: "oarrivemsg (λσ msg_zhops m) σ a"
from or and am have "castmsg (msg_fresh σ) a" by (auto dest!: ostep_invariantD [OF node_rreq_rrep_msg_zhops])
moreover from or and am have "castmsg (msg_zhops m) a" by (auto dest!: ostep_invariantD [OF node_anycast_msg_zhops])
ultimately show "castmsg (λσ m. msg_fresh σ m ∧ msg_zhops m) a" by (case_tac a) auto
qed
thus "(i : opaodv i ⟨⟨i qmsg : R⟩⟩o ⊆ A)
(λσ _, oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m) σ, check
other quality_increases \{i} \rightarrow) \text{globa} (\lambda (\sigma, a, \_).
\text{castmsg} (\lambda m. \text{msg\_fresh} \sigma m \land \text{msg\_zhops} m) a)
by rule auto

next
fix i R
show "(i : \text{opadv} i \langle i \text{\_ qmsg : R} \rangle_o \models_A
(\lambda \sigma \_. \text{oarrivemsg} (\lambda \sigma m. \text{msg\_fresh} \sigma m \land \text{msg\_zhops} m) \sigma,
other quality_increases \{i\} \rightarrow) \text{globa} (\lambda (\sigma, a, \_).
a \neq \tau \land (\forall i d. a \neq i: \text{deliver}(d)) \rightarrow \sigma i = \sigma' i)"
by (rule ostep\_invariant\_weakenE [OF node\_silent\_change\_only]) auto

2.11.4 Lift to closed networks

lemma onet\_nhop\_quality\_increases:
shows "\text{o\text{closed} (opnet (\lambda i. \text{opadv} i \langle i \text{\_ qmsg} \rangle p)}
\models (\lambda \_. \_. \text{True}, \text{other quality\_increases} (\text{net\_tree\_ips} p) \rightarrow)
\text{global} (\lambda \sigma. \forall i \in \text{net\_tree\_ips} p. \forall dip.
\text{let nhip} = \text{the} (\text{nhop} (\text{rt} (\sigma i)) dip)
in dip \in \text{vD} (\text{rt} (\sigma i)) \cap \text{vD} (\text{rt} (\sigma nhip)) \land nhip \neq dip
\rightarrow (\text{rt} (\sigma i)) \sqsubseteq dip (\text{rt} (\sigma nhip)))"
(is "\models (\_, \_U \rightarrow) \text{?inv}\"
proof (rule inclosed\_closed)
from opnet\_nhop\_quality\_increases
show "opnet (\lambda i. \text{opadv} i \langle i \text{\_ qmsg} \rangle p)
\models (\text{o\text{otherwith} (op=} (\text{net\_tree\_ips} p) \text{\text{in\text{closed}, \_U \rightarrow}}) \text{?inv}"
proof (rule oinvariant\_weakenE)
fix \sigma \sigma' :: "\text{ip} \Rightarrow \text{state}" and a :: "\text{msg node\_action}"
assume "\text{o\text{otherwith} (op=} (\text{net\_tree\_ips} p) \text{\text{in\text{closed} \sigma \sigma' a}}"
thus "\text{otherwith} (op=} (\text{net\_tree\_ips} p)
\text{\text{oarrivemsg} (\lambda \sigma m. \text{msg\_fresh} \sigma m \land \text{msg\_zhops} m) \sigma \sigma' a}"
proof (rule otherwith\_EI)
fix \sigma :: "\text{ip} \Rightarrow \text{state}" and a :: "\text{msg node\_action}"
assume "\text{\text{in\text{closed} \sigma a}}"
thus "\text{\text{oarrivemsg} (\lambda \sigma m. \text{msg\_fresh} \sigma m \land \text{msg\_zhops} m) \sigma a}"
proof (cases a)
fix ii ni ms
assume "a = ii\text{-ni: arrive}(ms)"
moreover with \text{\text{in\text{closed} \sigma a} obtai} d di where "ms = \text{\text{newpkt}(d, di)}"
by (cases ms) auto
ultimately show \text{?thesis} by simp
qed simp\_all
qed

2.11.5 Transfer into the standard model

interpretation aodv\_openproc: openproc paodv \text{opaodv} id
where "aodv\_openproc\_init\text{missing} = \text{init\text{missing}}"
proof -
show "openproc paodv \text{opaodv} id"
proof unfold\_locales
fix i :: ip
have "\{ (\sigma, \zeta). (\sigma i, \zeta) \in \sigma\text{AODV} i \land (\forall j. j \neq i \rightarrow \sigma j \in \text{fst'} \sigma\text{AODV} j) \} \subseteq \sigma\text{AODV}'"
unfolding \sigma\text{AODV\_def} \sigma\text{AODV'}\_def
proof (rule equality\_D1)

show "\( f \cdot \{(\sigma, \zeta) . \ (\sigma \ i, \zeta) \in \{(f \ i, p)\} \ \land \ (\forall j . \ j \neq i \rightarrow \sigma \ j \in \text{fst} \ ' \{(f \ j, p)\}) = \{(f, p)\}\)"

by (rule set_eqI) auto

qed

thus "\( \{ (\sigma, \zeta) \ / \sigma \in \text{init} (paodv i) \land (\sigma \ i, \zeta) = \text{id} \ s \land (\forall j . \ j \neq i \rightarrow \sigma \ j \in (\text{fst} \ o \ \text{id} \ ' \ \text{init} (paodv j)) \} \subseteq \text{init} (opaodv i)\)"

by simp

next

show "\( \forall j. \ \text{init} \ (paodv j) \neq \{\}\)"

unfolding \(\sigma_{AODV\_def}\) by simp

next

fix \(i\) \(s\) a \(s'\) \(\sigma\) \(\sigma'\)

assume "\(\sigma \ i = \text{fst} \ (\text{id} \ s)\)"

and "\(\sigma' \ i = \text{fst} \ (\text{id} \ s')\)"

and "\((\sigma, a, \sigma') \in \text{trans} \ (paodv i)\)"

then obtain \(q\) \(q'\) where "\(s = (\sigma \ i, q)\)"

and "\(s' = (\sigma' \ i, q')\)"

and "\(((\sigma \ i, q), a, (\sigma' \ i, q')) \in \text{trans} \ (paodv i)\)"

by (cases \(s\), cases \(s'\)) auto

from this(3) have "\(((\sigma, q), a, (\sigma', q')) \in \text{trans} \ (opaodv i)\)"

by simp (rule open_seqp_action [OF aodv_wf])

with \(\langle s = (\sigma \ i, q)\rangle\) and \(\langle s' = (\sigma' \ i, q')\rangle\)

show "\(((\sigma, \text{snd} \ (\text{id} \ s)), a, (\sigma', \text{snd} \ (\text{id} \ s'))) \in \text{trans} \ (opaodv i)\)"

by simp

qed

then interpret op: openproc paodv opaodv id .

have [simpl]: "\(\forall i. \ (\text{SOME} \ x. \ x \in (\text{fst} \ o \ \text{id} \ ' \ \text{init} (paodv i)) = \text{aodv_init} \ i\)"

unfolding \(\sigma_{AODV\_def}\) by simp

hence "\(\forall i. \ \text{openproc.initmissing} \ (\lambda i. \ \text{paodv} \ i \langle\langle qmsg) (\lambda (p, q). (\text{fst} \ (\text{id} \ p), \text{snd} \ (\text{id} \ p), q)) P \sigma \ = \text{initmissing} \ P \sigma\)"

by simp (rule openproc.initmissing_def op someinit_def initmissing_def)

thus "\(\text{openproc.initmissing} \ (\lambda i. \ \text{paodv} \ i \langle\langle qmsg) (\lambda (p, q). (\text{fst} \ (\text{id} \ p), \text{snd} \ (\text{id} \ p), q)) = \text{initmissing}\)" ..

qed

interpretation aodv_openproc_par_qmsg: openproc_parq paodv opaodv id qmsg

where "\(\text{aodv.openproc_par_qmsg.netglobal} = \text{netglobal}\)"

and "\(\text{aodv.openproc_par_qmsg.initmissing} = \text{initmissing}\)"

proof -

show "\(\text{openproc_parq} \ \text{paodv} \ \text{opaodv} \ \text{id} \ \text{qmsg}\)"

by (unfold_locales) simp

then interpret opq: openproc_parq paodv opaodv id qmsg .

have im: "\(\forall \sigma. \ \text{openproc.initmissing} \ (\lambda i. \ \text{paodv} \ i \langle\langle qmsg) (\lambda (p, q). (\text{fst} \ (\text{id} \ p), \text{snd} \ (\text{id} \ p), q)) \sigma \ = \text{initmissing} \ \sigma\)"

unfolding opq.initmissing_def opq someinit_def initmissing_def

unfolding \(\sigma_{AODV\_def}\) \(\sigma_{QMSG\_def}\) by (clarsimp cong: option.case_cong)

thus "\(\text{openproc.initmissing} \ (\lambda i. \ \text{paodv} \ i \langle\langle qmsg) (\lambda (p, q). (\text{fst} \ (\text{id} \ p), \text{snd} \ (\text{id} \ p), q)) \ P \sigma \ = \text{netglobal} \ P \ \sigma\)"

unfolding opq netglobal_def netglobal_def opq initmissing_def initmissing_def opq someinit_def

unfolding \(\sigma_{AODV\_def}\) \(\sigma_{QMSG\_def}\)

by (clarsimp cong: option.case_cong)

simp del: One_nat_def

simp add: fst initmissing netmap_default aodv init netlift

[symmetric, unfolded initmissing_def]

thus "\(\text{openproc.netglobal} \ (\lambda i. \ \text{paodv} \ i \langle\langle qmsg) (\lambda (p, q). (\text{fst} \ (\text{id} \ p), \text{snd} \ (\text{id} \ p), q)) \ = \text{netglobal}\)"

by auto

qed

lemma net_nhup_quality_increases:

assumes "\(\text{wf_net_tree} \ n\)"
shows "closed (pnet (λi. paodv i ⟨⟨ qmsg⟩⟩ n) \models netglobal
(λσ. \forall i dip. let nhip = the (nhop (rt (σ i)) dip)
in dip ∈ vD (rt (σ i)) \cap vD (rt (σ nhip)) \land nhip \neq dip
→ (rt (σ i)) \sqsubseteq dip (rt (σ nhip)))"
(is "\models netglobal (λσ. \forall i. ?inv σ i")

proof -
from ⟨wf_net_tree n⟩
  have proto: "closed (pnet (λi. paodv i ⟨⟨ qmsg⟩⟩ n) \models netglobal (λσ. \forall i\in net_tree_ips n. \forall dip.
    let nhip = the (nhop (rt (σ i)) dip)
in dip ∈ vD (rt (σ i)) \cap vD (rt (σ nhip)) \land nhip \neq dip
→ (rt (σ i)) \sqsubseteq dip (rt (σ nhip)))"
  by (rule aodv_openproc_par_qmsg.close_opnet [OF _ onet_nhop_quality_increases])
show ?thesis
unfolding invariant_def opnet_sos.opnet_tau1
proof (rule, simp only: aodv_openproc_par_qmsg.netglobalsimp
  fst_initmissing_netgmap_pair_fst, rule allI)
  fix σ i
  assume sr: "σ ∈ reachable (closed (pnet (λi. paodv i ⟨⟨ qmsg⟩⟩ n)) TT"
  hence "\forall i\in net_tree_ips n. ?inv (fst (initmissing (netgmap fst σ))) i"
    by - (drule invariantD [OF proto],
      simp only: aodv_openproc_par_qmsg.netglobalsimp
      fst_initmissing_netgmap_pair_fst)
  thus "?inv (fst (initmissing (netgmap fst σ))) i"
proof (cases "i\in net_tree_ips n")
  assume "i\notin net_tree_ips n"
  from sr have "σ ∈ reachable (pnet (λi. paodv i ⟨⟨ qmsg⟩⟩ n) TT" ..
  hence "net_ips σ = net_tree_ips n" ..
  with "i\notin net_tree_ips n" have "i\notin net_ips σ" by simp
  hence "(fst (initmissing (netgmap fst σ))) i = aodv_init i"
    by simp
  thus ?thesis by simp
qed metis

qed

2.11.6 Loop freedom of AODV

theorem aodv_loop_freedom:
assumes "wf_net_tree n"
shows "closed (pnet (λi. paodv i ⟨⟨ qmsg⟩⟩ n) \models netglobal (λσ. \forall dip. irrefl ((rt_graph σ dip)⁺))"
using assms by (rule aodv_openproc_par_qmsg.netglobal_weakenE
  [OF net_nhop_quality_increases inv_to_loop_freedom])
end
Chapter 3

Variant C: From Groupcast to Broadcast

Explanation [4, §10.4]: A node maintains a set of ‘precursor nodes’ for each of its valid routes. If the link to a route’s next hop is lost, an error message is groupcast to the associated precursor nodes. The idea is to reduce the number of messages received and handled. However, precursor lists are incomplete. They are updated only when a RREP message is sent. This can lead to packet loss. A possible solution is to abandon precursors and to replace every groupcast by a broadcast. At first glance this strategy seems to need more bandwidth, but this is not the case. Sending error messages to a set of precursors is implemented at the link layer by broadcasting the message anyway; a node receiving such a message then checks the header to determine whether it is one of the intended recipients. Instead of analysing the header only, a node can just as well read the message and decide whether the information contained in the message is of use. To be more precise: an error message is useful for a node if the node has established a route to one of the nodes listed in the message, and the next hop to a listed node is the sender of the error message. In case a node finds useful information inside the message, it should update its routing table and distribute another error message.

3.1 Predicates and functions used in the AODV model

theory C_Aodv_Data
imports C_Gtobcast
begin

3.1.1 Sequence Numbers

Sequence numbers approximate the relative freshness of routing information.

definition inc :: "sqn ⇒ sqn"
  where "inc sn ≡ if sn = 0 then sn else sn + 1"

lemma less_than_inc [simp]: "x ≤ inc x"
  unfolding inc_def by simp

lemma inc_minus_suc_0 [simp]:
  "inc x - Suc 0 = x"
  unfolding inc_def by simp

lemma inc_never_one' [simp, intro]: "inc x ≠ Suc 0"
  unfolding inc_def by simp

lemma inc_never_one [simp, intro]: "inc x ≠ 1"
  by simp

3.1.2 Modelling Routes

A route is a 5-tuple, (dsn, dsk, flag, hops, nhip) where dsn is the ‘destination sequence number’, dsk is the ‘destination-sequence-number status’, flag is the route status, hops is the number of hops to the destination, and nhip is the next hop toward the destination. In this variant, the set of ‘precursor nodes’ is not modelled.

type_synonym r = "sqn × k × f × nat × ip"
definition proj2 :: \( r \rightarrow sqn \) ("\( \pi_2 \)"
where "\( \pi_2 \equiv \lambda (dsn, _, _, _, _). dsn \)"
definition proj3 :: \( r \rightarrow k \) ("\( \pi_3 \)"
where "\( \pi_3 \equiv \lambda (_, dsk, _, _, _). dsk \)"
definition proj4 :: \( r \rightarrow f \) ("\( \pi_4 \)"
where "\( \pi_4 \equiv \lambda (_, _, flag, _, _). flag \)"
definition proj5 :: \( r \rightarrow nat \) ("\( \pi_5 \)"
where "\( \pi_5 \equiv \lambda (_, _, _, hops, _). hops \)"
definition proj6 :: \( r \rightarrow ip \) ("\( \pi_6 \)"
where "\( \pi_6 \equiv \lambda (_, _, _, _, nhip). nhip \)"

lemma projs [simp]:
"\( \pi_2 (dsn, dsk, flag, hops, nhip) = dsn \)"
"\( \pi_3 (dsn, dsk, flag, hops, nhip) = dsk \)"
"\( \pi_4 (dsn, dsk, flag, hops, nhip) = flag \)"
"\( \pi_5 (dsn, dsk, flag, hops, nhip) = hops \)"
"\( \pi_6 (dsn, dsk, flag, hops, nhip) = nhip \)"
by (clarsimp simp: proj2_def proj3_def proj4_def proj5_def proj6_def)+

lemma proj3_pred [intro]: "\( P \, kno; \, P \, unk \) \implies P (\( \pi_3 \, x \))"
by (rule k.induct)

lemma proj4_pred [intro]: "\( P \, val; \, P \, inv \) \implies P (\( \pi_4 \, x \))"
by (rule f.induct)

lemma proj6_pair_snd [simp]:
fixes dsn' r
shows "\( \pi_6 (dsn', snd (r)) = \pi_6 (r) \)"
by (cases r) simp

3.1.3 Routing Tables
Routing tables map ip addresses to route entries.
type synonym rt = "ip \rightarrow r"
syntax
"_Sigma_route" :: "rt \Rightarrow ip \Rightarrow r" ("\sigma_{route}'(\_, \_')")
translations
"\sigma_{route}(rt, dip)" \Rightarrow "rt dip"
definition sqn :: "rt \Rightarrow ip \Rightarrow sqn"
where "sqn rt dip \equiv case \sigma_{route}(rt, dip) of Some r \Rightarrow \pi_2 (r) \mid None \Rightarrow 0"
definition sqnf :: "rt \Rightarrow ip \Rightarrow k"
where "sqnf rt dip \equiv case \sigma_{route}(rt, dip) of Some r \Rightarrow \pi_3 (r) \mid None \Rightarrow unk"
abbreviation flag :: "rt \Rightarrow ip \Rightarrow f"
where "flag rt dip \equiv map_option \pi_4 (\sigma_{route}(rt, dip))"
abbreviation dhops :: "rt \Rightarrow ip \Rightarrow nat"
where "dhops rt dip \equiv map_option \pi_5 (\sigma_{route}(rt, dip))"
abbreviation nhop :: "rt \Rightarrow ip \Rightarrow ip"
where "nhop rt dip \equiv map_option \pi_6 (\sigma_{route}(rt, dip))"
definition vD :: "rt \Rightarrow ip set"
where "vD rt \equiv \{ dip. flag rt dip = Some val \)"

254
definition \( iD :: \text{"rt} \Rightarrow \text{ip set} \) \\
where \( "iD \ rt \equiv \{\text{dip. flag rt dip} = \text{Some inv}\}" \)

definition \( kD :: \text{"rt} \Rightarrow \text{ip set} \) \\
where \( "kD \ rt \equiv \{\text{dip. rt dip} \neq \text{None}\}" \)

lemma \( kD\_is\_vD\_and\_iD: "kD \ rt = vD \ rt \cup iD \ rt" \) \\
unfolding \( kD\_def \ vD\_def \ iD\_def \) by auto

lemma \( vD\_iD\_gives\_kD [simp]: "\forall \text{ip rt. ip} \in \text{vD} \ rt \Rightarrow \text{ip} \in \text{kD} \ rt" \) \\
"\forall \text{ip rt. ip} \in \text{iD} \ rt \Rightarrow \text{ip} \in \text{kD} \ rt" \\
unfolding \( kD\_is\_vD\_and\_iD \) by simp_all

lemma \( kD\_Some [dest]: \) \\
fixes dip rt \\
assumes "dip \in kD \ rt" \\
shows "\exists dsn \ dsk \ flag \ hops \ nhip. \sigma\_route(rt, dip) = \text{Some (dsn, dsk, flag, hops, nhip)}" \\
using assms unfolding \( kD\_def \) by simp

lemma \( kD\_None [dest]: \) \\
fixes dip rt \\
assumes "dip \notin kD \ rt" \\
shows "\sigma\_route(rt, dip) = \text{None}" \\
using assms unfolding \( kD\_def \) by (metis (mono_tags) mem_Collect_eq)

lemma \( vD\_Some [dest]: \) \\
fixes dip rt \\
assumes "dip \in vD \ rt" \\
shows "\exists dsn \ dsk \ hops \ nhip. \sigma\_route(rt, dip) = \text{Some (dsn, dsk, val, hops, nhip)}" \\
using assms unfolding \( vD\_def \) by simp

lemma \( vD\_empty [simp]: "vD \text{ Map.empty} = \{\}" \) \\
unfolding \( vD\_def \) by simp

lemma \( iD\_Some [dest]: \) \\
fixes dip rt \\
assumes "dip \in iD \ rt" \\
shows "\exists dsn \ dsk \ hops \ nhip. \sigma\_route(rt, dip) = \text{Some (dsn, dsk, inv, hops, nhip)}" \\
using assms unfolding \( iD\_def \) by simp

lemma \( \text{val\_is\_vD [elim]}: \) \\
fixes ip rt \\
assumes "ip\in kD(rt)" \\
and "the (flag rt ip) = val" \\
shows "ip\in vD(rt)" \\
using assms unfolding \( vD\_def \) by auto

lemma \( \text{inv\_is\_iD [elim]}: \) \\
fixes ip rt \\
assumes "ip\in kD(rt)" \\
and "the (flag rt ip) = inv" \\
shows "ip\in iD(rt)" \\
using assms unfolding \( iD\_def \) by auto

lemma \( \text{iD\_flag\_is\_inv [elim, simp]}: \) \\
fixes ip rt \\
assumes "ip\in iD(rt)" \\
shows "the (flag rt ip) = inv" \\
proof -
from \(\forall ip \in \text{id}(rt)\) have "ip \in \text{id}(rt)" by auto
with assms show ?thesis unfolding id_def by auto
qed

lemma \text{kd\_but\_not\_vd\_is\_id} [elim]:
  fixes ip rt
  assumes "ip \in \text{kd}(rt)"
  and "ip \notin \text{vd}(rt)"
  shows "ip \in \text{id}(rt)"
proof
  from \(\forall ip \in \text{kd}(rt)\) obtain dsn dsk f hops nhop
  where rtip: "rt ip = Some (dsn, dsk, f, hops, nhop)"
    by (metis kD_Some)
  from \(\forall ip \notin \text{vd}(rt)\) have "f \neq \text{val}"
proof (rule contrapos_nn)
  assume "f = \text{val}"
  with rtip have "the (flag rt ip) = \text{val}"
    by simp
  with \(\forall ip \in \text{kd}(rt)\) show "\(\forall ip \notin \text{vd}(rt)\)" ..
  qed
  with rtip have "the (flag rt ip)= \text{inv}" by simp
  with \(\forall ip \in \text{kd}(rt)\) show "\(\forall ip \in \text{id}(rt)\)" ..
  qed

lemma \text{vd\_or\_id} [elim]:
  fixes ip rt
  assumes "ip \in \text{kd}(rt)"
  and "ip \in \text{vd}(rt) = \Rightarrow P rt ip"
  and "ip \in \text{id}(rt) = \Rightarrow P rt ip"
  shows "P rt ip"
proof
  from \(\forall ip \in \text{kd}(rt)\) have "ip \in \text{vd}(rt) \cup \text{id}(rt)"
    by (simp add: kD_is_vD_and_id)
  thus ?thesis by (auto elim: assms(2-3))
  qed

lemma \text{proj5\_eq\_dhop}:
"\(\forall dip rt. dip \in \text{kd}(rt) = \Rightarrow \pi_5(\text{the}(rt dip)) = \text{the}(\text{dhop}(rt dip))\)"
unfolding sqn_def by (drule kD_Some) clarsimp

lemma \text{proj4\_eq\_flag}:
"\(\forall dip rt. dip \in \text{kd}(rt) = \Rightarrow \pi_4(\text{the}(rt dip)) = \text{the}(\text{flag}(rt dip))\)"
unfolding sqn_def by (drule kD_Some) clarsimp

lemma \text{proj2\_eq\_sqn}:
"\(\forall dip rt. dip \in \text{kd}(rt) = \Rightarrow \pi_2(\text{the}(rt dip)) = \text{sqn}(rt dip)\)"
unfolding sqn_def by (drule kD_Some) clarsimp

lemma \text{kD\_sqnf\_is\_proj3} [simp]:
"\(\forall ip rt. ip \in \text{kd}(rt) = \Rightarrow \text{sqnf}(rt ip) = \pi_3(\text{the}(rt ip))\)"
unfolding sqnf_def by auto

lemma \text{vd\_flag\_val} [simp]:
"\(\forall dip rt. dip \in \text{vd}(rt) = \Rightarrow \text{the}(\text{flag}(rt dip)) = \text{val}\)"
unfolding vd_def by clarsimp

lemma \text{kD\_update} [simp]:
"\(\forall rt nip v. \text{kd}(rt(nip \mapsto v)) = \text{insert}(nip \text{kd}(rt))\)"
unfolding kd_def by auto

lemma \text{kD\_empty} [simp]: "\text{kd Map.empty} = \{\}\"
unfolding kd_def by simp

lemma \text{ip\_equal\_or\_known} [elim]:
  fixes rt ip ip'
  assumes "ip = ip' \lor ip \in \text{kd}(rt)"
  and "ip = ip' = \Rightarrow P rt ip ip'"
  and "[ ip \neq ip'; ip \in \text{kd}(rt)] = \Rightarrow P rt ip ip'"
3.1.4 Updating Routing Tables

Routing table entries are modified through explicit functions. The properties of these functions are important in invariant proofs.

Updating route entries

**Lemma in_kD_case [simp]:**
fixes dip rt
assumes "dip ∈ kD(rt)"
shows "(case rt dip of None ⇒ en | Some r ⇒ es r) = es (the (rt dip))"
using assms [THEN kD_Some] by auto

**Lemma not_in_kD_case [simp]:**
fixes dip rt
assumes "dip /∈ kD(rt)"
shows "(case rt dip of None ⇒ en | Some r ⇒ es r) = en"
using assms [THEN kD_None] by auto

**Lemma rt_Some_sqn [dest]:**
fixes rt and ip dsn dsk flag hops nhip
assumes "rt ip = Some (dsn, dsk, flag, hops, nhip)"
shows "sqn rt ip = dsn"
unfolding sqn_def using assms by simp

**Definition update_arg_wf :: "r ⇒ bool"**
where "update_arg_wf r ≡ π₄(r) = val ∧
((π₂(r) = 0) = (π₃(r) = unk) ∧
(π₃(r) = unk −→ π₅(r) = 1))"

**Lemma update_arg_wf_gives_cases:**
"∀ r. update_arg_wf r =⇒ (π₂(r) = 0) = (π₃(r) = unk)"
unfolding update_arg_wf_def by simp

**Lemma update_arg_wf_tuples [simp]:**
"∀ nhip. update_arg_wf (0, unk, val, Suc 0, nhip)"
"∀ n hops nhip. update_arg_wf (Suc n, kno, val, hops, nhip)"
unfolding update_arg_wf_def by auto

**Lemma update_arg_wf_tuples' [elim]:**
"∀ n hops nhip. Suc 0 ≤ n =⇒ update_arg_wf (n, kno, val, hops, nhip)"
unfolding update_arg_wf_def by auto

**Lemma wf_r_cases [intro]:**
fixes P r
assumes "update_arg_wf r"
and c1: "∀ nhip. P (0, unk, val, Suc 0, nhip)"
and c2: "∀ dsn hops nhip. dsn > 0 =⇒ P (dsn, kno, val, hops, nhip)"
shows "P r"
proof -
obtain dsn dsk flag hops nhip
where *: "r = (dsn, dsk, flag, hops, nhip)" by (cases r)
with "update_arg_wf r": have wf1: "flag = val"
and wf2: "(dsn = 0) = (dsk = unk)"
and wf3: "dsk = unk \rightarrow (hops = 1)"

unfolding update_arg_wf_def by auto
have "P (dsn, dsk, flag, hops, nhip)"
proof (cases dsk)
  assume "dsk = unk"
  moreover with wf2 wf3 have "dsn = 0" and "hops = Suc 0" by auto
  ultimately show \(?thesis using \langle flag = val \rangle\) by simp (rule c1)
next
  assume "dsk = kno"
  moreover with wf2 have "dsn > 0" by simp
  ultimately show \(?thesis using \langle flag = val \rangle\) by simp (rule c2)
qed

with * show "P r" by simp

definition update :: "rt \Rightarrow ip \Rightarrow r \Rightarrow rt"
where
  "update rt ip r \equiv case \sigma _route(rt, ip) of
    None \Rightarrow rt (ip \mapsto r)
  | Some s \Rightarrow if \pi _2(s) < \pi _2(r) then rt (ip \mapsto r)
    else if \pi _2(s) = \pi _2(r) \land (\pi _5(s) > \pi _5(r) \lor \pi _4(s) = inv)
    then rt (ip \mapsto r)
    else if \pi _3(r) = unk
    then rt (ip \mapsto (\pi _2(s), snd (r)))
    else rt (ip \mapsto s)"

lemma update_simps [simp]:
  fixes r s nrt nr' ns rt ip
  defines "s \equiv the \sigma _route(rt, ip)"
  and "nr' \equiv (\pi _2(s), \pi _3(r), \pi _4(r), \pi _5(r), \pi _6(r))"
  shows
    "[i\ pi \notin kD(rt)]\ \Rightarrow update rt ip r = rt (ip \mapsto r)"
  "[i\ ip \in kD(rt); sqn rt ip < \pi _2(r)]\ \Rightarrow update rt ip r = rt (ip \mapsto r)"
  "[i\ ip \in kD(rt); sqn rt ip = \pi _2(r);
    the (dhops rt ip) > \pi _5(r)]\ \Rightarrow update rt ip r = rt (ip \mapsto r)"
  "[i\ ip \in kD(rt); sqn rt ip = \pi _2(r);
    flag rt ip = Some inv]\ \Rightarrow update rt ip r = rt (ip \mapsto r)"
  "[i\ ip \in kD(rt); \pi _3(r) = unk; (\pi _2(r) = 0) = (\pi _3(r) = unk)]\ \Rightarrow update rt ip r = rt (ip \mapsto nr')"
  "[i\ ip \in kD(rt); sqn rt ip \geq \pi _2(r); \pi _3(r) = kno;
    sqn rt ip = \pi _2(r) \Rightarrow the (dhops rt ip) \leq \pi _5(r) \land the (flag rt ip) = val ]\ \Rightarrow update rt ip r = rt (ip \mapsto s)"

proof -
  assume "i\ ip \notin kD(rt)"
  hence "\sigma _route(rt, ip) = None" ..
  thus "update rt ip r = rt (ip \mapsto r)"
  unfolding update_def by simp
next
  assume "i\ ip \in kD(rt)"
  and "sqn rt ip < \pi _2(r)"
  from this(1) obtain dsn dsk fl hops nhip
    where "rt ip = Some (dsn, dsk, fl, hops, nhip)"
    by (metis kD_Some)
  with \langle sqn rt ip < \pi _2(r)\rangle show "update rt ip r = rt (ip \mapsto r)"
  unfolding update_def s_def by auto
next
  assume "i\ ip \in kD(rt)"
  and "sqn rt ip = \pi _2(r)"
  and "the (dhops rt ip) > \pi _5(r)"
  from this(1) obtain dsn dsk fl hops nhip
    where "rt ip = Some (dsn, dsk, fl, hops, nhip)"
    by (metis kD_Some)
  with \langle sqn rt ip = \pi _2(r)\rangle and \langle the (dhops rt ip) > \pi _5(r)\rangle
show "update rt ip r = rt (ip ↦→ r)"
  unfolding update_def s_def by auto

next
  assume "ip ∈ kD(rt)"
  and "sqn rt ip = π₂(r)"
  and "flag rt ip = Some inv"
  from this(1) obtain dsn dsk fl hops nhip
    where "rt ip = Some (dsn, dsk, fl, hops, nhip)"
    by (metis kD_Some)
  with "sqn rt ip = π₂(r)" and "flag rt ip = Some inv"
  show "update rt ip r = rt (ip ↦→ r)"
    unfolding update_def s_def by auto

next
  assume "ip ∈ kD(rt)"
  and "π₃(r) = unk"
  and "(π₂(r) = 0) = (π₃(r) = unk)"
  from this(1) obtain dsn dsk fl hops nhip
    where "rt ip = Some (dsn, dsk, fl, hops, nhip)"
    by (metis kD_Some)
  with "(π₂(r) = 0) = (π₃(r) = unk)" and "π₃(r) = unk"
  show "update rt ip r = rt (ip ↦→ nr')"
    unfolding update_def nr'_def s_def
    by (cases r) simp

next
  assume "ip ∈ kD(rt)"
  and otherassms: "sqn rt ip ≥ π₂(r)"
  "π₃(r) = kno"
  "sqn rt ip = π₂(r) ∈⇒ the (dhops rt ip) ≤ π₅(r) ∧ the (flag rt ip) = val"
  from this(1) obtain dsn dsk fl hops nhip
    where "rt ip = Some (dsn, dsk, fl, hops, nhip)"
    by (metis kD_Some)
  with otherassms show "update rt ip r = rt (ip ↦→ s)"
    unfolding update_def s_def by auto

qed

lemma update_cases [elim]:
  assumes "(π₂(r) = 0) = (π₃(r) = unk)"
  and c1: "[\[ \[ ip /∈ kD(rt) \] \] ⇒ P (rt (ip ↦→ r))"
  and c2: "[\[ \[ ip ∈ kD(rt); sqn rt ip < π₂(r) \] \] ⇒ P (rt (ip ↦→ r))"
  and c3: "[\[ \[ ip ∈ kD(rt); sqn rt ip = π₂(r); the (dhops rt ip) > π₅(r) \] \] ⇒ P (rt (ip ↦→ r))"
  and c4: "[\[ \[ ip ∈ kD(rt); sqn rt ip = π₂(r); the (flag rt ip) = inv \] \] ⇒ P (rt (ip ↦→ r))"
  and c5: "[\[ \[ ip ∈ kD(rt); π₃(r) = unk \] \] ⇒ P (rt (ip ↦→ (π₃(the σroute(rt, ip)), π₃(r), π₄(r), π₅(r), π₆(r))))"
  and c6: "[\[ \[ ip ∈ kD(rt); sqn rt ip ≥ π₂(r); π₃(r) = kno; sqn rt ip = π₂(r) ∈⇒ the (dhops rt ip) ≤ π₅(r) ∧ the (flag rt ip) = val \] \] ⇒ P (rt (ip ↦→ the σroute(rt, ip))))"
  shows "(P (update rt ip r))"

proof (cases "ip ∈ kD(rt)")
  assume "ip /∈ kD(rt)"
  with c1 show thesis
    by simp

next
  assume "ip ∈ kD(rt)"
  moreover then obtain dsn dsk fl hops nhip
    where rteq: "rt ip = Some (dsn, dsk, fl, hops, nhip)"
    by (metis kD_Some)
  moreover obtain dsn' dsk' fl' hops' nhip'
    where req: "r = (dsn', dsk', fl', hops', nhip')"
    by (cases r) metis
ultimately show \( \vdash \text{thesis} \)

using \( (\pi_2(r) = 0) = (\pi_3(r) = \text{unk}) \)

c2 \[OF \ (ip \in kD(rt))\]

c3 \[OF \ (ip \in kD(rt))\]

c4 \[OF \ (ip \in kD(rt))\]

c5 \[OF \ (ip \in kD(rt))\]

c6 \[OF \ (ip \in kD(rt))\]

unfolding update_def sqn_def by auto

qed

lemma update_cases_kD:
assumes 
"(\pi_2(r) = 0) = (\pi_3(r) = \text{unk})"
and 
"ip \in kD(rt)"
and 
\[sqn rt ip < \pi_2(r) \Longrightarrow P(rt (ip \mapsto r))\]
and 
\[\llbracket sqn rt ip = \pi_2(r); \text{the (dhops rt ip)} > \pi_5(r) \rrbracket \Longrightarrow P(rt (ip \mapsto r))\]
and 
\[\llbracket sqn rt ip = \pi_2(r); \text{the (flag rt ip) = inv} \rrbracket \Longrightarrow P(rt (ip \mapsto r))\]
and 
\[\pi_3(r) = \text{unk} \Longrightarrow P(rt (ip \mapsto (\pi_2(\sigma_{route}(rt, ip)), \pi_3(r), \pi_4(r), \pi_5(r), \pi_6(r))))\]
and 
\[\llbracket sqn rt ip \geq \pi_2(r); \pi_3(r) = \text{kno}; sqn rt ip = \pi_2(r) \Longrightarrow \text{the (dhops rt ip) } \leq \pi_5(r) \land \text{the (flag rt ip) } = \text{val} \rrbracket \Longrightarrow P(rt (ip \mapsto \text{the } \sigma_{route}(rt, ip)))\]

shows "(P(update rt ip r))"

using assms(1) proof (rule update_cases)

assume "sqn rt ip < \pi_2(r)"
thus "P(rt(ip \mapsto r))" by (rule c2)

next
assume "sqn rt ip = \pi_2(r)"
and "the (dhops rt ip) > \pi_5(r)"
thus "P(rt(ip \mapsto r))"
by (rule c3)

next
assume "sqn rt ip = \pi_2(r)"
and "the (flag rt ip) = inv"
thus "P(rt(ip \mapsto r))"
by (rule c4)

next
assume "\pi_3(r) = \text{unk}"
thus "P(rt (ip \mapsto (\pi_2(\sigma_{route}(rt, ip)), \pi_3(r), \pi_4(r), \pi_5(r), \pi_6(r))))"
by (rule c5)

next
assume "sqn rt ip \geq \pi_2(r)"
and "\pi_3(r) = \text{kno}"
and "sqn rt ip = \pi_2(r) \Longrightarrow \text{the (dhops rt ip) } \leq \pi_5(r) \land \text{the (flag rt ip) } = \text{val}" 
thus "P(rt (ip \mapsto \text{the } (rt ip)))"
by (rule c6)

qed (simp add: \langle ip \in kD(rt) \rangle)

lemma in_kD_after_update [simp]:
fixes rt nip dsn dsk flag hops nhip
shows "kD(update rt nip (dsn, dsk, flag, hops, nhip)) = insert nip (kD rt)"

unfolding update_def
by (cases "rt nip") auto

lemma nhop_of_update [simp]:
fixes rt dip dsn dsk flag hops nhip
assumes "rt \neq \text{update rt dip (dsn, dsk, flag, hops, nhip)}"
shows "the \(\text{nhop (update rt dip (dsn, dsk, flag, hops, nhip)) dip}\) = nhip"

proof -
from assms
have update_neq: "\forall v. \text{rt dip = Some } v \Longrightarrow 
\text{update rt dip (dsn, dsk, flag, hops, nhip) } 
\neq \text{rt dip } \Rightarrow \text{the (rt dip))}"

260
by auto
show ?thesis
proof (cases "rt dip = None")
  assume "rt dip = None"
  thus "?thesis" unfolding update_def by clarsimp
next
  assume "rt dip ≠ None"
  then obtain v where "rt dip = Some v" by (metis not_None_eq)
  with update_neq [OF this] show ?thesis
    unfolding update_def by auto
qed
qed

lemma sqn_if_updated:
  fixes rip v rt ip
  shows "sqn (∀x. if x = rip then Some v else rt x) ip
    = (if ip = rip then π2(v) else sqn rt ip)"
  unfolding sqn_def by simp

lemma update_sqn [simp]:
  fixes rt dip rip dsn dsk hops nhip
  assumes "(dsn = 0) = (dsk = unk)"
  shows "sqn rt dip ≤ sqn (update rt rip (dsn, dsk, val, hops, nhip)) dip"
  proof (rule update_cases)
    show "(π2 (dsn, dsk, val, hops, nhip) = 0) = (π3 (dsn, dsk, val, hops, nhip) = unk)"
      by simp (rule assms)
  qed (clarsimp simp: sqn_if_updated sqn_def)+

lemma sqn_update_bigger [simp]:
  fixes rt ip ip' dsn dsk flag hops nhip
  assumes "1 ≤ hops"
  shows "sqn rt ip ≤ sqn (update rt ip' (dsn, dsk, flag, hops, nhip)) ip"
  using assms unfolding update_def sqn_def
  by (clarsimp split: option.split) auto

lemma dhops_update [intro]:
  fixes rt dsn dsk flag hops ip rip nhip
  assumes "ip ≠ hops"
  shows "∀ip∈kD rt. the (dhops rt ip) ≥ 1"
  using ip proof
    assume "ip = rip ∧ Suc 0 ≤ hops" thus ?thesis
      unfolding update_def using ex
    by (cases "rip ∈ kD rt") (drule(1) bspec, auto)
next
  assume "ip ≠ rip ∧ ip∈kD rt" thus ?thesis
  using ex unfolding update_def
  by (cases "rip∈kD rt") auto
qed

lemma update_another [simp]:
  fixes dip ip rt dsn dsk flag hops nhip
  assumes "ip ≠ dip"
  shows "(update rt dip (dsn, dsk, flag, hops, nhip)) ip = rt ip"
  using assms unfolding update_def
  by (clarsimp split: option.split)

lemma nhop_update_another [simp]:
  fixes dip ip rt dsn dsk flag hops nhip
  assumes "ip ≠ dip"
  shows "nhop (update rt dip (dsn, dsk, flag, hops, nhip)) ip = nhop rt ip"
  using assms unfolding update_def
  by (clarsimp split: option.split)
lemma dhops_update_another [simp]:
  fixes dip ip rt dsn dsk flag hops nhip
assumes "ip ≠ dip"
shows "dhops (update rt dip (dsn, dsk, flag, hops, nhip)) ip = dhops rt ip"
using assms unfolding update_def
by (clarsimp split: option.split)

lemma sqn_update_same [simp]:
  "∀rt ip dsn dsk flag hops nhip. sqn (rt(ip ↦→ v)) ip = π2(v)"
unfolding sqn_def by simp

lemma dhops_update_changed [simp]:
  fixes rt dip osn hops nhip
assumes "rt ≠ update rt dip (osn, kno, val, hops, nhip)"
shows "the (dhops (update rt dip (osn, kno, val, hops, nhip)) dip) = hops"
using assms unfolding update_def
by (clarsimp split: option.split_asm option.split split_if_asm) auto

lemma nhop_update_unk_val [simp]:
  "∀rt dip dsn hops. the (nhop (update rt dip (dsn, unk, val, hops, ip)) dip) = ip"
unfolding update_def
by (clarsimp split: option.split)

lemma nhop_update_changed [simp]:
  fixes rt dip dsn dsk flg hops sip
assumes "update rt dip (dsn, dsk, flg, hops, sip) ≠ rt"
shows "the (nhop (update rt dip (dsn, dsk, flg, hops, sip)) dip) = sip"
using assms unfolding update_def
by (clarsimp split: option.splits split_if_asm) auto

lemma update_rt_split_asm:
  "∀rt ip dsn dsk flag hops sip.
P (update rt ip (dsn, dsk, flag, hops, sip))
  =
  (∼(rt = update rt ip (dsn, dsk, flag, hops, sip) ∧ ∼P rt
  ∨ rt ≠ update rt ip (dsn, dsk, flag, hops, sip)
  ∧ ∼P (update rt ip (dsn, dsk, flag, hops, sip)))))" by auto

lemma sqn_update [simp]: "∀rt dip dsn flg hops sip.
rt ≠ update rt dip (dsn, kno, flg, hops, sip)
⇒ sqn (update rt dip (dsn, kno, flg, hops, sip)) dip = dsn"
unfolding update_def by (clarsimp split: option.split split_if_asm) auto

lemma sqnf_update [simp]: "∀rt dip dsn flg hops sip.
rt ≠ update rt dip (dsn, dsk, flg, hops, sip)
⇒ sqnf (update rt dip (dsn, dsk, flg, hops, sip)) dip = dsk"
unfolding update_def sqnf_def
by (clarsimp split: option.splits split_if_asm) auto

lemma update_kno_dsn_greater_zero:
  "∀rt dip ip dsn hops. 1 ≤ dsn ⇒ 1 ≤ (sqn (update rt dip (dsn, kno, val, hops, ip)) dip)"
unfolding update_def
by (clarsimp split: option.splits)

lemma proj3_update [simp]: "∀rt dip dsn flg hops sip.
rt ≠ update rt dip (dsn, dsk, flg, hops, sip)
⇒ π3(the (update rt dip (dsn, dsk, flg, hops, sip)) dip) = dsk"
unfolding update_def sqnf_def
by (clarsimp split: option.splits split_if_asm) auto

lemma nhop_update_changed_kno_val [simp]: "∀rt ip dsn dsk hops nhip.
rt ≠ update rt ip (dsn, kno, val, hops, nhip)
⇒ the (nhop (update rt ip (dsn, kno, val, hops, nhip)) ip) = nhip"
lemma flag_update [simp]: "∀ rt dip dsn flg hops sip. 
rt ≠ update rt dip (dsn, kno, flg, hops, sip) 
⇒ the ((flag (update rt dip (dsn, kno, flg, hops, sip)) dip) = flg"

unfolding update_def
by (clarsimp split: option.split split_if_asm) auto

lemma the_flag_Some [dest!]:
fixes ip rt
assumes "the (flag rt ip) = x"
and "ip ∈ kD rt"
shows "flag rt ip = Some x"
using assms by auto

lemma kD_update_unchanged [dest]:
fixes rt dip dsn dsk flag hops nhip
assumes "rt = update rt dip (dsn, dsk, flag, hops, nhip)"
shows "dip ∈ kD rt"
proof -
have "dip ∈ kD(update rt dip (dsn, dsk, flag, hops, nhip)) by simp
with assms show ?thesis by simp
qed

lemma nhop_update [simp]: "∀ rt dip dsn dsk flg hops sip. 
rt ≠ update rt dip (dsn, dsk, flg, hops, sip) 
⇒ the (nhop (update rt dip (dsn, dsk, flg, hops, sip)) dip) = sip"

unfolding update_def sqnf_def
by (clarsimp split: option.splits split_if_asm) auto

lemma sqn_update_another [simp]:
fixes dip ip rt dsn dsk flag hops nhip
assumes "ip ≠ dip"
shows "sqn (update rt dip (dsn, dsk, flag, hops, nhip)) ip = sqn rt ip"
using assms unfolding update_def sqn_def
by (clarsimp split: option.splits) auto

lemma sqnf_update_another [simp]:
fixes dip ip rt dsn dsk flag hops nhip
assumes "ip ≠ dip"
shows "sqnf (update rt dip (dsn, dsk, flag, hops, nhip)) ip = sqnf rt ip"
using assms unfolding update_def sqnf_def
by (clarsimp split: option.splits) auto

lemma vD_update_val [dest]:
"∀ dip rt dip' dsn hops nhip. 
(\∈ vD(update rt dip' (dsn, hops, nhip)) \⇒ (dip ∈ vD rt ∨ dip = dip'))"

unfolding update_def vD_def by (clarsimp split: option.split_asm split_if_asm)

Invalidating route entries

definition invalidate :: "rt ⇒ (ip ⇒ sqn) ⇒ rt"
where "invalidate rt dests ≡ λip. case (rt, dests) of 
    (None, _) ⇒ None 
    | (Some s, None) ⇒ Some s 
    | (Some (dsk, _, hops, nhip), Some rsn) ⇒ 
        Some (rsn, dsk, inv, hops, nhip)"

lemma proj3_invalidate [simp]:
"∀ dip. π₃(invalidate rt dests dip) = π₃(rt dip)"

unfolding invalidate_def by (clarsimp split: option.split)
lemma proj5_invalidate [simp]:
"\land dip. \pi_5(\text{(invalidate rt dests) dip}) = \pi_5(\text{(rt dip)})"
unfolding invalidate_def by (clarsimp split: option.split)

lemma proj6_invalidate [simp]:
"\land dip. \pi_6(\text{(invalidate rt dests) dip}) = \pi_6(\text{(rt dip)})"
unfolding invalidate_def by (clarsimp split: option.split)

lemma invalidate_kD_inv [simp]:
"\land rt dests. kD (invalidate rt dests) = kD rt"
unfolding invalidate_def kD_def by (simp split: option.split)

lemma invalidate_sqn:
fixes rt dip dests
assumes "\forall rsn. dests dip = Some rsn \rightarrow sqn rt dip \leq rsn"
shows "sqn rt dip \leq sqn (invalidate rt dests) dip"
proof (cases "dip /\in kD(rt)"
  assume "\neg dip /\in kD(rt)"
  hence "dip/\in kD(rt)" by simp
  then obtain dsn dsk flag hops nhip pre where "rt dip = Some (dsn, dsk, flag, hops, nhip)"
    by (metis kD_Some)
  with assms show "sqn rt dip \leq sqn (invalidate rt dests) dip"
    by (cases "dests dip") (auto simp add: invalidate_def sqn_def)
qed simp

lemma sqn_invalidate_in_dests [simp]:
fixes dests ipa rsn rt
assumes "dests ipa = Some rsn"
and "ipa/\in kD(rt)"
shows "sqn (invalidate (rt \xi) (dests \xi)) dip = sqn (rt \xi) dip"
unfolding sqnf_def invalidate_def using assms(1) assms(2) [THEN kD_Some]
by clarsimp

lemma dhops_invalidate [simp]:
"\land dip. dhops (invalidate rt dests) dip = dhops rt dip"
unfolding invalidate_def by (clarsimp split: option.split)

lemma sqnf_invalidate [simp]:
"\land dip. sqnf (invalidate (rt \xi) (dests \xi)) dip = sqnf (rt \xi) dip"
unfolding sqnf_def invalidate_def by (clarsimp split: option.split)

lemma nhop_invalidate [simp]:
"\land dip. nhop (invalidate (rt \xi) (dests \xi)) dip = nhop (rt \xi) dip"
unfolding invalidate_def by (clarsimp split: option.split)

lemma invalidate_other [simp]:
fixes rt dests dip
assumes "dip/\in dom(dests)"
shows "invalidate rt dests dip = rt dip"
using assms unfolding invalidate_def by (clarsimp split: option.split_asm)

lemma invalidate_none [simp]:
fixes rt dests dip
assumes "dip/\not/\in kD(rt)"
shows "invalidate rt dests dip = None"
using assms unfolding invalidate_def by clarsimp

lemma vD_invalidate_vD_not_dests:
"\land dip rt dests. dip/\in vD(invalidate rt dests) \rightarrow dip/\in vD(rt) \land dests dip = None"
unfolding invalidate_def vD_def
by (clarsimp split: option.split_asm)

lemma sqn_invalidate_not_in_dests [simp]:
  fixes dests dip rt
  assumes "dip \in dom(dests)"
  shows "sqn (invalidate rt dests) dip = sqn rt dip"
  using assms unfolding sqn_def by simp

lemma invalidate_changes:
  fixes rt dests dip dsn dsk flag hops nhip
  assumes "invalidate rt dests dip = Some (dsn, dsk, flag, hops, nhip)"
  shows "dsn = (case dests dip of None ⇒ π₂(the (rt dip)) | Some rsn ⇒ rsn)
    ∧ dsk = π₃(the (rt dip))
    ∧ flag = (if dests dip = None then π₄(the (rt dip)) else inv)
    ∧ hops = π₅(the (rt dip))
    ∧ nhip = π₆(the (rt dip))"
  using assms unfolding invalidate_def
  by (cases "rt dip", clarsimp, cases "dests dip") auto

lemma proj3_inv: "∀ dip rt dests. dip \in kD (rt)
  ⇒ π₃(the (invalidate rt dests dip)) = π₃(the (rt dip))"
  by (clarsimp simp: invalidate_def kD_def split: option.split)

lemma dests_iD_invalidate [simp]:
  assumes "dests ip = Some rsn"
  and "ip \in kD(rt)"
  shows "ip \in iD(invalidate rt dests)"
  using assms(1) assms(2) [THEN kD_Some]
  unfolding invalidate_def iD_def
  by (clarsimp split: option.split)

3.1.5 Route Requests

Generate a fresh route request identifier.

definition nrreqid :: "(ip \times rreqid) set ⇒ ip ⇒ rreqid"
  where "nrreqid rreqs ip ≡ Max ({n. (ip, n) \in rreqs} \union {0}) + 1"

3.1.6 Queued Packets

Functions for sending data packets.

type_synonym store = "ip ⇒ (p \times data list)"

definition sigma_queue :: "store ⇒ ip ⇒ data list" ("σ'queue(_, _, _)")
  where "σ'queue(store, dip) ≡ case store dip of None ⇒ [] | Some (p, q) ⇒ q"

definition qD :: "store ⇒ ip set"
  where "qD ≡ dom"

definition add :: "data ⇒ ip ⇒ store ⇒ store"
  where "add d dip store ≡ case store dip of
    None ⇒ store (dip := None) | Some (p, q) ⇒ store (dip := (p, q \union [d]))"

lemma qD_add [simp]:
  fixes d dip store
  shows "qD(add d dip store) = insert dip (qD store)"
  unfolding add_def Let_def qD_def
  by (clarsimp split: option.split)

definition drop :: "ip ⇒ store ⇒ store"
  where "drop dip store ≡ map_option (λ(p, q). if tl q = [] then store (dip := None)
    else store (dip := (p, tl q))) (store dip)"
definition $\sigma_p\_flag :: \text{store } \Rightarrow \text{ip } \mapsto \text{p} \equiv \text{map_option fst (store dip)}$

where $\sigma_p\_flag(\text{store, dip}) \equiv \text{map_option fst (store dip)}$

definition unsetRRF :: "store ⇒ ip ⇒ store" where "unsetRRF store dip ≡ case store dip of
  None ⇒ store
  | Some (p, q) ⇒ store (dip ↦ (noreq, q))"

definition setRRF :: "store ⇒ (ip ⇒ sqn) ⇒ store" where "setRRF store dests ≡ λdip. if dests dip = None then store dip
  else map_option (λ(_, q). (req, q)) (store dip)"

3.1.7 Comparison with the original technical report

The major differences with the AODV technical report of Fehnker et al are:

1. $\text{nhop}$ is partial, thus a ‘the’ is needed, similarly for $\text{dhops}$ and $\text{addpreRT}$.

2. $\text{precs}$ is partial.

3. $\sigma_p\_flag(\text{store, dip})$ is partial.

4. The routing table ($\text{rt}$) is modelled as a map ($\text{ip } \mapsto \text{r option}$) rather than a set of 7-tuples, likewise, the $\text{r}$ is a 6-tuple rather than a 7-tuple, i.e., the destination ip-address ($\text{dip}$) is taken from the argument to the function, rather than a part of the result. Well-definedness then follows from the structure of the type and more related facts are available automatically, rather than having to be acquired through tedious proofs.

5. Similar remarks hold for the $\text{dests}$ mapping passed to $\text{invalidate}$, and $\text{store}$.

end

3.2 AODV protocol messages

theory C_Aodv_Message
imports C_Stobcast
begin

datatype msg =
  Rreq nat rreqid ip sqn k ip sqn ip
| Rrep nat ip sqn ip ip
| Rerr "ip ⇒ sqn" ip
| Newpkt data ip
| Pkt data ip ip

instantiation msg :: msg
begin
  definition newpkt_def [simp]: "newpkt ≡ λ(d, dip). Newpkt d dip"
  definition eq_newpkt_def: "eq_newpkt m ≡ case m of Newpkt d dip ⇒ True | _ ⇒ False"

  instance by intro_classes (simp add: eq_newpkt_def)
end

The $\text{msg}$ type models the different messages used within AODV. The instantiation as a $\text{msg}$ is a technicality due to the special treatment of $\text{newpkt}$ messages in the AWN SOS rules. This use of classes allows a clean separation of the AWN-specific definitions and these AODV-specific definitions.

definition rreq :: "nat × rreqid × ip × sqn × k × ip × sqn × ip ⇒ msg"
where "rreq ≡ λ(hops, rreqid, dip, dsn, dsk, oip, osn, sip). Rreq hops rreqid dip dsn dsk oip osn sip"

lemma rreq_simp [simp]:
"rreq(hops, rreqid, dip, dsn, dsk, oip, osn, sip) = Rreq hops rreqid dip dsn dsk oip osn sip"
unfolding rreq_def by simp

266
definition \( rrep :: \text{nat} \times \text{ip} \times \text{sqn} \times \text{ip} \times \text{ip} \Rightarrow \text{msg} \)
where \( rrep \equiv \lambda (\text{hops}, \text{dip}, \text{dsn}, \text{oip}, \text{sip}). \text{Rrep hops dip dsn oip sip} \)

lemma \( rrep\_simp \) [simp]:
"\( rrep(\text{hops}, \text{dip}, \text{dsn}, \text{oip}, \text{sip}) = \text{Rrep hops dip dsn oip sip} \)"
unfolding \( rrep\_def \) by simp

definition \( rerr :: (\text{ip} \rightarrow \text{sqn}) \times \text{ip} \Rightarrow \text{msg} \)
where \( rerr \equiv \lambda (\text{dests}, \text{sip}). \text{Rerr dests sip} \)

lemma \( rerr\_simp \) [simp]:
"\( rerr(\text{dests}, \text{sip}) = \text{Rerr dests sip} \)"
unfolding \( rerr\_def \) by simp

lemma not_eq_newpkt_rreq [simp]: "\( \neg \text{eq_newpkt (Rreq hops rreqid dip dsn dsk oip osn sip)} \)"
unfolding eq_newpkt_def by simp

lemma not_eq_newpkt_rrep [simp]: "\( \neg \text{eq_newpkt (Rrep hops dip dsn oip sip)} \)"
unfolding eq_newpkt_def by simp

lemma not_eq_newpkt_rerr [simp]: "\( \neg \text{eq_newpkt (Rerr dests sip)} \)"
unfolding eq_newpkt_def by simp

lemma not_eq_newpkt_pkt [simp]: "\( \neg \text{eq_newpkt (Pkt d dip sip)} \)"
unfolding eq_newpkt_def by simp

definition \( pkt :: \text{data} \times \text{ip} \times \text{ip} \Rightarrow \text{msg} \)
where \( pkt \equiv \lambda (\text{d}, \text{dip}, \text{sip}). \text{Pkt d dip sip} \)

lemma \( pkt\_simp \) [simp]:
"\( pkt(\text{d}, \text{dip}, \text{sip}) = \text{Pkt d dip sip} \)"
unfolding \( pkt\_def \) by simp

end

3.3 The AODV protocol

theory C_Aodv
imports C_Aodv_Data C_Aodv_Message
"../../../AWN/AWN_SOS_Labels" "../../../AWN/AWN_Invariants"
begin

3.3.1 Data state

record state =
  ip :: "ip"
  sn :: "sqn"
  rt :: "rt"
  rreqs :: "(ip \times rreqid) set"
  store :: "store"

msg :: "msg"
data :: "data"
dests :: "ip \rightarrow sqn"
rreqid :: "rreqid"
dip :: "ip"
oip :: "ip"
hops :: "nat"
dsn :: "sqn"
dsk :: "k"
osn :: "sqn"
sip :: "ip"
abbreviation aodv_init :: "ip ⇒ state"
where "aodv_init i ≡ (
    ip = i,
    sn = 1,
    rt = empty,
    rreqs = {},
    store = empty,
    msg = (SOME x. True),
    data = (SOME x. True),
    dests = (SOME x. True),
    rreqid = (SOME x. True),
    dip = (SOME x. True),
    oip = (SOME x. True),
    hops = (SOME x. True),
    dsn = (SOME x. True),
    dsk = (SOME x. True),
    osn = (SOME x. True),
    sip = (SOME x. x ≠ i)
)
"

lemma some_neq_not_eq [simp]: "¬((SOME x :: nat. x ≠ i) = i)"
  by (subst some_eq_ex) (metis zero_neq_numeral)

definition clear_locals :: "state ⇒ state"
where "clear_locals ξ = ξ (msg := (SOME x. True),
               data := (SOME x. True),
               dests := (SOME x. True),
               rreqid := (SOME x. True),
               dip := (SOME x. True),
               oip := (SOME x. True),
               hops := (SOME x. True),
               dsn := (SOME x. True),
               dsk := (SOME x. True),
               osn := (SOME x. True),
               sip := (SOME x. x ≠ ip ξ))
"

lemma clear_locals_sip_not_ip [simp]: "¬(sip (clear_locals ξ) = ip ξ)"
  unfolding clear_locals_def by simp

lemma clear_locals_but_not_globals [simp]:
  "ip (clear_locals ξ) = ip ξ"
  "sn (clear_locals ξ) = sn ξ"
  "rt (clear_locals ξ) = rt ξ"
  "rreqs (clear_locals ξ) = rreqs ξ"
  "store (clear_locals ξ) = store ξ"
  unfolding clear_locals_def by auto

3.3.2 Auxilliary message handling definitions

definition is_newpkt
where "is_newpkt ξ ≡ case msg ξ of
    Newpkt data' dip' ⇒ { ξ[|data := data', dip := dip'|] }
  | _ ⇒ {}"

definition is_pkt
where "is_pkt ξ ≡ case msg ξ of
    Pkt data' dip' oip' ⇒ { ξ[|data := data', dip := dip', oip := oip'|] }
  | _ ⇒ {}"

definition is_rreq
where "is_rreq ξ ≡ case msg ξ of
Rreq hops' rreqid' dip' dsn' dsk' oip' osn' sip' \Rightarrow
\{\xi\xi hops := hops', rreqid := rreqid', dip := dip', ds := dsn',
dsk := dsk', oip := oip', osn := osn', sip := sip' \}\}
l_\_ \Rightarrow \{\}"

lemma is_rreq_asm [dest!]:
assumes "\(\xi' \in \text{is_rreq } \xi\)"
shows "(\(\exists \text{hops' rreqid' dip' dsn' dsk' oip' osn' sip'}.\)
\(msg \xi' = \text{Rreq hops' rreqid' dip' dsn' dsk' oip' osn' sip' \&}
\(\xi' = \{\xi\xi hops := hops', rreqid := rreqid', dip := dip', ds := dsn',
dsk := dsk', oip := oip', osn := osn', sip := sip' \}\)"
using assms unfolding is_rreq_def
by (cases "msg \xi") simp_all

definition is_rrep
where "is_rrep \xi \equiv \text{case msg } \xi of\nRrep hops' dip' dsn' oip' sip' \Rightarrow
\{\xi\xi hops := hops', dip := dip', ds := dsn', oip := oip', sip := sip' \}\}
l_\_ \Rightarrow \{\}"

lemma is_rrep_asm [dest!]:
assumes "\(\xi' \in \text{is_rrep } \xi\)"
shows "(\(\exists \text{hops' dip' dsn' oip' sip'}.\)
\(msg \xi' = \text{Rrep hops' dip' dsn' oip' sip' \&}
\(\xi' = \{\xi\xi hops := hops', dip := dip', ds := dsn', oip := oip', sip := sip' \}\)"
using assms unfolding is_rrep_def
by (cases "msg \xi") simp_all

definition is_rerr
where "is_rerr \xi \equiv \text{case msg } \xi of\nRerr dests' sip' \Rightarrow
\{\xi\xi dests := dests', sip := sip' \}\}
l_\_ \Rightarrow \{\}"

lemma is_rerr_asm [dest!]:
assumes "\(\xi' \in \text{is_rerr } \xi\)"
shows "(\(\exists \text{dests' sip'}.\)
\(msg \xi' = \text{Rerr dests' sip' \&}
\(\xi' = \{\xi\xi dests := dests', sip := sip' \}\)"
using assms unfolding is_rerr_def
by (cases "msg \xi") simp_all

lemmas is_msg_defs =
is_rerr_def is_rreq_def is_pkt_def is_newpkt_def

lemma is_msg_inv_ip [simp]:
"\(\xi' \in \text{is_rerr } \xi \Rightarrow \text{ip } \xi' = \text{ip } \xi\)"
"\(\xi' \in \text{is_rrep } \xi \Rightarrow \text{ip } \xi' = \text{ip } \xi\)"
"\(\xi' \in \text{is_rreq } \xi \Rightarrow \text{ip } \xi' = \text{ip } \xi\)"
"\(\xi' \in \text{is_pkt } \xi \Rightarrow \text{ip } \xi' = \text{ip } \xi\)"
"\(\xi' \in \text{is_newpkt } \xi \Rightarrow \text{ip } \xi' = \text{ip } \xi\)"
unfolding is_msg_defs
by (cases "msg \xi", clarsimp+) +

lemma is_msg_inv_sn [simp]:
"\(\xi' \in \text{is_rerr } \xi \Rightarrow \text{sn } \xi' = \text{sn } \xi\)"
"\(\xi' \in \text{is_rrep } \xi \Rightarrow \text{sn } \xi' = \text{sn } \xi\)"
"\(\xi' \in \text{is_rreq } \xi \Rightarrow \text{sn } \xi' = \text{sn } \xi\)"
"\(\xi' \in \text{is_pkt } \xi \Rightarrow \text{sn } \xi' = \text{sn } \xi\)"
"\(\xi' \in \text{is_newpkt } \xi \Rightarrow \text{sn } \xi' = \text{sn } \xi\)"
unfolding is_msg_defs
by (cases "msg \xi", clarsimp+) +

lemma is_msg_inv_rt [simp]:
"\(\xi' \in \text{is_rerr } \xi \Rightarrow \text{rt } \xi' = \text{rt } \xi\)"
unfolding is_msg_defs by (cases "msg ς", clarsimp+)+

lemma is_msg_inv_rreqs [simp]:
"ς' ∈ is_rerr ς ⇒ rreqs ς' = rreqs ς"
"ς' ∈ is_rrep ς ⇒ rreqs ς' = rreqs ς"
"ς' ∈ is_rreq ς ⇒ rreqs ς' = rreqs ς"
"ς' ∈ is_pkt ς ⇒ rreqs ς' = rreqs ς"
"ς' ∈ is_newpkt ς ⇒ rreqs ς' = rreqs ς"
unfolding is_msg_defs by (cases "msg ς", clarsimp+)+

lemma is_msg_inv_store [simp]:
"ς' ∈ is_rerr ς ⇒ store ς' = store ς"
"ς' ∈ is_rrep ς ⇒ store ς' = store ς"
"ς' ∈ is_rreq ς ⇒ store ς' = store ς"
"ς' ∈ is_pkt ς ⇒ store ς' = store ς"
"ς' ∈ is_newpkt ς ⇒ store ς' = store ς"
unfolding is_msg_defs by (cases "msg ς", clarsimp+)+

lemma is_msg_inv_sip [simp]:
"ς' ∈ is_pkt ς ⇒ sip ς' = sip ς"
"ς' ∈ is_newpkt ς ⇒ sip ς' = sip ς"
unfolding is_msg_defs by (cases "msg ς", clarsimp+)+

3.3.3 The protocol process

datatype pseqp =
  PAodv
| PNewPkt
| PPkt
| PRreq
| PRrep
| PRerr

fun nat_of_seqp :: "pseqp ⇒ nat"
where
  "nat_of_seqp PAodv = 1"
| "nat_of_seqp PPkt = 2"
| "nat_of_seqp PNewPkt = 3"
| "nat_of_seqp PRreq = 4"
| "nat_of_seqp PRrep = 5"
| "nat_of_seqp PRerr = 6"

instantiation "pseqp" :: ord begin
definition less_eq_seqp [iff]: "l1 ≤ l2 = (nat_of_seqp l1 ≤ nat_of_seqp l2)"
definition less_seqp [iff]: "l1 < l2 = (nat_of_seqp l1 < nat_of_seqp l2)"
instance ..
end

abbreviation AODV
where
  "AODV ≡ λ_. [clear_locals] call(PAodv)"

abbreviation PKT
where
  "PKT args ≡"
abbreviation NEWPKT
where
"NEWPKT args ≡
[ξ. let (data, dip, oip) = args ξ in
 (clear_locals ξ) (| data := data, dip := dip, oip := oip |)]
call(PNewPkt)"

abbreviation RREQ
where
"RREQ args ≡
[ξ. let (hops, rreqid, dip, dsn, dsk, oip, osn, sip) = args ξ in
 (clear_locals ξ) (| hops := hops, rreqid := rreqid, dip := dip,
 dsn := dsn, dsk := dsk, oip := oip,
 osn := osn, sip := sip |)]
call(PRreq)"

abbreviation RREP
where
"RREP args ≡
[ξ. let (hops, dip, dsn, dsk, oip, sip) = args ξ in
 (clear_locals ξ) (| hops := hops, dip := dip,
 dsn := dsn,
 oip := oip, sip := sip |)]
call(PRrep)"

abbreviation RERR
where
"RERR args ≡
[ξ. let (dests, sip) = args ξ in
 (clear_locals ξ) (| dests := dests, sip := sip |)]
call(PRerr)"

fun Γ_AODV :: "("state, msg, pseqp, pseqp label) seqp_env"
where
"Γ_AODV PAadv = labelled PAadv |
receive(λmsg' ξ. ξ (| msg := msg' |)).
 ( (is_newpkt) NEWPKT(λξ. (data ξ, ip ξ))
 ⊕ (is_pkt) PKT(λξ. (data ξ, dip ξ, oip ξ))
 ⊕ (is_rreq)
 [ξ. ξ (| rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ) |)]
 RREQ(λξ. (hops ξ, rreqid ξ, dip ξ, dsn ξ, dsk ξ, oip ξ, osn ξ, sip ξ))
 ⊕ (is_rrep)
 [ξ. ξ (| rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ) |)]
 RREP(λξ. (hops ξ, dip ξ, dsn ξ, oip ξ, sip ξ))
 ⊕ (is_rerr)
 [ξ. ξ (| rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ) |)]
 RERR(λξ. (dests ξ, sip ξ))
)
 ⊕ (λξ. { ξ (| dip := dip |) | dip. dip ∈ qD(store ξ) ∩ vD(rt ξ) } )
 [ξ. ξ (| data := hd(σ_queue(store ξ, dip ξ)) |)]
 unicast(λξ. the (nhop (rt ξ) (dip ξ)), λξ. pkt(data ξ, dip ξ, ip ξ)).
 [ξ. ξ (| store := the (drop (dip ξ) (store ξ)) |)]
 AODV()
 ⊕ [ξ. ξ (| dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (dip ξ))
 then Some (inc (sqn (rt ξ rip)) else None) |)]
 [ξ. ξ (| rt := invalidate (rt ξ) (dests ξ) |)]
 [ξ. ξ (| store := setRRF (store ξ) (dests ξ)) ]
 broadcast(λξ. rerr(dests ξ, ip ξ)). AODV()
 ⊕ (λξ. { ξ (| dip := dip |) | dip. dip ∈ qD(store ξ) ∩ vD(rt ξ) ∧
 (σ_p-flag(store ξ, dip ξ)) = req } )
 [ξ. ξ (| store := unsetRRF (store ξ) (dip ξ) |)]
\[ \Gamma_{AODV} \text{ PNewPkt = labelled PNewPkt (} \\
\langle \xi, \text{ dip } \xi = \text{ ip } \xi \rangle \text{).AODV()} \\
\oplus \langle \xi, \text{ dip } \xi \neq \text{ ip } \xi \rangle \text{. store := add (data } \xi) \text{ (dip } \xi) \text{ (store } \xi) \rangle \text{. AODV()}" \\
/ \]

\[
\Gamma_{AODV} \text{ PPkt = labelled PPkt (} \\
\langle \xi, \text{ dip } \xi = \text{ ip } \xi \rangle \text{. data } \xi)\text{.AODV()} \\
\oplus \langle \xi, \text{ dip } \xi \neq \text{ ip } \xi \rangle \text{. dip } \xi \in vD (rt } \xi) \\
\text{ unicast(} \lambda \xi. \text{ the (nhop (rt } \xi) \text{ (dip } \xi)), \lambda \xi. \text{ pkt(data } \xi) \text{, dip } \xi, \text{ oip } \xi))\text{.AODV()} \text{. dip } \xi \in iD (rt } \xi) \\
\text{ broadcast(} \lambda \xi. \text{ rerr(dests } \xi, \text{ ip } \xi))\text{.AODV()} \\
\oplus \langle \xi, \text{ dip } \xi \neq iD (rt } \xi) \rangle \text{. AODV()} \\
) \\
)" \\
/ \]

\[
\Gamma_{AODV} \text{ PRreq = labelled PRreq (} \\
\langle \xi, \text{ oip } \xi, \text{ rreqid } \xi \in \text{ rreqs } \xi \rangle \text{.AODV()} \\
\oplus \langle \xi, \text{ oip } \xi, \text{ rreqid } \xi \notin \text{ rreqs } \xi \rangle \text{. rreqs := rreqs (} \xi) \text{ (} \text{oip } \xi, \text{ rreqid } \xi) \rangle \\
\text{ (} \xi, \text{ rreqs := rreqs } \xi \cup \{\text{(oip } \xi, \text{ rreqid } \xi)\}\rangle \\
\langle \xi, \text{ dip } \xi = \text{ ip } \xi \rangle \text{. sn := inc (sn } \xi) \text{. pkt(data } \xi) \text{, dip } \xi, \text{ oip } \xi)\text{.AODV()} \text{. dip } \xi \neq \text{ ip } \xi \rangle \\
\text{ unicast(} \lambda \xi. \text{ the (nhop (rt } \xi) \text{ (oip } \xi)), \lambda \xi. \text{ rrep(0, dip } \xi, \text{ sn } \xi, \text{ oip } \xi, \text{ ip } \xi))\text{.AODV()} \text{. dip } \xi \in vD (rt } \xi) \text{ \& dip } \xi \neq \text{ ip } \xi \rangle \\
\text{ unicast(} \lambda \xi. \text{ the (nhop (rt } \xi) \text{ (oip } \xi)), \lambda \xi. \text{ rrep(dips (rt } \xi) \text{ (dip } \xi), \text{ dip } \xi, \text{ sn } \xi, \text{ oip } \xi, \text{ ip } \xi))\text{.AODV()} \text{. dip } \xi \neq \text{ ip } \xi \rangle \\
\text{ store := setRRF (store } \xi) \text{ (dests } \xi)\rangle \text{. AODV()} \\
\text{ broadcast(} \lambda \xi. \text{ rerr(dests } \xi, \text{ ip } \xi))\text{.AODV()} \\
\oplus \langle \xi, \text{ dip } \xi \neq iD (rt } \xi) \rangle \text{. AODV()} \\
) \text{. dip } \xi \in vD (rt } \xi) \text{ \& dip } \xi \neq \text{ ip } \xi \rangle \\
\text{ unicast(} \lambda \xi. \text{ the (nhop (rt } \xi) \text{ (dip } \xi)), \lambda \xi. \text{ rrep(dips (rt } \xi) \text{ (dip } \xi), \text{ dip } \xi, \text{ sn } \xi, \text{ oip } \xi, \text{ ip } \xi))\text{.AODV()} \text{. dip } \xi \neq \text{ ip } \xi \rangle \\
\text{ store := setRRF (store } \xi) \text{ (dests } \xi)\rangle \text{. AODV()} \\
\text{ broadcast(} \lambda \xi. \text{ rerr(dests } \xi, \text{ ip } \xi))\text{.AODV()} \\
\oplus \langle \xi, \text{ dip } \xi \neq iD (rt } \xi) \rangle \text{. AODV()} \\
) \\
)" \\
/ \]

\[
\Gamma_{AODV} \text{ PNewPkt = labelled PNewPkt (} \\
\langle \xi, \text{ dip } \xi = \text{ ip } \xi \rangle \text{.data } \xi)\text{.AODV()} \\
\oplus \langle \xi, \text{ dip } \xi \neq \text{ ip } \xi \rangle \text{. store := add (data } \xi) \text{ (dip } \xi) \text{ (store } \xi) \rangle \text{. AODV()} \\
)" \\
/ \]

\[
\Gamma_{AODV} \text{ PPkt = labelled PPkt (} \\
\langle \xi, \text{ dip } \xi = \text{ ip } \xi \rangle \text{. data } \xi)\text{.AODV()} \\
\oplus \langle \xi, \text{ dip } \xi \neq \text{ ip } \xi \rangle \text{. dip } \xi \in vD (rt } \xi) \\
\text{ unicast(} \lambda \xi. \text{ the (nhop (rt } \xi) \text{ (dip } \xi)), \lambda \xi. \text{ pkt(data } \xi) \text{, dip } \xi, \text{ oip } \xi)\text{.AODV()} \text{. dip } \xi \in iD (rt } \xi) \\
\text{ broadcast(} \lambda \xi. \text{ rerr(dests } \xi, \text{ ip } \xi))\text{.AODV()} \\
\oplus \langle \xi, \text{ dip } \xi \neq iD (rt } \xi) \rangle \text{. AODV()} \\
) \\
)" \\
/ \]

\[
\Gamma_{AODV} \text{ PRreq = labelled PRreq (} \\
\langle \xi, \text{ oip } \xi, \text{ rreqid } \xi \in \text{ rreqs } \xi \rangle \text{.AODV()} \\
\oplus \langle \xi, \text{ oip } \xi, \text{ rreqid } \xi \notin \text{ rreqs } \xi \rangle \text{. rreqs := rreqs (} \xi) \text{ (} \text{oip } \xi, \text{ rreqid } \xi) \rangle \\
\text{ (} \xi, \text{ rreqs := rreqs } \xi \cup \{\text{(oip } \xi, \text{ rreqid } \xi)\}\rangle \\
\langle \xi, \text{ dip } \xi = \text{ ip } \xi \rangle \text{. sn := inc (sn } \xi) \text{.pkt(data } \xi) \text{, dip } \xi, \text{ oip } \xi)\text{.AODV()} \text{. dip } \xi \neq \text{ ip } \xi \rangle \\
\text{ unicast(} \lambda \xi. \text{ the (nhop (rt } \xi) \text{ (oip } \xi)), \lambda \xi. \text{ rrep(0, dip } \xi, \text{ sn } \xi, \text{ oip } \xi, \text{ ip } \xi))\text{.AODV()} \text{. dip } \xi \in vD (rt } \xi) \text{ \& dip } \xi \neq \text{ ip } \xi \rangle \\
\text{ unicast(} \lambda \xi. \text{ the (nhop (rt } \xi) \text{ (dip } \xi)), \lambda \xi. \text{ rrep(dips (rt } \xi) \text{ (dip } \xi), \text{ dip } \xi, \text{ sn } \xi, \text{ oip } \xi, \text{ ip } \xi))\text{.AODV()} \text{. dip } \xi \neq \text{ ip } \xi \rangle \\
\text{ store := setRRF (store } \xi) \text{ (dests } \xi)\rangle \text{. AODV()} \\
\text{ broadcast(} \lambda \xi. \text{ rerr(dests } \xi, \text{ ip } \xi))\text{.AODV()} \\
\oplus \langle \xi, \text{ dip } \xi \neq iD (rt } \xi) \rangle \text{. AODV()} \\
) \text{. dip } \xi \in vD (rt } \xi) \text{ \& dip } \xi \neq \text{ ip } \xi \rangle \\
\text{ unicast(} \lambda \xi. \text{ the (nhop (rt } \xi) \text{ (dip } \xi)), \lambda \xi. \text{ rrep(dips (rt } \xi) \text{ (dip } \xi), \text{ dip } \xi, \text{ sn } \xi, \text{ oip } \xi, \text{ ip } \xi))\text{.AODV()} \text{. dip } \xi \neq \text{ ip } \xi \rangle \\
\text{ store := setRRF (store } \xi) \text{ (dests } \xi)\rangle \text{. AODV()} \\
\text{ broadcast(} \lambda \xi. \text{ rerr(dests } \xi, \text{ ip } \xi))\text{.AODV()} \\
\oplus \langle \xi, \text{ dip } \xi \neq iD (rt } \xi) \rangle \text{. AODV()} \\
) \\
)" \\
/ \]
broadcast(\( \lambda \xi \cdot \text{rreq}(hops \xi + 1, \text{rreqid} \xi, \text{dip} \xi, \text{max}(\text{sqn}(\text{rt} \xi)(\text{dip} \xi)), \text{dsn} \xi, \text{dsk} \xi, \text{oip} \xi, \text{osn} \xi, \text{ip} \xi)) \).

AODV()
)
)
"\( \Gamma_{AODV} \) \text{PRrep} = \text{labelled PRrep} (\xi, \text{rt} \xi \neq \text{update}(\text{rt} \xi)(\text{dip} \xi)(\text{dsn} \xi, \text{kno}, \text{val}, \text{hops} \xi + 1, \text{sip} \xi) \})
(\xi, \xi \ni \text{rt} := \text{update}(\text{rt} \xi)(\text{dip} \xi)(\text{dsn} \xi, \text{kno}, \text{val}, \text{hops} \xi + 1, \text{sip} \xi) \})
(\xi, \text{oip} \xi = \text{ip} \xi)
AODV()
\oplus (\xi, \text{oip} \xi \neq \text{ip} \xi)
(\xi, \text{oip} \xi \in \text{vD}(\text{rt} \xi))
\text{unicast}(\lambda \xi \cdot \text{the}(\text{nhop}(\text{rt} \xi)(\text{oip} \xi)), \lambda \xi \cdot \text{rrep}(hops \xi + 1, \text{dip} \xi, \text{dsn} \xi, \text{oip} \xi, \text{ip} \xi))
AODV()
\oplus (\xi, \text{oip} \xi \not\in \text{vD}(\text{rt} \xi))
AODV()
)
)
\oplus (\xi, \text{rt} = \text{update}(\text{rt} \xi)(\text{dip} \xi)(\text{dsn} \xi, \text{kno}, \text{val}, \text{hops} \xi + 1, \text{sip} \xi) \})
AODV()
)
"

\( \Gamma_{AODV} \) \text{PRerr} = \text{labelled PRerr} (\xi, \text{dests} := (\lambda \text{rip}. \text{if}(\text{rip} \in \text{vD}(\text{rt} \xi) \land \text{nhop}(\text{rt} \xi)(\text{rip}) = \text{nhop}(\text{rt} \xi)(\text{oip} \xi)) \text{then Some}(\text{inc}(\text{sqn}(\text{rt} \xi)(\text{rip})) \text{else None}) \})
(\xi, \text{rt} := \text{invalidate}(\text{rt} \xi)(\text{dests} \xi) \})
(\xi, \text{store} := \text{setRRF}(\text{store} \xi)(\text{dests} \xi) \})
\text{broadcast}(\lambda \xi \cdot \text{rerr}(\text{dests} \xi, \text{ip} \xi)).\text{AODV}()
\oplus (\xi, \text{dests} \xi \neq \text{Map}(.empty) \})
\text{broadcast}(\lambda \xi \cdot \text{rerr}(\text{dests} \xi, \text{ip} \xi)).\text{AODV}()
\oplus (\xi, \text{dests} \xi = \text{Map}.empty)
AODV()
)
"

declare \( \Gamma_{AODV}.\text{simps} \) [simp del, code del]
lemmas \( \Gamma_{AODV}\_\text{simps} \) [simp, code] = \( \Gamma_{AODV}.\text{simps} \) [simplified]

fun \( \Gamma_{AODV}\_\text{skeleton} \)
where
"\( \Gamma_{AODV}\_\text{skeleton} \) \text{PAodv} = \text{seqp}\_\text{skeleton}(\Gamma_{AODV} \text{PAodv})"
"\( \Gamma_{AODV}\_\text{skeleton} \) \text{PNewPkt} = \text{seqp}\_\text{skeleton}(\Gamma_{AODV} \text{PNewPkt})"
"\( \Gamma_{AODV}\_\text{skeleton} \) \text{PPkt} = \text{seqp}\_\text{skeleton}(\Gamma_{AODV} \text{PPkt})"
"\( \Gamma_{AODV}\_\text{skeleton} \) \text{PRreq} = \text{seqp}\_\text{skeleton}(\Gamma_{AODV} \text{PRreq})"
"\( \Gamma_{AODV}\_\text{skeleton} \) \text{PRrep} = \text{seqp}\_\text{skeleton}(\Gamma_{AODV} \text{PRrep})"
"\( \Gamma_{AODV}\_\text{skeleton} \) \text{PRerr} = \text{seqp}\_\text{skeleton}(\Gamma_{AODV} \text{PRerr})"

lemma \( \Gamma_{AODV}\_\text{skeleton}\_\text{wf} \) [simp]:
"\text{wellformed} \( \Gamma_{AODV}\_\text{skeleton} \)"
proof (rule, intro allI)
fix pn pn'
show "call(pn') \notin \text{stermsl} (\Gamma_{AODV\text{-skeleton}} pn)"
  by (cases pn) simp_all
qed

declare \Gamma_{AODV\text{-skeleton}}.simpsons [simp del, code del]
lemmas \Gamma_{AODV\text{-skeleton}}.simpsons [simp, code]
  = \Gamma_{AODV\text{-skeleton}}.simpsons [simplified \Gamma_{AODV}\text{-simps} seqp\_skeleton\_simpsons]

lemma aodv_proc_cases [dest]:
  fixes p pn
  shows "p \in \text{ctermsl} (\Gamma_{AODV} pn) \Rightarrow
    (p \in \text{ctermsl} (\Gamma_{AODV} PAodv) \lor
     p \in \text{ctermsl} (\Gamma_{AODV} PNewPkt) \lor
     p \in \text{ctermsl} (\Gamma_{AODV} PPkt) \lor
     p \in \text{ctermsl} (\Gamma_{AODV} PRreq) \lor
     p \in \text{ctermsl} (\Gamma_{AODV} PRrep) \lor
     p \in \text{ctermsl} (\Gamma_{AODV} PRerr))"
  using assms
  by (cases pn) simp_all

definition \sigma_{AODV} :: "ip \Rightarrow (state \times (state, msg, pseqp, pseqp label) seqp) set"
where "\sigma_{AODV} i \equiv \{(aodv_init i, \Gamma_{AODV} PAodv)\}

abbreviation paodv
  :: "ip \Rightarrow (state \times (state, msg, pseqp, pseqp label) seqp, msg seq_action) automaton"
where "paodv i \equiv (init = \sigma_{AODV} i, trans = seqp_sos \Gamma_{AODV})"

lemma aodv_trans: "trans (paodv i) = seqp_sos \Gamma_{AODV}"
  by simp

lemma aodv_control_within [simp]: "control_within \Gamma_{AODV} (init (paodv i))"
  unfolding \sigma_{AODV}\_def
  by (rule control_withinI) (auto simp del: \Gamma_{AODV}\_simpsons)

lemma aodv_wf [simp]:
  "wellformed \Gamma_{AODV}"
  proof (rule, intro allI)
    fix pn pn'
    show "call(pn') \notin \text{stermsl} (\Gamma_{AODV} pn)"
      by (cases pn) simp_all
    qed

lemmas aodv_labels_not_empty [simp] = labels_not_empty [OF aodv_wf]

lemma aodv_ex_label [intro]: "\exists l. l \in labels \Gamma_{AODV} p"
  by (metis aodv_labels_not_empty all_not_in_conv)

lemma aodv_ex_labelE [elim]:
  assumes "\forall l \in labels \Gamma_{AODV} p. P l p"
  and "\exists p l. P l p \Rightarrow Q"
  shows "Q"
  using assms by (metis aodv_ex_label)

lemma aodv_simple_labels [simp]: "simple_labels \Gamma_{AODV}"
  proof
    fix p
    assume "p \in \text{subterms}(\Gamma_{AODV} pn)"
    thus "\exists l. labels \Gamma_{AODV} p = \{l\}"
      by (cases pn) (simp_all cong: seqp_congs | elim disjE)+
    qed

lemma \sigma_{AODV}\_labels [simp]: "(\xi, p) \in \sigma_{AODV} i \Rightarrow labels \Gamma_{AODV} p = \{PAodv-:0\}"
  unfolding \sigma_{AODV}\_def by simp
lemma aodv_init_kD_empty [simp]: "(ξ, p) ∈ σ_AODV i ⇒ kD (rt ξ) = {}"
    unfolding σ_AODV_def kD_def by simp

lemma aodv_init_sip_not_ip [simp]: "¬(sip (aodv_init i) = i)" by simp

lemma aodv_init_sip_not_ip' [simp]:
    assumes "(ξ, p) ∈ σ_AODV i"
    shows "sip ξ ≠ ip ξ"
    using assms unfolding σ_AODV_def by simp

lemma aodv_init_sip_not_i [simp]:
    assumes "(ξ, p) ∈ σ_AODV i"
    shows "sip ξ ≠ i"
    using assms unfolding σ_AODV_def by simp

lemma clear_locals_sip_not_ip':
    assumes "ip ξ = i"
    shows "¬(sip (clear_locals ξ) = i)"
    using assms by auto

Stop the simplifier from descending into process terms.

declare seqp_congs [cong]

Configure the main invariant tactic for AODV.

declare Γ_AODV_simps [cterms_env]
aodv_proc_cases [ctermsl_cases]
seq_invariant_ctermsI [OF aodv_wf aodv_control_within aodv_simple_labels aodv_trans, cterms_intros]
seq_step_invariant_ctermsI [OF aodv_wf aodv_control_within aodv_simple_labels aodv_trans, cterms_intros]

end

3.4 Invariant assumptions and properties

theory C_Aodv_Predicates
imports C_Aodv
begin

Definitions for expression assumptions on incoming messages and properties of outgoing messages.

abbreviation not_Pkt :: "msg ⇒ bool"
    where "not_Pkt m ≡ case m of Pkt _ _ _ ⇒ False | _ ⇒ True"

definition msg_sender :: "msg ⇒ ip"
    where "msg_sender m ≡ case m of Rreq _ _ _ _ _ _ _ ipc ⇒ ipc
             | Rrep _ _ _ _ ipc ⇒ ipc
             | Rerr _ ipc ⇒ ipc
             | Pkt _ _ ipc ⇒ ipc"

lemma msg_sender_simps [simp]:
    "⋀ hops rreqid dip dsn dsk oip osn sip.
      msg_sender (Rreq hops rreqid dip dsn dsk oip osn sip) = sip"
    "⋀ hops dip dsn oip sip. msg_sender (Rrep hops dip dsn oip sip) = sip"
    "⋀ dests sip. msg_sender (Rerr dests sip) = sip"
    "⋀ d dip sip. msg_sender (Pkt d dip sip) = sip"
    unfolding msg_sender_def by simp_all

definition msg_zhops :: "msg ⇒ bool"
    where "msg_zhops m ≡ case m of
               Rreq hopsc _ dip _ _ dsk oip osn sip ⇒ hopsc = 0 → oipc = sipc"
lemma `msg_zhops_simps [simp]:
"∀hops rreqid dip dsn dsnc oip osn sip.
  msg_zhops (Rreq hops rreqid dip dsn dsnc oip osn sip) = (hops = 0 → oip = sip)"
"∀hops dip dsn oip sip. msg_zhops (Rrep hops dip dsn oip sip) = (hops = 0 → dip = sip)"
"∀dests sip. msg_zhops (Rerr dests sip) = True"
"∀d dip. msg_zhops (Newpkt d dip) = True"
"∀d dip sip. msg_zhops (Pkt d dip sip) = True"
unfolding `msg_zhops_def by simp_all

definition `rreq_rrep_sn :: (msg ⇒ bool)
where `rreq_rrep_sn m ≡
case m of
  Rreq _ _ _ _ _ _ osnc _ ⇒ osnc ≥ 1 |
  Rrep _ _ dsnc _ _ ⇒ dsnc ≥ 1 |
  _ ⇒ True"
lemma `rreq_rrep_sn_simps [simp]:
"∀hops rreqid dip dsn dsnc oip osn sip.
  rreq_rrep_sn (Rreq hops rreqid dip dsn dsnc oip osn sip) = (osn ≥ 1)
| Rrep _ _ dsnc _ _ ⇒ dsnc ≥ 1
| _ ⇒ True"

lemma `rreq_rrep_sn_simps [simp]:
"∀hops rreqid dip dsn dsnc oip osn sip.
  \ rreq_rrep_sn (Rreq hops rreqid dip dsn dsnc oip osn sip) = (osn ≥ 1)"
"∀hops dip dsn oip sip. \ rreq_rrep_sn (Rrep hops dip dsn oip sip) = (dsn ≥ 1)"
"∀dests sip. \ rreq_rrep_sn (Rerr dests sip) = True"
"∀d dip. \ rreq_rrep_sn (Newpkt d dip) = True"
"∀d dip sip. \ rreq_rrep_sn (Pkt d dip sip) = True"
unfolding `rreq_rrep_sn_def by simp_all

definition `rreq_rrep_fresh :: rt ⇒ msg ⇒ bool
where `rreq_rrep_fresh crt m ≡
case m of
  Rreq hopsc _ _ _ _ oipc osnc ipcc ⇒
  (ipcc ≠ oipc → oipc ∈ kD(crt) ∧ (sqn crt oipc > osnc ∨ (sqn crt oipc = osnc ∧ the (dhops crt oipc) ≤ hopsc ∧ the (flag crt oipc) = val)))
| Rrep hopsc dip dsnc _ ipcc ⇒ (ipcc ≠ dip → dip ∈ kD(crt) ∧ sqn crt dip = dsnc ∧ the (dhops crt dip) = hopsc ∧ the (flag crt dip) = val))
| _ ⇒ True"
lemma `rreq_rrep_fresh [simp]:
"∀hops rreqid dip dsn dsnc oip osn sip.
  rreq_rrep_fresh crt (Rreq hops rreqid dip dsn dsnc oip osn sip) =
  (sip ≠ oip → oip ∈ kD(crt))
  ∧ (sqn crt oip > osn ∨ (sqn crt oip = osn ∧ the (dhops crt oip) ≤ hopsc ∧ the (flag crt oip) = val)))"
"∀hops dip dsn oip sip. \ rreq_rrep_fresh crt (Rrep hops dip dsn oip sip) =
  (sip ≠ dip → dip ∈ kD(crt) ∧ sqn crt dip = dsnc ∧ the (dhops crt dip) = hopsc ∧ the (flag crt dip) = val)"
"∀dests sip. \ rreq_rrep_fresh crt (Rerr dests sip) = True"
"∀d dip. \ rreq_rrep_fresh crt (Newpkt d dip) = True"
"∀d dip sip. \ rreq_rrep_fresh crt (Pkt d dip sip) = True"
unfolding `rreq_rrep_fresh_def by simp_all

definition `rerr_invalid :: rt ⇒ msg ⇒ bool
where `rerr_invalid crt m ≡
case m of
  Rerr destsc _ ⇒ (∀ripc∈dom(destsc).
    (ripc ∈ iD(crt) ∧ the (destsc ripc) = sqn crt ripc))
| _ ⇒ True"
lemma `rerr_invalid [simp]:
"∀hops rreqid dip dsn dsnc oip osn sip.
rerr_invalid crt (Rreq hops rreqid dip dsn dsk oip osn sip) = True
"\wedge hops dip dsn oip sip. rerr_invalid crt (Rreq hops dip dsn oip sip) = True"
"\wedge dests sip. rerr_invalid crt (Rerr dests sip) = (\forall rip \in dom(dests).
  rip \in id(crt) \land the (dests rip) = sqn crt rip)"
"\wedge d dip. rerr_invalid crt (Newpkt d dip) = True"
"\wedge d dip sip. rerr_invalid crt (Pkt d dip sip) = True"

unfolding rerr_invalid_def by simp_all

definition
  initmissing :: "(nat \Rightarrow state option) \times 'a \Rightarrow (nat \Rightarrow state) \times 'a"
where
  "initmissing \sigma = (\lambda i. case (fst \sigma) i of None \Rightarrow aodv_init i | Some \sigma \Rightarrow \sigma, snd \sigma)"

lemma not_in_net_ips_fst_init_missing [simp]:
  assumes "i /\in net_ips \sigma"
  shows "fst (initmissing (netgmap fst \sigma)) i = aodv_init i"
using assms unfolding initmissing_def by simp

lemma fst_initmissing_netgmap_pair_fst [simp]:
  "fst (initmissing (netgmap (\lambda (p, q). (fst (id p), snd (id p), q)) s))
  = fst (initmissing (netgmap fst s))"

unfolding initmissing_def by auto

We introduce a streamlined alternative to initmissing with netgmap to simplify invariant statements and thus facilitate their comprehension and presentation.

lemma fst_initmissing_netgmap_default_aodv_init_netlift:
  "fst (initmissing (netgmap fst s)) = default aodv_init (netlift fst s)"
unfolding initmissing_def default_def
by (simp add: fst_netgmap_netlift del: One_nat_def)

definition
  netglobal :: "((nat \Rightarrow state) \Rightarrow bool) \Rightarrow ((state \times 'b) \times 'c) \Rightarrow bool"
where
  "netglobal P \equiv (\lambda s. P (default aodv_init (netlift fst s)))"

end

3.5 Quality relations between routes

theory C_Fresher
imports C_Aodv_Data
begin

3.5.1 Net sequence numbers

On individual routes

definition
  nsqn_r :: "r \Rightarrow sqn"
where
  "nsqn_r r \equiv if \pi_4 (r) = val \lor \pi_2 (r) = 0 then \pi_2 (r) else (\pi_2 (r) - 1)"

lemma nsqn_r_def':
  "nsqn_r r = (if \pi_4 (r) = inv then \pi_2 (r) - 1 else \pi_2 (r))"
unfolding nsqn_r_def by simp

lemma nsqn_r_zero [simp]:
  "\wedge dsn dsk flag hops nhhip. nsqn_r (0, dsk, flag, hops, nhhip) = 0"
unfolding nsqn_r_def by clarsimp

lemma nsqn_r_val [simp]:
  "\wedge dsn dsk hops nhhip. nsqn_r (dsn, dsk, val, hops, nhhip) = dsn"
unfolding nsqn_r_def by clarsimp
lemma nsqn_r_inv [simp]:
"∀dsn dsk hops nhip. nsqn_r (dsn, dsk, inv, hops, nhip) = dsn - 1"
unfolding nsqn_r_def by clarsimp

lemma nsqn_r_lte_dsn [simp]:
"∀dsn dsk flag hops nhip. nsqn_r (dsn, dsk, flag, hops, nhip) ≤ dsn"
unfolding nsqn_r_def by clarsimp

On routes in routing tables

definition nsqn :: "rt ⇒ ip ⇒ sqn"
where
"nsqn ≡ λrt dip. case σroute(rt, dip) of None ⇒ 0 | Some r ⇒ nsqn_r(r)"

lemma nsqn_sqn_def:
"∀rt dip. nsqn rt dip = (if flag rt dip = Some val ∨ sqn rt dip = 0
then sqn rt dip else sqn rt dip - 1)"
unfolding nsqn_def sqn_def by (clarsimp split: option.split) auto

lemma not_in_kD_nsqn [simp]:
assumes "dip /∈ kD(rt)"
shows "nsqn rt dip = 0"
using assms unfolding nsqn_def by simp

lemma kD_nsqn:
assumes "dip ∈ kD(rt)"
shows "nsqn rt dip = nsqn_r(the (σroute(rt, dip)))"
using assms [THEN kD_Some] unfolding nsqn_def byclarsimp

lemma nsqnr_r_flag_pred [simp, intro]:
fixes dsn dsk flag hops nhip
assumes "P (nsqn_r (dsn, dsk, val, hops, nhip))"
and "P (nsqn_r (dsn, dsk, inv, hops, nhip))"
shows "P (nsqn_r (dsn, dsk, flag, hops, nhip))"
using assms by (cases flag) auto

lemma sqn_nsqn:
"∀rt dip. sqn rt dip - 1 ≤ nsqn rt dip"
unfolding sqn_def nsqn_def by (clarsimp split: option.split)

lemma nsqn_sqn: "nsqn rt dip ≤ sqn rt dip"
unfolding sqn_def nsqn_def by (cases "rt dip") auto

lemma val_nsqn_sqn [elim, simp]:
assumes "ip ∈ kD(rt)"
and "the (flag rt ip) = val"
shows "nsqn rt ip = sqn rt ip"
using assms unfolding nsqn_sqn_def by auto

lemma vD_nsqn_sqn [elim, simp]:
assumes "ip ∈ vD(rt)"
shows "nsqn rt ip = sqn rt ip"
proof -
from (ip ∈ vD(rt)) have "ip ∈ kD(rt)"
and "the (flag rt ip) = val" by auto
thus ?thesis ..
qed

lemma inv_nsqn_sqn [elim, simp]:
assumes "ip ∈ kD(rt)"
and "the (flag rt ip) = inv"
shows "nsqn rt ip = sqn rt ip - 1"
using assms unfolding nsqn_sqn_def by auto
lemma $iD_{nsqn,sqn}$ [elim, simp]:
assumes "$ip \in iD(rt)$"
shows "$nsqn rt ip = sqn rt ip - 1$"
proof -
  from $ip \in iD(rt)$ have "$ip \in kD(rt)$"
  and "the (flag rt ip) = inv" by auto
  thus $\exists$thesis ..
qed

lemma $nsqn_{update\_changed\_kno\_val}$ [simp]: "\(rt \; ip \; dsn \; dsk \; hops \; nhip.\) 
rt \neq update rt ip (dsn, kno, val, hops, nhip) 
\implies nsqn (update rt ip (dsn, kno, val, hops, nhip)) ip = dsn"
using assms unfolding nsqn_,_def update_def
by (clarsimp simp: kD_nsqn split: option.split_asm option.split split_if_asm)
  (metis fun_upd_triv)

lemma $nsqn_{update\_other}$ [simp]:
fixes dsn dsk flag hops dip nhip rt ip
assumes "dip \neq ip"
shows "nsqn (update rt ip (dsn, dsk, flag, hops, nhip)) dip = nsqn rt dip"
using assms unfolding nsqn_def
by (clarsimp split: option.split)

lemma $nsqn_{invalidate\_eq}$:
assumes "dip \in kD(rt)"
and "dests dip = Some rsn"
shows "nsqn (invalidate rt dests) dip = rsn - 1"
using assms
proof -
  from assms obtain dsk hops nhip pre
  where "invalidate rt dests dip = Some (rsn, dsk, inv, hops, nhip)"
    unfolding invalidate_def
  by auto
  moreover from $dip \in kD(rt)$ have "dip \in kD(invalidate rt dests)" by simp
  ultimately show $\exists$thesis
    using $\langle$dests dip = Some rsn$\rangle$ by simp
qed

lemma $nsqn_{invalidate\_other}$ [simp]:
assumes "dip \in kD(rt)"
and "dip \notin dom dests"
shows "nsqn (invalidate rt dests) dip = nsqn rt dip"
using assms by (clarsimp simp add: kD_nsqn)

3.5.2 Comparing routes

definition $fresher$ :: "r \Rightarrow r \Rightarrow bool" ("(_/ \sqsubseteq _)") [51, 51] 50
where
"fresher r r' \equiv ((nsqn, r < nsqn, r') \lor (nsqn, r = nsqn, r' \land \pi_5(r) \geq \pi_5(r')))"

lemma $fresher_{I1}$ [intro]:
assumes "nsqn, r < nsqn, r'" 
shows "$r \sqsubseteq r'" 
unfolding fresher_def using assms by simp

lemma $fresher_{I2}$ [intro]:
assumes "nsqn, r = nsqn, r'" 
and "$\pi_5(r) \geq \pi_5(r')$"
shows "$r \sqsubseteq r'" 
unfolding fresher_def using assms by simp

lemma $fresher_{I}$ [intro]:

279
assumes 

\[ \text{(nsqn}_r \ r < \text{nsqn}_r \ r') \lor (\text{nsqn}_r \ r = \text{nsqn}_r \ r' \land \pi_5(r) \geq \pi_5(r')) \]

shows "\( r \sqsubseteq r' \)"

unfolding freshness_def using assms .

lemma freshnessE [elim]:
assumes "\( r \sqsubseteq r' \)"
and "\( \text{nsqn}_r \ r < \text{nsqn}_r \ r' \Rightarrow P \ r \ r' \)"
and "\( \text{nsqn}_r \ r = \text{nsqn}_r \ r' \land \pi_5(r) \geq \pi_5(r') \Rightarrow P \ r \ r' \)"
shows "\( P \ r \ r' \)"
using assms unfolding freshness_def by auto

lemma freshness_refl [simp]: "r \sqsubseteq r"
unfolding freshness_def by simp

lemma freshness_trans [elim, trans]:
"[ x \sqsubseteq y; y \sqsubseteq z ] \Rightarrow x \sqsubseteq z"
unfolding freshness_def by auto

lemma not_fresher_trans [elim, trans]:
"[ \neg(x \sqsubseteq y); \neg(z \sqsubseteq x) ] \Rightarrow \neg(z \sqsubseteq y)"
unfolding freshness_def by auto

lemma freshness DSM_flag_hops_const [simp]:
fixes dsn dsk dsk' flag hops nhip nhip'
shows "(dsn, dsk, flag, hops, nhip) \sqsubseteq (dsn, dsk', flag, hops, nhip')"
unfolding freshness_def by (cases flag) simp_all

3.5.3 Comparing routing tables

definition
rt_fresher :: "ip \Rightarrow rt \Rightarrow rt \Rightarrow bool"
where
"rt_fresher \equiv \lambda dip \ rt \ rt'. (the (\sigma_{route}(rt, dip))) \sqsubseteq (the (\sigma_{route}(rt', dip)))"

abbreviation
rt_fresher_syn :: "rt \Rightarrow ip \Rightarrow rt \Rightarrow bool" ("(_/ \sqsubseteq \_)") [51, 999, 51] 50
where
"rt1 \sqsubseteq\_ rt2 \equiv rt_fresher i rt1 rt2"

lemma rt_fresher_def':
"(rt1 \sqsubseteq\_ rt2) = (\text{nsqn}_r (the (rt1 i)) < \text{nsqn}_r (the (rt2 i)) \lor \text{nsqn}_r (the (rt1 i)) = \text{nsqn}_r (the (rt2 i)) \land \pi_5 (the (rt2 i)) \leq \pi_5 (the (rt1 i)))"
unfolding rt_fresher_def freshness_def by (rule refl)

lemma single_rt_fresher [intro]:
assumes "the (rt1 ip) \sqsubseteq the (rt2 ip)"
shows "rt1 \sqsubseteq\_rt2"
using assms unfolding rt_fresher_def .

lemma rt_fresher_single [intro]:
assumes "rt1 \sqsubseteq\_ ip rt2"
shows "the (rt1 ip) \sqsubseteq the (rt2 ip)"
using assms unfolding rt_fresher_def .

lemma rt_fresher_def2: 
assumes "dip \in kD(rt1)"
and "dip \in kD(rt2)"
shows "(rt1 \sqsubseteq\_ dip rt2) = (\text{nsqn}_r rt1 dip < \text{nsqn}_r rt2 dip \lor (\text{nsqn}_r rt1 dip = \text{nsqn}_r rt2 dip \land the (dhops rt1 dip) \geq the (dhops rt2 dip)))"
using assms unfolding rt_fresher_def freshness_def by (simp add: kD_nsqn proj5_eq_dhops)

lemma rt_fresherI1 [intro]:
assumes "dip \in kD(rt1)"

and "dip ∈ kD(rt2)"
and "nsqn rt1 dip < nsqn rt2 dip"
shows "rt1 ⊑ dip rt2"

unfolding rt_fresher_def2 [OF assms(1-2)] using assms(3) by simp

lemma rt_fresherI2 [intro]:
assumes "dip ∈ kD(rt1)"
and "dip ∈ kD(rt2)"
and "nsqn rt1 dip = nsqn rt2 dip"
and "the (dhops rt1 dip) ≥ the (dhops rt2 dip)"
shows "rt1 ⊑ dip rt2"

unfolding rt_fresher_def2 [OF assms(1-2)] using assms(3-4) by simp

lemma rt_fresherE [elim]:
assumes "rt1 ⊑ dip rt2"
and "dip ∈ kD(rt1)"
and "dip ∈ kD(rt2)"
and 
"[ nsqn rt1 dip < nsqn rt2 dip ] \implies P rt1 rt2 dip"
and "[ nsqn rt1 dip = nsqn rt2 dip; the (dhops rt1 dip) ≥ the (dhops rt2 dip) ] \implies P rt1 rt2 dip"
shows "P rt1 rt2 dip"
using assms(1) unfolding rt_fresher_def2 [OF assms(2-3)]
using assms(4-5) by auto

lemma rt_fresher_refl [simp]: "rt ≈ dip rt"
unfolding rt_fresher_def by simp

lemma rt_fresher_trans [elim, trans]:
assumes "rt1 ⊑ dip rt2"
and "rt2 ⊑ dip rt3"
shows "rt1 ⊑ dip rt3"
using assms unfolding rt_fresher_def by auto

lemma rt_fresher_if_Some [intro!]:
assumes "the (rt dip) ⊑ r"
shows "rt ⊑ dip (\ip. if ip = dip then Some r else rt ip)"
using assms unfolding rt_fresher_def by simp

definition rt_fresh_as :: "ip ⇒ rt ⇒ rt ⇒ bool"
where
"rt_fresh_as ≡ λdip rt1 rt2. (rt1 ⊑ dip rt2) ∧ (rt2 ⊑ dip rt1)"

abbreviation
rt_fresh_as_syn :: "rt ⇒ ip ⇒ rt ⇒ bool" ("(_/≈_/_)" [51, 999, 51] 50)
where
"rt1 ≈ i rt2 ≡ rt_fresh_as i rt1 rt2"

lemma rt_fresh_as_refl [simp]: "∀rt dip. rt ≈ dip rt"
unfolding rt_fresh_as_def by simp

lemma rt_fresh_as_trans [simp, intro, trans]:
"∀rt1 rt2 rt3 dip. [ rt1 ≈ dip rt2; rt2 ≈ dip rt3 ] \implies rt1 ≈ dip rt3"
unfolding rt_fresh_as_def rt_fresher_def
by (metis (mono_tags) fresher_trans)

lemma rt_fresh_asI [intro!]:
assumes "rt1 ⊑ dip rt2"
and "rt2 ⊑ dip rt1"
shows "rt1 ≈ dip rt2"
using assms unfolding rt_fresh_as_def by simp

lemma rt_fresh_as_fresherI [intro]:
assumes "dip∈kD(rt1)"
and "dip∈kD(rt2)"
and "the (rt1 dip) ⊑ the (rt2 dip)"
and "the (rt2 dip) ⊑ the (rt1 dip)"
sows "rt1 ≈_{dip} rt2"

using asms unfolding rt_fresh_as_def
by (clarsimp dest!: single_rt_fresher)

lemma nsqn_rt_fresh_asI:
assumes "dip ∈ kD(rt)"
and "dip ∈ kD(rt')"
and "nsqn rt dip = nsqn rt' dip"
and "π₅(the (rt dip)) = π₅(the (rt' dip))"
sows "rt ≈_{dip} rt'"

proof
from asms(1-2,4) have dhops': "the (dhops rt' dip) ≤ the (dhops rt dip)"
by (simp add: proj5_eq_dhops)
with asms(1-3) show "rt ⊑_{dip} rt'"
by (rule rt_fresherI2)

next
from asms(1-2,4) have dhops: "the (dhops rt dip) ≤ the (dhops rt' dip)"
by (simp add: proj5_eq_dhops)
with asms(2,1) asms(3) [symmetric] show "rt' ⊑_{dip} rt"
by (rule rt_fresherI2)

qed

lemma rt_fresh_asE [elim]:
assumes "rt1 ≈_{dip} rt2"
and "[ rt1 ⊑_{dip} rt2; rt2 ⊑_{dip} rt1 ] ⇒ P rt1 rt2 dip"
sows "P rt1 rt2 dip"

using asms unfolding rt_fresh_as_def by simp

lemma rt_fresh_asD1 [dest]:
assumes "rt1 ≈_{dip} rt2"
shows "rt1 ⊑_{dip} rt2"

using asms unfolding rt_fresh_as_def by simp

lemma rt_fresh_asD2 [dest]:
assumes "rt1 ≈_{dip} rt2"
shows "rt2 ⊑_{dip} rt1"

using asms unfolding rt_fresh_as_def by simp

lemma rt_fresh_as_sym:
assumes "rt1 ≈_{dip} rt2"
shows "rt2 ≈_{dip} rt1"

using asms unfolding rt_fresh_as_def by simp

lemma not_rt_fresh_asI1 [intro]:
assumes "¬ (rt1 ⊑_{dip} rt2)"
shows "¬ (rt1 ≈_{dip} rt2)"

proof
assume "rt1 ⊑_{dip} rt2"

with "¬ (rt1 ⊑_{dip} rt2)" show False ..

qed

lemma not_rt_fresh_asI2 [intro]:
assumes "¬ (rt2 ⊑_{dip} rt1)"
shows "¬ (rt1 ≈_{dip} rt2)"

proof
assume "rt1 ≈_{dip} rt2"

with "¬ (rt2 ⊑_{dip} rt1)" show False ..

qed

lemma not_single_rt_fresher [elim]:
assumes "¬(the (rt1 ip) ⊑ the (rt2 ip))"
  shows "¬(rt1 ⊑ip rt2)"
proof
  assume "rt1 ⊑ip rt2"
  hence "the (rt1 ip) ⊑ the (rt2 ip)"
  with ¬(the (rt1 ip) ⊑ the (rt2 ip)) show False ..
qed

lemmas not_single_rt_fresh_asI1 [intro] = not_rt_fresh_asI1 [OF not_single_rt_fresher]
lemmas not_single_rt_fresh_asI2 [intro] = not_rt_fresh_asI2 [OF not_single_rt_fresher]

lemma not_rt_fresher_single [elim]:
  assumes "¬(rt1 ⊑ ip rt2)"
  shows "¬(the (rt1 ip) ⊑ the (rt2 ip))"
proof
  assume "the (rt1 ip) ⊑ the (rt2 ip)"
  hence "rt1 ⊑ ip rt2" ..
  with ¬(rt1 ⊑ip rt2) show False ..
qed

lemma rt_fresh_as_nsqnr:
  assumes "dip ∈ kD(rt1)"
  and "dip ∈ kD(rt2)"
  and "rt1 ≈ dip rt2"
  shows "nsqn r (the (rt2 dip)) = nsqn r (the (rt1 dip))"
using assms unfolding rt_fresh_as_def
by (auto simp: rt_fresher_def2 [OF ⟨dip ∈ kD(rt1)⟩ ⟨dip ∈ kD(rt2)⟩]
  kD_nsqn [OF ⟨dip ∈ kD(rt1)⟩]
  kD_nsqn [OF ⟨dip ∈ kD(rt2)⟩])

lemma rt_fresher_mapupd [intro!]:
  assumes "dip∈kD(rt)"
  and "the (rt dip) ⊑ r"
  shows "rt ⊑dip rt(dip ↦→ r)"
using assms unfolding rt_fresher_def by simp

lemma rt_fresher_map_update_other [intro!]:
  assumes "dip∈kD(rt)"
  and "dip ≠ ip"
  shows "rt ⊑dip rt(ip ↦→ r)"
using assms unfolding rt_fresher_def by simp

lemma rt_fresher_update_other [simp]:
  assumes inkD: "dip∈kD(rt)"
  and "dip ≠ ip"
  shows "rt ⊑dip update rt ip r"
using assms unfolding update_def
by (clarsimp split: option.split) (fastforce)

theorem rt_fresher_update [simp]:
  assumes "dip∈kD(rt)"
  and "the (dhops rt dip) ≥ 1"
  and "update_arg_wf r"
  shows "rt ⊑dip update rt ip r"
proof (cases "dip = ip")
  assume "dip ≠ ip" with ⟨dip∈kD(rt)⟩ show ?thesis
  by (rule rt_fresher_update_other)
next
  assume "dip = ip"
from ⟨dip∈kD(rt)⟩ obtain dsn_n dsk_n f_n hops_n nhip_n
  where rt [simp]: "the (rt dip) = (dsn_n, dsk_n, f_n, hops_n, nhip_n)"
by (metis prod_cases5)
with \( \langle \text{dhops } rt \ dip \rangle \geq 1 \) and \( \langle \text{dip} \in kD(rt) \rangle \) have "hops \_n \geq 1"
by \( \text{metis proj5_eq_dhops projs(4)} \)
from \( \langle \text{dip} \in kD(rt) \rangle \) \( \text{rtn} \) have [simp]: "sqn rt dip = dsn\_n"
and [simp]: "the (dhops rt dip) = hops\_n"
and [simp]: "the (flag rt dip) = f\_n"
by (simp add: sqn_def proj5_eq_dhops [symmetric]
proj4_eq_flag [symmetric])+
from \( \langle \text{update_arg_wf } r \rangle \) have "(dsn\_n, dsk\_n, f\_n, hops\_n, nhip\_n)
\subseteq \text{the (update } rt \ dip \ (0, unk, val, Suc 0, nhip) dip)"
proof (rule wf_r_cases)
fix nhip pre
from \( \langle \text{hops } n \geq 1 \rangle \) have "\( \forall \text{pre'. (dsn}\_n, dsk\_n, f\_n, hops\_n, nhip\_n) \}\n\subseteq \text{the (update } rt \ dip \ (0, unk, val, Suc 0, nhip) dip)"
unfolding fresher_def sqn_def by (cases f\_n) auto
thus "(dsn\_n, dsk\_n, f\_n, hops\_n, nhip\_n)
\subseteq \text{the (update } rt \ dip \ (0, unk, val, Suc 0, nhip) dip)"
unfolding fresher_def sqn_def by simp
next
fix dsn :: sqn and hops nhip pre
assume "0 < dsn"
show "(dsn\_n, dsk\_n, f\_n, hops\_n, nhip\_n)
\subseteq \text{the (update } rt \ dip \ (dsn, kno, val, hops, nhip) dip)"
proof (rule update_cases_kD [OF _ \( \langle \text{dip} \in kD(rt) \rangle \)], simp_all add: \( \langle 0 < dsn \rangle \))
assume "dsn\_n < dsn"
thus "(dsn\_n, dsk\_n, f\_n, hops\_n, nhip\_n)
\subseteq (dsn, kno, val, hops, nhip)"
unfolding fresher_def by auto
next
assume "dsn\_n = dsn"
and "hops < hops\_n"
thus "(dsn, dsk\_n, f\_n, hops\_n, nhip\_n)
\subseteq (dsn, kno, val, hops, nhip)"
unfolding fresher_def nsqn\_r_def by simp
next
assume "dsn\_n = dsn"
with \( 0 < dsn \)
show "(dsn, dsk\_n, inv, hops\_n, nhip\_n)
\subseteq (dsn, kno, val, hops, nhip)"
unfolding fresher_def by simp
qed
qed

hence "rt \subseteq dip update rt dip r"
by (rule single_rt_fresher, simp)
with \( \langle dip = ip \rangle \) show ?thesis by simp
qed

theorem rt_fresher_invalidate [simp]:
assumes "\( \text{dip} \in kD(rt) \rangle \)"
and indests: "\( \forall \text{rip} \in \text{dom(dests)}. rip \in vD(rt) \wedge sqn rt rip < the (dests rip) \)"
shows "rt \subseteq dip invalidate rt dests"
proof (cases "\( \text{dip} \in \text{dom(dests)} \)"
assume "\( \text{dip} \in \text{dom(dests)} \)"
thus ?thesis using \( \langle \text{dip} \in kD(rt) \rangle \)
by (rule single_rt_fresher, simp)
next
assume "\( \text{dip} \in \text{dom(dests)} \)"
moreover with indests have "\( \text{dip} \in vD(rt) \)"
and "sqn rt dip < the (dests dip)"
by auto
ultimately show ?thesis
unfolding invalidate_def sqn_def
by (rule single_rt_fresher, auto simp: fresher_def)
qed
lemma \textit{nsqn\_invalidate} [simp]:
assumes "\textit{dip\_\in\textit{kD}(rt)}"
and "\textit{dip\_\in\textit{dom}(dests)}"
shows "\textit{nsqn} \textit{\_invalidate} (\textit{the (invalidate rt dests dip)}) = \textit{the (dests dip)} - 1"
using \textit{assms} unfolding \textit{invalidate\_def} by auto

lemma \textit{rt\_fresh\_as\_inc\_invalidate} [simp]:
assumes "\textit{dip\_\in\textit{kD}(rt)}"
and "\\forall \textit{rip\_\in\textit{dom}(dests). rip\_\in\textit{vD}(rt) \land \textit{the (dests rip)} = \textit{inc (sqn rt rip)}}"
shows "\textit{rt \approx dip invalidate rt dests}"
proof (cases "\textit{dip\_\in\textit{dom}(dests)}")
assume "\textit{dip\_/\notin\textit{dom}(dests)}"
with \textit{\langle dip\_\in\textit{kD}(rt) \rangle} have "\textit{dip\_\in\textit{kD}(invalidate rt dests)}" by simp
with \textit{\langle dip\_\in\textit{kD}(rt) \rangle} show \textit{?thesis} by rule (simp_all add: \textit{\langle dip\_\notin\textit{dom}(dests) \rangle})
next
assume "\textit{dip\_\in\textit{dom}(dests)}"
with \\\\assms(2) have "\textit{sqn rt dip = nsqn r (the (invalidate rt dests dip))}" by auto
moreover then have "\textit{dip\_\in\textit{kD}(invalidate rt dests)}" by simp
ultimately show \textit{?thesis} proof (rule \textit{nsqn\_rt\_fresh\_asI})
from \textit{\langle dip\_\in\textit{kD}(rt) \rangle} have "\textit{nsqn rt dip = nsqn r (the (invalidate rt dests dip))}" by simp
also have "\textit{sqn rt dip = nsqn r (the (validate rt dests dip))}" by autolemma \textit{rt\_fresher\_inc\_invalidate} [simp] = \textit{rt\_fresh\_as\_inc\_invalidate} [THEN \textit{rt\_fresh\_asD1}]

3.5.4 Strictly comparing routing tables

definition \textit{rt\_strictly\_fresher} :: "ip \Rightarrow rt \Rightarrow rt \Rightarrow bool"
where
"\textit{rt\_strictly\_fresher} \equiv \lambda dip rt1 rt2. (rt1 \sqsubseteq dip rt2) \land \neg(\textit{rt1 \approx dip rt2})"

abbreviation
\textit{rt\_strictly\_fresher\_syn} :: "rt \Rightarrow ip \Rightarrow rt \Rightarrow bool" ("(/ _ _ _)"
where
"rt1 \sqsubseteq \_ rt2 \equiv \textit{rt\_strictly\_fresher i rt1 rt2}"

lemma \textit{rt\_strictly\_fresher\_def'}:
"rt1 \sqsubseteq \_ rt2 = ((rt1 \sqsubseteq dip rt2) \land \neg(rt2 \sqsubseteq dip rt1))"

unfolding \textit{rt\_strictly\_fresher\_def} rt\_fresh\_as\_def\_by\_auto

lemma \textit{rt\_strictly\_fresherI'} [intro]:
assumes "rt1 \sqsubseteq dip rt2"
and "\neg(rt2 \sqsubseteq dip rt1)"
shows "rt1 \sqsubseteq \_ rt2"
using \assms unfolding \textit{rt\_strictly\_fresher\_def'} by simp

lemma \textit{rt\_strictly\_fresherE'} [elim]:

285
assumes "rt1 ⊏ i rt2"
  and "[ rt1 ⊑ i rt2; ¬(rt2 ⊑ i rt1) ] ⇒ P rt1 rt2 i"
  shows "P rt1 rt2 i"
using assms unfolding rt_strictly_fresher_def'' by simp

lemma rt_strictly_fresherI [intro]:
  assumes "rt1 ⊏ i rt2"
  and "¬(rt1 ≈ i rt2)"
  shows "rt1 ⊑ i rt2"
unfolding rt_strictly_fresher_def using assms ..

lemmas rt_strictly_fresher_singleI [elim] = rt_strictly_fresherI [OF single_rt_fresher]

lemma rt_strictly_fresherE [elim]:
  assumes "rt1 ⊏ i rt2"
  and "[ rt1 ⊑ i rt2; ¬(rt1 ≈ i rt2) ] ⇒ P rt1 rt2 i"
  shows "P rt1 rt2 i"
using assms(1) unfolding rt_strictly_fresher_def
by rule (erule(1) assms(2))

lemma rt_strictly_fresher_def':
  "rt1 ⊏ i rt2 =
  (nsqr_r (the (rt1 i)) < nsqr_r (the (rt2 i))
  ∨ (nsqr_r (the (rt1 i)) = nsqr_r (the (rt2 i)) ∧ π_5(the (rt1 i)) > π_5(the (rt2 i)))"
unfolding rt_strictly_fresher_def'' rt_fresher_def fresher_def by auto

lemma rt_strictly_fresher_fresherD [dest]:
  assumes "rt1 ⊏ dip rt2"
  shows "the (rt1 dip) ⊑ the (rt2 dip)"
using assms unfolding rt_strictly_fresher_def rt_fresher_def by auto

lemma rt_strictly_fresher_not_fresh_asD [dest]:
  assumes "rt1 ⊏ dip rt2"
  shows "¬ rt1 ≈ dip rt2"
using assms unfolding rt_strictly_fresher_def by auto

lemma rt_strictly_fresher_trans [elim, trans]:
  assumes "rt1 ⊏ dip rt2"
  and "rt2 ⊏ dip rt3"
  shows "rt1 ⊏ dip rt3"
using assms proof -
  from rt1 ⊏ dip rt2 obtain "the (rt1 dip) ⊑ the (rt2 dip)" by auto
  also from rt2 ⊏ dip rt3 obtain "the (rt2 dip) ⊑ the (rt3 dip)" by auto
  finally have "the (rt1 dip) ⊑ the (rt3 dip)".
  moreover have "¬ (rt1 ≈ dip rt3)"
  proof -
    from rt1 ⊏ dip rt2 obtain "¬(the (rt2 dip) ⊑ the (rt1 dip))" by auto
    also from rt2 ⊏ dip rt3 obtain "¬(the (rt3 dip) ⊑ the (rt2 dip))" by auto
    finally have "¬(the (rt3 dip) ⊑ the (rt1 dip))".
    thus ?thesis ..
  qed
  ultimately show "rt1 ⊏ dip rt3" ..
  qed

lemma rt_strictly_fresher_irefl [simp]: "¬ (rt ⊏ dip rt)"
using assms unfolding rt_strictly_fresher_def
by clarsimp

lemma rt_fresher_trans_rt_strictly_fresher [elim, trans]:
  assumes "rt1 ⊏ dip rt2"
  and "rt2 ⊏ dip rt3"
  shows "rt1 ⊏ dip rt3"
proof -
from $rt1 \sqsubseteq dip rt2$: have "$rt1 \sqsubseteq dip rt2$"
  and "$(rt2 \sqsubseteq dip rt1)$"
  unfolding $rt\_strictly\_fresher\_def''$ by auto
from this(1) and $rt2 \sqsubseteq dip rt3$ have "$rt1 \sqsubseteq dip rt3$ ..

moreover from $(\neg(rt2 \sqsubseteq dip rt1))$ have "$(\neg(rt3 \sqsubseteq dip rt1))$"
proof (rule contrapos_nn)
  assume "$rt3 \sqsubseteq dip rt1"
  with $rt2 \sqsubseteq dip rt3$ show "$rt2 \sqsubseteq dip rt1$ ..
qed

ultimately show "$rt1 \sqsubseteq dip rt3$"
  unfolding $rt\_strictly\_fresher\_def''$ by auto
qed

lemma $rt\_fresher\_trans\_rt\_strictly\_fresher' [elim, trans]$: 
assumes "$rt1 \sqsubseteq dip rt2$"
  and "$rt2 \sqsubseteq dip rt3$"
shows "$rt1 \sqsubseteq dip rt3$"
proof -
  from $rt2 \sqsubseteq dip rt3$: have "$rt2 \sqsubseteq dip rt3$"
    and "$(rt3 \sqsubseteq dip rt2)$"
    unfolding $rt\_strictly\_fresher\_def''$ by auto
  from $rt1 \sqsubseteq dip rt2$: and this(1) have "$rt1 \sqsubseteq dip rt3$ ..
moreover from $(\neg(rt3 \sqsubseteq dip rt2))$ have "$(\neg(rt3 \sqsubseteq dip rt1))$"
proof (rule contrapos_nn)
  assume "$rt3 \sqsubseteq dip rt1"
  thus "$rt3 \sqsubseteq dip rt2$" using $rt1 \sqsubseteq dip rt2$ ..
qed

ultimately show "$rt1 \sqsubseteq dip rt3$"
  unfolding $rt\_strictly\_fresher\_def''$ by auto
qed

lemma $rt\_fresher\_imp\_nsqn\_le$: 
assumes "$rt1 \sqsubseteq ip rt2$"
  and "$ip \in kD rt1$"
  and "$ip \in kD rt2$"
shows "$nsqn rt1 ip \leq nsqn rt2 ip$"
using assms(1)
by (auto simp add: $rt\_fresher\_def2$ [OF assms(2-3)])

lemma $rt\_strictly\_fresher\_ltI [intro]$: 
assumes "dip \in kD(rt1)"
  and "dip \in kD(rt2)"
  and "$nsqn rt1 dip < nsqn rt2 dip$"
shows "$rt1 \sqsubseteq dip rt2$"
proof
  from assms show "$rt1 \sqsubseteq dip rt2$ ..
next 
show "$(rt1 \approx dip rt2)$"
proof
  assume "$rt1 \approx dip rt2"
  hence "$rt2 \sqsubseteq dip rt1$ ..
  hence "$nsqn rt2 dip \leq nsqn rt1 dip$"
    using dip \in kD(rt2) dip \in kD(rt1)
    by (rule $rt\_fresher\_imp\_nsqn\_le$)
  with $nsqn rt1 dip < nsqn rt2 dip$ show "$False$"
    by simp
qeda

lemma $rt\_strictly\_fresher\_eqI [intro]$: 

assumes "i ∈ kD(rt1)"
and "i ∈ kD(rt2)"
and "nsqn rt1 i = nsqn rt2 i"
and "π₅(the (rt2 i)) < π₅(the (rt1 i))"
shows "rt1 △ i rt2"

using assms unfolding rt_strictly_fresher_def' by (auto simp add: kD_nsqn)

lemma invalidate_rtsf_left [simp]:
"\set\set\dests\ dip\ rt\'.\ \ dests\ dip\ =\ None\ \ \ \implies\ \ invalidate\ rt\ dests\ △ dip\ rt') = (rt △ dip\ rt')"

unfolding invalidate_def rt_strictly_fresher_def'
by (rule iffI) (auto split: option.split_asm)

lemma vD_invalidate_rt_strictly_fresher [simp]:
assumes "dip ∈ vD(invalidate rt1 dests)"
shows "(invalidate rt1 dests △ dip rt2) = (rt1 △ dip rt2)"
proof (cases "dip ∈ dom(dests)"
assume "dip ∈ dom(dests)"
hence "dip /∈ vD(invalidate rt1 dests)"
unfolding invalidate_def vD_def
by clarsimp (metis assms option.simps(3) vD_invalidate_vD_not_dests)
with ⟨dip ∈ vD(invalidate rt1 dests)⟩
show ?thesis by simp
next
assume "dip /∈ dom(dests)"
hence "dests dip = None" by auto
moreover with ⟨dip ∈ vD(invalidate rt1 dests)⟩ have "dip ∈ vD(rt1)"
unfolding invalidate_def vD_def
by clarsimp (metis (hide_lams, no_types) assms vD_Some vD_invalidate_vD_not_dests)
ultimately show ?thesis
unfolding invalidate_def rt_strictly_fresher_def' by auto
qed

lemma rt_strictly_fresher_update_other [elim!]:
"∃ dip ip rt rt'.\ \ dip\ ≠\ ip;\ \ rt\ △ dip\ rt'\ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ →\ \ update\ rt\ ip\ r\ △ dip\ rt'"
unfolding rt_strictly_fresher_def'
by clarsimp

lemma lt_sqn_imp_update_strictly_fresher:
assumes "dip ∈ vD (rt2 nhip)"
and *: "osn < sqn (rt2 nhip) dip"
and **: "rt / update rt dip (osn, kno, val, hops, nhip)"
shows "update rt dip (osn, kno, val, hops, nhip) △ dip rt2 nhip"
unfolding rt_strictly_fresher_def'
proof (rule disjI1)
from ** have "nsqn (update rt dip (osn, kno, val, hops, nhip)) dip = osn"
by (rule nsqn_update_changed_kno_val)
with ⟨dip ∈ vD(rt2 nhip)⟩
have "nsqn_r (the (update rt dip (osn, kno, val, hops, nhip) dip)) = osn"
by (simp add: kD_nsqn)
also have "osn < sqn (rt2 nhip) dip" by (rule *)
also have "sqn (rt2 nhip) dip = nsqn_r (the (rt2 nhip dip))"
unfolding nsqn_r_def using ⟨dip ∈ vD (rt2 nhip)⟩
by - (metis vD_flag_val proj2_eq_sqn proj4_eq_flag vD_iD_gives_kD(1))
finally show "nsqn_r (the (update rt dip (osn, kno, val, hops, nhip) dip)) < nsqn_r (the (rt2 nhip dip))".
qed

lemma dhops_le_hops_imp_update_strictly_fresher:
assumes "dip ∈ vD(rt2 nhip)"
and sqn: "sqn (rt2 nhip) dip = osn"
and hop: "the (dhops (rt2 nhip) dip) ≤ hops"
and **: "rt / update rt dip (osn, kno, val, Suc hops, nhip)"
shows "update rt dip (osn, kno, val, Suc hops, nhip) △ dip rt2 nhip"
unfolding rt_strictly_fresher_def'
proof (rule disjI2, rule conjI)
from ** have "nsqn (update rt dip (osn, kno, val, Suc hops, nhip)) dip = osn"
lemma nsqn_invalidate:
assumes "dip \in kD(rt)"
  and "\forall ip \in dom(dests). ip \in vD(rt) \land the(dests ip) = inc (sqn rt ip)"
shows "nsqn (invalidate rt dests) dip = nsqn rt dip"
proof -
  from \langle dip \in kD(rt) \rangle: have "dip \in kD(invalidate rt dests)" by simp

  from assms have "rt \approx dip invalidate rt dests"
    by (rule rt_fresh_as_inc_invalidate)
  with \langle dip \in kD(rt) \rangle \langle dip \in kD(invalidate rt dests) \rangle show ?thesis
    by (simp add: kD_nsqn del: invalidate_kD_inv)
(erule(2) rt_fresh_as_nsqnr)
qed

3.6 Invariant proofs on individual processes

theory C_Seq_Invariants
imports "../../../AWN/Invariants" C_Aodv C_Aodv_Data C_Aodv_Predicates C_Fresher
begin

The proposition numbers are taken from the December 2013 version of the Fehnker et al technical report.

Proposition 7.2

lemma sequence_number_increases:
  "paodv i \|= A onl1 Γ_{AODV} (\lambda ((ξ, _), _, (ξ', _)). sn ξ \leq sn ξ')"
  by inv_cterms

lemma sequence_number_one_or_bigger:
  "paodv i \|= onl1 Γ_{AODV} (\lambda (ξ, _). 1 \leq sn ξ)"
  by (rule onll_step_to_invariantI [OF sequence_number_increases])
    (auto simp: σ_{AODV_def})

We can get rid of the onl/onll if desired...

lemma sequence_number_increases':
  "paodv i \|= A (\lambda ((ξ, _), _, (ξ', _)). sn ξ \leq sn ξ')"
  by (rule step_invariant_weakenE [OF sequence_number_increases]) (auto dest!: onllD)

lemma sequence_number_one_or_bigger':
  "paodv i \|= (\lambda (ξ, _). 1 \leq sn ξ)"
  by (rule invariant_weakenE [OF sequence_number_one_or_bigger]) auto
lemma sip_in_kD:
"paodv i |= onl Γ AODV (λ(ξ, 1). 1 ∈ {PAodv-:7} ∪ {PAodv-:5} ∪ {PRrep:-0..PRrep:-1}) 
∪ {PRreq:-0..PRreq:-3}) → sip ξ ∈ kD (rt ξ))"
by inv_cterms

lemma rrep_1_update_changes:
"paodv i |= onl Γ AODV (λ(ξ, 1). (1 = PRrep:-1 → 
rt ξ ≠ update (rt ξ) (dip ξ) (dsn ξ, kno, val, hops ξ + 1, sip ξ)))"
by inv_cterms

Proposition 7.38
lemma includes_nhip:
"paodv i |= onl Γ AODV (λ(ξ, 1). ∀ dip ∈ kD(rt ξ). the (nhop (rt ξ) dip) ∈ kD(rt ξ)))"
proof -
{ fix ip and ξ ξ' :: state
  assume "∀ dip ∈ kD(rt ξ). the (nhop (rt ξ) dip) ∈ kD(rt ξ)"
  hence "∀ dip ∈ kD(rt ξ).
  the (nhop (update (rt ξ) ip (0, unk, val, Suc 0, ip)) dip) = ip
  ∨ the (nhop (update (rt ξ) ip (0, unk, val, Suc 0, ip)) dip) ∈ kD(rt ξ)"
  by clarsimp (metis nhop_update_unk_val update_another)
} note one_hop = this

{ fix ip sip sn hops and ξ ξ' :: state
  assume "∀ dip ∈ kD(rt ξ). the (nhop (rt ξ) dip) ∈ kD(rt ξ)"
  and "ξ' = ξ(rt := update (rt ξ) ip (sn, kno, val, Suc hops, sip))"
  and "sip ∈ kD(rt ξ)"
  hence "((the (nhop (update (rt ξ) ip (sn, kno, val, Suc hops, sip)) dip) = ip
  ∨ the (nhop (update (rt ξ) ip (sn, kno, val, Suc hops, sip)) dip) ∈ kD(rt ξ)))
  ∧ (∀ dip ∈ kD(rt ξ).
  the (nhop (update (rt ξ) ip (sn, kno, val, Suc hops, sip)) dip) = ip
  ∨ the (nhop (update (rt ξ) ip (sn, kno, val, Suc hops, sip)) dip) ∈ kD(rt ξ)))"
  by (metis kD_update_unchanged nhop_update_changed update_another)
} note nhip_is_sip = this
show ?thesis
by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf sip_in_kD]
solve: one_hop nhip_is_sip)
qed

Proposition 7.4
lemma known_destinations_increase:
"paodv i |=A onll Γ AODV (λ((ξ, _), _, (ξ', _)). kD (rt ξ) ⊆ kD (rt ξ'))"
by (inv_cterms simp add: subset_insertI)

Proposition 7.5
lemma rreqs_increase:
"paodv i |=A onll Γ AODV (λ((ξ, _), _, (ξ', _)). rreqs ξ ⊆ rreqs ξ')"
by (inv_cterms simp add: subset_insertI)

lemma dests_bigger_than_sqn:
"paodv i |= onl Γ AODV (λ(ξ, 1). 1 ∈ {PAodv-:15..PAodv-:17}
∪ {PPkt-:7..PPkt-:9}
∪ {PRreq-:9..PRreq-:11}
∪ {PRreq-:17..PRreq-:19}
∪ {PRrep-:8..PRrep-:10}
∪ {PRerr-:1..PRerr-:4} ∪ {PRerr-:6})
→ (∀ ip ∈ dom(dests ξ). ip ∈ kD(rt ξ) ∧ sqn (rt ξ) ip ≤ the (dests ξ ip)))"
proof -
  have sqninv:
  "∀ dests rt rsn ip.
  [ ∀ ip ∈ dom(dests). ip ∈ kD(rt) ∧ sqn rt ip ≤ the (dests ip); dests ip = Some rsn ]
  → sqn (invalidate rt dests) ip ≤ rsn"
  by (rule sqn_invalidate_in_dests [THEN eq_imp_le], assumption) auto
  have indests:
"⋀ dests rt rsn ip.
[ ∀ ip∈dom(dests). ip∈kD(rt) ∧ sqn rt ip ≤ the (dests ip); dests ip = Some rsn ]
⇒ ip∈kD(rt) ∧ sqn rt ip ≤ rsn"
by (metis domI option.sel)

show ?thesis
by inv_cterms
(clarsimp split: split_if_asm option.split_asm elim!: sqninv dests)

qed

Proposition 7.6

lemma sqns_increase:
"paodv i |=_A onll Γ\text{AODV} (λ((ξ, _), _, (ξ', _)). ∀ ip. sqn (rt ξ) ip ≤ sqn (rt ξ') ip)"
proof -
{ fix ξ :: state
assume *: "∀ ip∈dom(dests ξ). ip ∈ kD (rt ξ) ∧ sqn (rt ξ) ip ≤ the (dests ξ ip)"
have "∀ ip. sqn (rt ξ) ip ≤ sqn (invalidate (rt ξ) (dests ξ)) ip"
proof
fix ip
from * have "ip ∉ dom(dests ξ) ∨ sqn (rt ξ) ip ≤ the (dests ξ ip)" by simp
thus "sqn (rt ξ) ip ≤ sqn (invalidate (rt ξ) (dests ξ)) ip"
by (metis domI invalidate_sqn option.sel)
qed
} note solve_invalidate = this
show ?thesis
by (inv_cterms inv add: onll_invvariant_sterms [OF aodv_wf dests_bigger_than_sqn]
simp add: solve_invalidate)

qed

Proposition 7.7

lemma ip_constant:
"paodv i |= onll Γ\text{AODV} (λ((ξ, _), a, _). ip ξ = i)"
by (inv_cterms simp add: σ\text{AODV}_def)

Proposition 7.8

lemma sender_ip_valid':
"paodv i |=_A onll Γ\text{AODV} (λ((ξ, _), a, _). anycast (λm. not_Pkt m −→ msg_sender m = ip ξ) a)"
by inv_cterms

lemma sender_ip_valid:
"paodv i |=_A onll Γ\text{AODV} (λ((ξ, _), a, _). anycast (λm. not_Pkt m −→ msg_sender m = i) a)"
by (rule step_invvariant_weaken_with_invvariantE [OF ip_constant sender_ip_valid'])
(auto dest!: onlD onllD)

lemma received_msg_inv:
"paodv i |= (recvmsg P →) onll Γ\text{AODV} (λ(ξ, 1). l ∈ {PAodv−:1} → P (msg ξ))"
by inv_cterms

Proposition 7.9

lemma sip_not_ip':
"paodv i |= (recvmsg (λm. not_Pkt m −→ msg_sender m ≠ i) →) onll Γ\text{AODV} (λ(ξ, _). sip ξ ≠ ip ξ)"
by (inv_cterms inv add: onll_invvariant_sterms [OF aodv_wf received_msg_inv]
onll_invvariant_sterms [OF aodv_wf ip_constant [THEN invariant_restrict_inD]])
simp add: clear_locals_sip_not_ip') clarsimp+

lemma sip_not_ip:
"paodv i |= (recvmsg (λm. not_Pkt m −→ msg_sender m ≠ i) →) onll Γ\text{AODV} (λ(ξ, _). sip ξ ≠ i)"
by (inv_cterms inv add: onll_invvariant_sterms [OF aodv_wf received_msg_inv]
onll_invvariant_sterms [OF aodv_wf ip_constant [THEN invariant_restrict_inD]])
simp add: clear_locals_sip_not_ip') clarsimp+

Neither sip_not_ip' nor sip_not_ip is needed to show loop freedom.

Proposition 7.10
lemma hop_count_positive:
"paodv i |= onl Γ_AODV (λ(ξ, _). ∀ip∈kD (rt ξ). the (dhops (rt ξ) ip) ≥ 1)"
by (inv_cterms) auto

lemma rreq_dip_in_vD_dip_eq_ip:
"paodv i |= onl Γ_AODV (λ(ξ, l). (∀ip∈kD (rt ξ). the (dhops (rt ξ) ip) ≥ 1))"
by (inv_cterms) auto

Proposition 7.11
lemma anycast_msg_zhops:
"⋀rreqid dip dsn dsk oip osn sip. paodv i |= onl Γ_AODV (λ(_, a, _). anycast msg_zhops a)"
proof (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf rreq_dip_in_vD_dip_eq_ip [THEN invariant_restrict_inD]]
onl_invariant_sterms [OF aodv_wf hop_count_positive [THEN invariant_restrict_inD]], elim conjE)
fix l ξ a pp p' pp'
assume "((ξ, pp) ∈ reachable (paodv i) TT"
and "{PRreq-:14}unicast(λξ. the (nhop (rt ξ) (oip ξ)),
λξ. Rrep (the (dhops (rt ξ) (dip ξ))) (dip ξ) (sqn (rt ξ) (dip ξ)) (oip ξ) (ip ξ)).
p' ⊢ pp' ∈ sterms Γ_AODV pp"
and "l = PRreq-:14"
and "a = unicast (the (nhop (rt ξ) (oip ξ)))
(Rrep (the (dhops (rt ξ) (dip ξ))) (dip ξ) (sqn (rt ξ) (dip ξ)) (oip ξ) (ip ξ))"
and *: "∀ip∈kD (rt ξ). Suc 0 ≤ the (dhops (rt ξ) ip)"
and "dip ξ ∈ vD (rt ξ)"
from dip ξ ∈ vD (rt ξ) have "dip ξ ∈ kD (rt ξ)"
by (rule vD_iD_gives_kD(1))
with * have "Suc 0 ≤ the (dhops (rt ξ) (dip ξ))" ..
thus "0 < the (dhops (rt ξ) (dip ξ))" by simp
qed

lemma hop_count_zero_oip_dip_sip:
"paodv i |= (recvmsg msg_zhops →) onl Γ_AODV (λ(ξ, _). aodv wf rreq_dip_in_vD_dip_eq_ip [THEN invariant_restrict_inD])
onl_invariant_sterms [OF aodv_wf hop_count_positive [THEN invariant_restrict_inD]], elim conjE)
fix l ξ a pp p' pp'
assume "((ξ, pp) ∈ reachable (paodv i) TT"
and "{PAodv-:4..PAodv-:5} ∪ {PRreq-:n|n. True} →
((∃l∈{PAodv-:4..PAodv-:5} ∪ {PRreq-:n|n. True} →
the (dhops (rt ξ) ip) = 0 → oip ξ = sip ξ)))"
by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]) auto

lemma osn_rreq:
"paodv i |= (recvmsg rreq_rrep_sn →) onl Γ_AODV (λ(ξ, _). aodv wf received_msg_inv)
1 ∈ {PAodv-:4, PAodv-:5} ∪ {PRreq-:n|n. True} → 1 ≤ osn ξ)"
by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]) clarsimp

lemma osn_rreq':
"paodv i |= (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) →) onl Γ_AODV (λ(ξ, _).
1 ∈ {PAodv-:4, PAodv-:5} ∪ {PRreq-:n|n. True} → 1 ≤ osn ξ)"
proof (rule invariant_weakenE [OF osn_rreq])
fix a
assume "recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) a"
thus "recvmsg (λm. rreq_rrep_sn m) a"
by (cases a) simp_all
qed

lemma dsn_rrep:
"paodv i |= (recvmsg rreq_rrep_sn →) onl Γ_AODV (λ(ξ, _).
1 ∈ {PAodv-:6, PAodv-:7} ∪ {PRreq-:n|n. True} → 1 ≤ dsn ξ)"
by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]) clarsimp

lemma dsn_rrep':

292
proof (rule invariant_weakenE [OF dsn_rrep])
  fix a
  assume "recvmsg (\m. rreq_rrep_sn m \& msg_zhops m) a"
  thus "recvmsg rreq_rrep_sn a"
  by (cases a) simp_all
qed

lemma hop_count_zero_oip_dip_sip' :
  "paodv i |||= (recvmsg (\m. rreq_rrep_sn m \& msg_zhops m) → onl Γ\_AODV (\(x, _.\).
    \(\xi \in \{PAodv-:4..PAodv-:5\} \cup \{PRreq-:n|n. True\} \rightarrow (hops \(\xi = 0 \rightarrow oip \(\xi = sip \(\xi)))\) ∧
    \((1\in\{PAodv-:6..PAodv-:7\} \cup \{PRreq-:n|n. True\} \rightarrow (hops \(\xi = 0 \rightarrow dip \(\xi = sip \(\xi))))\))")
  proof
  -
    { fix dip and \(x\) :: state and P
      assume "sqn (invalidate (rt \(\xi\)) (dests \(\xi\))) dip = 0"
      and all: "\((\forall dip \in kD(rt). (sqn (rt \(\xi\)) dip = 0 \rightarrow sqnf (rt \(\xi\)) dip = unk) ∧
        (sqnf (rt \(\xi\)) dip = unk \rightarrow the (dhops (rt \(\xi\)) dip) = 1) ∧
        (the (dhops (rt \(\xi\)) dip) = 1 \rightarrow the (nhop (rt \(\xi\)) dip) = dip))\)")
      have "P \(\xi\) dip"
      proof
        from all have "sqn (invalidate (rt \(\xi\)) (dests \(\xi\))) dip = 0" ..
        with \(\forall dip \in kD(rt). (sqn (rt \(\xi\)) dip = 0 \rightarrow sqnf (rt \(\xi\)) dip = unk) ∧
          (sqnf (rt \(\xi\)) dip = unk \rightarrow the (dhops (rt \(\xi\)) dip) = 1) ∧
          (the (dhops (rt \(\xi\)) dip) = 1 \rightarrow the (nhop (rt \(\xi\)) dip) = dip))\)")
        have "P \(\xi\) dip" by (rule *)
      qed
    } note sqn_invalidate_zero [elim!] = this

    { fix dip and \(\xi\) :: state and P
      assume "sqn (invalidate (rt \(\xi\)) (dests \(\xi\))) dip = 0"
      and all: "\((\forall dip \in kD(rt). (sqn (rt \(\xi\)) dip = 0 \rightarrow sqnf (rt \(\xi\)) dip = unk) ∧
        (sqnf (rt \(\xi\)) dip = unk \rightarrow the (dhops (rt \(\xi\)) dip) = 1) ∧
        (the (dhops (rt \(\xi\)) dip) = 1 \rightarrow the (nhop (rt \(\xi\)) dip) = dip))\)")
      have "P \(\xi\) dip"
      proof
        from all have "sqn (invalidate (rt \(\xi\)) (dests \(\xi\))) dip = 0" ..
        with \(\forall dip \in kD(rt). (sqn (rt \(\xi\)) dip = 0 \rightarrow sqnf (rt \(\xi\)) dip = unk) ∧
          (sqnf (rt \(\xi\)) dip = unk \rightarrow the (dhops (rt \(\xi\)) dip) = 1) ∧
          (the (dhops (rt \(\xi\)) dip) = 1 \rightarrow the (nhop (rt \(\xi\)) dip) = dip))\)")
        have "P \(\xi\) dip" by (rule *)
      qed
    } note prreq_ok1 [simp] = this

  { fix dsn hops :: nat and sip oip rt and ip dip :: ip
    assume "\(\forall dip \in kD(rt). (sqn (rt dip = 0 \rightarrow \pi_3 (the (rt dip)) = unk) ∧
      (\pi_3 (the (rt dip)) = unk \rightarrow the (dhops rt dip) = Suc 0) ∧
      (the (dhops rt dip) = Suc 0 \rightarrow the (nhop rt dip) = dip))\)"
    and "hops = 0 \rightarrow sip = dip"
    and "Suc 0 \leq dsn"
    and "ip \neq dip \rightarrow ip \in kD(rt)"
    hence "the (dhops (update rt dip (dsn, kno, val, Suc hops, sip)) ip) = Suc 0 \rightarrow the (nhop (update rt dip (dsn, kno, val, Suc hops, sip)) ip) = ip" by - (rule update_cases, auto simp add: sqn_def dest!: bepec)
  } note prreq_ok1 [simp] = this

  { fix ip dsn hops sip oip rt dip
    assume "\(\forall dip \in kD(rt). (sqn (rt dip = 0 \rightarrow \pi_3 (the (rt dip)) = unk) ∧
      (\pi_3 (the (rt dip)) = unk \rightarrow the (dhops rt dip) = Suc 0) ∧
      (the (dhops rt dip) = Suc 0 \rightarrow the (nhop rt dip) = dip))\)"
    and "Suc 0 \leq dsn"
    and "ip \neq dip \rightarrow ip \in kD(rt)"
    hence "\pi_3 (the (update rt dip (dsn, kno, val, Suc hops, sip) ip)) = unk \rightarrow the (dhops (update rt dip (dsn, kno, val, Suc hops, sip) ip)) = Suc 0"
by - (rule update_cases, auto simp add: sqn_def sqnf_def dest!: bspec)
} note prreq_ok2 [simp] = this

{ fix ip dsn hops sip oip rt dip
  assume "\forall dip \in kD rt.
    (sqn rt dip = 0 \longrightarrow \pi_3 (the (rt dip)) = unk) \land
    (\pi_3 (the (rt dip)) = unk \longrightarrow the (dhops rt dip) = Suc 0) \land
    (the (dhops rt dip) = Suc 0 \longrightarrow the (nhop rt dip) = dip)"
  and "Suc 0 \leq dsn"
  and "ip \neq dip \longrightarrow ip \in kD rt"
  hence "sqn (update rt dip (dsn, kno, val, Suc hops, sip)) ip = 0 \longrightarrow
    \pi_3 (the (update rt dip (dsn, kno, val, Suc hops, sip) ip)) = unk"
by - (rule update_cases, auto simp add: sqn_def dest!: bspec)
} note prreq_ok2 [simp] = this

{ fix ip dsn hops sip oip rt dip
  assume "\forall dip \in kD rt.
    (sqn rt dip = 0 \longrightarrow \pi_3 (the (rt dip)) = unk) \land
    (\pi_3 (the (rt dip)) = unk \longrightarrow the (dhops rt dip) = Suc 0) \land
    (the (dhops rt dip) = Suc 0 \longrightarrow the (nhop rt dip) = dip)"
  and "Suc 0 \leq dsn"
  and "ip \neq dip \longrightarrow ip \in kD rt"
  hence "sqn (update rt dip (dsn, kno, val, Suc hops, sip)) ip = 0 \longrightarrow
    \pi_3 (the (update rt dip (dsn, kno, val, Suc hops, sip) ip)) = unk"
by - (rule update_cases, auto simp add: sqn_def sqnf_def)
} note prreq_ok3 [simp] = this

{ fix rt sip
  assume "\forall dip \in kD rt.
    (sqn rt dip = 0 \longrightarrow \pi_3 (the (rt dip)) = unk) \land
    (\pi_3 (the (rt dip)) = unk \longrightarrow the (dhops rt dip) = Suc 0) \land
    (the (dhops rt dip) = Suc 0 \longrightarrow the (nhop rt dip) = dip)"
  and "Suc 0 \leq dsn"
  and "ip \neq dip \longrightarrow ip \in kD rt"
  hence "sqn (update rt dip (0, unk, val, Suc 0, sip)) dip = 0 \longrightarrow
    \pi_3 (the (update rt dip (0, unk, val, Suc 0, sip) dip)) = unk"
by (rule update_cases, simp_all add: sqn_def)
} note prreq_ok4 [simp] = this

have prreq_ok5 [simp]: "\forall sip rt.
  \pi_3 (the (update rt dip (0, unk, val, Suc 0, sip)) sip) = unk \Rightarrow
  the (dhops (update rt dip (0, unk, val, Suc 0, sip)) sip) = Suc 0"
by (rule update_cases) simp_all

have prreq_ok6 [simp]: "\forall sip rt.
  sqn (update rt dip (0, unk, val, Suc 0, sip)) sip = 0 \Rightarrow
  \pi_3 (the (update rt dip (0, unk, val, Suc 0, sip)) sip) = unk"
by (rule update_cases) simp_all

show ?thesis
  by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf hop_count_zero_oip_dip_sip']
      seq_step_invariant_sterms_TT [OF sqns_increase aodv_wf aodv_trans]
      onl_invariant_sterms [OF aodv_wf osn_rreq']
      onl_invariant_sterms [OF aodv_wf dsn_rrep']) clarsimp+ qed

lemma zero_seq_unk_hops_one:
  "paodv i \mid
  (recvmsg (\lambda m. rreq_rrep_sn m \land msg_zhops m) \rightarrow
     \forall dip \in kD rt.
       (sqn rt dip = 0 \longrightarrow \pi_3 (the (rt dip)) = unk) \land
       (\pi_3 (the (rt dip)) = unk \longrightarrow the (dhops rt dip) = Suc 0) \land
       (the (dhops rt dip) = Suc 0 \longrightarrow the (nhop rt dip) = dip))"
  by (rule invariant_weakenE [OF zero_seq_unk_hops_one']) auto

lemma kD_unk_or_atleast_one:
  "paodv i \mid
  (recvmsg rreq_rrep_sn \rightarrow
     \forall dip \in kD rt.
       (sqn rt dip = 0 \longrightarrow \pi_3 (the (rt dip)) = unk) \land
       (\pi_3 (the (rt dip)) = unk \lor 1 \leq \pi_2 (the (rt dip))))"
  by (rule invariant_weakenE [OF zero_seq_unk_hops_one']) auto

proof -
  { fix sip rt dsn1 dsn2 dsk1 dsk2 flag1 flag2 hops1 hops2 nhip1 nhip2
    assume "dsk1 = unk \lor Suc 0 \leq dsn2"
    hence "\pi_3 (the (update rt dip (dsn1, dsk1, flag1, hops1, nhip1) sip)) = unk"
    \land
    Suc 0 \leq sqn (update rt dip (dsn2, dsk2, flag2, hops2, nhip2) sip)"
    unfolding update_def by (cases "dsk1 = unk") (clarsimp split: option.split)+
  } note fromsip [simp] = this

294
{ fix dip sip rt dsn1 dsn2 dsk1 dsk2 flag1 flag2 hops1 hops2 nhip1 nhip2
   assume allkd: "∀ dip ∈ kD(rt). π₃ (the (rt dip)) = unk ∨ Suc 0 ≤ sqn rt dip"
   and **: "dsn1 = unk ∨ Suc 0 ≤ dsn2"
have "∀ dip ∈ kD(rt). π₃ (the (update rt sip (dsn1, dsk1, flag1, hops1, nhip1) dip)) = unk
   ∨ Suc 0 ≤ sqn (update rt sip (dsn2, dsk2, flag2, hops2, nhip2)) dip"
   (is "∀ dip ∈ kD(rt). ?prop dip")
   proof
   fix dip
   assume "dip ∈ kD(rt)"
   thus "?prop dip"
   proof (cases "dip = sip")
     assume "dip = sip" with **
     show ?thesis by (simp ..)
   next
     assume "dip ≠ sip" with ⟨dip ∈ kD(rt)⟩ allkd
     show ?thesis by simp
   qed
   qed
 }

note solve_update [simp] = this

{ fix dip rt dests
   assume *: "∀ ip ∈ dom(dests). ip ∈ kD(rt) ∧ sqn rt ip ≤ the (dests ip)"
   and **: "∀ ip ∈ kD(rt). π₃ (the (rt ip)) = unk ∨ Suc 0 ≤ sqn rt ip"
have "∀ dip ∈ kD(rt). π₃ (the (rt dip)) = unk ∨ Suc 0 ≤ sqn (invalidate rt dests) dip"
   proof
   fix dip
   assume "dip ∈ kD(rt)"
   with **
   have "π₃ (the (rt dip)) = unk ∨ Suc 0 ≤ sqn rt dip" ..
   thus "π₃ (the (rt dip)) = unk ∨ Suc 0 ≤ sqn (invalidate rt dests) dip"
   proof
     assume "π₃ (the (rt dip)) = unk" thus ?thesis ..
   next
     have "Suc 0 ≤ sqn (invalidate rt dests) dip"
     proof (cases "dip ∈ dom(dests)"
       assume "dip ∈ dom(dests)"
       with *
       have "sqn rt dip ≤ the (dests dip)" by simp
       with ⟨Suc 0 ≤ sqn rt dip⟩ have "Suc 0 ≤ the (dests dip)" by simp
       with ⟨dip ∈ dom(dests)⟩ ⟨dip ∈ kD(rt)⟩ [THEN kD_Some] show ?thesis
       unfolding invalidate_def sqn_def by auto
     next
     assume "dip ∉ dom(dests)"
     with ⟨Suc 0 ≤ sqn rt dip⟩ ⟨dip ∈ kD(rt)⟩ [THEN kD_Some] show ?thesis
     unfolding invalidate_def sqn_def by auto
     qed
     thus ?thesis by (rule disjI2)
   qed
   qed
 }

note solve_invalidate [simp] = this

show ?thesis
   by (inv_c terms inv add: onl_invariant_sterms [OF aodv_wf dests_bigger_than_sqn
   [THEN invariant_restrict_inD]]
      onl_invariant_sterms [OF aodv_wf osn_rreq]
      onl_invariant_sterms [OF aodv_wf dsn_rrep]
      simp add: proj3_inv proj2_eq_sqn)
   qed

Proposition 7.13

lemma rreq_rrep_sn_any_step_invariant:
  "paodv i |=ₐ (recvmsg rreq_rrep_sn →) onll ΓₐOADV (λ(_, a, _). anycast rreq_rrep_sn a)"

295
proof -

have sqnf_kno: "paodv i \models onl \Gamma_{AODV} (\lambda(\xi, 1).
(1 \in \{PRreq:-14\} \rightarrow dip \xi \in kD (rt \xi) \land sqn (rt \xi) (dip \xi) (sqn_kno) )
by (inv_cterms)

show \?thesis
by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf sequence_number_one_or_bigger
THEN invariant_restrict_inD])
onl_invariant_sterms [OF aodv_wf kD_unk_or_atleast_one]
onl_invariant_sterms_TT [OF aodv_wf sqnf_kno]
onl_invariant_sterms [OF aodv_wf osn_rreq]
onl_invariant_sterms [OF aodv_wf dsn_rrep])
(auto simp: proj2_eq_sqn)
qed

Proposition 7.14

lemma rreq_rrep_fresh_any_step_invariant:
"paodv i \models onl \Gamma_{AODV} (\lambda((\xi, _), a, _). anycast (rreq_rrep_fresh (rt \xi)) a)
proof -

have rreq_oip: "paodv i \models onl \Gamma_{AODV} (\lambda((\xi, l). (l \in \{PRreq:-3, PRreq:-4, PRreq:-13, PRreq:-21\}
\rightarrow oip \xi \in kD (rt \xi)
\land (sqn (rt \xi) (oip \xi) > (osn \xi)
\lor (sqn (rt \xi) (oip \xi) = (osn \xi)
\land the (dhops (rt \xi) (oip \xi)) \leq Suc (hops \xi)
\land the (flag (rt \xi) (oip \xi) = val))))"
proof inv_cterms
fix l \xi l' pp p'
assume "((\xi, pp) \in reachable (paodv i) TT"
and "{PRreq:-2}\\lambda. \xi (rt := update (rt \xi) (oip \xi) (osn \xi, kno, val, Suc (hops \xi), sip \xi)) \} \ p' \in sterms \Gamma_{AODV} pp"
and "l' = PRreq:-3"
show "osn \xi < sqn (update (rt \xi) (oip \xi) (osn \xi, kno, val, Suc (hops \xi), sip \xi)) (oip \xi)
\lor (sqn (update (rt \xi) (oip \xi) (osn \xi, kno, val, Suc (hops \xi), sip \xi)) (oip \xi) = osn \xi
\land the (dhops (update (rt \xi) (oip \xi) (osn \xi, kno, val, Suc (hops \xi), sip \xi)) (oip \xi)) \leq Suc (hops \xi)
\land the (flag (update (rt \xi) (oip \xi) (osn \xi, kno, val, Suc (hops \xi), sip \xi)) (oip \xi) = val)"
unfolding update_def by (clarsimp split: option.split)
(metis linorder_neqE_nat not_less)
qed

have rrep_prrep: "paodv i \models onl \Gamma_{AODV} (\lambda(\xi, 1).
(1 \in \{PRrep:-2..PRrep:-5\} \rightarrow (dip \xi \in kD (rt \xi)
\land sqn (rt \xi) (dip \xi) = dsn \xi
\land the (dhops (rt \xi) (dip \xi)) \leq Suc (hops \xi)
\land the (flag (rt \xi) (dip \xi)) = val
\land the (nhop (rt \xi) (dip \xi)) \in kD (rt \xi))))"
by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf rrep_i_update_changes]
onl_invariant_sterms [OF aodv_wf sip_in_kD])

show \?thesis
by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf rreq_oip]
onl_invariant_sterms [OF aodv_wf rreq_dip_in_vD_dip_eq_ip]
onl_invariant_sterms [OF aodv_wf rrep_prrep])
qed

Proposition 7.15

lemma rerr_invalid_any_step_invariant:
"paodv i \models onll \Gamma_{AODV} (\lambda((\xi, _), a, _). anycast (rerr_invalid (rt \xi)) a)
proof -

have dests_inv: "paodv i \models onl \Gamma_{AODV} (\lambda(\xi, l). (l \in \{PAodv:-15, PPkt:-7, PRreq:-9, PRreq:-17, PRrep:-8, PRerr:-1\}
0296
\[ \forall ip \in \text{dom}(\text{dests } \xi). \, ip \in VD(rt \xi) \]
\[ \land (1 \in \{PAodv\:-\:16..PAodv\:-\:17\} \]
\[ \cup \{PPkt\:-\:8..PPkt\:-\:9\} \]
\[ \cup \{PRreq\:-\:10..PRreq\:-\:11\} \]
\[ \cup \{PRreq\:-\:18..PRreq\:-\:19\} \]
\[ \cup \{PRrep\:-\:9..PRrep\:-\:10\} \]
\[ \cup \{PRerr\:-\:2..PRerr\:-\:4\} \rightarrow (\forall ip \in \text{dom}(\text{dests } \xi). \, ip \in iD(rt \xi) \]
\[ \land \text{the (dests } \xi \, ip) = \text{sqn} (rt \xi) \, ip)) \]
\[ \land (1 = PPkt\:-\:12 \rightarrow dip \xi \in iD(rt \xi)) \]
\[ \land \{\text{PAodv-:15, PPkt-:7, PRreq-:9, PRreq-:17, PRrep-:8}\} \]
\[ \land \{\text{PAodv-:16, PAodv-:17}\} \]
\[ \land \{\text{PAodv-:18, PAodv-:19}\} \]
\[ \land \{\text{PAodv-:20, PAodv-:21}\} \]
\[ \land \{\text{PAodv-:22, PAodv-:23}\} \]

by inv_cterms (clarsimp split: split_if_asm option.split_asm simp: domIff)+

Proposition 7.16

Some well-definedness obligations are irrelevant for the Isabelle development:

1. In each routing table there is at most one entry for each destination: guaranteed by type.
2. In each store of queued data packets there is at most one data queue for each destination: guaranteed by structure.
3. Whenever a set of pairs \((rip, rsn)\) is assigned to the variable \(\text{dests}\) of type \(ip \rightarrow \text{sqn}\), or to the first argument of the function \(\text{rerr}\), this set is a partial function, i.e., there is at most one entry \((rip, rsn)\) for each destination \(rip\): guaranteed by type.

Lemma \(\text{dests_vD_inc_sqn}\):

\[ \text{paodv i } \models \]
\[ \text{onl } \Gamma_{AODV} (\lambda(\xi, \_). \, (1 \in \{\text{PAodv-:15, PPkt-:7, PRreq-:9, PRreq-:17, PRrep-:8}\} \]
\[ \land (1 \in \{\text{PRreq-:2}\} \rightarrow (\forall ip \in \text{dom}(\text{dests } \xi). \, ip \in VD(rt \xi) \land \text{the (dests } \xi \, ip) = \text{inc} \text{ sqn} (rt \xi) \, ip)) \]
\[ \land (1 = \text{PPkt-:12} \rightarrow dip \xi \in iD(rt \xi)) \]
\[ \land (1 = \text{PRreq-:2} \land \text{the (dests } \xi \, ip) > \text{sqn} (rt \xi) \, ip)) \]

by inv_cterms (clarsimp split: split_if_asm option.split_asm)+

Proposition 7.27

Lemma \(\text{route_tables_fresher}\):

\[ \text{paodv i } \models_{A} (\text{recvmsg rreq_rrep_sn } \rightarrow \text{onl } \Gamma_{AODV} (\lambda((\xi, \_), \_, (\xi', \_)). \] 
\[ \forall dip \in \text{kd} (rt \xi). \, rt \xi \sqsubseteq dip \, rt \xi' \]) \]

proof (inv_cterms inv add:

\[ \text{onl_invariant_sterms [OF aodv_wf dests_vD_inc_sqn [THEN invariant_restrict_inD]]} \]
\[ \text{onl_invariant_sterms [OF aodv_wf hop_count_positive [THEN invariant_restrict_inD]]} \]
\[ \text{onl_invariant_sterms [OF aodv_wf osn_rreq]} \]
\[ \text{onl_invariant_sterms [OF aodv_wf dsn_rrep]} \]
\[ \text{onl_invariant_sterms [OF aodv_wf invariant_restrict_inD]} \]

fix \(\xi\) pp p'

assume "\((\xi, pp) \in \text{reachable (paodv i) (recvmsg rreq_rrep_sn)}\)"

and "\{\text{PRreq-:2}\} [\lambda \xi. \, (\xi', \_). (\xi', \_). (\text{rerr} \_)]

proof (inv_cterms inv add:

\[ \text{onl_invariant_sterms [OF aodv_wf dests_vD_inc_sqn [THEN invariant_restrict_inD]]} \]
\[ \text{onl_invariant_sterms [OF aodv_wf hop_count_positive [THEN invariant_restrict_inD]]} \]
\[ \text{onl_invariant_sterms [OF aodv_wf osn_rreq]} \]
\[ \text{onl_invariant_sterms [OF aodv_wf dsn_rrep]} \]
\[ \text{onl_invariant_sterms [OF aodv_wf invariant_restrict_inD]} \]

fix ip

assume "ip \in \text{kd} (rt \xi)"

moreover with * have "1 \leq \text{dhops (rt \xi) ip}" by simp

moreover from \(Suc 0 \leq \text{osn} \xi\)

have "update_arg_wf (osn \xi, kno, val, Suc (hops \xi), sip \xi)" ..

ultimately show "rt \xi \sqsubseteq dip update (rt \xi) (oip \xi) (osn \xi, kno, val, Suc (hops \xi), sip \xi)"

by (rule rt_fresher_update)

qed

next

fix ip

assume "ip \in \text{kd} (rt \xi)"

moreover with * have "1 \leq \text{dhops (rt \xi) ip}" by simp

moreover from \(Suc 0 \leq \text{osn} \xi\)

have "update_arg_wf (osn \xi, kno, val, Suc (hops \xi), sip \xi)" ..

ultimately show "rt \xi \sqsubseteq dip update (rt \xi) (oip \xi) (osn \xi, kno, val, Suc (hops \xi), sip \xi)"

by (rule rt_fresher_update)

qed
and "{PRrep:-1][λξ. ξ≡ update (rt ξ) (dip ξ) (dsn ξ, kno, val, Suc (hops ξ), sip ξ)]
  p' ∈ terms Γ AODV pp"
and "Suc 0 ≤ dsn ξ"
and "∀ ip∈kD (rt ξ). Suc 0 ≤ the (dhops (rt ξ) ip)"
show "∀ ip∈kD (rt ξ). rt ξ ⊑ ip update (rt ξ) (dip ξ) (dsn ξ, kno, val, Suc (hops ξ), sip ξ)"

proof
fix ip
assume "ip∈kD (rt ξ)"
moreover with * have "1 ≤ the (dhops (rt ξ) ip)" by simp
moreover from ⟨Suc 0 ≤ dsn ξ⟩ have "update_arg_wf (dsn ξ, kno, val, Suc (hops ξ), sip ξ)" ..
ultimately show "rt ξ ⊑ ip update (rt ξ) (dip ξ) (dsn ξ, kno, val,Suc (hops ξ), sip ξ)"
by (rule rt_fresher_update)
qed
qed
end

3.7 The quality increases predicate

theory C_Quality_Increases
imports C_Aodv_Predicates C_Fresher
begin

definition quality_increases :: "state ⇒ state ⇒ bool"
where "quality_increases ξ ξ' ≡ (∀ dip∈kD(rt ξ). dip∈kD(rt ξ') ∧ rt ξ ⊑ dip rt ξ') ∧ (∀ dip. sqn (rt ξ) dip ≤ sqn (rt ξ') dip)"

lemma quality_increasesI [intro!]:
assumes "∀ dip. dip∈kD(rt ξ) =⇒ dip∈kD(rt ξ')" and "∀ dip. [ dip∈kD(rt ξ); dip∈kD(rt ξ') ] =⇒ rt ξ ⊑ dip rt ξ'"
and "∀ dip. sqn (rt ξ) dip ≤ sqn (rt ξ') dip"
shows "quality_increases ξ ξ'"
unfolding quality_increases_def using assms by clarsimp

lemma quality_increasesE [elim]:
fixes dip
assumes "quality_increases ξ ξ'" and "dip∈kD(rt ξ)" and "[ dip∈kD(rt ξ'); rt ξ ⊑ dip rt ξ'; sqn (rt ξ) dip ≤ sqn (rt ξ') dip ] =⇒ R dip ξ ξ'"
shows "R dip ξ ξ'"
using assms unfolding quality_increases_def by clarsimp

lemma quality_increases_rt_fresherD [dest]:
fixes ip
assumes "quality_increases ξ ξ'" and "ip∈kD(rt ξ)"
shows "rt ξ ⊑ ip rt ξ'"
using assms by auto

lemma quality_increases_sqnE [elim]:
fixes dip
assumes "quality_increases ξ ξ'" and "sqn (rt ξ) dip ≤ sqn (rt ξ') dip =⇒ R dip ξ ξ'"
shows "R dip ξ ξ'"
using assms unfolding quality_increases_def by clarsimp

lemma quality_increases_refl [intro, simp]: "quality_increases ξ ξ"
by rule simp_all

lemma strictly_fresher_quality_increases_right [elim]:
fixes σ σ' dip
assumes "rt (σ i) ⊑ dip rt (σ nhip)" and qinc: "quality_increases (σ nhip) (σ' nhip)"

end
and \( \text{dip} \in kD(rt (\sigma \text{nhip})) \)

shows \( rt (\sigma i) \sqsubseteq \text{dip} \) \( rt (\sigma' \text{nhip}) \)

proof -
from qinc have \( rt (\sigma \text{nhip}) \sqsubseteq \text{dip} \) \( rt (\sigma' \text{nhip}) \) using \( \text{dip} \in kD(rt (\sigma \text{nhip})) \)
by auto
with \( rt (\sigma i) \sqsubseteq \text{dip} \) \( rt (\sigma \text{nhip}) \) show \(?\)thesis ..
qed

**lemma** \( kD\_quality\_increases \) [elim]:
assumes \( i \in kD(rt \xi) \)
and \( \text{quality\_increases} \ \xi \ \xi' \)
shows \( i \in kD(rt \xi') \)
using assms by auto

**lemma** \( kD\_nsqn\_quality\_increases \) [elim]:
assumes \( i \in kD(rt \xi) \)
and \( \text{quality\_increases} \ \xi \ \xi' \)
shows \( i \in kD(rt \xi') \land \text{nsqn} (rt \xi) \ i \leq \text{nsqn} (rt \xi') \ i \)
proof -
from assms have \( i \in kD(rt \xi') \) ..
moreover with assms have \( rt \xi \sqsubseteq i \) \( rt \xi' \) by auto
ultimately have \( \text{nsqn} (rt \xi) \ i \leq \text{nsqn} (rt \xi') \ i \)
using \( i \in kD(rt \xi) \) by - (erule(2) rt_fresher_imp_nsqn_le)
with \( i \in kD(rt \xi') \) show \(?\)thesis ..
qed

**lemma** \( nsqn\_quality\_increases \) [elim]:
assumes \( i \in kD(rt \xi) \)
and \( \text{quality\_increases} \ \xi \ \xi' \)
shows \( \text{nsqn} (rt \xi) \ i \leq \text{nsqn} (rt \xi') \ i \)
using assms by (rule kD_nsqn_quality_increases [THEN conjunct2])

**lemma** \( kD\_nsqn\_quality\_increases\_trans \) [elim]:
assumes \( i \in kD(rt \xi) \)
and \( s \leq \text{nsqn} (rt \xi) \ i \)
and \( \text{quality\_increases} \ \xi \ \xi' \)
shows \( i \in kD(rt \xi') \land s \leq \text{nsqn} (rt \xi') \ i \)
proof -
from \( i \in kD(rt \xi) \) and \( \text{quality\_increases} \ \xi \ \xi' \) show \( i \in kD(rt \xi') \) ..
next
from \( i \in kD(rt \xi) \) and \( \text{quality\_increases} \ \xi \ \xi' \) have \( \text{nsqn} (rt \xi) \ i \leq \text{nsqn} (rt \xi') \ i \) ..
with \( s \leq \text{nsqn} (rt \xi) \ i \) show \( s \leq \text{nsqn} (rt \xi') \ i \) by (rule le_trans)
qed

**lemma** \( nsqn\_quality\_increases\_nsqn\_lt\_lt \) [elim]:
assumes \( i \in kD(rt \xi) \)
and \( \text{quality\_increases} \ \xi \ \xi' \)
and \( s < \text{nsqn} (rt \xi) \ i \)
shows \( s < \text{nsqn} (rt \xi') \ i \)
proof -
from assms(1-2) have \( \text{nsqn} (rt \xi) \ i \leq \text{nsqn} (rt \xi') \ i \) ..
with \( s < \text{nsqn} (rt \xi) \ i \) show \( s < \text{nsqn} (rt \xi') \ i \) by simp
qed

**lemma** \( nsqn\_quality\_increases\_dhops \) [elim]:
assumes \( i \in kD(rt \xi) \)
and \( \text{quality\_increases} \ \xi \ \xi' \)
and \( \text{nsqn} (rt \xi) \ i = \text{nsqn} (rt \xi') \ i \)
shows \( \text{the} (\text{dhops} (rt \xi) \ i) \geq \text{the} (\text{dhops} (rt \xi') \ i) \)
using assms unfolding quality_increases_def
by (clarsimp) (drule(1) bspec, clarsimp simp: rt_fresher_def2)

**lemma** \( nsqn\_quality\_increases\_nsqn\_eq\_le \) [elim]:
assumes \( i \in kD(rt \xi) \)
and \( \text{quality\_increases} \ \xi \ \xi' \)
and \( \text{nsqn} (rt \xi) \ i = \text{nsqn} (rt \xi') \ i \)
shows \( \text{the} (\text{dhops} (rt \xi) \ i) = \text{the} (\text{dhops} (rt \xi') \ i) \)
using assms unfolding quality_increases_def
by (clarsimp) (drule(1) bspec, clarsimp simp: rt_fresher_def2)
and "quality_increases \( \xi \xi \)"
and "s = nsqn (rt \( \xi \) ) i"
shows "s < nsqn (rt \( \xi \) ) i \lor (s = nsqn (rt \( \xi \) ) i \land the (dhops (rt \( \xi \) ) i) \geq the (dhops (rt \( \xi \) ) i))"
using assms by (metis nat_less_le nsqn_quality_increases nsqn_quality_increases_dhops)

lemma quality_increases_rreq_rrep_props [elim]:
fixes sn ip hops sip
assumes qinc: "quality_increases (\( \sigma \) sip) (\( \sigma \)' sip)"
and "1 \leq sn"
*: "ip \in kD (rt (\( \sigma \) sip)) \land sn \leq nsqn (rt (\( \sigma \) sip)) ip
\land (nsqn (rt (\( \sigma \) sip)) ip = sn
\lor the (flag (rt (\( \sigma \) sip)) ip) = inv))"
shows "ip \in kD (rt (\( \sigma \)' sip)) \land sn \leq nsqn (rt (\( \sigma \)' sip)) ip
\land (nsqn (rt (\( \sigma \)' sip)) ip = sn
\lor the (flag (rt (\( \sigma \)' sip)) ip) = inv))"
(is "_ \land ?nsqnafter")
proof -
from * obtain "ip \in kD (rt (\( \sigma \) sip))" and "sn \leq nsqn (rt (\( \sigma \) sip)) ip" by auto
from (quality_increases (\( \sigma \) sip) (\( \sigma \)' sip)) have "sqn (rt (\( \sigma \) sip)) ip \leq sqn (rt (\( \sigma \)' sip)) ip" ..
from (quality_increases (\( \sigma \) sip) (\( \sigma \)' sip)) and (ip \in kD (rt (\( \sigma \) sip))) have "ip \in kD (rt (\( \sigma \)' sip))" ..
from (sn \leq nsqn (rt (\( \sigma \) sip)) ip) have ?nsqnafter
proof
assume "sn < nsqn (rt (\( \sigma \) sip)) ip"
also from (ip \in kD (rt (\( \sigma \) sip))) and (quality_increases (\( \sigma \) sip) (\( \sigma \)' sip))
have "... \leq nsqn (rt (\( \sigma \)' sip)) ip" ..
finally have "sn < nsqn (rt (\( \sigma \)' sip)) ip" ..
thus ?thesis by simp
next
assume "sn = nsqn (rt (\( \sigma \) sip)) ip"
with (ip \in kD (rt (\( \sigma \) sip))) and (quality_increases (\( \sigma \) sip) (\( \sigma \)' sip))
have "sn < nsqn (rt (\( \sigma \)' sip)) ip
\lor (nsqn (rt (\( \sigma \)' sip)) ip = sn \land (the (dhops (rt (\( \sigma \)' sip)) ip) \leq hops
\lor the (flag (rt (\( \sigma \)' sip)) ip) = inv))"
proof
assume "sn < nsqn (rt (\( \sigma \)' sip)) ip" thus ?thesis ..
next
assume "sn = nsqn (rt (\( \sigma \) sip)) ip
\land the (dhops (rt (\( \sigma \) sip)) ip) \geq the (dhops (rt (\( \sigma \) sip)) ip)"
hence "sn = nsqn (rt (\( \sigma \)' sip)) ip"
and "the (dhops (rt (\( \sigma \)' sip)) ip) \leq the (dhops (rt (\( \sigma \) sip)) ip)" by auto
from * and (sn = nsqn (rt (\( \sigma \) sip)) ip) have "the (dhops (rt (\( \sigma \) sip)) ip) \leq hops
\lor the (flag (rt (\( \sigma \) sip)) ip) = inv"
by simp
thus ?thesis
proof
assume "the (dhops (rt (\( \sigma \) sip)) ip) \leq hops"
with (the (dhops (rt (\( \sigma \) sip)) ip) \leq the (dhops (rt (\( \sigma \) sip)) ip))
have "the (dhops (rt (\( \sigma \) sip)) ip) \leq hops" by simp
with (sn = nsqn (rt (\( \sigma \) sip)) ip) show ?thesis by simp
next
assume "the (flag (rt (\( \sigma \) sip)) ip) = inv"
with (ip \in kD (rt (\( \sigma \) sip))) have "nsqn (rt (\( \sigma \) sip)) ip = sqn (rt (\( \sigma \) sip)) ip - 1" ..
with (sn \geq 1) and (sn = nsqn (rt (\( \sigma \) sip)) ip)
have "sqn (rt (σ sip)) ip > 1" by simp

from \(\text{ip} \in \text{kD}(rt (σ' sip))\) show \(?\text{thesis}\)

proof (rule vD_or_iD)

assume "\(\text{ip} \in \text{iD}(rt (σ' sip))\)"

hence "the (flag (rt (σ' sip)) ip) = inv" ..

with \(\text{sn} = \text{nsqn (rt (σ' sip)) ip}\) show \(?\text{thesis}\)

by simp

next

assume "\(\text{ip} \in \text{vD}(rt (σ' sip))\)"

hence "\(\text{nsqn (rt (σ' sip)) ip} = \text{sqn (rt (σ' sip)) ip}\)" ..

with \(\text{sn} = \text{nsqn (rt (σ' sip)) ip}\) have "\(\text{nsqn (rt (σ' sip)) ip} \geq \text{sqn (rt (σ' sip)) ip}\)" by simp

with \(\text{sqn (rt (σ' sip)) ip} > 1\)

have "\(\text{nsqn (rt (σ' sip)) ip} > \text{sqn (rt (σ' sip)) ip} - 1\)" by simp

with \(\text{nsqn (rt (σ' sip)) ip} = \text{sn}\) have "\(\text{nsqn (rt (σ' sip)) ip} > \text{nsqn (rt (σ' sip)) ip}\)" by simp

thus \(?\text{thesis}\) ..

qed

thus \(?\text{thesis}\) by (metis (mono_tags) le_cases not_le)

qed

lemma quality_increases_rreq_rrep_props':

fixes sn ip hops sip

assumes "\(\forall j. \text{quality_increases (σ j) (σ' j)}\)"

and "\(1 \leq \text{sn}\)"

and \(*\): "\(\text{ip} \in \text{kD}(rt (σ sip)) \land \text{sn} \leq \text{nsqn (rt (σ sip)) ip}\)

\(\land (\text{nsqn (rt (σ sip)) ip} = \text{sn}\)

\(\rightarrow (\text{the (dhops (rt (σ sip)) ip}) \leq \text{hops}

\lor \text{the (flag (rt (σ sip)) ip}) = \text{inv}))\)"

shows "\(\text{ip} \in \text{kD}(rt (σ' sip)) \land \text{sn} \leq \text{nsqn (rt (σ' sip)) ip}\)

\(\land (\text{nsqn (rt (σ' sip)) ip} = \text{sn}\)

\(\rightarrow (\text{the (dhops (rt (σ' sip)) ip}) \leq \text{hops}

\lor \text{the (flag (rt (σ' sip)) ip}) = \text{inv}))\)"

proof -

from \(\text{assms}(1)\) have "\(\text{quality_increases (σ sip) (σ' sip)}\)" ..

thus \(?\text{thesis}\) using \(\text{assms}(2-3)\) by (rule quality_increases_rreq_rrep_props)

qed

lemma rteq_quality_increases:

assumes "\(\forall j. j \neq i \rightarrow \text{quality_increases (σ j) (σ' j)}\)"

and "\(\text{rt (σ' i) = rt (σ i)}\)"

shows "\(\forall j. \text{quality_increases (σ j) (σ' j)}\)"

using \(\text{assms}\) by clarsimp (metis order_refl quality_increasesI rt_fresher_refl)

definition msg_fresh :: "(ip ⇒ state) ⇒ msg ⇒ bool"

where "msg_fresh σ m ≡ 

case m of Rreq hopsc _ _ _ oipc osnc sipc ⇒ osnc ≥ 1 ∧ (sipc ≠ oipc → 

(oipc∈kD(rt (σ sipc)) ∧ nsqn (rt (σ sipc)) oipc ≥ osnc

\land (nsqn (rt (σ sipc)) oipc = osnc

\rightarrow (hopsc ≥ the (dhops (rt (σ sipc)) oipc)

\lor the (flag (rt (σ sipc)) oipc) = \text{inv})))

| Rrep hopsc dipc dsnc _ sipc ⇒ dsnc ≥ 1 ∧ (sipc ≠ dipc → 

dipc∈kD(rt (σ sipc)) ∧ nsqn (rt (σ sipc)) dipc ≥ dsnc

\land (nsqn (rt (σ sipc)) dipc = dsnc

\rightarrow (hopsc ≥ the (dhops (rt (σ sipc)) dipc)

| ...

301
lemma msg_fresh [simp]:
"∀ hops rreqid dip dsn dsk oip osn sip.
msg_fresh σ (Rreq hops rreqid dip dsn dsk oip osn sip) =
(osn ≥ 1 ∧ (sip ≠ oip → oip∈kD(rt (σ sip)))
∧ nsqn (rt (σ sip)) oip ≥ osn
∧ (nsqn (rt (σ sip))) oip = osn
→ (hops ≥ the (dhops (rt (σ sip)) oip)
∨ the (flag (rt (σ sip))) oip = inv)))"

"∀ hops dip dsn oip sip. msg_fresh σ (Rrep hops dip dsn oip sip) =
(dsn ≥ 1 ∧ (sip ≠ dip → dip∈kD(rt (σ sip)))
∧ nsqn (rt (σ sip)) dip ≥ dsn
∧ (nsqn (rt (σ sip))) dip = dsn
→ (hops ≥ the (dhops (rt (σ sip)) dip))
∨ the (flag (rt (σ sip))) dip = inv)))"

"∀ dests sip. msg_fresh σ (Rerr dests sip) =
(∀ ripc∈dom(dests). (ripc∈kD(rt (σ sip)))
∧ the (dests ripc) - 1 ≤ nsqn (rt (σ sip)) ripc))"

"∀ d dip. msg_fresh σ (Newpkt d dip) = True"

"∀ d dip sip. msg_fresh σ (Pkt d dip sip) = True"

unfolding msg_fresh_def by simp_all

lemma msg_fresh_inc_sn [simp, elim]:
"msg_fresh σ m =⇒ rreq_rrep_sn m"
by (cases m) simp_all

lemma recv_msg_fresh_inc_sn [simp, elim]:
"orecvmsg (msg_fresh) σ m =⇒ recvmsg rreq_rrep_sn m"
by (cases m) simp_all

lemma rreq_nsqn_is_fresh [simp]:
fixes σ msg hops rreqid dip dsn dsk oip osn sip
assumes "rreq_rrep_fresh (rt (σ sip)) (Rreq hops rreqid dip dsn dsk oip osn sip)"
and "rreq_rrep_sn (Rreq hops rreqid dip dsn dsk oip osn sip)"
shows "msg_fresh σ (Rreq hops rreqid dip dsn dsk oip osn sip)"
(is "msg_fresh σ ?msg")

proof -
let ?rt = "rt (σ sip)"
from assms(2) have "1 ≤ osn" by simp
thus ?thesis

unfolding msg_fresh_def

proof (simp only: msg.case, intro conjI impI)
assume "sip ≠ oip"
with assms(1) show "oip∈kD(?rt)" by simp

next
assume "sip ≠ oip"
and "nsqn ?rt oip = osn"
show "the (dhops ?rt oip) ≤ hops ∨ the (flag ?rt oip) = inv"

proof (cases "oip∈vD(?rt)")
assume "oip∈vD(?rt)"
hence "nsqn ?rt oip = sqn ?rt oip" ..
with assms(1) have "sqn ?rt oip = osn" by simp
with assms(1) and ⟨sip ≠ oip⟩ have "the (dhops ?rt oip) ≤ hops" by simp
thus ?thesis ..

next
assume "oip∈vD(?rt)"
moreover from assms(1) and ⟨sip ≠ oip⟩ have "oip∈kD(?rt)" by simp
ultimately have "oip∈iD(?rt)" by auto
hence "the (flag ?rt oip) = inv" ..
thus \( ? \)thesis ..

qed

next

assume "sip \( \neq \) oip"
with assms(1) have "osn \( \leq \) sqn \( ? \)rt oip" by auto
thus "osn \( \leq \) nsqn \( ? \)rt oip" by (rule sqn_nsqn)

proof (rule nat_le_eq_or_lt)
assume "osn = sqn \( ? \)rt oip"
with assms(1) have "osn \( \leq \) nsqn ?rt oip" by simp
finally show "osn \( \leq \) nsqn \( ? \)rt oip".

next

assume "osn \( \neq \) sqn ?rt oip"
with assms(1) and \( \langle \) sip \( \neq \) oip \( \rangle \)
have "osn \( \leq \) nsqn ?rt oip" by simp
thus "osn \( \leq \) nsqn ?rt oip" by simp
qed

lemma rrep_nsqn_is_fresh [simp]:
fixes \( \sigma \) msg hops dip dsn oip sip
assumes "rreq_rrep_fresh \( \sigma \) (Rrep hops dip dsn oip sip)"
and "rreq_rrep_sn (Rrep hops dip dsn oip sip)"
shows "msg_fresh \( \sigma \) (Rrep hops dip dsn oip sip)"
(is "msg_fresh \( \sigma \) ?msg")
proof -
let \( ? \)rt = "rt \( \sigma \) sip"
from assms have "sip \( \neq \) dip \( \rightarrow \) dip\( \in \)kD(?rt) \( \land \) sqn ?rt dip = dsn \( \land \) the (flag ?rt dip) = val" by simp
hence "sip \( \neq \) dip \( \rightarrow \) dip\( \in \)kD(?rt) \( \land \) nsqn ?rt dip \( \geq \) dsn" by clarsimp
with assms show "msg_fresh \( \sigma \) ?msg" by clarsimp
qed

lemma rerr_nsqn_is_fresh [simp]:
fixes \( \sigma \) msg dests sip
assumes "rerr_invalid \( \sigma \) (Rerr dests sip)"
shows "msg_fresh \( \sigma \) (Rerr dests sip)"
(is "msg_fresh \( \sigma \) ?msg")
proof -
let \( ? \)rt = "rt \( \sigma \) sip"
from assms have \( \langle \forall \) rip \( \in \)dom(dests). (rip\( \in \)kD(rt \( \sigma \) sip)) \( \land \) the (dests rip) = sqn (rt \( \sigma \) sip) rip)\( \rangle \) by clarsimp
have \( \langle \forall \) rip \( \in \)dom(dests). (rip\( \in \)kD(rt \( \sigma \) sip)) \( \land \) the (dests rip) - 1 \( \leq \) nsqn (rt \( \sigma \) sip) rip)\( \rangle \) by clarsimp

proof
fix rip
assume "rip \( \in \) dom dests"
with \( \langle \rangle \) have "rip\( \in \)kD(rt \( \sigma \) sip)" and "the (dests rip) = sqn (rt \( \sigma \) sip) rip" by auto
from this(2) have "the (dests rip) - 1 = sqn (rt \( \sigma \) sip) rip - 1" by simp
also have "\( \ldots \) \( \leq \) nsqn (rt \( \sigma \) sip) rip" by (rule sqn_nsqn)
finally have "the (dests rip) - 1 \( \leq \) nsqn (rt \( \sigma \) sip) rip".
with \( \langle \rangle \)
show "rip\( \in \)kD(rt \( \sigma \) sip) \( \land \) the (dests rip) - 1 \( \leq \) nsqn (rt \( \sigma \) sip) rip" by clarsimp

303
thus "msg_fresh σ ?msg"
  by simp
qed

lemma quality_increases_msg_fresh [elim]:
assumes qinc: "∀ j. quality_increases (σ j) (σ' j)"
  and "msg_fresh σ m"
shows "msg_fresh σ' m"
using assms(2)
proof (cases m)
  fix hops rreqid dip dsn dsk oip osn sip
  assume [simp]: "m = Rreq hops rreqid dip dsn dsk oip osn sip"
  and "msg_fresh σ m"
  then have "osn ≥ 1" and "sip = oip ∨ (oip∈kD(rt (σ sip))) ∧ osn ≤ nsqn (rt (σ sip)) oip
        ∧ (nsqn (rt (σ sip)) oip = osn
            → (the (dhops (rt (σ sip)) oip) ≤ hops
                ∨ the (flag (rt (σ sip)) oip) = inv))"
        by auto
  from this(2) show ?thesis
  proof
    assume "sip = oip" with ⟨osn ≥ 1⟩ show ?thesis by simp
  next
    assume "oip∈kD(rt (σ sip)) ∧ osn ≤ nsqn (rt (σ sip)) oip
        ∧ (nsqn (rt (σ sip)) oip = osn
            → (the (dhops (rt (σ sip)) oip) ≤ hops
                ∨ the (flag (rt (σ sip)) oip) = inv))"
    moreover from qinc have "quality_increases (σ sip) (σ' sip)" ..
    ultimately have "oip∈kD(rt (σ sip)) ∧ osn ≤ nsqn (rt (σ sip)) oip
        ∧ (nsqn (rt (σ sip)) oip = osn
            → (the (dhops (rt (σ sip)) oip) ≤ hops
                ∨ the (flag (rt (σ sip)) oip) = inv))"
        using ⟨osn ≥ 1⟩ by (rule quality_increases_rreq_rrep_props [rotated 2])
    with ⟨osn ≥ 1⟩ show "msg_fresh σ' m"
    by (clarsimp)
  qed
next
  fix hops dip dsn oip sip
  assume [simp]: "m = Rrep hops dip dsn oip sip"
  and "msg_fresh σ m"
  then have "dsn ≥ 1" and "sip = dip ∨ (dip∈kD(rt (σ sip))) ∧ dsn ≤ nsqn (rt (σ sip)) dip
        ∧ (nsqn (rt (σ sip)) dip = dsn
            → (the (dhops (rt (σ sip)) dip) ≤ hops
                ∨ the (flag (rt (σ sip)) dip) = inv))"
        by auto
  from this(2) show "?thesis"
  proof
    assume "sip = dip" with ⟨dsn ≥ 1⟩ show ?thesis by simp
  next
    assume "dip∈kD(rt (σ sip)) ∧ dsn ≤ nsqn (rt (σ sip)) dip
        ∧ (nsqn (rt (σ sip)) dip = dsn
            → (the (dhops (rt (σ sip)) dip) ≤ hops
                ∨ the (flag (rt (σ sip)) dip) = inv))"
    moreover from qinc have "quality_increases (σ sip) (σ' sip)" ..
    ultimately have "dip∈kD(rt (σ sip)) ∧ dsn ≤ nsqn (rt (σ sip)) dip
        ∧ (nsqn (rt (σ sip)) dip = dsn
            → (the (dhops (rt (σ sip)) dip) ≤ hops
                ∨ the (flag (rt (σ sip)) dip) = inv))"
        using ⟨dsn ≥ 1⟩ by (rule quality_increases_rreq_rrep_props [rotated 2])
    with ⟨dsn ≥ 1⟩ show "msg_fresh σ' m"
    by clarsimp
  qed
next
  fix dests sip
assume [simp]: "m = Rerr dests sip"
and "msg_fresh σ m"
then have *: "∀ rip∈dom(dests). rip∈kD(rt (σ sip))
∧ the (dests rip) - 1 ≤ nsqn (rt (σ sip)) rip"
by simp
have "∀ rip∈dom(dests). rip∈kD(rt (σ' sip))
∧ the (dests rip) - 1 ≤ nsqn (rt (σ' sip)) rip"
proof
fix rip
assume "rip∈dom(dests)"
with * have "rip∈kD(rt (σ sip))" and "the (dests rip) - 1 ≤ nsqn (rt (σ sip)) rip"
by - (drule(1) bspec, clarsimp)+
moreover from qinc have "quality_increases (σ sip) (σ' sip)" by simp
ultimately show "rip∈kD(rt (σ' sip)) ∧ the (dests rip) - 1 ≤ nsqn (rt (σ' sip)) rip" ..
qed
thus ?thesis by simp
qed simp_all
end

3.8 The 'open' AODV model

theory C_AODV
imports C_AODV ../../../AWN/OAWN_SOS_Labels ../../../AWN/OAWN_Convert
begin
Definitions for stating and proving global network properties over individual processes.
definition σ_AODV' :: "((ip ⇒ state) × ((state, msg, pseqp, pseqp label) seqp)) set"
where "σ_AODV' ≡ {λi. aodv_init i, ΓAODV PAodv}"
abbreviation opaodv :: "ip ⇒ ((ip ⇒ state) × (state, msg, pseqp, pseqp label) seqp, msg seq_action) automaton"
where "opaodv i ≡ (init = σ_AODV', trans = oseqp_sos ΓAODV i)"
lemma initiali_aodv [intro!, simp]: "initiali i (init (opaodv i)) (init (paodv i))"
unfolding σ_AODV_def σ_AODV'_def by rule simp_all
lemma oadvc_control_within [simp]: "control_within ΓAODV (init (opaodv i))"
unfolding σ_AODV'_def by (rule control_withinI) (auto simp del: ΓAODV_simps)
lemma σ_AODV'_labels [simp]: "(σ, p) ∈ σ_AODV' ⇒ labels ΓAODV p = {PAodv-:0}"
unfolding σ_AODV'_def by simp
lemma oadvc_init_kD_empty [simp]: 
"(σ, p) ∈ σ_AODV' ⇒ kD (rt (σ i)) = {}"
unfolding σ_AODV'_def kD_def by simp
lemma oadvc_init_vD_empty [simp]: 
"(σ, p) ∈ σ_AODV' ⇒ vD (rt (σ i)) = {}"
unfolding σ_AODV'_def vD_def by simp
lemma oadvc_trans: "trans (opaodv i) = oseqp_sos ΓAODV i"
by simp
declare oseq_invariant_ctermsI [OF aodvc_wf oadvc_control_within aodvc_simple_labels oadvc_trans, cterms_intros]
oseq_step_invariant_ctermsI [OF aodvc_wf oadvc_control_within aodvc_simple_labels oadvc_trans, cterms_intros]
end
3.9 Global invariant proofs over sequential processes

theory C_Global_Invariants
imports C_Seq_Invariants
C_Aodv_Predicates
C_Fresher
C_Quality_Increases
"../../../AWN/OAWN_Convert"
C_OAodv
begin

lemma other_quality_increases [elim]:
  assumes other: "∀ j. quality_increases (σ j) (σ' j)"
  shows "∀ j. quality_increases (σ j) (σ' j)"
  using assms by (rule, clarsimp) (metis quality_increases_refl)

lemma weaken_otherwith [elim]:
  fixes m
  assumes *: "otherwith P I (orecvmsg Q) σ σ' (receive m)
  and weakenP: "∀ σ m. P σ m =⇒ P' σ m"
  and weakenQ: "∀ σ m. Q σ m =⇒ Q' σ m"
  shows "otherwith P' I (orecvmsg Q') σ σ' (receive m)"
  proof
    fix j
    assume "j /∈ I" with * have "P (σ j) (σ' j)"
    thus "P' (σ j) (σ' j)" by (rule weakenP)
  next
    from * have "orecvmsg Q σ a" by auto
    thus "orecvmsg Q' σ a" by rule (erule weakenQ)
  qed

definition oreceived_msg_inv:
  assumes other: "∀ σ σ' m. [ P σ m; other Q {i} (σ σ' m) ] =⇒ P σ' m"
  and local: "∀ σ m. P σ m =⇒ P (σ (msg := m)) m"
  shows "opaodv i⇧= (otherwith Q {i} (orecvmsg P), other Q {i} a) onl Γ AODV (λ (σ, l). l ∈ {PAodv-:1} −→ P σ (msg (σ i)))"
  proof (inv_cterms, intro impI)
    fix σ σ' l
    assume "l = PAodv-:1 =⇒ P σ (msg (σ i))"
    and "l = PAodv-:1"
    and "other Q {i} σ σ'"
    from this(1-2) have "P σ (msg (σ i))" .. hence "P σ' (msg (σ i))" using "other Q {i} σ σ'" by (rule other)
    moreover from "other Q {i} σ σ'" have "σ i = σ i" .. ultimately show "P σ' (msg (σ i))" by simp
  next
    fix σ σ' msg
    assume other: "otherwith Q {i} (orecvmsg P) σ σ' (receive msg)"
    and "σ i = σ i[msg := msg]"
    from this(1) have "P σ msg" and "∀ j. j ≠ i =⇒ Q (σ j) (σ' j)" by auto
    from this(1) have "P (σ i := σ i[msg := msg]) msg" by (rule local)
    thus "P σ' msg" proof (rule other)
      from "σ i = σ i[msg := msg]" and "∀ j. j ≠ i =⇒ Q (σ j) (σ' j)"
      show "other Q {i} (σ i := σ i[msg := msg]) σ'" by - (rule otherI, auto)
    qed
  qed

(Equivalent to) Proposition 7.27
lemma local_quality_increases:
"paodv i |=A (recvmsg req_rrep_sn) onll Γ_{AODV} \(\lambda((\xi, \_), \_\), (\xi', \_)). quality_increases \((\xi', \_))\)
proof (rule step_invariantI)
fix s a s'
assume sr: "s ∈ reachable (paodv i) (recvmsg req_rrep_sn)"
and tr: "(s, a, s') ∈ trans (paodv i)"
and rm: "recvmsg req_rrep_sn a"
from sr have srTT: "s ∈ reachable (paodv i) TT " ..
from route_tables_fresher sr tr rm
have "onll Γ_{AODV} (\lambda((\xi, \_), \_\), (\xi', \_)). quality_increases \xi ξ'
∀ dip∈kd (rt \xi). rt \xi ⊑ dip rt \xi' (s, a, s')" by (rule step_invariantD)
moreover from known_destinations_increase srTT tr TT_True
have "onll Γ_{AODV} (\lambda((\xi, \_), \_\), (\xi', \_)). kD (rt \xi) ⊆ kD (rt \xi') (s, a, s')" by (rule step_invariantD)
moreover from sqns_increase srTT tr TT_True
have "onll Γ_{AODV} (\lambda((\xi, \_), \_\), (\xi', \_)). ∀ ip. sqn (rt \xi) ip ≤ sqn (rt \xi') ip) (s, a, s')" by (rule step_invariantD)
ultimately show "onll Γ_{AODV} (\lambda((\xi, \_), \_\), (\xi', \_)). quality_increases \xi ξ' (s, a, s')" unfolding onll_def by auto
qed
lemmas olocal_quality_increases = open_seq_step_invariant [OF local_quality_increases initiali_aodv oaodv_trans aodv_trans, simplified seqll_onll_swap]

lemma equality_increases:
"opaodv i |=A (otherwith quality_increases {i} (orecvmsg (λ_. rreq_rrep_sn)),
other quality_increases {i} →) onll Γ_{AODV} (\lambda((\sigma, \_), \_\), (\sigma', \_)). ∀ j. quality_increases (\sigma j) (\sigma' j))"
(is "-_|=A (??S, _→)_")
proof (rule onll_ostep_invariantI, simp)
fix σ p a σ' p' l'
assume or: "(\sigma, p) ∈ reachable (opaodv i) ?S (other quality_increases {i})"
and 1: "l ∈ labels Γ_{AODV} p"
and "??S ??σ ??a" and tr: "((\sigma, p), a, (\sigma', p')) ∈ oseqp_sos Γ_{AODV} i"
and 1': "l' ∈ labels Γ_{AODV} p'"
from this(1-3) have "orecvmsg (λ_. rreq_rrep_sn) σ a"
by (auto dest!: oreachable_weakenE [where QS="act (recvmsg rreq_rrep_sn)"
and QU="other quality_increases {i} ‘"]
otherwith_actionD)
with or have orw: "(\sigma, p) ∈ reachable (opaodv i) (act (recvmsg rreq_rrep_sn))
(other quality_increases {i})"
by - (erule oreachable_weakenE, auto)
with tr ll ll' and orecvmsg (λ_. rreq_rrep_sn) σ a have "quality_increases (\sigma i) (\sigma' i)"
by - (drule onll_ostep_invariantD [OF olocal_quality_increases], auto simp: seqll_def)
with QS σ σ' a show "∀ j. quality_increases (\sigma j) (\sigma' j)"
by (auto dest!: otherwith_syncD)
qed

lemma rreq_rrep_nsqn_fresh_any_step_invariant:
"opaodv i |=A (act (recvmsg rreq_rrep_sn), other A {i} →) onll Γ_{AODV} (\lambda((\sigma, \_), a, \_). anycast (msg_fresh \sigma) a)"
proof (rule ostep_invariantI, simp del: act.simps)
fix σ p a σ' p'
assume or: "(\sigma, p) ∈ reachable (opaodv i) (act (recvmsg rreq_rrep_sn)) (other A {i})"
and "((\sigma, p), a, (\sigma', p')) ∈ oseqp_sos Γ_{AODV} i"
and recv: "act (recvmsg rreq_rrep_sn) σ σ' a"
obtain l l' where "l∈labels Γ_{AODV} p" and "l'∈labels Γ_{AODV} p'"
by (metis aodv_ex_label)

307
have "anycast (rreq_rrep_fresh (rt (σ i))) a"
proof -
  have "opaodv i \|= A (act (recvmsg rreq_rrep_sn), other A {i} →)
    onll Γ_AODV (seqll i (λ((ξ, _), a, _). anycast (rreq_rrep_fresh (rt ξ)) a))"
    by (rule ostep_invariant_weakenE [OF
      open_seq_step_invariant [OF rreq_rrep_fresh_any_step_invariant initiali_aodv,
      simplified seqll_onll_swap]) auto
  hence "onll Γ_AODV (seqll i (λ((ξ, _), a, _). anycast (rreq_rrep_fresh (rt ξ)) a))
    ((σ, p), a, (σ', p'))"
  using or tr recv by - (erule(4) ostep_invariantE)
  thus ?thesis using ⟨l∈labels Γ_AODV p⟩ and ⟨l'∈labels Γ_AODV p'⟩ by auto
qed

moreover have "anycast (rerr_invalid (rt (σ i))) a"
proof -
  from or tr recv
  have "onll Γ_AODV (seqll i (λ((ξ, _), a, _). anycast (rerr_invalid (rt ξ)) a))
    ((σ, p), a, (σ', p'))"
  using or tr recv by - (erule(4) ostep_invariantE)
  thus ?thesis using ⟨l∈labels Γ_AODV p⟩ and ⟨l'∈labels Γ_AODV p'⟩ by auto
qed

moreover have "anycast rreq_rrep_sn a"
proof -
  from or tr recv
  have "onll Γ_AODV (seqll i (λ(_, a, _). anycast rreq_rrep_sn a)) ((σ, p), a, (σ', p'))"
  by (rule ostep_invariantE [OF
    open_seq_step_invariant [OF rreq_rrep_sn_any_step_invariant initiali_aodv,
    simplified seqll_onll_swap]) auto
  thus ?thesis using ⟨l∈labels Γ_AODV p⟩ and ⟨l'∈labels Γ_AODV p'⟩ by auto
qed

moreover have "anycast (λm. not_Pkt m −→ msg_sender m = i) a"
proof -
  have "opaodv i \|= A (act (recvmsg rreq_rrep_sn), other A {i} →)
    onll Γ_AODV (seqll i (λ((ξ, _), a, _). anycast (λm. not_Pkt m −→ msg_sender m = i) a))"
    by (rule ostep_invariant_weakenE [OF
      open_seq_step_invariant [OF sender_ip_valid initiali_aodv,
      oadv_trans aodv_trans,
      simplified seqll_onll_swap]]) auto
  thus ?thesis using or tr recv (l∈labels Γ_AODV p) and (l'∈labels Γ_AODV p') by auto
qed

ultimately have "anycast (msg_fresh σ) a"
by (simp_all add: anycast_def
  del: msg_fresh
  split: seq_action.split_asm msg.split_asm) simp_all
thus "onll Γ_AODV (λ((σ, _), a, _). anycast (msg_fresh σ) a) ((σ, p), a, (σ', p'))"
by auto
qed

lemma oreceived_rreq_rrep_nsqn_fresh_inv:
"opaodv i \|= (otherwith quality_increases {i} (orecvmsg msg_fresh),

ultimately have "anycast (msg_fresh σ) a"
by (simp_all add: anycast_def
  del: msg_fresh
  split: seq_action.split_asm msg.split_asm) simp_all
thus "onll Γ_AODV (λ((σ, _), a, _). anycast (msg_fresh σ) a) ((σ, p), a, (σ', p'))"
by auto
qed
other quality_increases {i} →
onl Γ_{AODV} (λ(σ, l). 1 ∈ \{PAodv-:1\} → msg_fresh σ (msg (σ i)))

proof (rule oreceived_msg_inv)
fix σ σ' m
assume #: "msg_fresh σ m"
and "other quality_increases {i} σ σ'"
from this(2) have "∀ j. quality_increases (σ j) (σ' j)" ..
thus "msg_fresh σ' m" using * ..

next
fix σ m
assume "msg_fresh σ m"
thus "msg_fresh (σ_i := σ i (msg := m)) m"

proof (cases m)
fix dests sip
assume "m = Rerr dests sip"
with msg_fresh σ m show ?thesis by auto
qed auto

lemma quality_increases_nsqn_fresh:
"opaodv i |= (otherwith quality_increases {i} (orecvmsg msg_fresh),
other quality_increases {i} i) →
onl Γ_{AODV} (λ((σ, _), _, (_, _)). ∀ j. quality_increases (σ j) (σ' j))"

by (rule ostep_invariant_weakenE [OF quality_increases]) auto

lemma oosn_rreq:
"opaodv i |= (otherwith quality_increases {i} (orecvmsg msg_fresh),
other quality_increases {i} i) →
onl Γ_{AODV} (seq_l (λ(ξ, l). 1 ∈ \{PAodv-:4, PAodv-:5\} ∪ \{PRreq-n \mid n. True\} → 1 ≤ osn ξ))"

by (rule oinvariant_weakenE [OF open_seq_invariant [OF oosn_rreq initiali_aodv]])
(auto simp: seql_onl_swap)

lemma rreq_sip:
"opaodv i |= (otherwith quality_increases {i} (orecvmsg msg_fresh),
other quality_increases {i} i) →
onl Γ_{AODV} (λ(σ, l).
(1 ∈ \{PAodv-:4, PAodv-:5, PRreq-:0, PRreq-:2\} ∧ sip (σ i) ≠ oip (σ i))
→ oip (σ i) ∈ kd(rt (σ (sip (σ i))))
∧ nsqn (rt (σ (sip (σ i)))) (oip (σ i)) ≥ osn (σ i)
∧ (nsqn (rt (σ (sip (σ i)))) (oip (σ i)) = osn (σ i))
→ (hops (σ i) ≥ the (dhops (rt (σ (sip (σ i)))) (oip (σ i))))
∨ the (flag (rt (σ (sip (σ i)))) (oip (σ i))) = inv))"

(is "_ |= (?S, ?U → _)"

proof (inv_cterms inv add: oseq_step_invariant_sterms [OF quality_increases_nsqn_fresh]
aodv_wf oaadv_trans]
onl_oinvariant_sterms [OF aodv_wf oreceived_rreq_rrep_nsqn_fresh_inv]
onl_oinvariant_sterms [OF aodv_wf oosn_rreq]
simp add: seql_into)
(simp del: One_nat_def, rule implI)

fix σ σ' p l
assume "(σ, p) ∈ oreachable (opaodv i) ?S ?U"
and "l ∈ labels Γ_{AODV} p"
and pre:
"(1 = PAodv-:4 ∨ 1 = PAodv-:5 ∨ 1 = PRreq-:0 ∨ 1 = PRreq-:2) ∧ sip (σ i) ≠ oip (σ i)
→ oip (σ i) ∈ kd(rt (σ (sip (σ i))))
∧ osn (σ i) ≤ nsqn (rt (σ (sip (σ i)))) (oip (σ i))
∧ (nsqn (rt (σ (sip (σ i)))) (oip (σ i)) = osn (σ i))
→ the (dhops (rt (σ (sip (σ i)))) (oip (σ i))) ≤ hops (σ i)
∨ the (flag (rt (σ (sip (σ i)))) (oip (σ i))) = inv)"

and "other quality_increases {i} σ σ'"
and hyp: "(1 = PAodv-:4 ∨ 1 = PAodv-:5 ∨ 1 = PRreq-:0 ∨ 1 = PRreq-:2) ∧ sip (σ' i) ≠ oip (σ' i)"
(is "?labels ∧ sip (σ' i) ≠ oip (σ' i)"
from this(4) have "σ' i = σ i" ..
with hyp have hyp': "?labels ∧ sip (σ i) ≠ oip (σ i)" by simp
show "oip (σ' i) ∈ kD (rt (σ' (sip (σ' i))))
  ∧ osn (σ' i) ≤ nsqn (rt (σ' (sip (σ' i)))) (oip (σ' i))
  ∧ (nsqn (rt (σ' (sip (σ' i)))) (oip (σ' i)) = osn (σ' i)
    → the (dhops (rt (σ' (sip (σ' i)))) (oip (σ' i))) ≤ hops (σ' i)
    ∨ the (flag (rt (σ' (sip (σ' i)))) (oip (σ' i))) = inv"
proof
  qed (auto elim!: quality_increases_rreq_rrep_props')
  show "oip (σ i) = i"
  assume "oip (σ i) = i"
  from (other quality increases {i} σ σ')
  have "quality_increases (σ (sip (σ i))) (σ (sip (σ i)))"
    by (rule quality_increases_rreq_rrep_props)
  moreover from ⟨(σ, p) ∈ orreachable (oapodv i) ?S ?U | l ∈ labels Γ_{AODV} p⟩
    and hyp
  have "1 ≤ osn (σ i)"
    by (auto dest!: oinvariant_weakenD [OF oosn_rreq]
      simp add: seqlsimp)
  moreover from (sip (σ i) = i) hyp' and pre
  have "oip (σ' i) ∈ kD (rt (σ (sip (σ i))))
    ∧ osn (σ' i) ≤ nsqn (rt (σ (sip (σ i)))) (oip (σ' i))
    ∧ (nsqn (rt (σ (sip (σ i)))) (oip (σ' i)) = osn (σ' i)
      → the (dhops (rt (σ (sip (σ i)))) (oip (σ' i))) ≤ hops (σ' i)
      ∨ the (flag (rt (σ (sip (σ i)))) (oip (σ' i))) = inv"
    by (auto simp: σ' i = σ i)
  ultimately show ?thesis
    by (rule quality_increases_rreq_rrep_props)
next
  assume "oip (σ i) = i" thus ?thesis
  using σ' i = σ i hyp and pre by auto
qed

lemma odsn_rrep:
  "oapodv i ⊢ (otherwith quality_increases {i} (orecvmsg msg_fresh),
    other quality increases {i} →
    onl Γ_{AODV} (seq l (λ(i, l). l ∈ {PAodv-:6, PAodv-:7} ∪ {PRrep-:n|n. True} → 1 ≤ dsn ξ)))"
by (rule oinvariant_weakenE [OF open_seq_invariant [OF dsn_rrep_initiali_aodv]])
(auto simp: seql_onl_swap)

lemma rrep_sip:
  "oapodv i ⊢ (otherwith quality_increases {i} (orecvmsg msg_fresh),
    other quality increases {i} →
    onl Γ_{AODV} (λ(σ, i).
      (l ∈ {PAodv-:6, PAodv-:7, PRrep-:0, PRrep-:1} ∧ sip (σ i) ≠ dip (σ i))
      → dip (σ i) ∈ kD (rt (σ (sip (σ i))))
      ∧ nsqn (rt (σ (sip (σ i)))) (dip (σ i)) ≥ dsn (σ i)
      ∧ (nsqn (rt (σ (sip (σ i)))) (dip (σ i)) = dsn (σ i)
        → (hops (σ i) ≥ the (dhops (rt (σ (sip (σ i)))) (dip (σ i)))
          ∨ the (flag (rt (σ (sip (σ i)))) (dip (σ i))) = inv))"
(is "_ ⊢ (?S, ?U → _)"
proof (inv cterms inv add: oseq_step_invariant_sterms [OF quality_increases nsqn fresh aodv wf oapodv_trans]
  onl_o invariant sterms [OF aodv_wf oreceived rreq rrep nsqn fresh inv]
  onl_o invariant sterms [OF aodv_wf dsn rrep]
  simp del: OneNatDef, rule impI)
fix σ σ' p l
assume "σ, p ∈ orreachable (oapodv i) ?S ?U"
and "l ∈ labels Γ_{AODV} p" and
and pref:
  "(l = PAodv-:6 ∨ l = PAodv-:7 ∨ l = PRrep-:0 ∨ l = PRrep-:1) ∧ sip (σ i) ≠ dip (σ i)
    → dip (σ i) ∈ kD (rt (σ (sip (σ i))))
    ∧ dsn (σ i) ≤ nsqn (rt (σ (sip (σ i)))) (dip (σ i))
    ∧ (nsqn (rt (σ (sip (σ i)))) (dip (σ i)) = dsn (σ i)
      → the (dhops (rt (σ (sip (σ i)))) (dip (σ i))) ≤ hops (σ i)
      ∨ the (flag (rt (σ (sip (σ i)))) (dip (σ i))) = inv"
and "other quality increases {i} σ σ'"
and hyp: "(l=PAodv-:6 ∨ l=PAodv-:7 ∨ l=PRrep-:0 ∨ l=PRrep-:1) ∧ sip (σ' i) ≠ dip (σ' i)"
lemma rerr_sip:

is "labels ∧ sip (σ' i) ≠ dip (σ' i)"

from this(4) have "σ' i = σ i ..

with hyp have hyp': "labels ∧ sip (σ i) ≠ dip (σ i)" by simp

show "dip (σ' i) ∈ kD (rt (σ' (sip (σ' i)))) ∧
dsn (σ' i) ≤ nsqn (rt (σ' (sip (σ' i)))) (dip (σ' i)) ∧
(nsqn (rt (σ' (sip (σ' i)))) (dip (σ' i)) = dsn (σ' i)
→ the (dhops (rt (σ' (sip (σ' i)))) (dip (σ' i))) ≤ hops (σ' i)
∧ the (flag (rt (σ' (sip (σ' i)))) (dip (σ' i)) = inv)"

proof (cases "sip (σ i) = i")

assume "sip (σ i) ≠ i"

from other quality_increases {i} σ σ' have "quality_increases (σ (sip (σ i))) (σ' (sip (σ' i)))" by (rule otherE) (clarsimp simp: (sip (σ i) ≠ i))

moreover from (σ, p) ∈ oreachable (opaodv i) ?S ?U. l ∈ labels Γ_AODV p and hyp have "l ≤ dsn (σ' i)"

by (auto dest!: onl_onvariant_weakenD [OF odsn_rrep]
simp add: sqeqlsimp σ' i = σ i)

moreover from 'sip (σ i) ≠ i' hyp' and pre have "dip (σ' i) ∈ kD (rt (σ (sip (σ i)))) ∧
dsn (σ' i) ≤ nsqn (rt (σ (sip (σ i)))) (dip (σ' i)) ∧
(nsqn (rt (σ (sip (σ i)))) (dip (σ' i)) = dsn (σ' i)
→ the (dhops (rt (σ (sip (σ i)))) (dip (σ' i))) ≤ hops (σ' i)
∧ the (flag (rt (σ (sip (σ i)))) (dip (σ' i)) = inv)"

by (auto simp: 'σ' i = σ i)

ultimately show ?thesis

next

assume "sip (σ i) = i" thus ?thesis

using σ' i = σ i hyp and pre by auto

qed

qed (auto simp add: sqeqlsimp elim!: quality_increases_rreq_rrep_props')

lemma rerr_sip:

"opaodv i |-- (otherwith quality_increases {i} (orecvmsg msg_fresh),
other quality_increases {i} →)
onl Γ_AODV (λ(σ, l).
1 ∈ {PAodv−:8, PAodv−:9, PRerr−:0, PRerr−:1} →
(∀rip∈dom(dests (σ i)). rip∈kD(rt (σ (sip (σ i)))) ∧
the (dests (σ i) ripc) - 1 ≤ nsqn (rt (σ (sip (σ i)))) rip))"

(is "_ |-- (?S, ?U →) _")

proof -

{ fix dests rip sip rsn and σ σ' :: "ip ⇒ state"
  assume qinc: "∀j. quality_increases (σ j) (σ' j)"
  and *: "∀rip∈dom dests. rip ∈ kD (rt (σ sip))
∧ the (dests ripc) - 1 ≤ nsqn (rt (σ sip)) rip"
  and "dests rip = Some rsn"

  from this(3) have "rip∈dom dests" by auto
  with * and (dests rip = Some rsn) have "rip∈kD(rt (σ sip))"
  and "rsn - 1 ≤ nsqn (rt (σ sip)) rip"

  by (auto dest!: bspec)

  from qinc have "quality_increases (σ sip) (σ' sip)" ..
  have "rip ∈ kD(rt (σ' sip)) ∧ rsn - 1 ≤ nsqn (rt (σ' sip)) rip"

  proof
  from rip∈kD(rt (σ sip))) and (quality_increases (σ sip) (σ' sip))
  show "rip ∈ kD(rt (σ' sip))") ..

  next
  from rip∈kD(rt (σ sip))) and (quality_increases (σ sip) (σ' sip))
  have "nsqn (rt (σ sip)) rip ≤ nsqn (rt (σ' sip)) rip" ..
  with rsn - 1 ≤ nsqn (rt (σ sip)) rip) show "rsn - 1 ≤ nsqn (rt (σ' sip)) rip"
  by (rule le_trans)

  qed
  }
  note partial = this

  show ?thesis

show ?thesis
by (inv_cterms inv add: oseq_step_invariant_sterms [OF equality_increases nsqn_fresh aodv_wf aodv_trans]
onl_oInvariant_sterms [OF aodv_wf oreceived_rreq_rrep nsqn_fresh_inv]
other_quality_increases other_localD
simp del: One_nat_def, intro conjI)
(clarsimp simp del: One_nat_def split: split_if_asm option.split_asm, erule(2) partial)+
qed

lemma prerr_guard: "paodv i \|=\n onl \Gamma_{AODV} (\lambda(\xi, l). (l = PRerr-:1 \rightarrow (\forall ip \in \text{dom}(\text{dests} \ \xi). ip \in vD(\text{rt} \ \xi) \land \text{sqn} (\text{rt} \ \xi) \ ip < \text{the}(\text{dests} \ \xi \ ip)))"
by (inv_cterms) (clarsimp split: option.split_asm split_if_asm)

lemmas odests_vD_inc_sqn =
open_seq_invariant [OF dests_vD_inc_sqn initiali_aodv oadv_trans oadv_trans,
simplified seql_onl_swap,
THEN oinvariant_anyact]

lemmas oprerr_guard =
open_seq_invariant [OF prerr_guard initiali_aodv oadv_trans oadv_trans,
simplified seql_onl_swap,
THEN oinvariant_anyact]

Proposition 7.28

lemma seq_compare_next_hop':
"opaodv i \|= \ (\text{otherwith quality_increases} \ \{i\} (\text{orecvmsg msg_fresh}),
other_quality_increases \ \{i\} \rightarrow) onl \ \Gamma_{AODV} (\lambda(\sigma, _). \forall dip. let nhip = \text{the}(\text{nhop}(\text{rt} \ (\sigma i)) dip)
in dip \in kD(\text{rt} \ (\sigma i)) \land nhip \neq dip \rightarrow
dip \in kD(\text{rt} \ (\sigma nhip)) \land \text{nsqn} (\text{rt} \ (\sigma i)) dip \leq \text{nsqn} (\text{rt} \ (\sigma nhip)) dip)"
(is "\_ \|= \ (?S, ?U \rightarrow) \ _")
proof -

{ fix nhop and \sigma \sigma' :: "ip \Rightarrow state"
  assume pre: "\forall dip \in kD(\text{rt} \ (\sigma i)). nhop dip \neq dip \rightarrow
dip \in kD(\text{rt} \ (\sigma (\text{nhop dip}))) \land \text{nsqn} (\text{rt} \ (\sigma i)) dip \leq \text{nsqn} (\text{rt} \ (\sigma (\text{nhop dip}))) dip"
  and qinc: "\forall j. quality_increases (\sigma j) (\sigma' j)"

  have "\forall dip \in kD(\text{rt} \ (\sigma i)). nhop dip \neq dip \rightarrow
dip \in kD(\text{rt} \ (\sigma' (\text{nhop dip}))) \land \text{nsqn} (\text{rt} \ (\sigma i)) dip \leq \text{nsqn} (\text{rt} \ (\sigma' (\text{nhop dip}))) dip"
    (is "\ _ \|= \ (?S, ?U \rightarrow) \ _")
proof (intro ballI impI)
  fix dip
  assume "dip \in kD(\text{rt} \ (\sigma i))"
  and "nhop dip \neq dip"
  with pre have "dip \in kD(\text{rt} \ (\sigma (\text{nhop dip})))"
    and "\text{nsqn} (\text{rt} \ (\sigma i)) dip \leq \text{nsqn} (\text{rt} \ (\sigma (\text{nhop dip}))) dip"
    by auto
  from qinc have "quality_increases (\sigma (\text{nhop dip})) (\sigma' (\text{nhop dip})))" ..
  with "dip \in kD(\text{rt} \ (\sigma (\text{nhop dip})))" have "dip \in kD(\text{rt} \ (\sigma' (\text{nhop dip})))" ..

  moreover have "\text{nsqn} (\text{rt} \ (\sigma i)) dip \leq \text{nsqn} (\text{rt} \ (\sigma' (\text{nhop dip}))) dip"
proof -
  from "dip \in kD(\text{rt} \ (\sigma (\text{nhop dip})))" qinc_nhop
  have "\text{nsqn} (\text{rt} \ (\sigma (\text{nhop dip}))) dip \leq \text{nsqn} (\text{rt} \ (\sigma' (\text{nhop dip}))) dip" ..
  with "\text{nsqn} (\text{rt} \ (\sigma i)) dip \leq \text{nsqn} (\text{rt} \ (\sigma (\text{nhop dip}))) dip" show thesis
    by simp
  qed

  ultimately show "dip \in kD(\text{rt} \ (\sigma' (\text{nhop dip})))
    \land \text{nsqn} (\text{rt} \ (\sigma i)) dip \leq \text{nsqn} (\text{rt} \ (\sigma' (\text{nhop dip}))) dip" ..
  qed
}

} note basic = this
{ fix nhop and σ σ' :: "ip ⇒ state"

assume pre: "∀ dip ∈ kD(rt (σ i)). nhop dip ≠ dip → dip ∈ kD(rt (σ (nhop dip)))
∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (nhop dip))) dip"

and ndest: "∀ ripc ∈ dom (dests (σ i)). ripc ∈ kD (rt (σ (sip (σ i))))
∧ the (dests (σ i) ripc) - 1 ≤ nsqn (rt (σ (sip (σ i)))) ripc"

and issip: "∀ dip ∈ dom (dests (σ i)). nhop ip = sip (σ i)"

and qinc: "∀ j. quality_increases (σ j) (σ' j)"

have "∀ dip ∈ kD(rt (σ i)). nhop dip ≠ dip → dip ∈ kD (rt (σ' (nhop dip)))
∧ nsqn (invalidate (rt (σ i)) (dests (σ i))) dip ≤ nsqn (rt (σ' (nhop dip))) dip"

proof (intro ballI impl)

fix dip
assume "dip ∈ kD (rt (σ i))"

and "nhop dip ≠ dip"

with pre and qinc have "dip ∈ kD (rt (σ' (nhop dip)))"

and "nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (nhop dip))) dip"

by (auto dest!: basic)

have "nsqn (invalidate (rt (σ i)) (dests (σ i))) dip ≤ nsqn (rt (σ' (nhop dip))) dip"

proof (cases "dip ∈ dom (dests (σ i))")

assume "dip ∈ dom (dests (σ i))"

with ⟨dip ∈ kD(rt (σ i))⟩ obtain dsn where "dests (σ i) dip = Some dsn"

by auto

with ⟨dip ∈ kD(rt (σ i))⟩ have "nsqn (invalidate (rt (σ i)) (dests (σ i))) dip = dsn - 1"

by (rule nsqn_invalidate_eq)

moreover have "dsn - 1 ≤ nsqn (rt (σ' (nhop dip))) dip"

proof -

from ⟨dests (σ i) dip = Some dsn⟩ have "the (dests (σ i) dip) = dsn" by simp

with ndest and ⟨dip ∈ dom (dests (σ i))⟩ have "dip ∈ kD (rt (σ (sip (σ i))))"

"dsn - 1 ≤ nsqn (rt (σ (sip (σ i)))) dip"

by auto

moreover from issip and ⟨dip ∈ dom (dests (σ i))⟩ have "nhop dip = sip (σ i)" ..

ultimately have "dip ∈ kD (rt (σ (nhop dip)))"

and "dsn - 1 ≤ nsqn (rt (σ (nhop dip))) dip" by auto

with qinc show "dsn - 1 ≤ nsqn (rt (σ' (nhop dip))) dip"

by simp (metis kD_nsqn_quality_increases_trans)

qed

ultimately show ?thesis by simp

next

assume "dip ∉ dom (dests (σ i))"

with ⟨dip ∈ kD(rt (σ i))⟩:

have "nsqn (invalidate (rt (σ i)) (dests (σ i))) dip = nsqn (rt (σ i)) dip"

by (rule nsqn_invalidate_other)

with ⟨nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (nhop dip))) dip⟩ show ?thesis by simp

qed

with ⟨dip ∈ kD(rt (σ' (nhop dip)))⟩:

show "dip ∈ kD (rt (σ' (nhop dip)))"

"∧ nsqn (invalidate (rt (σ i)) (dests (σ i))) dip ≤ nsqn (rt (σ' (nhop dip))) dip" ..

qed

} note basic_prerr = this

{ fix σ σ' :: "ip ⇒ state"

assume a1: "∀ dip ∈ kD(rt (σ i)). the (nhop (rt (σ i)) dip) ≠ dip → dip ∈ kD (rt (σ (the (nhop (rt (σ i)) dip))))
∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (the (nhop (rt (σ i)) dip)))) dip"

and a2: "∀ j. quality_increases (σ j) (σ' j)"

have "∀ dip ∈ kD(rt (σ i)).

the (nhop (update (rt (σ i)) (sip (σ i)) (0, unk, val, Suc 0, sip (σ i))) dip) ≠ dip →
dip ∈ kD (rt (σ' (the (nhop (update (rt (σ i)) (sip (σ i)) (0, unk, val, Suc 0, sip (σ i)) dip))))
∧ nsqn (update (rt (σ i)) (sip (σ i)) (0, unk, val, Suc 0, sip (σ i))) dip ≤ nsqn (rt (σ' (the (nhop (update (rt (σ i)) (sip (σ i)) (0, unk, val, Suc 0, sip (σ i)))) dip)))"
dip" (is "∀ dip ∈ kD (rt (σ i)). ?P dip")

proof
fix dip
assume "dip ∈ kD (rt (σ i))"
with a1 and a2
have "the (nhop (rt (σ i)) dip) ≠ dip \implies dip ∈ kD (rt (σ' (the (nhop (rt (σ i)) dip))))
∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (the (nhop (rt (σ i)) dip)))) dip"
by \-(drule(1) basic, auto)
thus "?P dip" by (cases "dip = sip (σ i)") auto
qed

} note nhop_update_sip = this

{ fix σ σ' oip sip osn hops
assume pre: "∀ dip ∈ kD (rt (σ i)). the (nhop (rt (σ i)) dip) ≠ dip
\implies dip ∈ kD (rt (σ' (the (nhop (rt (σ i)) dip))))
∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (the (nhop (rt (σ i)) dip)))) dip"
and qinc: "∀ j. quality_increases (σ j) (σ' j)"
and *: "sip ≠ oip \implies oip ∈ kD (rt (σ sip))
∧ osn ≤ nsqn (rt (σ sip)) oip
∧ (nsqn (rt (σ sip)) oip = osn
\implies the (dhops (rt (σ sip)) oip) ≤ hops
∨ the (flag (rt (σ sip)) oip) = inv)"

from pre and qinc
have pre': "∀ dip ∈ kD (rt (σ i)). the (nhop (rt (σ i)) dip) ≠ dip
\implies dip ∈ kD (rt (σ' (the (nhop (rt (σ i)) dip))))
∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (the (nhop (rt (σ i)) dip)))) dip"
by (rule basic)

have "(the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip)) oip) ≠ oip
\implies oip ∈ kD (rt (σ' (the (nhop (update (rt (σ i)) oip
(osn, kno, val, Suc hops, sip)) oip))))
∧ nsqn (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip)) oip
≤ nsqn (rt (σ' (the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip)) oip)))) oip)"
(is "?nhop_not_oip \implies ?oip_in_kD ∧ ?nsqn_le_nsqn")

proof (rule, split update_rt_split_asm)
assume "rt (σ i) = update (rt (σ i)) oip (osn, kno, val, Suc hops, sip)"
and the (nhop (rt (σ i)) oip) ≠ oip"
with pre' show "?oip_in_kD ∧ ?nsqn_le_nsqn" by auto
next
assume rtnot: "rt (σ i) ≠ update (rt (σ i)) oip (osn, kno, val, Suc hops, sip)"
and notoip: ?nhop_not_oip
with * qinc have ?oip_in_kD
by (clarsimp elim!: kD_quality_increases)
moreover with * pre qinc rtnot notoip have ?nsqn_le_nsqn
by simp (metis kD_nsqn_quality_increases_trans)
ultimately show "?oip_in_kD ∧ ?nsqn_le_nsqn" ..
qed

} note update1 = this

{ fix σ σ' oip sip osn hops
assume pre: "∀ dip ∈ kD (rt (σ i)). the (nhop (rt (σ i)) dip) ≠ dip
\implies dip ∈ kD (rt (σ' (the (nhop (rt (σ i)) dip))))
∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (the (nhop (rt (σ i)) dip)))) dip"
and qinc: "∀ j. quality_increases (σ j) (σ' j)"
and *: "sip ≠ oip \implies oip ∈ kD (rt (σ sip))
∧ osn ≤ nsqn (rt (σ sip)) oip
∧ (nsqn (rt (σ sip)) oip = osn
\implies the (dhops (rt (σ sip)) oip) ≤ hops
∨ the (flag (rt (σ sip)) oip) = inv)"

from pre and qinc
have pre': "∀ dip ∈ kD (rt (σ i)). the (nhop (rt (σ i)) dip) ≠ dip
\implies dip ∈ kD (rt (σ' (the (nhop (rt (σ i)) dip))))
∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (the (nhop (rt (σ i)) dip)))) dip"
by (rule basic)
have "∀ dip ∈ kD(rt (σ i)). 
the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip)) dip) ≠ dip 
→ dip ∈ kD(rt (σ' (the (nhop (update (rt (σ i)) oip 
(osn, kno, val, Suc hops, sip)) dip))))
∧ nsqn (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip)) dip 
≤ nsqn (rt (σ' (the (nhop (update (rt (σ i)) oip 
(osn, kno, val, Suc hops, sip)) dip)))) dip"
(is "∀ dip ∈ kD(rt (σ i)). _ → ?dip_in_kD dip ∧ ?nsqn_le_nsqn dip")

proof (intro ballI impI, split update_rt_split_asm)
  fix dip
  assume "dip ∈ kD(rt (σ i))"
  and "the (nhop (rt (σ i)) dip) ≠ dip"
  and "rt (σ i) = update (rt (σ i)) oip (osn, kno, val, Suc hops, sip)"
  with pre' show "?dip_in_kD dip ∧ ?nsqn_le_nsqn dip" by simp

next
  fix dip
  assume "dip ∈ kD(rt (σ i))"
  and notdip: "the (nhop (update (rt (σ i)) oip 
(osn, kno, val, Suc hops, sip)) dip) ≠ dip"
  and rtnot: "rt (σ i) ≠ update (rt (σ i)) oip (osn, kno, val, Suc hops, sip)"
  show "?dip_in_kD dip ∧ ?nsqn_le_nsqn dip"
  proof (cases "dip = oip")
    assume "dip ≠ oip"
    with pre' ⟨dip ∈ kD(rt (σ i))⟩ notdip *
    have "?dip_in_kD dip" by simp (metis kD_quality_increases)
    moreover from ⟨dip = oip⟩ rtnot qinc ⟨dip ∈ kD(rt (σ i))⟩ notdip *
    have "?nsqn_le_nsqn dip" by simp (metis kD_nsqn_quality_increases_trans)
    ultimately show ?thesis ..
  qed

qed

qed

} note update2 = this

have "opaodv i |= (?S, ?U →) onl Γ AODV (λ(σ, _).
∀ dip ∈ kD(rt (σ i)). the (nhop (rt (σ i)) dip) ≠ dip 
→ dip ∈ kD(rt (σ (the (nhop (rt (σ i)) dip)))))
∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (the (nhop (rt (σ i)) dip)))) dip)"
by (inv_cterms inv add: oseq_step_invariant_sterms [OF oquality_increases_nsqn_fresh aodv_wf
 oaadv_trans] onl_oinvariant_sterms [OF aodv_wf odests_vD_inc_sqn]
 onl_oinvariant_sterms [OF aodv_wf oprerr_guard]
 onl_oinvariant_sterms [OF aodv_wf rreq_sip]
 onl_oinvariant_sterms [OF aodv_wf rrep_sip]
 onl_oinvariant_sterms [OF aodv_wf rerr_sip]
 other_quality_increases
 other_localD
 solve: basic basic_prerr
 simp add: seqlsimp nsqn_invalidate nhop_update_sip
 simp del: One_nat_def)
(rule conjI, erule(2) update1, erule(2) update2)++

thus ?thesis unfolding Let_def by auto

qed

Proposition 7.30

lemmas okD_unk_or_atleast_one =
  open_seq_invariant [OF kD_unk_or_atleast_one initiali_aodv,
  simplified seql_onl_swap]

lemmas ozero_seq_unk_hops_one =
lemma oreachable_fresh_okD_unk_or_atleast_one:
  fixes dip
  assumes "(σ, p) ∈ oreachable (opaodv i)
    (otherwith (op=) (i) (orecvmsg (λσ m. msg_fresh σ m
     ∧ msg_zhops m)))
    (otherquality Increases {i})"
  shows "∃l. l ∈ labels Γ AODV p" by (metis aodv_ex_label)
  with assms(1)
  have "∀dip ∈ kD (rt (σ i)). ?P dip" by -
    (drule oinvariant_weakenD [OF okD_unk_or_atleast_one [OF oaodv_trans aodv_trans]],
     auto dest!: onlD otherwith_actionD simp: seqlsimp)
  with ⟨dip ∈ kD (rt (σ i))⟩ show ?thesis by simp
  qed

lemma oreachable_fresh_ozero_seq_unk_hops_one:
  fixes dip
  assumes "(σ, p) ∈ oreachable (opaodv i)
    (otherwith (op=) (i) (orecvmsg (λσ m. msg_fresh σ m
     ∧ msg_zhops m)))
    (other quality Increases {i})"
  and "dip ∈ kD (rt (σ i))"
  shows "sqn (rt (σ i)) dip = 0 ←→
    sqnf (rt (σ i)) dip = unk
    ∧ the (dhops (rt (σ i)) dip) = 1
    ∧ the (nhop (rt (σ i)) dip) = dip"
  (is "?P dip")
  proof -
    have "∃l. l ∈ labels Γ AODV p" by (metis aodv_ex_label)
    with assms(1)
    have "∀dip ∈ kD (rt (σ i)). ?P dip" by -
      (drule oinvariant_weakenD [OF ozero_seq_unk_hops_one [OF oaodv_trans aodv_trans]],
       auto dest!: onlD otherwith_actionD simp: seqlsimp)
    with ⟨dip ∈ kD (rt (σ i))⟩ show ?thesis by simp
    qed

lemma seq_nhop_quality_increases':
  shows "opaodv i |= (otherwith (op=) {i}
    (orecvmsg (λσ m. msg_fresh σ m
     ∧ msg_zhops m)),
    other quality Increases {i} →)
    onl Γ AODV (λ(σ, _). ∀dip. let nhip = the (nhop (rt (σ i)) dip)
     in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip))
     ∧ nhip ≠ dip
     → (rt (σ i)) ⊑ dip (rt (σ nhip)))"
(is "_ |= (?S i, _ → ) _")
  proof -
    have weaken:
      "∀p I Q R P. p |= (otherwith quality Increases I (orecvmsg Q), other quality Increases I →) P
       → p |= (otherwith (op=) I (orecvmsg (λσ m. Q σ m ∧ R σ m)), other quality Increases I →) P"
      by auto
    \{ fix i a and σ σ' :: "ip ⇒ state"
      assume ai: "∀dip. dip ∈ vD (rt (σ i))
       ∧ dip ∈ vD (rt (σ (the (nhop (rt (σ i)) dip))))
       ∧ (the (nhop (rt (σ i)) dip)) ≠ dip
       → rt (σ i) ⊑ dip rt (σ (the (nhop (rt (σ i)) dip)))"
      and ov: "?S i σ σ' a"
      have "∀dip. dip ∈ vD (rt (σ i))
       ∧ dip ∈ vD (rt (σ' (the (nhop (rt (σ i)) dip))))
       ∧ (the (nhop (rt (σ i)) dip)) ≠ dip
       \}
proof clarify
fix dip
assume a2: "dip ∈ vD (rt (σ i))"
and a3: "dip ∈ vD (rt (σ’ (the (nhop (rt (σ i)) dip))))"
and a4: "(the (nhop (rt (σ i)) dip)) ≠ dip"
from \(\forall j. j \neq i \longrightarrow σ j = σ’ j\) by auto
show "rt (σ i) ⊏ dip rt (σ’ (the (nhop (rt (σ i)) dip)))"
proof (cases "(the (nhop (rt (σ i)) dip)) = i")
assume "(the (nhop (rt (σ i)) dip)) = i"
with \(\exists \sigma i. \sigma i \neq i\) have "dip ∈ vD (rt (σ (the (nhop (rt (σ i)) dip))))" by simp
show "dip ∈ vD (rt (σ’ (the (nhop (rt (σ i)) dip))))" by simp
with \(\exists \sigma i. \sigma i \neq i\) have "rt (σ i) ⊏ dip rt (σ (the (nhop (rt (σ i)) dip)))" by simp
hence False by simp
thus \(\exists \sigma i. \sigma i \neq i\)
next
assume "(the (nhop (rt (σ i)) dip)) ≠ i"
with \(\exists \sigma j. j \neq i \longrightarrow σ j = σ’ j\) have "dip ∈ vD (rt (σ (the (nhop (rt (σ i)) dip))))" by simp
show "dip ∈ vD (rt (σ’ (the (nhop (rt (σ i)) dip))))" by simp
with \(\exists \sigma j. j \neq i \longrightarrow σ j = σ’ j\) have "rt (σ i) ⊏ dip rt (σ (the (nhop (rt (σ i)) dip)))" by simp
with * show \(\exists \sigma i. \sigma i \neq i\) by simp
qed
qed

{ fix \(σ σ’\) a dip sip i
assume a1: "∀ dip. dip ∈ vD (rt (σ i))
\land dip ∈ vD (rt (σ (the (nhop (rt (σ i)) dip))))
\land (the (nhop (rt (σ i)) dip)) ≠ dip
\longrightarrow rt (σ i) ⊏ dip rt (σ (the (nhop (rt (σ i)) dip))))"
and ov: "∀ i. \(σ i \neq σ’ i\)" have "∀ dip. dip ∈ vD (update (rt (σ i)) sip (0, unk, val, Suc 0, sip))
\land dip ∈ vD (rt (σ’ (the (nhop (update (rt (σ i)) sip (0, unk, val, Suc 0, sip))) dip))))
\land (the (nhop (update (rt (σ i)) sip (0, unk, val, Suc 0, sip)) dip)) ≠ dip
\longrightarrow update (rt (σ i)) sip (0, unk, val, Suc 0, sip)
\subset dip rt (σ’ (the (nhop (update (rt (σ i)) sip (0, unk, val, Suc 0, sip))) dip)))"
proof clarify
fix dip
assume a2: "dip ∈ vD (update (rt (σ i)) sip (0, unk, val, Suc 0, sip))"
and a3: "dip ∈ vD (rt (σ’ (the (nhop (update (rt (σ i)) sip (0, unk, val, Suc 0, sip)) dip))))"
and a4: "the (nhop (update (rt (σ i)) sip (0, unk, val, Suc 0, sip)) dip)) ≠ dip"
show "update (rt (σ i)) sip (0, unk, val, Suc 0, sip)
\subset dip rt (σ’ (the (nhop (update (rt (σ i)) sip (0, unk, val, Suc 0, sip)) dip)))"
proof (cases "dip = sip")
assume "dip = sip"
with \(\exists \sigma i. \sigma i \neq \sigma’ i\) have False by simp
thus \(\exists \sigma i. \sigma i \neq \sigma’ i\)
next
assume [simp]: "dip ≠ sip"
from a2 have "dip ∈ vD (rt (σ i)) \lor dip = sip"
by (rule vD_update_val)
with \(\exists \sigma i. \sigma i \neq \sigma’ i\) have "dip ∈ vD (rt (σ i))" by simp
moreover from a3 have "dip ∈ vD (rt (σ’ (the (nhop (rt (σ i)) dip))))" by simp
moreover from a4 have "the (nhop (rt (σ i)) dip) ≠ dip" by simp
ultimately have "rt (σ i) ⊏ dip rt (σ’ (the (nhop (rt (σ i)) dip)))"
using a1 ov by - (drule(1) basic, simp)
with \(\exists \sigma i. \sigma i \neq \sigma’ i\) show \(\exists \sigma i. \sigma i \neq \sigma’ i\)
by - (erule rt_strictly_fresher_update_other, simp)
qed
qed
\{ fix σ a σ' nhop \\
\quad assume pre: "∀ dip. dip ∈ vD (rt (σ i)) ∧ dip ∈ vD (rt (σ (nhop dip))) ∧ nhop dip ≠ dip \\
\qquad → rt (σ i) ⊑ dip rt (σ (nhop dip))" \\
\quad and ow: "?S i σ σ' a" \\
\quad have "∀ dip. dip ∈ vD (invalidate (rt (σ i)) (dests (σ i))) \\
\qquad ∧ dip ∈ vD (rt (σ' (nhop dip))) ∧ nhop dip ≠ dip \\
\qquad → rt (σ i) ⊑ dip rt (σ' (nhop dip))" \\
\quad proof clarify \\
\quad fix dip \\
\quad assume "dip ∈ vD (invalidate (rt (σ i)) (dests (σ i)))" \\
\qquad and "dip ∈ vD (rt (σ' (nhop dip)))" \\
\qquad and "nhop dip ≠ dip" \\
\quad from this have "dip ∈ vD (rt (σ i))" \\
\quad by (clarsimp dest!: vD_invalidate_vD_not_dests) \\
\quad moreover from ow have "∀ j. j ≠ i → σ j = σ' j" by auto \\
\quad ultimately have "rt (σ i) ⊑ dip rt (σ (nhop dip))" \\
\quad using pre "dip ∈ vD (rt (σ' (nhop dip))) ∧ nhop dip ≠ dip" \\
\quad by metis \\
\quad with "∀ j. j ≠ i → σ j = σ' j" show "rt (σ i) ⊑ dip rt (σ' (nhop dip))" \\
\quad by (metis rt_strictly_fresher_irefl) \\
\quad qed \\
\} note invalidate = this \\
\{ fix σ a σ' dip oip osn sip hops i \\
\quad assume pre: "∀ dip. dip ∈ vD (rt (σ i)) \\
\qquad ∧ dip ∈ vD (rt (σ (the (nhop (rt (σ i)) dip)))) \\
\qquad ∧ the (nhop (rt (σ i)) dip) ≠ dip \\
\qquad → rt (σ i) ⊑ dip rt (σ (the (nhop (rt (σ i)) dip)))" \\
\quad and ow: "?S i σ σ' a" \\
\quad and "Suc 0 ≤ osn" \\
\quad and a6: "sip ≠ oip → oip ∈ kD (rt (σ sip)) \\
\qquad ∧ osn ≤ nsqn (rt (σ sip)) oip \\
\qquad ∧ (nsqn (rt (σ sip)) oip = osn \\
\qquad → the (dhops (rt (σ sip))) oip ≤ hops \\
\qquad ∨ the (flag (rt (σ sip))) oip = inv)" \\
\quad and after: "σ' i = σ i(\rt := update (rt (σ i)) oip (osn, kno, val, Suc hops, sip))" \\
\quad have "∀ dip. dip ∈ vD (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip)) \\
\qquad ∧ dip ∈ vD (rt (σ' (the (nhop (update (rt (σ i)) oip \\
\qquad (osn, kno, val, Suc hops, sip))) dip)))) \\
\qquad ∧ the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip))) dip) ≠ dip \\
\qquad → update (rt (σ i)) oip (osn, kno, val, Suc hops, sip) \\
\qquad ⊑ dip \\
\qquad rt (σ' (the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip))) dip)))" \\
\quad proof clarify \\
\quad fix dip \\
\quad assume a2: "dip ∈ vD (update (rt (σ i)) oip (osn, kno, val, Suc (hops), sip))" \\
\quad and a3: "dip ∈ vD (rt (σ' (the (nhop (update (rt (σ i)) oip \\
\qquad (osn, kno, val, Suc hops, sip))) dip))))" \\
\quad and a4: "the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip))) dip) ≠ dip" \\
\quad from ow have a5: "∀ j. j ≠ i → σ j = σ' j" by auto \\
\quad show "update (rt (σ i)) oip (osn, kno, val, Suc hops, sip) \\
\qquad ⊑ dip \\
\qquad rt (σ' (the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip))) dip)))" \\
\quad (is "?rt1 ⊑ dip ?rt2 dip") \\
\quad proof (cases "?rt1 = rt (σ i)") \\
\quad assume nochange [simp]: \\
\qquad "update (rt (σ i)) oip (osn, kno, val, Suc hops, sip) = rt (σ i)" \\
\quad from after have "σ' i = σ i" by simp \\
\quad with a5 have "∀ j. σ j = σ' j" by metis \\
\quad from a2 have "dip ∈ vD (rt (σ i))" by simp
moreover from \(a3\) have "\(\text{dip} \in \text{vD}(\text{rt} (\sigma (\text{the} (\text{nhop} (\text{rt} (\sigma i)) \text{dip}))))\)"
  using nochange and \(\forall j. \sigma j = \sigma' j\) by clarsimp
moreover from \(a4\) have "\(\text{the} (\text{nhop} (\text{rt} (\sigma i)) \text{dip}) \neq \text{dip}\)" by simp
ultimately have "\(\text{rt} (\sigma i) \sqsubseteq_{\text{dip}} \text{rt} (\sigma (\text{the} (\text{nhop} (\text{rt} (\sigma i)) \text{dip}))))\)"
  using pre by simp

hence "\(\text{rt} (\sigma i) \sqsubseteq_{\text{dip}} \text{rt} (\sigma' (\text{the} (\text{nhop} (\text{rt} (\sigma i)) \text{dip}))))\)"
  using \(\forall j. \sigma j = \sigma' j\) by simp
thus "\(?\text{thesis}\)" by simp
next
assume change: "\(?\text{rt}1 \neq \text{rt} (\sigma i)\)"
from after \(a2\) have "\(\text{dip} \in \text{kD}(\text{rt} (\sigma' i))\)" by auto
show "\(?\text{thesis}\)"
proof (cases "\(\text{dip} = \text{oip}\)"
  assume "\(\text{dip} \neq \text{oip}\)"
  with \(a2\) have "\(\text{dip} \in \text{vD} (\text{rt} (\sigma i))\)" by auto
  moreover with \(a3 \ a5\) after and \(\text{dip} \neq \text{oip}\)
    have "\(\text{dip} \in \text{vD}(\text{rt} (\sigma (\text{the} (\text{nhop} (\text{rt} (\sigma i)) \text{dip}))))\)"
    by simp metis
  moreover from \(a4\) and \(\text{dip} \neq \text{oip}\) have "\(\text{the} (\text{nhop} (\text{rt} (\sigma i)) \text{dip}) \neq \text{dip}\)" by simp
ultimately have "\(\text{rt} (\sigma i) \sqsubseteq_{\text{dip}} \text{rt} (\sigma (\text{the} (\text{nhop} (\text{rt} (\sigma i)) \text{dip}))))\)"
  using pre by simp

with after and \(a5\) and \(\text{dip} \neq \text{oip}\) show "\(?\text{thesis}\)"
  by simp (metis rt_strictly_fresher_update_other rt_strictly_fresher_irefl)
next
assume "\(\text{dip} = \text{oip}\)"

with \(a4\) and change have "\(\text{sip} \neq \text{oip}\)" by simp
with \(a6\) have "\(\text{oip} \in \text{kD}(\text{rt} (\sigma \text{sip})))\"
  and "\(\text{osn} \leq \text{nsqn} (\text{rt} (\sigma \text{sip})) \text{oip}\)" by auto

from \(a3\) change \(\text{dip} = \text{oip}\) have "\(\text{oip} \in \text{vD}(\text{rt} (\sigma' \text{sip})))\" by simp
hence "\(\text{the} (\text{flag} (\text{rt} (\sigma' \text{sip})) \text{oip}) = \text{val}\)" by simp

from \(\text{oip} \in \text{kD}(\text{rt} (\sigma \text{sip})))\)
have "\(\text{osn} < \text{nsqn} (\text{rt} (\sigma' \text{sip})) \text{oip} \lor (\text{osn} = \text{nsqn} (\text{rt} (\sigma' \text{sip})) \text{oip})\)
\(\land \text{the} (\text{dhops} (\text{rt} (\sigma' \text{sip})) \text{oip}) \leq \text{hops}\)"
proof
  assume "\(\text{oip} \in \text{vD}(\text{rt} (\sigma \text{sip})))\"
  hence "\(\text{the} (\text{flag} (\text{rt} (\sigma \text{sip})) \text{oip}) = \text{val}\)" by simp
  with \(a6 \ \text{\langle sip} \neq \text{oip}\) have "\(\text{nsqn} (\text{rt} (\sigma \text{sip})) \text{oip} = \text{osn} \rightarrow \text{the} (\text{dhops} (\text{rt} (\sigma \text{sip})) \text{oip}) \leq \text{hops}\)"
    by simp
show "\(?\text{thesis}\)"
proof (cases "\(\text{sip} = \text{i}\)"
  assume "\(\text{sip} \neq \text{i}\)"
  with \(a5\) have "\(\sigma \text{sip} = \sigma' \text{sip}\)" by simp
  with \(\text{osn} \leq \text{nsqn} (\text{rt} (\sigma \text{sip})) \text{oip}\)
  and \(\text{\langle nsqn} (\text{rt} (\sigma \text{sip})) \text{oip} = \text{osn} \rightarrow \text{the} (\text{dhops} (\text{rt} (\sigma \text{sip})) \text{oip}) \leq \text{hops}\)"
  show "\(?\text{thesis}\)" by auto
next
— alternative to using sip_not_ip
assume [simp]: "\(\text{sip} = \text{i}\)"
have "\(?\text{rt}1 = \text{rt} (\sigma i)\)"
proof (rule update_cases_kD, simp_all)
  from \(\text{Suc} 0 \leq \text{osn}\) show "\(0 < \text{osn}\)" by simp
next
  from \(\text{oip} \in \text{kD}(\text{rt} (\sigma \text{sip})))\) and \(\text{\langle sip} = \text{i}\) show "\(\text{oip} \in \text{kD}(\text{rt} (\sigma \text{i}))\)"
    by simp
next
  assume "\(\text{sqn} (\text{rt} (\sigma \text{i})) \text{oip} < \text{osn}\)"
also from \( \text{osn} \leq \text{nsqn} (\sigma (\text{sip})) \text{ oip} \)
  have "... \leq \text{nsqn} (\sigma (\text{i})) \text{ oip}" by simp
also have "... \leq \text{sqn} (\sigma (\text{i})) \text{ oip}"
  by (rule nsqn_sqn)
finally have "\text{sqn} (\sigma (\text{i})) \text{ oip} < \text{sqn} (\sigma (\text{i})) \text{ oip}" .
hence False by simp
thus "(\lambda a. if a = \text{oip} then Some (\text{osn}, \kno, \val, \text{Suc hops}, \text{i}) else \text{rt} (\sigma \text{i}) a) = \text{rt} (\sigma \text{i})" ..

next
assume "\text{sqn} (\sigma (\text{i})) \text{ oip} = \text{osn}"
  and "\text{Suc hops} < \text{the} (\text{dhops} (\sigma (\text{i})) \text{ oip})" by simp
with \( \text{nsqn} (\sigma (\text{i})) \text{ oip} = \text{osn} \rightarrow \text{the} (\text{dhops} (\sigma (\text{i})) \text{ oip}) \leq \text{hops} \)
  have "the (\text{dhops} (\sigma (\text{i})) \text{ oip}) \leq \text{hops}" by simp
with \( \text{Suc hops} < \text{the} (\text{dhops} (\sigma (\text{i})) \text{ oip})\) have False by simp
thus "(\lambda a. if a = \text{oip} then Some (\text{osn}, \kno, \val, \text{Suc hops}, \text{i}) else \text{rt} (\sigma \text{i}) a) = \text{rt} (\sigma \text{i})" ..

next
assume "the (\text{flag} (\sigma (\text{i})) \text{ oip}) = \text{inv}"
  with \( \text{the} (\text{flag} (\sigma (\text{i})) \text{ oip}) = \text{val} \) have False by simp
thus "(\lambda a. if a = \text{oip} then Some (\text{osn}, \kno, \val, \text{Suc hops}, \text{i}) else \text{rt} (\sigma \text{i}) a) = \text{rt} (\sigma \text{i})" ..

next
from \( \text{oip} \in \text{kd}(\sigma (\text{i})) \)
  show "(\lambda a. if a = \text{oip} then Some (\text{osn}, \kno, \val, \text{Suc hops}, \text{i}) else \text{rt} (\sigma \text{i}) a) = \text{rt} (\sigma \text{i})" by (auto dest!: kd_Some)
qed with change have False ..
thus \(?\text{thesis}\) ..
qed

next
assume "\text{oip} \in \text{id}(\sigma (\text{i}))"
  with \( \text{the} (\text{flag} (\sigma (\text{ip})) \text{ oip}) = \text{val} \) and \( a5 \) have "\text{sip} = \text{i}" by (metis f.distinct(1) id_flag_is_inv)
from \( \text{oip} \in \text{id}(\sigma (\text{i})) \)
  have "the (\text{flag} (\sigma (\text{i})) \text{ oip}) = \text{inv}" by auto
with \( \text{sip} = \text{i} : \text{Suc 0} \leq \text{osn} \) change after \( \text{oip} \in \text{kd}(\sigma (\text{sip})) \)
  have "\text{nsqn} (\sigma (\text{sip})) \text{ oip} < \text{nsqn} (\sigma (\text{ip})) \text{ oip}" unfolding update_def
    by (clarsimp split: option.split_asm split_if_asm)
    (auto simp: sqn_def)
with \( \text{osn} \leq \text{nsqn} (\sigma (\text{sip})) \text{ oip} \) have "\text{osn} < \text{nsqn} (\sigma (\text{ip})) \text{ oip}" by simp
thus \(?\text{thesis}\) ..
qed thus \(?\text{thesis}\) proof
assumoesnltn: "\text{osn} < \text{nsqn} (\sigma (\text{ip})) \text{ oip}"
from \( \text{dip} \in \text{kd}(\sigma (\text{i})) \)
  and \( \text{dip} = \text{oip} \) have "\text{dip} \in \text{kd} (?\text{rt1})" by simp
moreover from \( a3 \) have "\text{dip} \in \text{kd}(\text{rt2 dip})" by simp
moreover have "\text{nsqn} ?\text{rt1 dip} < \text{nsqn} (?\text{rt2 dip}) \text{ dip}"
  proof -
    have "\text{nsqn} ?\text{rt1 oip} = \text{osn}"
      by (simp add: \text{dip} = \text{oip} nsqn_update_changed_kno_val [OF change [THEN not_sym]])
    also have "... < \text{nsqn} (\sigma (\text{ip})) \text{ oip}" using osnlt .
    also have "... = \text{nsqn} (?\text{rt2 oip}) \text{ oip}" by (simp add: change)
    finally show \(?\text{thesis}\)
      using \( \text{dip} = \text{oip} \) by simp
    qed
ultimately show \(?\text{thesis}\)
  by (rule rt_strictly_fresher_ltI)
next
assume osneq: "osn = nsqn (rt (σ' sip)) oip ∧ the (dhops (rt (σ' sip)) oip) ≤ hops"

have "oip ∈ kD(?rt1)" by simp
moreover from a3 ⟨dip = oip⟩ have "oip ∈ kD(?rt2 oip)" by simp

moreover have "nsqn ?rt1 oip = nsqn (?rt2 oip)"
proof -
from osneq have "osn = nsqn (rt (σ' sip)) oip" ..
also have "osn = nsqn ?rt1 oip"
  by (simp add: ⟨dip = oip⟩ nsqn_update_changed_kno_val [OF change [THEN not_sym]])
also have "nsqn (rt (σ' sip)) oip = nsqn (?rt2 oip) oip"
  by (simp add: change)
finally show ?thesis .
qed

moreover have "π5((the (?rt2 oip)) < π5((the (?rt1 oip))"
proof -
from osneq have "the (dhops (rt (σ' sip)) oip) ≤ hops" ..
moreover from ⟨oip ∈ vD (rt (σ' sip))⟩ have "oip ∈ kD(rt (σ' sip))" by auto
ultimately have "π5((the (rt (σ' sip)) oip)) ≤ hops"
  by (auto simp add: proj5_eq_dhops)
also from change after have "hops < π5((the (rt (σ' i)) oip))"
  by (auto simp add: proj5_eq_dhops) (metis dhops_update_changed_lessI)
finally have "π5((the (rt (σ' sip)) oip)) < π5((the (rt (σ' i)) oip))" .
with change after show ?thesis by simp
qed

ultimately have "?rt1 ⊑ oip ?rt2 oip"
by (rule rt_strictly_fresher_eqI)
with ⟨dip = oip⟩ show ?thesis by simp
qed

} note rreq_rrep_update = this

have "opaodv i | (otherwith (op=) {i}) (orecvmsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
onl Γ_{AODV} onl
(λ(σ, _). ∀dip. dip ∈ vD (rt (σ i)) ∩ vD (rt (σ (the (nhop (rt (σ i)) dip))))
∧ the (nhop (rt (σ i)) dip) ≠ dip
→ rt (σ i) ⊑ dip rt (σ (the (nhop (rt (σ i)) dip))))"
proof (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf rreq_sip [THEN weaken]]
onl_invariant_sterms [OF aodv_wf rrep_sip [THEN weaken]]
onl_invariant_sterms [OF aodv_wf rerr_sip [THEN weaken]]
onl_invariant_sterms [OF aodv_wf oosn_rreq [THEN weaken]]
onl_invariant_sterms [OF aodv_wf odsn_rrep [THEN weaken]]
solve: basic update_0_unk invalidate rreq_rrep_update
simp add: seqlsimp)

fix σ σ' p l
assume or: "((σ, p) ∈ oreachable (opaodv i) (?S i) (other quality_increases {i}))"
  and "other quality_increases {i} σ σ'"
and ll: "l ∈ labels Γ_{AODV} p"
and pre: "∀dip. dip ∈ vD (rt (σ i))
∧ dip ∈ vD (rt (σ (the (nhop (rt (σ i)) dip))))
∧ the (nhop (rt (σ i)) dip) ≠ dip
→ rt (σ i) ⊑ dip rt (σ (the (nhop (rt (σ i)) dip))))"
from this(1-2)
  have or': "(σ', p) ∈ oreachable (opaodv i) (?S i) (other quality_increases {i})"
    by (rule oreachable_other')
from or and ll have next_hop: "∀dip. let nhip = the (nhop (rt (σ i)) dip)"
\[
\begin{align*}
\text{in dip} & \in kD(rt (\sigma i)) \land \text{nhip} \neq \text{dip} \\
\rightarrow & \text{dip} \in kD(rt (\sigma \text{nhip})) \\
\land & \text{nsqn (rt (\sigma i)) dip} \leq \text{nsqn (rt (\sigma \text{nhip})) dip}
\end{align*}
\]

by (auto dest!: onl_oInvariant_weakenD [OF seq_compare_next_hop'])

from or and 11 have unk_hops_one: "\(\forall \text{dip} \in kD (rt (\sigma i)). \text{sqn (rt (\sigma i)) dip} = 0 \rightarrow \text{sqnf (rt (\sigma i)) dip} = \text{unk} \\
\land \text{the (dhops (rt (\sigma i)) dip)} = 1 \\
\land \text{the (nhop (rt (\sigma i)) dip)} = \text{dip}\)"

by (auto dest!: onl_oInvariant_weakenD [OF ozero_seq_unk_hops_one [OF oaodv_trans aodv_trans]]

otherwith_actionD
simp: seqlsimp)

from \{other quality Increases \{i\} \sigma \sigma'\} have "\(\sigma' \equiv \sigma i\)" by auto

hence \{other quality Increases \{i\} \sigma \sigma'\} have "\(\forall j. \text{quality Increases (}\sigma j) (\sigma' j)\)"

by - (erule otherE, metis singleton_iff)

show "\(\forall \text{dip}. \text{dip} \in vD (rt (\sigma i)) \\
\land \text{dip} \in vD (rt (\sigma' (\text{the (nhop (rt (\sigma i)) dip))))) \\
\land \text{the (nhop (rt (\sigma i)) dip)} \neq \text{dip} \\
\rightarrow rt (\sigma i) \sqsubseteq dip rt (\sigma' (\text{the (nhop (rt (\sigma i)) dip))))\)

proof clarify

fix dip

assume "\(\text{dip} \in vD(rt (\sigma i))\)"

and "\(\text{dip} \in vD(rt (\sigma' (\text{the (nhop (rt (\sigma i)) dip))))\)"

and "\(\text{the (nhop (rt (\sigma i)) dip)} \neq \text{dip}\)"

from this(1) and (\(\sigma' i = \sigma i\)) have "\(\text{dip} \in vD(rt (\sigma i))\)"

and "\(\text{dip} \in kD(rt (\sigma i))\)"

by auto

from \(\text{dip} \in kD(rt (\sigma i)) \neq \text{dip}\) and (\(\sigma' i = \sigma i\)) have "\(\text{the (nhop (rt (\sigma i)) dip)} \neq \text{dip}\)" (is "\(\text{nhip} \neq \_\)"

by simp

with \(\text{dip} \in kD(rt (\sigma \text{nhip}))\) and next_hop

have "\(\text{dip} \in kD(rt (\sigma ?\text{nhip}))\)"

and nsqns: "\(\text{nsqn (rt (\sigma i)) dip} \leq \text{nsqn (rt (\sigma ?\text{nhip})) dip}\)"

by (auto simp: Let_def)

have "0 < \text{sqn (rt (\sigma i)) dip}"

proof (rule neq0_conv [THEN iffD1, OF notI])

assume "\(\text{sqn (rt (\sigma i)) dip} = 0\)"

with \(\text{dip} \in kD(rt (\sigma i))\) and unk_hops_one

have "\(\text{nhip} = \text{dip}\)" by simp

with \(\text{nhip} \neq \text{dip}\) show False ..

qed

also have "\(\ldots = \text{nsqn (rt (\sigma i)) dip}\)"

by (rule vD_nsqn_sqn [OF \(\text{dip} \in vD(rt (\sigma i))\), THEN sym])

also have "\(\ldots \leq \text{nsqn (rt (\sigma ?\text{nhip})) dip}\)"

by (rule nsqns)

also have "\(\ldots \leq \text{sqn (rt (\sigma ?\text{nhip})) dip}\)"

by (rule nsqn_sqn)

finally have "0 < \text{sqn (rt (\sigma ?\text{nhip})) dip}\)".

have "\(\text{rt (\sigma i) \sqsubseteq dip rt (\sigma ?\text{nhip})}\)"

proof (cases "\(\text{dip} \in vD(rt (\sigma ?\text{nhip}))\)"

assume "\(\text{dip} \in vD(rt (\sigma ?\text{nhip}))\)"

with pre \(\text{dip} \in vD(rt (\sigma i))\) and \(\text{nhip} \neq \text{dip}\)

have "\(\text{rt (\sigma i) \sqsubseteq dip rt (\sigma ?\text{nhip})}\)" by auto

moreover from \(\forall j. \text{quality Increases (}\sigma j) (\sigma' j)\) have "\(\text{quality Increases (\sigma ?\text{nhip}) (\sigma' ?\text{nhip})}\)" ..

ultimately show \(\text{thesis}\)

using \(\text{dip} \in kD(rt (\sigma ?\text{nhip}))\)

by (rule strictly_fresher_quality_increases_right)
next 
assume "dip ∉ vD (rt (σ ?nhip))"
with ⟨dip ∈ kD (rt (σ ?nhip))⟩ have "dip ∈ iD (rt (σ ?nhip))" ..
hence "the (flag (rt (σ ?nhip)) dip) = inv"
by auto
have "nsqn (rt (σ i)) dip ≤ nsqn (rt (σ ?nhip)) dip"
by (rule nsqns)
also from ⟨dip ∈ kD (rt (σ ?nhip))⟩ have "... = sqn (rt (σ ?nhip)) dip" ..
also have "... < sqn (rt (σ' ?nhip)) dip" proof from ⟨∀ j. quality_increases (σ j) (σ' j)⟩ have "quality_increases (σ ?nhip) (σ' ?nhip)" ..
hence "∀ ip. sqn (rt (σ ?nhip)) ip ≤ sqn (rt (σ' ?nhip)) ip" by auto
hence "sqn (rt (σ ?nhip)) dip ≤ sqn (rt (σ' ?nhip)) dip" ..
with ⟨0 < sqn (rt (σ ?nhip)) dip⟩ show ?thesis by auto
qed
finally have "nsqn (rt (σ i)) dip < nsqn (rt (σ' ?nhip)) dip".
moreover from ⟨dip ∈ kD (rt (σ' (the (nhop (rt (σ' i)) dip))))⟩ and ⟨σ' i = σ i⟩
ultimately show "rt (σ i) ⊏ dip rt (σ' ?nhip)"
using ⟨dip ∈ kD (rt (σ i))⟩ by - (rule rt_strictly_fresher_ltI)
with ⟨σ' i = σ i⟩ show "rt (σ i) ⊏ dip rt (σ' (the (nhop (rt (σ' i)) dip)))"
by simp
qed

thus ?thesis unfolding Let_def .
qed

lemma seq_compare_next_hop:
  fixes w
  shows "opaodv i |= (otherwith (op=) {i} (orecmsg msg_fresh),
  other quality_increases {i} →)
  global (λσ. ∀ dip. let nhip = the (nhop (rt (σ i)) dip)
  in dip ∈ kD (rt (σ i)) ∧ nhip ≠ dip →
  dip ∈ kD (rt (σ nhip))
  ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ nhip)) dip)"
by (rule oinvariant_weakenE [OF seq_compare_next_hop']) (auto dest!: onlD)

lemma seq_nhop_quality_increases:
  shows "opaodv i |= (otherwith (op=) {i} (orecmsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
  other quality_increases {i} →)
  global (λσ. ∀ dip. let nhip = the (nhop (rt (σ i)) dip)
  in dip ∈ vD (rt (σ i)) ⊆ vD (rt (σ nhip)) ∧ nhip ≠ dip
  → (rt (σ i)) ⊑ dip (rt (σ nhip)))"
by (rule oinvariant_weakenE [OF seq_nhop_quality_increases']) (auto dest!: onlD)

end

3.10 Routing graphs and loop freedom

theory C_Loop_Freedom
imports C_Aodv_Predicates C_Fresher
begin
Define the central theorem that relates an invariant over network states to the absence of loops in the associate
routing graph.

**definition**

\[
rt_graph :: (ip ⇒ state) ⇒ ip ⇒ ip rel
\]

where

\[
rt_graph σ = (λdip. { (ip, ip') | ip ip' dsn dsk hops.
    \text{ip} \neq \text{dip} \land \text{rt} (σ \text{ip}) \text{dip} = \text{Some (dsn, dsk, val, hops, ip')}})
\]

Given the state of a network \(σ\), a routing graph for a given destination \(dip\) abstracts the details of routing tables into nodes (ip addresses) and vertices (valid routes between ip addresses).

**lemma** \(rt_graphE \{\text{elim}\}^\) fixes \(n\ dip\ ip\ ip'\)

assumes \(\langle (ip, ip') \in rt_graph σ dip \rangle\)

shows \(\text{ip} \neq \text{dip} \land (\exists r. \text{rt} (σ \text{ip}) = r \land (\exists dsn dsk hops. r dip = \text{Some (dsn, dsk, val, hops, ip')}))\)

using \text{assms unfolding } rt_graph_def by \text{auto}

**lemma** \(rt_graph_vD \{\text{dest}\}^\)

\(\bigwedge ip ip' σ dip. (ip, ip') \in rt_graph σ dip \implies dip ∈ vD(rt (σ ip))\)

unfolding \text{rt_graph_def vD_def by auto}

**lemma** \(rt_graph_vD_trans \{\text{dest}\}^\)

\(\bigwedge ip ip' σ dip. (ip, ip') ∈ (rt_graph σ dip) + \implies dip ∈ vD(rt (σ ip))\)

by \(\text{erule converse_tranclE auto}\)

**lemma** \(rt_graph_not_dip \{\text{dest}\}^\)

\(\bigwedge ip ip' σ dip. (ip, ip') \in rt_graph σ dip \implies \text{ip} \neq \text{dip}\)

unfolding \text{rt_graph_def} by \text{auto}

**lemma** \(rt_graph_not_dip_trans \{\text{dest}\}^\)

\(\bigwedge ip ip' σ dip. (ip, ip') \in (rt_graph σ dip) + \implies \text{ip} \neq \text{dip}\)

by \(\text{erule converse_tranclE auto}\)

NB: the property below cannot be lifted to the transitive closure

**lemma** \(rt_graph_nhip_is_nhop \{\text{dest}\}^\)

\(\bigwedge ip ip' σ dip. (ip, ip') \in rt_graph σ dip \implies ip' = \text{the (nhop (rt (σ ip)) dip)}\)

unfolding \text{rt_graph_def} by \text{auto}

**theorem** \(\text{inv_to_loop_freedom}\)

assumes \(\forall i dip. \text{let nhip = the (nhop (rt (σ i)) dip) in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) \land nhip \neq dip \implies (rt (σ i)) \cap dip (rt (σ nhip))}\)

shows \(\forall dip. \text{irrefl ((rt_graph σ dip) +)}\)

using \text{assms proof (intro allI)}

fix \(σ :: \text{"ip ⇒ state" and dip}\)

assume \(\text{inv} : \"\forall ip dip.
    \text{let nhip = the (nhop (rt (σ ip)) dip)
    in dip ∈ vD (rt (σ ip)) \cap vD(rt (σ nhip)) \land nhip \neq dip \implies (rt (σ ip)) \cap dip (rt (σ nhip))\}"\)

\{ fix \(ip ip'\)

assume \("(ip, ip') \in (rt_graph σ dip) +\)"

and \(" dip ∈ vD(rt (σ ip'))\)"

and \(" ip' \neq dip\)"

hence \(" rt (σ ip) \cap dip (rt (σ ip'))\)"

proof induction

fix \(nhip\)

assume \("(ip, nhip) ∈ rt_graph σ dip\)"

and \(" dip ∈ vD(rt (σ nhip))\)"

and \(" nhip \neq dip\)"

from \("(ip, nhip) ∈ rt_graph σ dip\) have "dip ∈ vD(rt (σ ip))"

and \(" nhip = the (nhop (rt (σ ip)) dip)\)"

by \text{auto}

from \(" dip ∈ vD(rt (σ ip))\) and \(" dip ∈ vD(rt (σ nhip))\)"}
have “dip ∈ vD(rt (σ ip)) ∩ vD(rt (σ nhip))” ..
with nhip = the (nhop (rt (σ ip)) dip)
and nhip ≠ dip
and inv
show “rt (σ ip) ⊑ dip rt (σ nhip)”
by (clarsimp simp: Let_def)
next
fix nhip nhip’
assume "((ip, nhip) ∈ rt_graph σ dip) +
and (nhip, nhip’) ∈ rt_graph σ dip"
and IH: “[\ dip ∈ vD(rt (σ nhip)); nhip ≠ dip ] ⇒ rt (σ ip) ⊑ dip rt (σ nhip)”
and “dip ∈ vD(rt (σ nhip’))”
and "nhip’ ≠ dip”
from ⟨(nhip, nhip’) ∈ rt_graph σ dip⟩ have 1: “dip ∈ vD(rt (σ nhip))”
and 2: “nhip ≠ dip”
and “nhip’ = the (nhop (rt (σ nhip)) dip)”
by auto
from 1 2 have "rt (σ ip) ⊑ dip rt (σ nhip)” by (rule IH)
also have "rt (σ nhip) ⊑ dip rt (σ nhip’)
proof -
from dip ∈ vD(rt (σ nhip)) and dip ∈ vD(rt (σ nhip’))
have "dip ∈ vD(rt (σ nhip)) ∩ vD(rt (σ nhip’))” ..
with nhip’ ≠ dip
and ‘nhip’ = the (nhop (rt (σ nhip)) dip)
and inv
show "rt (σ nhip) ⊑ dip rt (σ nhip’)
by (clarsimp simp: Let_def)
qed
finally show "rt (σ ip) ⊑ dip rt (σ nhip’)
. note fresher = this
qed
}
}
show "irrefl ((rt_graph σ dip) +”
unfolding irrefl_def proof (intro allI notI)
fix ip
assume "((ip, ip) ∈ rt_graph σ dip) +
moreover then have "dip ∈ vD(rt (σ ip))”
and "ip ≠ dip”
by auto
ultimately have "rt (σ ip) ⊑ dip rt (σ ip)” by (rule fresher)
thus False by simp
qed

end

3.11 Lift and transfer invariants to show loop freedom

theory C_Aodv_Loop_Freedom
imports "../../../AWN/OClosed_Transfer" "../../../AWN/Qmsg_Lifting" C_Global_Invariants C_Loop_Freedom
begin

3.11.1 Lift to parallel processes with queues

lemma par_step_no_change_on_send_or_receive:
fixes σ s a σ’ s’
assumes "((σ, s), a, (σ’, s’,)) ∈ oparp_sos i (oseqp_sos Γ_AODV i) (seqp_sos Γ_QMSG)"
and "a ≠ τ”
shows "σ’ i = σ i”
using assms by (rule qmsg_no_change_on_send_or_receive)

lemma par_nhop_quality_increases:
shows "opaodv i (\i qmsg \ otherwith (op=) {i} (orecvmsg (λσ m.
msg_fresh σ m ∧ msg_zhops m)), other quality_increases {i} →)"
lemma par_rreq_rrep_sn_quality_increases:
par_rreq_rrep_snquality_increases:

lemma par_anycast_msg_zhops:

proof (rule lift_into_qmsg [OF seq_nhop_quality_increases])

show "opaodv i \models_A (otherwith (op=) \{i\})
(orecvmsg (\lambda m. msg_fresh \sigma m \land msg_zhops m)),
other quality_increases \{i\} \rightarrow
globala (\lambda(\sigma, _, \sigma'). quality_increases (\sigma i) (\sigma' i))"

proof (rule ostep_invariant_weakenE [OF olocal_quality_increases], simp_all)

fix t :: "(((nat ⇒ state) ⇴ (state, msg, pseqp, pseqp label) seqp), msg seq_action) transition"
assume "onll \Gamma_{AODV} (\lambda(\sigma, _, _, (\sigma', _)). \forall j. quality_increases (\sigma j) (\sigma' j)) t"
thus "quality_increases (fst (fst t) i) (fst (snd (snd t)) i)"

by (cases t) (clarsimp dest!: onllD)

lemma par_rreq_rrep_sn_quality_increases:
"opaodv i \models_A (\lambda\sigma _. orecvmsg (\lambda_. rreq_rrep_sn) \sigma), other (\lambda_. True) \{i\} \rightarrow
globala (\lambda(\sigma, a, \sigma'). anycast (msg_fresh \sigma) a)"

proof -

have "opaodv i \models_A (\lambda\sigma _. (orecvmsg (\lambda_. rreq_rrep_sn)) \sigma), other (\lambda_. True) \{i\} \rightarrow
globala (\lambda(\sigma, a, \sigma'). anycast (msg_fresh \sigma) a)"

by (rule ostep_invariant_weakenE [OF req_rreq_rrep_snquality_increases])
(auto dest!: onllD elim!: aodv_ex_label)

hence "opaodv i \models_A (\lambda\sigma _. orecvmsg (\lambda_. rreq_rrep_sn)) \sigma), other (\lambda_. True) \{i\} \rightarrow
globala (\lambda(\sigma, a, \sigma'). anycast (msg_fresh \sigma) a)"

by (rule lift_step_into_qmsg_statelessassm) simp_all
thus ?thesis by rule auto

qed

lemma par_rreq_rrep_nsqn_fresh_any_step:
shows "opaodv i \models_A (\lambda\sigma _. orecvmsg (\lambda_. rreq_rrep_sn) \sigma),
other (\lambda_. True) \{i\} \rightarrow
globala (\lambda(\sigma, a, \sigma'). anycast (msg_fresh \sigma) a)"

proof -

have "opaodv i \models_A (\lambda\sigma _. (orecvmsg (\lambda_. rreq_rrep_sn)) \sigma), other (\lambda_. True) \{i\} \rightarrow
globala (\lambda(\sigma, a, \sigma'). anycast (msg_fresh \sigma) a)"

proof (rule ostep_invariant_weakenE [OF req_rreq_rrep_nsqn_fresh_any_step_invariant])

fix t
assume "onll \Gamma_{AODV} (\lambda(\sigma, a, _). anycast (msg_fresh \sigma) a) t"
thus "globala (\lambda(\sigma, a, \sigma'). anycast (msg_fresh \sigma) a) t"

by (cases t) (clarsimp dest!: onllD)

qed auto

hence "opaodv i \models_A (\lambda\sigma _. orecvmsg (\lambda_. rreq_rrep_sn)) \sigma), other (\lambda_. True) \{i\} \rightarrow
globala (\lambda(\sigma, a, \sigma'). anycast (msg_fresh \sigma) a)"

by (rule lift_step_into_qmsg_statelessassm) simp_all
thus ?thesis by rule auto

qed

lemma par_anycast_msg_zhops:
shows "opaodv i \models_A (\lambda\sigma _. orecvmsg (\lambda_. rreq_rrep_sn) \sigma), other (\lambda_. True) \{i\} \rightarrow
globala (\lambda(\_, a, \_.) anycast msg_zhops a)"

proof -

from anycast_msg_zhops initiali_aodv oadv_trans aodv_trans
have "opaodv i \models_A (act TT, other (\lambda_. True) \{i\} \rightarrow
seqll i (onll \Gamma_{AODV} (\lambda(\_, a, _). anycast msg_zhops a)))"

by (rule open_seq_step_invariant)

hence "opaodv i \models_A (\lambda\sigma _. orecvmsg (\lambda_. rreq_rrep_sn) \sigma), other (\lambda_. True) \{i\} \rightarrow
globala (\lambda(\_, a, \_.) anycast msg_zhops a)"
\[ globala (\lambda(_, a, _). \text{anycast msg\_zhops a}) \]  

proof (rule ostep\_invariant\_weakenE)  
fix \( t :: ((\text{nat} \Rightarrow \text{state}) \times (\text{state}, \text{msg}, \text{pseqp}, \text{pseqp label}) \text{seqp}), \text{msg seq\_action}) \) transition  
assume "seqll i (onll AODV (\lambda(_, a, _). \text{anycast msg\_zhops a})) t"  
thus "globala (\lambda(_, a, _). \text{anycast msg\_zhops a}) t"  
by (cases t) (clarsimp dest!: seqllD onllD, metis aodv\_ex\_label)  
qed simp\_all  
hence "opaodv i \langle\langle i \text{qmsg} | A \rangle = (\lambda\sigma_. \text{orecvmsg} (\lambda_. \text{rreq\_rrep\_sn}) \sigma, \text{other (\lambda_. True}) \langle i \rangle) \rightarrow  
globala (\lambda(_, a, _). \text{anycast msg\_zhops a})"  
by (rule lift\_step\_into\_qmsg\_statelessassm) simp\_all  
thus ?thesis by rule auto  
qed

### 3.11.2 Lift to nodes

lemma node\_step\_no\_change\_on\_send\_or\_receive:  
assumes "((\sigma, \text{NodeS i P R}), a, (\sigma', \text{NodeS i' P' R'})) \in \text{onode\_sos}  
(oparp\_sos i (oseqp\_sos \Gamma AODV i) (seqp\_sos \Gamma QMSG))"  
and "a \neq \tau"  
sorts "\sigma' i = \sigma i"  
using assms by (cases a) (auto elim!: par\_step\_no\_change\_on\_send\_or\_receive)

lemma node\_nhop\_quality\_increases:  
shows "\langle i : \text{opaodv i} \langle\langle i \text{qmsg} : R \rangle_o \Rightarrow  
\text{otherwith (op=) \{ (\sigma, a, \sigma') \rightarrow \text{globala (\lambda(\sigma, a, \sigma'). quality\_increases (\sigma i) (\sigma' i))} \rangle \}

by (rule node\_lift [OF par\_nhop\_quality\_increases]) auto

lemma node\_rreq\_rrep\_nsqn\_fresh\_any\_step:  
shows "\langle i : \text{opaodv i} \langle\langle i \text{qmsg} : R \rangle_o \Rightarrow  
(\lambda(\sigma, a, \sigma'). \text{castmsg (msg\_fresh (\sigma a))})"  
by (rule node\_lift\_anycast\_statelessassm [OF par\_rreq\_rrep\_nsqn\_fresh\_any\_step])

lemma node\_anycast\_msg\_zhops:  
shows "\langle i : \text{opaodv i} \langle\langle i \text{qmsg} : R \rangle_o \Rightarrow  
globala (\lambda(_, a, _). \text{anycast msg\_zhops a})"  
by (rule node\_lift\_anycast\_statelessassm [OF par\_anycast\_msg\_zhops])

lemma node\_silent\_change\_only:  
shows "\langle i : \text{opaodv i} \langle\langle i \text{qmsg} : R \rangle_o \Rightarrow  
globala (\lambda(\sigma, a, \sigma'). a \neq \tau \rightarrow \sigma' i = \sigma i)"  
proof (rule ostep\_invariantI, simp (no_asm), rule impl)  
fix \( \sigma' \)  
assume or: "\((\sigma, \zeta) \in \text{oreachable} \langle\langle i : \text{opaodv i} \langle\langle i \text{qmsg} : R_i\rangle_o \rangle  
(\lambda(\sigma, a, \sigma'). \text{castmsg (msg\_fresh (\sigma a))})\)\"  
and tr: "\((\sigma, \zeta), a, (\sigma', \zeta') \in \text{trans} \langle\langle i : \text{opaodv i} \langle\langle i \text{qmsg} : R_i\rangle_o \rangle \rangle\)  
and "a \neq \tau'"  
from or obtain p R where "\zeta = \text{NodeS i p R}"  
by (drule node\_net\_state, metis)
with \( \text{tr} \) have 
\[
((\sigma, \text{NodeS } i \ p \ R), a, (\sigma', \zeta')) \\
\in \text{node_sos} (\text{oparp_sos } i (\text{trans } (\text{opaodv } i)) (\text{trans qmsg}))
\]
by simp
thus "\( \sigma' = \sigma \)" using \( a \neq \tau_n \)
by (cases rule: node_sos.cases)
(auto elim: qmsg_no_change_on_send_or_receive)
qed

3.11.3 Lift to partial networks

lemma arrive_rreq_rrep_nsqn_fresh_inc_sn [simp]:
assumes "arrivemsg (\( \lambda \sigma \ m. \ \text{msg_fresh } \sigma \ m \land P \ \sigma \ m \)) \sigma \ m"
shows "arrivemsg (\( \lambda . \ \text{rreq_rrep_sn} \)) \sigma \ m"
using assms by (cases m) auto

lemma opnet_nhop_quality_increases:
shows "opnet (\( \lambda i. \ \text{opaodv } i \langle \langle \ i qmsg : R \rangle \ o \ | \ \rangle \ \text{global} (\sigma). \forall i \in \text{net_tree_ips } p. \forall \text{dip.}
\text{let } \text{nhip } = \text{the} (\text{nhop } (\text{rt } (\sigma i)) \text{ dip})
\text{in } \text{dip } \in \text{vD } (\text{rt } (\sigma i)) \cap \text{vD } (\text{rt } (\sigma \text{ nhip})) \land \text{nhip } \neq \text{dip}
\rightarrow (\text{rt } (\sigma i)) \sqsubseteq_{\text{dip}} (\text{rt } (\sigma \text{ nhip})))"
proof (rule pnet_lift [OF node_nhop_quality_increases])
fix \( i \ R \)
have "\( i : \text{opaodv } i \langle \langle \ i qmsg : R \rangle_0 \models_A (\lambda . \ \text{arrivemsg } (\lambda . \ \text{rreq_rrep_sn}) \sigma, \text{ other } (\lambda . \ \text{True}) \{i\} \rightarrow) \text{ globala } (\lambda (\sigma, a, \sigma').
\text{castmsg } (\lambda . \ \text{msg_fresh } \sigma \ m \land \text{msg_zhops } m) a)"
proof (rule ostep_invariantI, simp (no_asm))
fix \( \sigma \ s a \sigma' s' \)
assume or: "\((\sigma, s) \in \text{oreachable } ((i : \text{opaodv } i \langle \langle \ i qmsg : R \rangle_0) (\lambda . \ \text{arrivemsg } (\lambda . \ \text{rreq_rrep_sn}) \sigma, \text{ other } (\lambda . \ \text{True}) \{i\}))"
and tr: "\(((\sigma, s), a, (\sigma', s')) \in \text{trans } ((i : \text{opaodv } i \langle \langle \ i qmsg : R \rangle_0)"
and am: "\text{castmsg } (\lambda . \ \text{rreq_rrep_sn}) \sigma a"
from or tr am have "\text{castmsg } (\text{msg_fresh } \sigma ) a"
by (auto dest!: ostep_invariantD [OF node_rreq_rrep_nsqn_fresh_any_step])
moreover from or tr am have "\text{castmsg } (\text{msg_zhops } m) a"
by (auto dest!: ostep_invariantD [OF node_anycast_msg_zhops])
ultimately show "\text{castmsg } (\lambda . \ \text{msg_fresh } \sigma \ m \land \text{msg_zhops } m) a"
by (case_tac a) auto
qed
thus "\( i : \text{opaodv } i \langle \langle \ i qmsg : R \rangle_0 \models_A (\lambda . \ \text{arrivemsg } (\lambda . \ \text{rreq_rrep_sn}) \sigma, \text{ other } (\lambda . \ \text{msg_fresh } \sigma \ m \land \text{msg_zhops } m) a)"
by rule auto
next
fix \( i \ R \)
show "\( i : \text{opaodv } i \langle \langle \ i qmsg : R \rangle_0 \models_A (\lambda . \ \text{arrivemsg } (\lambda . \ \text{rreq_rrep_sn}) \sigma, \text{ other } (\lambda . \ \text{msg_fresh } \sigma \ m \land \text{msg_zhops } m) a, \ \text{a } \neq \tau \land (\forall i \ d. \ a \neq i : \text{deliver(d)}) \rightarrow (\sigma i = \sigma' i))"
by (rule ostep_invariant_weakenE [OF node_silent_change_only]) auto
next
fix \( i \ R \)
show "\( i : \text{opaodv } i \langle \langle \ i qmsg : R \rangle_0 \models_A (\lambda . \ \text{arrivemsg } (\lambda . \ \text{rreq_rrep_sn}) \sigma, \text{ other } (\lambda . \ \text{msg_fresh } \sigma \ m \land \text{msg_zhops } m) a = \tau \lor (\exists d. \ a = i : \text{deliver(d)}) \rightarrow \text{quality_increases } (\sigma i) (\sigma' i))"
by (rule ostep_invariant_weakenE [OF node_quality_increases]) auto
qed simp_all
3.11.4 Lift to closed networks

lemma onet_nhop_quality_increases:
shows "oclosed (opnet (λi. opaodv i ⟨⟨ i qmsg) p) /
    (λi. opaodv i (rt (σ i)) dip)
in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
    → (rt (σ i) ⊏ dip (rt (σ nhip))))"
(is " _ |= (?U →) ?inv")
proof (rule inclosed_closed)
  from opnet_nhop_quality_increases
  show "opnet (λi. opaodv i ⟨⟨ i qmsg) p |
      = (otherwith (op=) (net_tree_ips p) inoclosed, ?U →) ?inv"
  proof (rule oinvariant_weakenE)
    fix σ σ ' :: "ip ⇒ state" and a :: "msg node_action"
    assume "otherwith (op=) (net_tree_ips p) inoclosed σ σ ' a"
    thus "otherwith (op=) (net_tree_ips p)
        (oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m)) σ σ ' a"
    proof (rule otherwithEI)
      fix σ :: "ip ⇒ state" and a :: "msg node_action"
      assume "inoclosed σ a"
      thus "oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m) σ a"
      proof (cases a)
        fix ii ni ms
        assume "a = ii ¬ ni:arrive(ms)"
        moreover with ⟨inoclosed σ a⟩ obtain d di
        where "ms = newpkt(d, di)"
        by (cases ms) auto
        ultimately show ?thesis by simp
      qed
    qed
    qed
  qed
  qed

3.11.5 Transfer into the standard model

interpretation aodv_openproc: openproc paodv opaodv id
where "aodv_openproc.initmissing = initmissing"
proof -
  show "openproc paodv opaodv id"
  proof unfold_locales
    fix i :: ip
    have "{(σ, ζ). (σ i, ζ) ∈ σAODV i ∧ (∀ j. j ≠ i → σ j ∈ fst ' σAODV j)} ⊆ σAODV''
    unfolding σAODV_def σAODV'_def
    proof (rule equalityD1)
      show "∀ p. {(σ, ζ). (σ i, ζ) ∈ {(f i, p)} ∧ (∀ j. j ≠ i
          → σ j ∈ fst ' {f j, p})} = {(f, p)}"n
      by (rule set_eqI) auto
    qed
    thus "{(σ, ζ) | σ ζ s. s ∈ init (paodv i)
      ∧ (σ i, ζ) = id s
      ∧ (∀ j. j≠i → σ j ∈ (fst o id) ' init (paodv j))} ⊆ init (opaodv i)"
    by simp
next
  show "∀ j. init (paodv j) ≠ {}"
  unfolding σAODV_def by simp
next
  fix i s a s' σ σ'
  assume "σ i = fst (id s)"
  and "σ' i = fst (id s')"
  and "(s, a, s') ∈ trans (paodv i)"
  then obtain q q' where "s = (σ i, q)"
    and "s' = (σ' i, q')"
    and "((σ i, q), a, (σ' i, q')) ∈ trans (paodv i)"
  by (cases s, cases s') auto
from this (3) have "((σ, q), a, (σ', q')) ∈ trans (opaodv i)"
  by simp (rule open_seqp_action [OF aodv_wf])

with 's = (σ i, q) and 's' = (σ' i, q')
  show "((σ, snd (id s)), a, (σ', snd (id s'))) ∈ trans (opaodv i)"
  by simp
qed

then interpret op: openproc paodv opaodv id

have [simp]: "∀ i. (SOME x. x ∈ (fst o id) ' init (paodv i)) = aodv_init i"
  unfolding σ_AODV_def by simp
hence "∀ i. openproc.initmissing paodv id i = initmissing i"
  unfolding op.initmissing_def op.someinit_def initmissing_def
  by (auto split: option.split)
thus "openproc.initmissing paodv id = initmissing" ..
qed

interpretation aodv_openproc_par_qmsg: openproc_parq paodv opaodv id qmsg
  where "aodv_openproc_par_qmsg.netglobal = netglobal"
  and "aodv_openproc_par_qmsg.initmissing = initmissing"
proof -
  have im: "∀ σ. openproc.netglobal (λ i. paodv i ' (qmsg) (λ (p, q). (fst (id p), snd (id p), q))) P σ = netglobal P σ"
    unfolding opq.netglobal_def netglobal_def opq.initmissing_def initmissing_def opq.someinit_def
    by (clarsimp cong: option.case_cong
      simp del: One_nat_def
      simp add: fst_initmissing_netgmap_default_aodv_init_netlift [symmetric, unfolded initmissing_def])
  thus "openproc.netglobal (λ i. paodv i ' (qmsg) (λ (p, q). (fst (id p), snd (id p), q))) = netglobal" by auto
qed

lemma net_nhop_quality_increases:
  assumes "wf_net_tree n"
  shows "closed (pnet (λ i. paodv i ' (qmsg) n)) \|= netglobal
  (λσ. ∀ i dip. let nhip = the (nhop (rt (σ i)) dip)
    in dip ∈ vD (rt (σ i)) \cap vD (rt (σ nhip)) ∧ nhip \neq dip
    → (rt (σ i)) \sqsubseteq dip (rt (σ nhip)))"
(is "_ \|= netglobal (λσ. ∀ i. ?inv σ i)")
proof -
  from "wf_net_tree n"
  have proto: "closed (pnet (λ i. paodv i ' (qmsg) n)) \|= netglobal (λσ. ∀ i∈net_tree_ips n. ∀ dip.
    let nhip = the (nhop (rt (σ i)) dip)
    in dip ∈ vD (rt (σ i)) \cap vD (rt (σ nhip)) ∧ nhip \neq dip
    → (rt (σ i)) \sqsubseteq dip (rt (σ nhip)))"
    by (rule aodv_openproc_par_qmsg.close_opnet [OF _ onet_nhop_quality_increases])
  show thesis
    unfolding invariant_def opnet_sos.opnet_tau1
    proof (rule, simp only: aodv_openproc_par_qmsg.netglobalsimp
      fst_initmissing_netgmap_pair_fst, rule allI)
      fix σ i
      assume sr: "σ ∈ reachable (closed (pnet (λ i. paodv i ' (qmsg) n)) TT"
      hence "∀ i∈net_tree_ips n. ?inv (fst (initmissing (netgmap fst σ))) i"
        by (drule invariantD [OF proto],
...
thus "\( \tau(\text{initmissing}\ (\text{netgmap}\ \text{fst}\ \sigma))\) i"

proof (cases "i \in \text{net_tree_ips}\ n")

assume "i \notin \text{net_tree_ips}\ n"

from sr have "\sigma \in \text{reachable}\ (\text{pnet}\ (\lambda i. \text{paodv}\ i\ qmsg)\ n)\ \text{TT}"

hence "\text{net_ips}\ \sigma = \text{net_tree_ips}\ n"

with (i \notin \text{net_tree_ips}\ n) have "i \notin \text{net_ips}\ \sigma" by simp

hence "(\text{fst}\ (\text{initmissing}\ (\text{netgmap}\ \text{fst}\ \sigma)))\ i = \text{aodv_init}\ i"

by simp

thus ?thesis by simp

qed

qed

3.11.6 Loop freedom of AODV

theorem aodv_loop_freedom:

assumes "\text{wf_net_tree}\ n"

shows "\text{closed}\ (\text{pnet}\ (\lambda i. \text{paodv}\ i\ qmsg)\ n)\ |\ \text{netglobal}\ (\lambda \sigma. \forall \text{dip. irrefl}\ ((\text{rt_graph}\ \sigma\ \text{dip})^+))"

using assms by (rule aodv_openproc_par_qmsg.netglobal_weakenE [OF net_nhop_quality_increases inv_to_loop_freedom])

end
Chapter 4

Variant D: Forwarding the Route Request

Explanation [4, §10.5]: In AODV’s route discovery process, a destination node (or an intermediate node with an active route to the destination) will generate a RREP message in response to a received RREQ message. The RREQ message is then dropped and not forwarded. This termination of the route discovery process at the destination can lead to other nodes inadvertently creating non-optimal routes to the source node [5]. A possible modification to solve this problem is to allow the destination node to continue to forward the RREQ message. A route request is only stopped if it has been handled before. The forwarded RREQ message from the destination node needs to be modified to include a Boolean flag \texttt{handled} that indicates a RREP message has already been generated and sent in response to the former message. In case the flag is set to true, it prevents other nodes (with valid route to the destination) from sending a RREP message in response to their reception of the forwarded RREQ message.

4.1 Predicates and functions used in the AODV model

theory D_Aodv_Data
import D_Fwdrreqs
begin

4.1.1 Sequence Numbers

Sequence numbers approximate the relative freshness of routing information.

definition inc :: "sqn ⇒ sqn"
where "inc sn ≡ if sn = 0 then sn else sn + 1"

lemma less_than_inc [simp]: "x ≤ inc x"
  unfolding inc_def by simp

lemma inc_minus_suc_0 [simp]:
  "inc x - Suc 0 = x"
  unfolding inc_def by simp

lemma inc_never_one' [simp, intro]: "inc x ≠ Suc 0"
  unfolding inc_def by simp

lemma inc_never_one [simp, intro]: "inc x ≠ 1"
  by simp

4.1.2 Modelling Routes

A route is a 6-tuple, \((dsn, dsk, flag, hops, nhip, pre)\) where \(dsn\) is the ‘destination sequence number’, \(dsk\) is the ‘destination-sequence-number status’, \(flag\) is the route status, \(hops\) is the number of hops to the destination, \(nhip\) is the next hop toward the destination, and \(pre\) is the set of ‘precursor nodes’ those interested in hearing about changes to the route.

type_synonym r = "sqn × k × f × nat × ip × ip set"

definition proj2 :: "r ⇒ sqn" ("\pi_2")
where \( \pi_2 \equiv \lambda (dsn, _, _, _, _). dsn \)

definition \( \text{proj3} :: "r \Rightarrow k" \) ("\( \pi_3 \)"
where \( \pi_3 \equiv \lambda (_, dsk, _, _, _). dsk \)

definition \( \text{proj4} :: "r \Rightarrow f" \) ("\( \pi_4 \)"
where \( \pi_4 \equiv \lambda (_, _, flag, _, _, _). flag \)

definition \( \text{proj5} :: "r \Rightarrow \text{nat}" \) ("\( \pi_5 \)"
where \( \pi_5 \equiv \lambda (_, _, hops, _, _). hops \)

definition \( \text{proj6} :: "r \Rightarrow \text{ip}" \) ("\( \pi_6 \)"
where \( \pi_6 \equiv \lambda (_, _, _, nhip, _). nhip \)

definition \( \text{proj7} :: "r \Rightarrow \text{ip set}" \) ("\( \pi_7 \)"
where \( \pi_7 \equiv \lambda (_, _, _, _, _, pre). pre \)

lemma \( \text{projs \ [simp]}: \)
\[
\pi_2 (dsn, dsk, flag, hops, nhip, pre) = dsn
\]
\[
\pi_3 (dsn, dsk, flag, hops, nhip, pre) = dsk
\]
\[
\pi_4 (dsn, dsk, flag, hops, nhip, pre) = flag
\]
\[
\pi_5 (dsn, dsk, flag, hops, nhip, pre) = hops
\]
\[
\pi_6 (dsn, dsk, flag, hops, nhip, pre) = nhip
\]
\[
\pi_7 (dsn, dsk, flag, hops, nhip, pre) = pre
\]
by (clarsimp simp: proj2_def proj3_def proj4_def
proj5_def proj6_def proj7_def)+

lemma \( \text{proj3 \_pred \ [intro]}: \) "\( P \ k\no; P \ unk \ \Longrightarrow \ P \ (\pi_3 \ x) \)"
by (rule k.induct)

lemma \( \text{proj4 \_pred \ [intro]}: \) "\( P \ val; P \ inv \ \Longrightarrow \ P \ (\pi_4 \ x) \)"
by (rule f.induct)

lemma \( \text{proj6 \_pair \_snd \ [simp]}: \)
fixes dsn' r
shows "\( \pi_6 (dsn', \text{snd} (r)) = \pi_6 (r) \)"
by (cases r) simp

4.1.3 Routing Tables
Routing tables map ip addresses to route entries.

type_synonym \( \text{rt} = "\text{ip} \rightarrow r" \)

syntax
\"\_\_Sigma\_route\" :: "\text{rt} \Rightarrow \text{ip} \rightarrow r" ("\( \sigma\_\text{route}'(_, _)\)"
translations
\"\sigma\_\text{route}(rt, dip)\" \Rightarrow "rt dip"

definition \( \text{sqn} :: "rt \Rightarrow \text{ip} \Rightarrow \text{sqn}" \)
where \( \text{sqn} \ rt \ dip \equiv \text{case } \sigma\_\text{route}(rt, dip) \text{ of } \text{Some } r \Rightarrow \pi_2 (r) \ | \ \text{None} \Rightarrow 0 \)

definition \( \text{sqnf} :: "rt \Rightarrow \text{ip} \Rightarrow k" \)
where \( \text{sqnf} \ rt \ dip \equiv \text{case } \sigma\_\text{route}(rt, dip) \text{ of } \text{Some } r \Rightarrow \pi_3 (r) \ | \ \text{None} \Rightarrow \text{unk} \)

abbreviation \( \text{flag} :: "rt \Rightarrow \text{ip} \Rightarrow f" \)
where \( \text{flag} \ rt \ dip \equiv \text{map\_option } \pi_4 (\sigma\_\text{route}(rt, dip)) \)

abbreviation \( \text{dhops} :: "rt \Rightarrow \text{ip} \Rightarrow \text{nat}" \)
where \( \text{dhops} \ rt \ dip \equiv \text{map\_option } \pi_5 (\sigma\_\text{route}(rt, dip)) \)

abbreviation \( \text{nhop} :: "rt \Rightarrow \text{ip} \Rightarrow \text{ip}" \)
where \( \text{nhop} \ rt \ dip \equiv \text{map\_option } \pi_6 (\sigma\_\text{route}(rt, dip)) \)
abbreviation \textit{precs} :: \( \text{"rt} \Rightarrow \text{ip} \rightarrow \text{ip} \text{ set} \)
  where \( \text{"precs rt dip} = \text{map_option} \pi \gamma (\sigma_{\text{route}(rt, dip)}) \) 

definition \textit{vD} :: \( \text{"rt} \Rightarrow \text{ip} \) set
  where \( \text{\"vD rt} = \{\text{dip. flag rt dip = Some val}\} \) 

definition \textit{iD} :: \( \text{"rt} \Rightarrow \text{ip} \) set
  where \( \text{\"iD rt} = \{\text{dip. flag rt dip = Some inv}\} \) 

definition \textit{kD} :: \( \text{"rt} \Rightarrow \text{ip} \) set
  where \( \text{\"kD rt} = \{\text{dip. rt dip \neq None}\} \) 

lemma \textsf{kD_is_vD_and_iD}: \( \text{\"kD rt} = \text{\"vD rt} \cup \text{\"iD rt} \)
  unfolding \textit{kD_def} \textit{vD_def} \textit{iD_def} by auto 

lemma \textsf{vD_iD_gives_kD [simp]}:
  \( \forall \text{ip rt. ip} \in \text{\"vD rt} \Rightarrow \text{ip} \in \text{\"kD rt} \) 
  \( \forall \text{ip rt. ip} \in \text{\"iD rt} \Rightarrow \text{ip} \in \text{\"kD rt} \)
  unfolding \textsf{kD_is_vD_and_iD} by \textsf{simp_all} 

lemma \textsf{kD_Some [dest]}:
  fixes dip rt
  assumes \( \text{\"dip} \in \text{\"kD rt} \)
  shows \( \exists \text{dsn dsk flag hops nhip pre.} \\ \sigma_{\text{route}(rt, dip)} = \text{Some (dsn, dsk, flag, hops, nhip, pre)} \) 
  using \textsf{assms unfolding \textit{kD_def} \textsf{by simp} 

lemma \textsf{kD_None [dest]}:
  fixes dip rt
  assumes \( \text{\"dip} \notin \text{\"kD rt} \)
  shows \( \sigma_{\text{route}(rt, dip)} = \text{None} \) 
  using \textsf{assms \textit{kD_def \textsf{by (metis (mono_tags) mem_Collect_eq}} 

lemma \textsf{vD_Some [dest]}:
  fixes dip rt
  assumes \( \text{\"dip} \in \text{\"vD rt} \)
  shows \( \exists \text{dsn dsk hops nhip pre.} \\ \sigma_{\text{route}(rt, dip)} = \text{Some (dsn, dsk, val, hops, nhip, pre)} \) 
  using \textsf{assms \textit{vD_def \textsf{by simp} 

lemma \textsf{vD_empty [simp]}: \( \text{\"vD Map.empty} = \{\} \) 
  unfolding \textit{vD_def \textsf{by simp} 

lemma \textsf{iD_Some [dest]}:
  fixes dip rt
  assumes \( \text{\"dip} \in \text{\"iD rt} \)
  shows \( \exists \text{dsn dsk hops nhip pre.} \\ \sigma_{\text{route}(rt, dip)} = \text{Some (dsn, dsk, inv, hops, nhip, pre)} \) 
  using \textsf{assms \textit{iD_def \textsf{by simp} 

lemma \textsf{val_is_vD [elim]}:
  fixes ip rt
  assumes \( \text{\"ip} \in \text{\"kD(rt)} \) 
  \( \text{and \"the (flag rt ip) = val} \)
  shows \( \text{\"ip} \in \text{\"vD(rt)} \) 
  using \textsf{assms \textit{vD_def \textsf{by auto} 

lemma \textsf{inv_is_iD [elim]}:
  fixes ip rt
  assumes \( \text{\"ip} \in \text{\"kD(rt)} \) 
  \( \text{and \"the (flag rt ip) = inv} \)
  shows \( \text{\"ip} \in \text{\"iD(rt)} \) 
  using \textsf{assms \textit{iD_def \textsf{by auto} 

334
lemma \textit{id\_flag\_is\_inv} \ [\textit{elim, simp}]:
  \begin{itemize}
  \item \textbf{fixes} ip rt
  \item \textbf{assumes} "ip \in \textit{id}(rt)"
  \item \textbf{shows} "the (flag rt ip) = inv"
  \item \textbf{proof -}
  \item from \langle \textit{ip \in id}(rt) \rangle have "ip \in \textit{kD}(rt)" by auto
  \item with \textbf{assms} \textbf{show} ?thesis unfolding \textit{id\_def} by auto
  \item \textbf{qed}
  \end{itemize}

lemma \textit{kD\_but\_not\_vD\_is\_iD} \ [\textit{elim}]:
  \begin{itemize}
  \item \textbf{fixes} ip rt
  \item \textbf{assumes} "ip \in \textit{kD}(rt)"
  \item and "ip / \in \textit{vD}(rt)"
  \item \textbf{shows} "ip \in \textit{iD}(rt)"
  \item \textbf{proof -}
  \item from \langle \textit{ip \in kD}(rt) \rangle obtain dsn dsk f hops nhop pre
  \item where rtip: "rt ip = Some (dsn, dsk, f, hops, nhop, pre)"
  \item by (metis \textit{kD\_Some})
  \item from \langle \textit{ip / \in vD}(rt) \rangle have "f \neq \textit{val}"
  \item proof \item rule \textit{contrapos\_nn}
  \item assume "f = \textit{val}"
  \item with rtip have "the (flag rt ip) = \textit{val}" by simp
  \item with \langle \textit{ip \in kD}(rt) \rangle show "ip \in \textit{vD}(rt)" ..
  \item \textbf{qed}
  \item with \langle \textit{ip \in kD}(rt) \rangle show "the (flag rt ip) = inv" by simp
  \item with \langle \textit{ip \in iD}(rt) \rangle show "ip \in \textit{iD}(rt)" ..
  \item \textbf{qed}
  \end{itemize}

lemma \textit{vD\_or\_iD} \ [\textit{elim}]:
  \begin{itemize}
  \item \textbf{fixes} ip rt
  \item \textbf{assumes} "ip \in \textit{kD}(rt)"
  \item and "ip \in \textit{vD}(rt) = \Rightarrow \textit{P rt ip}"
  \item and "ip \in \textit{iD}(rt) = \Rightarrow \textit{P rt ip}"
  \item \textbf{shows} \textit{P rt ip}
  \item \textbf{proof -}
  \item from \langle \textit{ip \in kD}(rt) \rangle have "ip \in \textit{vD}(rt) \cup \textit{iD}(rt)"
  \item by (simp add: \textit{kD\_is\_vD\_and\_iD})
  \item thus \textbf{?thesis} by (auto \textbf{elim:} \textbf{assms}(2-3))
  \item \textbf{qed}
  \end{itemize}

lemma \textit{proj5\_eq\_dhops}: "\(\forall \textit{dip rt}. \textit{dip \in kD}(rt) = \Rightarrow \pi_5(\textit{the (rt dip)}) = \textit{the (dhops rt dip)})"
  unfolding \textit{sqn\_def} by (drule \textit{kD\_Some}) clarsimp

lemma \textit{proj4\_eq\_flag}: "\(\forall \textit{dip rt}. \textit{dip \in kD}(rt) = \Rightarrow \pi_4(\textit{the (rt dip)}) = \textit{the (flag rt dip)})"
  unfolding \textit{sqn\_def} by (drule \textit{kD\_Some}) clarsimp

lemma \textit{proj2\_eq\_sqn}: "\(\forall \textit{dip rt}. \textit{dip \in kD}(rt) = \Rightarrow \pi_2(\textit{the (rt dip)}) = \textit{sqn rt dip})"
  unfolding \textit{sqn\_def} by (drule \textit{kD\_Some}) clarsimp

lemma \textit{kd\_sqnf\_is\_proj3} \ [\textit{simp}]:
  "\(\forall \textit{ip rt}. \textit{ip \in kD}(rt) = \Rightarrow \textit{sqnf rt ip} = \pi_3(\textit{the (rt ip)})"
  unfolding \textit{sqnf\_def} by auto

lemma \textit{vD\_flag\_val} \ [\textit{simp}]:
  "\(\forall \textit{dip rt}. \textit{dip \in vD (rt)} = \Rightarrow \textit{the (flag rt dip)} = \textit{val})"
  unfolding \textit{vD\_def} by clarsimp

lemma \textit{kd\_update} \ [\textit{simp}]:
  "\(\forall \textit{rt nip v}. \textit{kD rt nip rt nip \Rightarrow v}) = \textit{insert nip (kD rt)}"
  unfolding \textit{kD\_def} by auto

lemma \textit{kd\_empty} \ [\textit{simp}]: "\textit{kD Map.empty = \{\}}"
  unfolding \textit{kD\_def} by simp
lemma ip_equal_or_known [elim]:
fixes rt ip ip'
assumes "ip = ip' ∨ ip ∈ kD(rt)"
and "ip = ip' ⟹ P rt ip ip'"
and "[ ip ≠ ip'; ip ∈ kD(rt) ] ⟹ P rt ip ip'"
shows "P rt ip ip'"
using assms by auto

4.1.4 Updating Routing Tables
Routing table entries are modified through explicit functions. The properties of these functions are important in invariant proofs.

Updating Precursor Lists

definition addpre :: "r ⇒ ip set ⇒ r"
where "addpre r npre ≡ let (dsn, dsk, flag, hops, nhip, pre) = r in
(dsn, dsk, flag, hops, nhip, pre ∪ npre)"

lemma proj2_addpre:
fixes v pre
shows "π_2(addpre v pre) = π_2(v)"
unfolding addpre_def by (cases v) simp

lemma proj3_addpre:
fixes v pre
shows "π_3(addpre v pre) = π_3(v)"
unfolding addpre_def by (cases v) simp

lemma proj4_addpre:
fixes v pre
shows "π_4(addpre v pre) = π_4(v)"
unfolding addpre_def by (cases v) simp

lemma proj5_addpre:
fixes v pre
shows "π_5(addpre v pre) = π_5(v)"
unfolding addpre_def by (cases v) simp

lemma proj6_addpre:
fixes dsn dsk flag hops nhip pre npre
shows "π_6(addpre v npre) = π_6(v)"
unfolding addpre_def by (cases v) simp

lemma proj7_addpre:
fixes dsn dsk flag hops nhip pre npre
shows "π_7(addpre v npre) = π_7(v) ∪ npre"
unfolding addpre_def by (cases v) simp

lemma addpre_empty: "addpre r {} = r"
unfolding addpre_def by simp

lemma addpre_r:
"addpre (dsn, dsk, f1, hops, nhip, pre) npre = (dsn, dsk, f1, hops, nhip, pre ∪ npre)"
unfolding addpre_def by simp

lemmas addpre_simps [simp] = proj2_addpre proj3_addpre proj4_addpre proj5_addpre
proj6_addpre proj7_addpre addpre_empty addpre_r

definition addpreRT :: "rt ⇒ ip set ⇒ rt"
where "addpreRT rt dip npre ≡ map_option (λs. rt (dip ↦ addpre s npre)) (σ_route(rt, dip))"
lemma snd_addpre [simp]:
"\( \forall dsn\ dsn'\ v \ pre. (dsn, snd (addpre (dsn', v) pre)) = addpre (dsn, v) pre" 
unfolding addpre_def by clarsimp

lemma proj2_addpreRT [simp]:
fixes ip rt ip' npre 
assumes "ip\(\in\)kd rt" 
and "ip'\(\in\)kd rt" 
shows "\\pi_2(\text{the}\ (\text{the}\ (addpreRT\ rt\ ip'\ npre)\ ip)) = \pi_2(\text{the}\ (rt\ ip))" 
using assms [THEN kD_Some] unfolding addpreRT_def by clarsimp

lemma proj3_addpreRT [simp]:
fixes ip rt ip' npre 
assumes "ip\(\in\)kd rt" 
and "ip'\(\in\)kd rt" 
shows "\\pi_3(\text{the}\ (\text{the}\ (addpreRT\ rt\ ip'\ npre)\ ip)) = \pi_3(\text{the}\ (rt\ ip))" 
using assms [THEN kD_Some] unfolding addpreRT_def by clarsimp

lemma proj5_addpreRT [simp]:
"\( \forall rt\ dip\ ip\ npre. dip\(\in\)kd(\text{rt})\\Rightarrow\ \pi_5(\text{the}\ (\text{addpreRT}\ rt\ dip\ npre)\ ip) = \pi_5(\text{the}\ (rt\ ip))" 
unfolding addpreRT_def by auto

lemma flag_addpreRT [simp]:
fixes rt pre ip dip 
assumes "dip\(\in\)kd rt" 
shows "flag (\text{the}\ (addpreRT\ rt\ dip\ npre))\ ip = flag rt ip" 
unfolding addpreRT_def 
using assms [THEN kD_Some] by (clarsimp)

lemma kD_addpreRT [simp]:
fixes rt dip npre 
assumes "dip\(\in\)kd rt" 
shows "kd (\text{the}\ (addpreRT\ rt\ dip\ npre)) = kd rt" 
unfolding kD_def addpreRT_def 
using assms [THEN kD_Some] by clarsimp blast

lemma vD_addpreRT [simp]:
fixes rt dip npre 
assumes "dip\(\in\)kd rt" 
shows "vD (\text{the}\ (addpreRT\ rt\ dip\ npre)) = vD rt" 
unfolding vD_def addpreRT_def 
using assms [THEN kD_Some] by clarsimp auto

lemma iD_addpreRT [simp]:
fixes rt dip npre 
assumes "dip\(\in\)kd rt" 
shows "iD (\text{the}\ (addpreRT\ rt\ dip\ npre)) = iD rt" 
unfolding iD_def addpreRT_def 
using assms [THEN kD_Some] by clarsimp auto

lemma nhop_addpreRT [simp]:
fixes rt pre ip dip 
assumes "dip\(\in\)kd rt" 
shows "nhop (\text{the}\ (addpreRT\ rt\ dip\ npre))\ ip = nhop rt ip" 
unfolding sqn_def addpreRT_def 
using assms [THEN kD_Some] by (clarsimp)

lemma sqn_addpreRT [simp]:
fixes rt pre ip dip 
assumes "dip\(\in\)kd rt" 
shows "sqn (\text{the}\ (addpreRT\ rt\ dip\ npre))\ ip = sqn rt ip" 
unfolding sqn_def addpreRT_def 
using assms [THEN kD_Some] by (clarsimp)
lemma dhops_addpreRT [simp]:
  fixes rt pre ip dip
  assumes "dip ∈ kD rt"
  shows "dhops (the (addpreRT rt dip pre)) ip = dhops rt ip"
unfolding addpreRT_def
using assms [THEN kD_Some] by (clarsimp)

lemma sqnf_addpreRT [simp]:
"\(\forall ip \ dip. \ \text{ip} \in \text{kD}(rt \ \xi) \implies sqnf (the (addpreRT (rt \ \xi) \ ip \ npre)) \ dip = sqnf (rt \ \xi) \ dip\)"
unfolding sqnf_def addpreRT_def by auto

Updating route entries

lemma in_kD_case [simp]:
  fixes dip rt
  assumes "dip ∈ kD(rt)"
  shows "\((\text{case } rt \ dip \ \text{of None } \Rightarrow \text{en} | \ Some r \Rightarrow es r) = es (the (rt dip))\)"
using assms [THEN kD_Some] by auto

lemma not_in_kD_case [simp]:
  fixes dip rt
  assumes "dip /∈ kD(rt)"
  shows "\((\text{case } rt \ dip \ \text{of None } \Rightarrow \text{en} | \ Some r \Rightarrow es r) = \text{en}\)"
using assms [THEN kD_None] by auto

lemma rt_Some_sqn [dest]:
  fixes rt and ip dsn dsk flag hops nhip pre
  assumes "rt ip = Some (dsn, dsk, flag, hops, nhip, pre)"
  shows "sqn rt ip = dsn"
unfolding sqn_def using assms by simp

lemma not_kD_sqn [simp]:
  fixes dip rt
  assumes "dip /∈ kD(rt)"
  shows "sqn rt dip = 0"
using assms unfolding sqn_def by simp

definition update_arg_wf :: "r ⇒ bool"
where "update_arg_wf r ≡ \(\pi_4(r) = \text{val} \land (\pi_2(r) = 0) = (\pi_3(r) = \text{unk}) \land (\pi_3(r) = \text{unk} \implies \pi_5(r) = 1)\)"

lemma update_arg_wf_gives_cases:
"\(\forall r. \ update_arg_wf r \implies (\pi_2(r) = 0) = (\pi_3(r) = \text{unk})\)"
unfolding update_arg_wf_def by simp

lemma update_arg_wf_tuples [simp]:
"\(\forall \text{nhip pre. update_arg_wf (0, unk, val, Suc 0, nhip, pre)}\)"
"\(\forall \text{n hops nhip pre. update_arg_wf (Suc n, kno, val, hops, nhip, pre)}\)"
unfolding update_arg_wf_def by auto

lemma update_arg_wf_tuples' [elim]:
"\(\forall \text{n hops nhip pre. Suc 0 ≤ n \implies update_arg_wf (n, kno, val, hops, nhip, pre)}\)"
unfolding update_arg_wf_def by auto

lemma wf_r_cases [intro]:
  fixes P r
  assumes "update_arg_wf r"
  and c1: "\(\forall \text{nhip pre. P (0, unk, val, Suc 0, nhip, pre)}\)"
  and c2: "\(\forall \text{dsn hops nhip pre. dsn > 0 \implies P (dsn, kno, val, hops, nhip, pre)}\)"
  shows "P r"
proof -
obtain dsn dsk flag hops nhip pre 
where *: "r = (dsn, dsk, flag, hops, nhip, pre)" by (cases r) 
with update_arg_wf r have wfl: "flag = val"
 and wf2: "(dsn = 0) = (dsk = unk)"
 and wf3: "dsk = unk ⌣→ (hops = 1)"
 unfolding update_arg_wf_def by auto 
have "P (dsn, dsk, flag, hops, nhip, pre)"
proof (cases r)
 assume "dsn = dsk" 
 moreover with wf2 wf3 have "dsn = 0" and "hops = Suc 0" by auto 
 ultimately show ?thesis using ⟨flag = val⟩ by simp (rule c1)
next 
 assume "dsk = kno" 
 moreover with wf2
 have "dsn > 0" by simp
 ultimately show ?thesis using ⟨flag = val⟩ by simp (rule c2)
qed 
with * show "P r" by simp 
qed

definition update :: "rt ⇒ ip ⇒ r ⇒ rt" 
where
"update rt ip r ≡ case σ_route(rt, ip) of
  None ⇒ rt (ip ↦→ r)
| Some s ⇒ if π_2(s) < π_2(r) then rt (ip ↦→ addpre r (π_7(s)))
  else if π_2(s) = π_2(r) ∧ (π_5(s) > π_5(r) ∨ π_4(s) = inv)
  then rt (ip ↦→ addpre r (π_7(s)))
  else if π_3(r) = unk
  then rt (ip ↦→ (π_3(s), snd (addpre r (π_7(s)))))
  else if π_3(r) = unk then rt (ip ↦→ addpre s (π_7(r)))
  else rt (ip ↦→ addpre s (π_7(r))))"

lemma update_simps [simp]: 
fixes r s nr nr' ns rt ip
defines "s ≡ the σ_route(rt, ip)"
and "nr ≡ addpre r (π_7(s))"
and "nr' ≡ (π_3(s), π_3(nr), π_4(nr), π_5(nr), π_6(nr), π_7(nr))"
and "ns ≡ addpre s (π_7(r))"
shows
"[¬ ip ∈ kD(rt)]" ⟹ update rt ip r = rt (ip ↦→ r)"
"[ip ∈ kD(rt); sqn rt ip < π_2(r)]" ⟹ update rt ip r = rt (ip ↦→ nr)"
"[ip ∈ kD(rt); sqn rt ip = π_2(r);
  the (dhops rt ip) > π_5(r)]" ⟹ update rt ip r = rt (ip ↦→ nr)
"[ip ∈ kD(rt); sqn rt ip = π_2(r);
  flag rt ip = Some inv]" ⟹ update rt ip r = rt (ip ↦→ nr)"
"[ip ∈ kD(rt); π_3(r) = unk; (π_2(r) = 0) = (π_3(r) = unk)]" ⟹ update rt ip r = rt (ip ↦→ nr')"
"[ip ∈ kD(rt); sqn rt ip ≥ π_2(r); π_3(r) = kno;
  sqn rt ip = π_2(r) ⌣→ the (dhops rt ip) ≤ π_5(r) ∧ the (flag rt ip) = val ]" ⟹ update rt ip r = rt (ip ↦→ ns)
proof -
assume "ip ∉ kD(rt)"
hence "σ_route(rt, ip) = None" ..
thus "update rt ip r = rt (ip ↦→ r)"
unfolding update_def by simp 
next 
assume "ip ∈ kD(rt)"
and "sqn rt ip < π_2(r)"
from this(1) obtain dsn dsk fl hops nhip pre 
where "rt ip = Some (dsn, dsk, fl, hops, nhip, pre)" 
by (metis kD_Some)
with ⟨sqn rt ip < π_2(r): show "update rt ip r = rt (ip ↦→ nr)"
unfolding update_def nr_def s_def by auto 
next 
assume "ip ∈ kD(rt)"

and "sqn rt ip = π₂(r)"
and "the (dhops rt ip) > π₅(r)"

from this(1) obtain dsn dsk fl hops nhip pre
where "rt ip = Some (dsn, dsk, fl, hops, nhip, pre)"
by (metis kD_Some)
with ⟨sqn rt ip = π₂(r)⟩ and ⟨the (dhops rt ip) > π₅(r)⟩
show "update rt ip r = rt (ip ↦→ nr)"
unching update_def nr_def s_def by auto

next
assume "ip ∈ kD(rt)"
and "sqn rt ip = π₃(r)"
and "flag rt ip = Some inv"
from this(1) obtain dsn dsk fl hops nhip pre
where "rt ip = Some (dsn, dsk, fl, hops, nhip, pre)"
by (metis kD_Some)
with ⟨sqn rt ip = π₃(r)⟩ and ⟨flag rt ip = Some inv⟩
show "update rt ip r = rt (ip ↦→ nr)"
unching update_def nr_def s_def by auto

next
assume "ip ∈ kD(rt)"
and "π₃(r) = unk"
and "(π₃(r) = 0) = (π₃(r) = unk)"
from this(1) obtain dsn dsk fl hops nhip pre
where "rt ip = Some (dsn, dsk, fl, hops, nhip, pre)"
by (metis kD_Some)
with ⟨π₃(r) = 0⟩ and ⟨π₃(r) = unk⟩
show "update rt ip r = rt (ip ↦→ nr)"
unching update_def nr_def n_def s_def by auto

next
assume "ip ∈ kD(rt)"
and otherassms: "sqn rt ip ≥ π₂(r)"
"π₃(r) = kno"
"sqn rt ip = π₂(r) ⇒ the (dhops rt ip) ≤ π₅(r) ∧ the (flag rt ip) = val"
from this(1) obtain dsn dsk fl hops nhip pre
where "rt ip = Some (dsn, dsk, fl, hops, nhip, pre)"
by (metis kD_Some)
with otherassms show "update rt ip r = rt (ip ↦→ ns)"
unching update_def ns_def s_def by auto

qed

lemma update_cases [elim]:
assumes "(π₂(r) = 0) = (π₃(r) = unk)"
and c1: "[| ip /∈ kD(rt) |] ⇒ P (rt (ip ↦→ r))"
and c2: "[| ip ∈ kD(rt); sqn rt ip < π₂(r) |]
⇒ P (rt (ip ↦→ addpre r (π₇(the σroute(rt, ip)))))"
and c3: "[| ip ∈ kD(rt); sqn rt ip = π₂(r); the (dhops rt ip) > π₅(r) |]
⇒ P (rt (ip ↦→ addpre r (π₇(the σroute(rt, ip)))))"
and c4: "[| ip ∈ kD(rt); sqn rt ip = π₂(r); the (flag rt ip) = inv |]
⇒ P (rt (ip ↦→ addpre r (π₇(the σroute(rt, ip)))))"
and c5: "[| ip ∈ kD(rt); π₃(r) = unk |]
⇒ P (rt (ip ↦→ (π₇(the σroute(rt, ip)), π₃(r),
π₄(r), π₅(r), π₆(r), π₇(addpre r (π₇(the σroute(rt, ip)))))))"
and c6: "[| ip ∈ kD(rt); sqn rt ip ≥ π₂(r); π₃(r) = kno; sqn rt ip = π₂(r) ⇒ the (dhops rt ip) ≤ π₅(r) ∧ the (flag rt ip) = val |]
⇒ P (rt (ip ↦→ addpre (the σroute(rt, ip)) (π₇(r)))))"
shows "(P (update rt ip r))"

proof (cases "ip ∈ kD(rt)")
assume "ip /∈ kD(rt)"
with c1 show ?thesis
by simp

next
assume "ip ∈ kD(rt)"
moreover then obtain dsn dsk fl hops nhip pre
where rteq: "rt ip = Some (dsn, dsk, fl, hops, nhip, pre)"
by (metis kD_Some)
mcorver obtain dsn' dsk' fl' hops' nhip' pre'
where req: "r = (dsn', dsk', fl', hops', nhip', pre')"
by (cases r) metis
ultimately show ?thesis
using ⟨
  \pi_2 (r) = 0 \equiv \pi_3 (r) = \text{unk}\rangle
c2 [OF \langle \text{ip} \in \text{kD} (rt) \rangle]
c3 [OF \langle \text{ip} \in \text{kD} (rt) \rangle]
c4 [OF \langle \text{ip} \in \text{kD} (rt) \rangle]
c5 [OF \langle \text{ip} \in \text{kD} (rt) \rangle]
c6 [OF \langle \text{ip} \in \text{kD} (rt) \rangle]

unfolding update_def sqn_def by auto
qed

lemma update_cases_kD:
  assumes "(\pi_2 (r) = 0) = (\pi_3 (r) = \text{unk})"
  and "\text{ip} \in \text{kD} (rt)"
  and \text{c2}: "\text{sqn} (rt \text{ ip} < \pi_2 (r) \equiv \text{P} (rt (\text{ip} \mapsto addpre r (\pi_7 (\text{route} (rt, \text{ip}))))))"
  and \text{c3}: "\text{sqn} rt \text{ ip} = \pi_2 (r); \text{the} (\text{dhops} \text{ rt} \text{ ip}) > \pi_5 (r) \equiv \text{P} (rt (\text{ip} \mapsto addpre r (\pi_7 (\text{route} (rt, \text{ip}))))))"
  and \text{c4}: "\text{sqn} rt \text{ ip} = \pi_2 (r); \text{the} (\text{flag} \text{ rt} \text{ ip}) = \text{inv} \equiv \text{P} (rt (\text{ip} \mapsto addpre r (\pi_7 (\text{route} (rt, \text{ip}))))))"
  and \text{c5}: "\pi_3 (r) = \text{unk} \equiv \text{P} (rt (\text{ip} \mapsto (\pi_2 (\text{the} (\text{route} (rt, \text{ip}))), \pi_3 (r), \pi_4 (r), \pi_5 (r), \pi_6 (r), \pi_7 (\text{addpre} r (\pi_7 (\text{the} (\text{route} (rt, \text{ip}))))))))"
  and \text{c6}: "\text{sqn} rt \text{ ip} \geq \pi_2 (r); \pi_3 (r) = \text{kno}; \text{sqn} rt \text{ ip} = \pi_2 (r) \equiv \text{the} (\text{dhops} \text{ rt} \text{ ip}) \leq \pi_5 (r) \land \text{the} (\text{flag} \text{ rt} \text{ ip}) = \text{val} \equiv \text{P} (rt (\text{ip} \mapsto addpre (\text{the} (\text{route} (rt, \text{ip}))) (\pi_7 (r))))"

shows "(\text{P} (\text{update} \text{ rt} \text{ ip} \text{ r}))"
using assms(1) proof (rule update_cases)
  assume "\text{sqn} rt \text{ ip} < \pi_2 (r)"
  thus "\text{P} (rt (\text{ip} \mapsto addpre r (\pi_7 (\text{the} (\text{rt} \text{ ip}))))))" by (rule c2)
next
  assume "\text{sqn} rt \text{ ip} = \pi_2 (r)"
  and "\text{the} (\text{dhops} \text{ rt} \text{ ip}) > \pi_5 (r)"
  thus "\text{P} (rt (\text{ip} \mapsto addpre r (\pi_7 (\text{the} (\text{rt} \text{ ip}))))))"
  by (rule c3)
next
  assume "\text{sqn} rt \text{ ip} = \pi_2 (r)"
  and "\text{the} (\text{flag} \text{ rt} \text{ ip}) = \text{inv}"
  thus "\text{P} (rt (\text{ip} \mapsto addpre r (\pi_7 (\text{the} (\text{rt} \text{ ip}))))))"
  by (rule c4)
next
  assume "\pi_3 (r) = \text{unk}"
  thus "\text{P} (rt (\text{ip} \mapsto (\pi_2 (\text{the} (\text{route} (rt, \text{ip}))), \pi_3 (r), \pi_4 (r), \pi_5 (r), \pi_6 (r), \pi_7 (\text{addpre} r (\pi_7 (\text{the} (\text{route} (rt, \text{ip}))))))))"
  by (rule c5)
next
  assume "\text{sqn} rt \text{ ip} \geq \pi_2 (r)"
  and "\pi_3 (r) = \text{kno}"
  and "\text{sqn} rt \text{ ip} = \pi_2 (r) \equiv \text{the} (\text{dhops} \text{ rt} \text{ ip}) \leq \pi_5 (r) \land \text{the} (\text{flag} \text{ rt} \text{ ip}) = \text{val}"
  thus "\text{P} (rt (\text{ip} \mapsto addpre (\text{the} (\text{route} (rt, \text{ip}))) (\pi_7 (r))))"
  by (rule c6)
qed (simp add: \langle \text{ip} \in \text{kD} (rt) \rangle)

lemma in_kD_after_update [simp]:
  fixes rt nip dsn dsk flag hops nhip pre
  shows "\text{kD} (\text{update} \text{ rt} \text{ nip} \langle \text{dsn}, \text{dsk}, \text{flag}, \text{hops}, \text{nhip}, \text{pre} \rangle) = \text{insert} \text{ nip} (\text{kD} \text{ rt})"
unfolding update_def
by (cases "\text{rt} \text{ nip}")

lemma nhop_of_update [simp]:
fixes rt dip dsn dsk flag hops nhip
assumes "rt ≠ update rt dip (dsn, dsk, flag, hops, nhip, {})"
shows "the (nhop (update rt dip (dsn, dsk, flag, hops, nhip, {})) dip) = nhip"
proof -
from assms
have update_neq: "∀ v. rt dip = Some v ⇒
  update rt dip (dsn, dsk, flag, hops, nhip, {})
  ≠ rt(dip ↦ addpre (the (rt dip)) (πτ (dsn, dsk, flag, hops, nhip, {})))"
  by auto
show ?thesis
proof (cases "rt dip = None")
  assume "rt dip = None"
  thus ?thesis
    unfolding update_def by clarsimp
next
  assume "rt dip ≠ None"
  then obtain v where "rt dip = Some v" by (metis not_None_eq)
  with update_neq [OF this] show ?thesis
    unfolding update_def by auto
qed
qed

lemma sqn_if_updated:
  fixes rip v rt ip
  shows "sqn (λ x. if x = rip then Some v else rt x) ip
        = (if ip = rip then π2(v) else sqn rt ip)"
  unfolding sqn_def by simp

lemma update_sqn [simp]:
  fixes rt dip rip dsn dsk hops nhip pre
  assumes "(dsn = 0) = (dsk = unk)"
  shows "sqn rt dip ≤ sqn (update rt rip (dsn, dsk, val, hops, nhip, pre)) dip"
proof (rule update_cases)
  show "(π2 (dsn, dsk, val, hops, nhip, pre) = 0) = (π3 (dsn, dsk, val, hops, nhip, pre) = unk)"
    by simp (rule assms)
qed (clarsimp simp: sqn_if_updated sqn_def)+

lemma sqn_update_bigger [simp]:
  fixes rt ip ip' dsn dsk flag hops nhip pre
  assumes "1 ≤ hops"
  shows "sqn rt ip ≤ sqn (update rt ip' (dsn, dsk, flag, hops, nhip, pre)) ip"
using assms unfolding update_def sqn_def
by (clarsimp split: option.split) auto

lemma dhops_update [intro]:
  fixes rt dsn dsk flag hops ip rip nhip pre
  assumes ex: "∀ ip∈kD rt. the (dhops rt ip) ≥ 1"
      and ip: "(ip = rip ∧ Suc 0 ≤ hops) ∨ (ip ≠ rip ∧ ip∈kD rt)"
  shows "Suc 0 ≤ the (dhops (update rt rip (dsn, dsk, flag, hops, nhip, pre)) ip)"
using ip proof
  assume "ip = rip ∧ Suc 0 ≤ hops" thus ?thesis
  unfolding update_def using ex
  by (cases "rip ∈ kD rt") (drule(1) bspec, auto)
next
  assume "ip ≠ rip ∧ ip∈kD rt" thus ?thesis
  using ex unfolding update_def
  by (cases "rip∈kD rt") auto
qed

lemma update_another [simp]:
  fixes dip ip rt dsn dsk flag hops nhip pre
  assumes "ip ≠ dip"
  shows "((update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = rt ip"
using assms unfolding update_def
by (clarsimp split: option.split)
lemma nhop_update_another [simp]:
  fixes dip ip rt dan dsk flag hops nhip pre 
  assumes "ip ≠ dip"
  shows "nhop (update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = nhop rt ip"
  using assms unfolding update_def
  by (clarsimp split: option.split)

lemma dhops_update_another [simp]:
  fixes dip ip rt dan dsk flag hops nhip pre 
  assumes "ip ≠ dip"
  shows "dhops (update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = dhops rt ip"
  using assms unfolding update_def
  by (clarsimp split: option.split)

lemma sqn_update_same [simp]:
  "∀ rt ip dsn dsk flag hops nhip pre. sqn (rt(ip ↦→ v)) ip = π₂(v)"
  unfolding sqn_def
  by simp

lemma nhop_update_unk_val [simp]:
  "∀ rt dip dsn hops npre. the (nhop (update rt dip (dsn, unk, val, hops, ip, npre)) dip) = ip"
  unfolding update_def
  by (clarsimp split: option.split)

lemma nhop_update_changed [simp]:
  fixes rt dip dsn dsk flg hops sip 
  assumes "update rt dip (dsn, dsk, flg, hops, sip, {}) ≠ rt"
  shows "the (nhop (update rt dip (dsn, dsk, flg, hops, sip, {})) dip) = sip"
  using assms unfolding update_def
  by (clarsimp split: option.splits split_if_asm) auto

lemma update_rt_split_asm:
  "∀ rt ip dsn dsk flag hops sip. 
   P (update rt ip (dsn, dsk, flag, hops, sip, {})) = 
   (¬(rt = update rt ip (dsn, dsk, flag, hops, sip, {})) ∧ ¬P rt 
    ∨ rt ≠ update rt ip (dsn, dsk, flag, hops, sip, {}}) ∧ ¬P (update rt ip (dsn, dsk, flag, hops, sip, {}}}))"
  by auto

lemma sqn_update [simp]: "∀ rt dip dsn flg hops sip. 
  rt ≠ update rt dip (dsn, kno, flg, hops, sip, {}) 
  ⇒ sqn (update rt dip (dsn, kno, flg, hops, sip, {})) dip = dsn"
  unfolding update_def by (clarsimp option.split split_if_asm) auto

lemma sqnf_update [simp]: "∀ rt dip dsn flg hops sip. 
  rt ≠ update rt dip (dsn, kno, flg, hops, sip, {}) 
  ⇒ sqnf (update rt dip (dsn, kno, flg, hops, sip, {})) dip = dsk"
  unfolding update_def sqnf_def
  by (clarsimp option.splits split_if_asm) auto

lemma update_kno_dsn_greater_zero:
  "∀ rt dip dsn hops npre. 1 ≤ dsn ⇒ 1 ≤ (sqn (update rt dip (dsn, kno, val, hops, ip, npre)) dip)"
  unfolding update_def
  by (clarsimp option.splits)

lemma proj3_update [simp]: "∀ rt dip dsn dsk flg hops sip.
$rt \neq \text{update}\ rt\ dip\ (dsn,\ dsk,\ flg,\ hops,\ sip,\ \{\})$
$\implies \pi_3(\text{the}\ (\text{update}\ rt\ dip\ (dsn,\ dsk,\ flg,\ hops,\ sip,\ \{\})\ dip)) = dsk$

unfolding update_def sqnf_def
by (clarsimp split: option.splits split_if_asm) auto

lemma nhop_update_changed_kno_val [simp]: "\rt\ ip\ dsn\ dsk\ hops\ nhip.\ rt \neq \text{update}\ rt\ ip\ (dsn,\ kno,\ val,\ hops,\ nhip,\ \{\})\ \implies\ \text{the}\ (\text{nhop}\ (\text{update}\ rt\ ip\ (dsn,\ kno,\ val,\ hops,\ nhip,\ \{\}))\ ip) = nhip"
using assms unfolding update_def
by (clarsimp split: option.split_asm option.split split_if_asm) auto

lemma flag_update [simp]: "\rt\ dip\ dsn\ flg\ hops\ sip.\ rt \neq \text{update}\ rt\ dip\ (dsn,\ kno,\ flg,\ hops,\ sip,\ \{\})\ \implies\ \text{the}\ (\text{flag}\ (\text{update}\ rt\ dip\ (dsn,\ kno,\ flg,\ hops,\ sip,\ \{\}))\ dip) = flg"
unfolding update_def
by (clarsimp split: option.split split_if_asm) auto

lemma the_flag_Some [dest!]:
fixes ip rt
assumes "the\ (\text{flag}\ rt\ ip) = x"
and "ip\ \in\ kD\ rt"
shows "\text{flag}\ rt\ ip = Some\ x"
using assms by auto

lemma kD_update_unchanged [dest]:
fixes rt dip dsn dsk flag hops nhip pre
assumes "rt = \text{update}\ rt\ dip\ (dsn,\ dsk,\ flag,\ hops,\ nhip,\ pre)"
shows "dip\ \in\ kD(rt)"
proof -
have "dip\ \in\ kD(update\ rt\ dip\ (dsn,\ dsk,\ flag,\ hops,\ nhip,\ pre))" by simp
with assms show \thesis \by simp
qed

lemma nhop_update [simp]: "\rt\ dip\ dsn\ dsk\ flg\ hops\ sip.\ rt \neq \text{update}\ rt\ dip\ (dsn,\ dsk,\ flg,\ hops,\ sip,\ \{\})\ \implies\ \text{the}\ (\text{nhop}\ (\text{update}\ rt\ dip\ (dsn,\ dsk,\ flg,\ hops,\ sip,\ \{\}))\ dip) = sip"
unfolding update_def sqnf_def
by (clarsimp split: option.splits split_if_asm) auto

lemma sqn_update_another [simp]:
fixes dip ip rt dsn dsk flag hops nhip pre
assumes "ip\ \neq\ dip"
shows "sqn\ (update\ rt\ dip\ (dsn,\ dsk,\ flag,\ hops,\ nhip,\ pre))\ ip = sqn\ rt\ ip"
using assms unfolding update_def sqnf_def
by (clarsimp split: option.splits) auto

lemma sqnf_update_another [simp]:
fixes dip ip rt dsn dsk flag hops nhip pre
assumes "ip\ \neq\ dip"
shows "sqnf\ (update\ rt\ dip\ (dsn,\ dsk,\ flag,\ hops,\ nhip,\ pre))\ ip = sqnf\ rt\ ip"
using assms unfolding update_def sqnf_def
by (clarsimp split: option.splits) auto

lemma vD_update_val [dest]:
"\{dip\ rt\ dip’\ dsn\ dsk\ hops\ nhip\ pre.\ dip\ \in\ vD(update\ rt\ dip’\ (dsn,\ dsk,\ val,\ hops,\ nhip,\ pre))\ \implies\ (dip\ \in\ vD(rt)\ \ve\ dip=dip’)"
unfolding update_def vD_def by (clarsimp split: option.split_asm split_if_asm)

Invalidating route entries

definition invalidate :: "%rt\ \Rightarrow\ (ip\ \Rightarrow\ sqn)\ \Rightarrow\ rt"
where "\text{invalidate}\ rt\ dests\ \equiv\ \\
\lambda ip.\ \text{case}\ (rt\ ip,\ dests\ ip)\ \of\ \\
(\text{None,}_)\ \Rightarrow\ \text{None}"

344
lemma proj3_invalidate [simp]:
"\( \forall dip. \pi_3 (\text{the } (\text{invalidate rt dests) dip)) = \pi_3 (\text{the } (rt dip)) \)"
unfolding invalidate_def by (clarsimp split: option.split)

lemma proj5_invalidate [simp]:
"\( \forall dip. \pi_5 (\text{the } (\text{invalidate rt dests) dip)) = \pi_5 (\text{the } (rt dip)) \)"
unfolding invalidate_def by (clarsimp split: option.split)

lemma proj6_invalidate [simp]:
"\( \forall dip. \pi_6 (\text{the } (\text{invalidate rt dests) dip)) = \pi_6 (\text{the } (rt dip)) \)"
unfolding invalidate_def by (clarsimp split: option.split)

lemma proj7_invalidate [simp]:
"\( \forall dip. \pi_7 (\text{the } (\text{invalidate rt dests) dip)) = \pi_7 (\text{the } (rt dip)) \)"
unfolding invalidate_def by (clarsimp split: option.split)

lemma invalidate_kD_inv [simp]:
"\( \forall rt \text{ dests}. kD (\text{invalidate rt dests}) = kD rt \)"
unfolding invalidate_def kD_def
by (simp split: option.split)

lemma invalidate_sqn:
fixes rt dip dests
assumes "\( \forall rsn. \text{ dests dip = Some rsn } \Rightarrow \text{ sqn rt dip } \leq rsn \)"
shows "\( \text{ sqn rt dip } \leq \text{ sqn (invalidate rt dests) dip} \)"
proof (cases "\( \text{ dip } \not\in \text{ kD(rt)} \)"
  assume "\( \text{ dip } \not\in \text{ kD(rt)} \)"
  hence "\( \text{ dip } \in \text{ kD(rt)} \)" by simp
  then obtain dsn dsk flag hops nhip pre where "\( rt dip = \text{ Some (dsn, dsk, flag, hops, nhip, pre)} \)"
    by (metis kD_Some)
  with assms show "\( \text{ sqn rt dip } \leq \text{ sqn (invalidate rt dests) dip} \)"
    by (cases "\( \text{ dests dip} \)" (auto simp add: invalidate_def sqn_def))
qed simp

lemma sqn_invalidate_in_dests [simp]:
fixes dests ipa rsn rt
assumes "\( \text{ dests ipa = Some rsn} \)"
and "\( \text{ ipa } \in \text{ kD(rt)} \)"
shows "\( \text{ sqn (invalidate rt dests) ipa } = \text{ rsn} \)"
unfolding invalidate_def sqn_def
using assms(1) assms(2) [THEN kD_Some]
by clarsimp

lemma dhops_invalidate [simp]:
"\( \forall dip. \text{ the } (\text{dhops (invalidate rt dests) dip}) = \text{ the } (\text{dhops rt dip}) \)"
unfolding invalidate_def by (clarsimp split: option.split)

lemma sqnf_invalidate [simp]:
"\( \forall dip. \text{ sqnf (invalidate rt dests (ξ)) (dests ξ)) dip } = \text{ sqnf (rt ξ) dip} \)"
unfolding sqnf_def invalidate_def by (clarsimp split: option.split)

lemma nhop_invalidate [simp]:
"\( \forall dip. \text{ the } (\text{nhop (invalidate rt dests (ξ)) (dests ξ)) dip } = \text{ the } (\text{nhop (rt ξ) dip}) \)"
unfolding invalidate_def by (clarsimp split: option.split)

lemma invalidate_other [simp]:
fixes rt dests dip
assumes "\( \text{ dip } \not\in \text{ dom(dests)} \)"
shows "\( \text{ invalidate rt dests dip } = rt dip \)"
using assms unfolding invalidate_def
by (clarsimp split: option.split_asm)

lemma invalidate_none [simp]:
  fixes rt dests dip
  assumes "dip \notin kD(rt)"
  shows "invalidate rt dests dip = None"
  using assms unfolding invalidate_def by clarsimp

lemma vD_invalidate_vD_not_dests:
  "dip rt dests dip \notin vD(invalidate rt dests) = dip \notin vD(rt) ∧ dests dip = None"
  unfolding invalidate_def vD_def
  by (clarsimp split: option.split_asm)

lemma sqn_invalidate_not_in_dests [simp]:
  fixes dests dip rt
  assumes "dip \notin dom(dests)"
  shows "sqn (invalidate rt dests) dip = sqn rt dip"
  using assms unfolding sqn_def by simp

lemma invalidate_changes:
  fixes rt dests dip dsn dsk flag hops nhip pre
  assumes "invalidate rt dests dip = Some (dsn, dsk, flag, hops, nhip, pre)"
  shows "dsn = (case dests dip of None ⇒ π_2(the (rt dip)) | Some rsn ⇒ rsn)
    ∧ dsk = π_3(the (rt dip))
    ∧ flag = (if dests dip = None then π_4(the (rt dip)) else inv)
    ∧ hops = π_5(the (rt dip))
    ∧ nhip = π_6(the (rt dip))
    ∧ pre = π_7(the (rt dip))"
  using assms unfolding invalidate_def
  by (cases "rt dip", clarsimp, cases "dests dip") auto

lemma proj3_inv: "dip rt dests dip \notin kD (rt)
  ⇒ π_3(the (invalidate rt dests dip)) = π_3(the (rt dip))"
  by (clarsimp simp: invalidate_def kD_def split: option.split)

lemma dests_iD_invalidate [simp]:
  assumes "dests ip = Some rsn"
  and "ip \notin kD(rt)"
  shows "ip \notin iD(invalidate rt dests)"
  using assms(1) assms(2) [THEN kD_Some] unfolding invalidate_def iD_def
  by (clarsimp split: option.split)

4.1.5 Route Requests

Generate a fresh route request identifier.

definition nrreqid :: "(ip × rreqid) set ⇒ ip ⇒ rreqid"
  where "nrreqid rreqs ip = Max ({n. (ip, n) \in rreqs} \cup \{0\}) + 1"

4.1.6 Queued Packets

Functions for sending data packets.

type_synonym store = "ip ⇒ (p × data list)"

definition sigma_queue :: "store ⇒ ip ⇒ data list" ("\sigma_queue(_,
_’)")
  where "\sigma_queue(store, dip) ≡ case store dip of None ⇒ [] | Some (p, q) ⇒ q"

definition qD :: "store ⇒ ip set"
  where "qD ≡ dom"

definition add :: "data ⇒ ip ⇒ store ⇒ store"
  where "add d dip store ≡ case store dip of
    None ⇒ store (dip ⇒ (req, [d]))"
lemma qD_add [simp]:
  fixes d dip store
  shows "qD(add d dip store) = insert dip (qD store)"
  unfolding add_def Let_def qD_def
  by (clarsimp split: option.split)

definition drop :: "ip ⇒ store → store"
  where "drop dip store ≡
    map_option (λ(p, q). if tl q = [] then store (dip := None)
      else store (dip ↦→ (p, tl q))) (store dip)"

definition sigma_p_flag :: "store ⇒ ip ⇒ p" ("σp-flag'(_, _'')")
  where "σp-flag(store, dip) ≡ map_option fst (store dip)"

definition unsetRRF :: "store ⇒ ip ⇒ store"
  where "unsetRRF store dip ≡
    case store dip of
      None ⇒ store
      | Some (p, q) ⇒ store (dip ↦→ (noreq, q))"

definition setRRF :: "store ⇒ (ip ⇒ sqn) ⇒ store"
  where "setRRF store dests ≡ λdip. if dests dip = None then store dip
    else map_option (λ(_, q). (req, q)) (store dip)"

4.1.7 Comparison with the original technical report

The major differences with the AODV technical report of Fehnker et al are:

1. nhop is partial, thus a 'the' is needed, similarly for dhops and addpreRT.
2. precs is partial.
3. σp-flag(store, dip) is partial.
4. The routing table (rt) is modelled as a map (ip ⇒ r option) rather than a set of 7-tuples, likewise, the r
   is a 6-tuple rather than a 7-tuple, i.e., the destination ip-address (dip) is taken from the argument to the
   function, rather than a part of the result. Well-definedness then follows from the structure of the type
   and more related facts are available automatically, rather than having to be acquired through tedious proofs.
5. Similar remarks hold for the dests mapping passed to invalidate, and store.

end

4.2 AODV protocol messages

theory D_Aodv_Message
imports D_Fwdrreqs
begin

datatype msg =
  Rreq nat rreqid ip sqn k ip sqn ip bool
  | Rrep nat ip sqn ip ip
  | Rerr "ip ⇒ sqn" ip
  | Newpkt data ip
  | Pkt data ip

instantiation msg :: msg
begin

definition newpkt_def [simp]: "newpkt ≡ λ(d, dip). Newpkt d dip"
definition eq_newpkt_def: "eq_newpkt m ≡ case m of Newpkt d dip ⇒ True | _ ⇒ False"

instance by intro_classes (simp add: eq_newpkt_def)
The \textit{msg} type models the different messages used within AODV. The instantiation as a \textit{msg} is a technicality due to the special treatment of \textit{newpkt} messages in the AWN SOS rules. This use of classes allows a clean separation of the AWN-specific definitions and these AODV-specific definitions.

\begin{verbatim}
definition rreq :: \textit{nat} \times \textit{rreqid} \times \textit{ip} \times \textit{sqn} \times \textit{k} \times \textit{ip} \times \textit{sqn} \times \textit{ip} \times \textit{bool} \Rightarrow \textit{msg}
where \textit{rreq} \equiv \lambda(hops, rreqid, dip, dsn, dsk, oip, osn, sip, handled).
Rreq hops rreqid dip dsn dsk oip osn sip handled

lemma rreq_simp [simp]:
"rreq(hops, rreqid, dip, dsn, dsk, oip, osn, sip, handled) = Rreq hops rreqid dip dsn dsk oip osn sip handled"
unfolding rreq_def by simp

definition rrep :: \textit{nat} \times \textit{ip} \times \textit{sqn} \times \textit{ip} \times \textit{ip} \Rightarrow \textit{msg}
where \textit{rrep} \equiv \lambda(hops, dip, dsn, oip, sip). Rrep hops dip dsn oip sip

lemma rrep_simp [simp]:
"rrep(hops, dip, dsn, oip, sip) = Rrep hops dip dsn oip sip"
unfolding rrep_def by simp

definition rerr :: \textit{(ip \rightarrow sqn)} \times \textit{ip} \Rightarrow \textit{msg}
where \textit{rerr} \equiv \lambda(dests, sip). Rerr dests sip

lemma rerr_simp [simp]:
"rerr(dests, sip) = Rerr dests sip"
unfolding rerr_def by simp

lemma not_eq_newpkt_rreq [simp]: "\neg \textit{eq_newpkt} (Rreq hops rreqid dip dsn dsk oip osn sip handled)"
unfolding eq_newpkt_def by simp

lemma not_eq_newpkt_rrep [simp]: "\neg \textit{eq_newpkt} (Rrep hops dip dsn oip sip)"
unfolding eq_newpkt_def by simp

lemma not_eq_newpkt_rerr [simp]: "\neg \textit{eq_newpkt} (Rerr dests sip)"
unfolding eq_newpkt_def by simp

lemma not_eq_newpkt_pkt [simp]: "\neg \textit{eq_newpkt} (Pkt d dip sip)"
unfolding eq_newpkt_def by simp

definition pkt :: \textit{data} \times \textit{ip} \times \textit{ip} \Rightarrow \textit{msg}
where \textit{pkt} \equiv \lambda(d, dip, sip). Pkt d dip sip

lemma pkt_simp [simp]:
"pkt(d, dip, sip) = Pkt d dip sip"
unfolding pkt_def by simp

end
\end{verbatim}

4.3 The AODV protocol

theory D_Aodv
imports D_Aodv_Data D_Aodv_Message
"../../../AWN/AWN_SOS_Labels" "../../../AWN/AWN_Invariants"
begin

4.3.1 Data state

record state =
ip :: "ip"
sn :: "sqn"
rt :: "rt"
rreqs :: "(ip \times rreqid) set"


abbreviation aodv_init :: "ip ⇒ state"
where "aodv_init i ≡ (
    ip = i,
    sn = 1,
    rt = empty,
    rreqs = {},
    store = empty,
    msg = (SOME x. True),
    data = (SOME x. True),
    dests = (SOME x. True),
    pre = (SOME x. True),
    rreqid = (SOME x. True),
    dip = (SOME x. True),
    oip = (SOME x. True),
    hops = (SOME x. True),
    dsn = (SOME x. True),
    dsk = (SOME x. True),
    osn = (SOME x. True),
    sip = (SOME x. x ≠ i),
    handled = (SOME x. True)
)

lemma some_neq_not_eq [simp]: "¬((SOME x :: nat. x ≠ i) = i)"
  by (subst some_eq_ex) (metis zero_neq_numeral)

definition clear_locals :: "state ⇒ state"
where "clear_locals ξ = ξ (\
    msg := (SOME x. True),
    data := (SOME x. True),
    dests := (SOME x. True),
    pre := (SOME x. True),
    rreqid := (SOME x. True),
    dip := (SOME x. True),
    oip := (SOME x. True),
    hops := (SOME x. True),
    dsn := (SOME x. True),
    dsk := (SOME x. True),
    osn := (SOME x. True),
    sip := (SOME x. x ≠ ip ξ),
    handled := (SOME x. True)
)"

lemma clear_locals_sip_not_ip [simp]: "¬(sip (clear_locals ξ) = ip ξ)"
  unfolding clear_locals_def by simp

lemma clear_locals_but_notGlobals [simp]:
  "ip (clear_locals ξ) = ip ξ"
\[ sn (\text{clear_locals} \ \xi) = sn \ \xi \]
\[ rt (\text{clear_locals} \ \xi) = rt \ \xi \]
\[ rreqs (\text{clear_locals} \ \xi) = rreqs \ \xi \]
\[ store (\text{clear_locals} \ \xi) = store \ \xi \]

unfolding \text{clear_locals_def} by auto

4.3.2 Auxilliary message handling definitions

definition \textit{is_newpkt} where
\[ \textit{is_newpkt} \ \xi \equiv \text{case} \ \text{msg} \ \xi \ \text{of} \]
\[ \ \text{Newpkt} \ \text{data'} \ \text{dip'} \Rightarrow \{ \xi[\text{data} := \text{data'}, \ \text{dip} := \text{dip'}] \} \]
\[ \_ \Rightarrow \{\} \]


definition \textit{is_pkt} where
\[ \textit{is_pkt} \ \xi \equiv \text{case} \ \text{msg} \ \xi \ \text{of} \]
\[ \ \text{Pkt} \ \text{data'} \ \text{dip'} \ \text{oip'} \Rightarrow \{ \xi[\text{data} := \text{data'}, \ \text{dip} := \text{dip'}, \ \text{oip} := \text{oip'}] \} \]
\[ \_ \Rightarrow \{\} \]


definition \textit{is_rreq} where
\[ \textit{is_rreq} \ \xi \equiv \text{case} \ \text{msg} \ \xi \ \text{of} \]
\[ \ \text{Rreq} \ \text{hops'} \ \text{rreqid'} \ \text{dip'} \ \text{dsn'} \ \text{dsk'} \ \text{oip'} \ \text{osn'} \ \text{sip'} \ \text{handled'} \Rightarrow \]
\[ \{ \xi[\text{hops} := \text{hops'}, \ \text{rreqid} := \text{rreqid'}, \ \text{dip} := \text{dip'}, \ \text{dsn} := \text{dsn'}, \]
\[ \ \text{dsk} := \text{dsk'}, \ \text{oip} := \text{oip'}, \ \text{osn} := \text{osn'}, \ \text{sip} := \text{sip'}, \]
\[ \ \text{handled} := \text{handled'}] \} \]
\[ \_ \Rightarrow \{\} \]


lemma \textit{is_rreq_asm [dest!]}:
assumes \[ \xi' \in \textit{is_rreq} \ \xi \]
shows \[ (\exists \text{hops} \ \text{rreqid} \ \text{dip} \ \text{dsn} \ \text{dsk} \ \text{oip} \ \text{osn} \ \text{sip} \ \text{handled}). \]
\[ \text{msg} \ \xi = \text{Rreq} \ \text{hops'} \ \text{rreqid'} \ \text{dip'} \ \text{dsn'} \ \text{dsk'} \ \text{oip'} \ \text{osn'} \ \text{sip'} \ \text{handled'} \wedge \]
\[ \xi' = \xi[\text{hops} := \text{hops'}, \ \text{rreqid} := \text{rreqid'}, \ \text{dip} := \text{dip'}, \ \text{dsn} := \text{dsn'}, \]
\[ \ \text{dsk} := \text{dsk'}, \ \text{oip} := \text{oip'}, \ \text{osn} := \text{osn'}, \ \text{sip} := \text{sip'}, \]
\[ \ \text{handled} := \text{handled'}]" 
using \text{assms unfolding} \textit{is_rreq_def} 
by (cases "\text{msg} \ \xi") simp_all 


definition \textit{is_rrep} where
\[ \textit{is_rrep} \ \xi \equiv \text{case} \ \text{msg} \ \xi \ \text{of} \]
\[ \ \text{Rrep} \ \text{hops'} \ \text{dip'} \ \text{dsn'} \ \text{oip'} \ \text{sip'} \Rightarrow \]
\[ \{ \xi[\text{hops} := \text{hops'}, \ \text{dip} := \text{dip'}, \ \text{dsn} := \text{dsn'}, \ \text{oip} := \text{oip'}, \ \text{sip} := \text{sip'}] \} \]
\[ \_ \Rightarrow \{\} \]


lemma \textit{is_rrep_asm [dest!]}:
assumes \[ \xi' \in \textit{is_rrep} \ \xi \]
shows \[ (\exists \text{hops} \ \text{dip} \ \text{dsn} \ \text{oip} \ \text{sip}). \]
\[ \text{msg} \ \xi = \text{Rrep} \ \text{hops'} \ \text{dip'} \ \text{dsn'} \ \text{oip'} \ \text{sip'} \wedge \]
\[ \xi' = \xi[\text{hops} := \text{hops'}, \ \text{dip} := \text{dip'}, \ \text{dsn} := \text{dsn'}, \ \text{oip} := \text{oip'}, \ \text{sip} := \text{sip'}]" 
using \text{assms unfolding} \textit{is_rrep_def} 
by (cases "\text{msg} \ \xi") simp_all 


definition \textit{is_rerr} where
\[ \textit{is_rerr} \ \xi \equiv \text{case} \ \text{msg} \ \xi \ \text{of} \]
\[ \ \text{Rerr} \ \text{dests'} \ \text{sip'} \Rightarrow \{ \xi[\text{dests} := \text{dests'}, \ \text{sip} := \text{sip'}] \} \]
\[ \_ \Rightarrow \{\} \]


lemma \textit{is_rerr_asm [dest!]}:
assumes \[ \xi' \in \textit{is_rerr} \ \xi \]
shows \[ (\exists \text{dests} \ \text{sip}). \]
\[ \text{msg} \ \xi = \text{Rerr} \ \text{dests'} \ \text{sip'} \wedge \]
\[ \xi' = \xi[\text{dests} := \text{dests'}, \ \text{sip} := \text{sip'}]" 
using \text{assms unfolding} \textit{is_rerr_def} 
by (cases "\text{msg} \ \xi") simp_all 

lemmas \textit{is_msg_defs} =
lemma is_msg_inv_ip [simp]:
"ξ' ∈ is_rerr ξ ⇒ ip ξ' = ip ξ"
"ξ' ∈ is_rrep ξ ⇒ ip ξ' = ip ξ"
"ξ' ∈ is_rreq ξ ⇒ ip ξ' = ip ξ"
"ξ' ∈ is_pkt ξ ⇒ ip ξ' = ip ξ"
"ξ' ∈ is_newpkt ξ ⇒ ip ξ' = ip ξ"
unfolding is_msg_defs
by (cases "msg ξ", clarsimp+)+

lemma is_msg_inv_sn [simp]:
"ξ' ∈ is_rerr ξ ⇒ sn ξ' = sn ξ"
"ξ' ∈ is_rrep ξ ⇒ sn ξ' = sn ξ"
"ξ' ∈ is_rreq ξ ⇒ sn ξ' = sn ξ"
"ξ' ∈ is_pkt ξ ⇒ sn ξ' = sn ξ"
"ξ' ∈ is_newpkt ξ ⇒ sn ξ' = sn ξ"
unfolding is_msg_defs
by (cases "msg ξ", clarsimp+)+

lemma is_msg_inv_rt [simp]:
"ξ' ∈ is_rerr ξ ⇒ rt ξ' = rt ξ"
"ξ' ∈ is_rrep ξ ⇒ rt ξ' = rt ξ"
"ξ' ∈ is_rreq ξ ⇒ rt ξ' = rt ξ"
"ξ' ∈ is_pkt ξ ⇒ rt ξ' = rt ξ"
"ξ' ∈ is_newpkt ξ ⇒ rt ξ' = rt ξ"
unfolding is_msg_defs
by (cases "msg ξ", clarsimp+)+

lemma is_msg_inv_rreqs [simp]:
"ξ' ∈ is_rerr ξ ⇒ rreqs ξ' = rreqs ξ"
"ξ' ∈ is_rrep ξ ⇒ rreqs ξ' = rreqs ξ"
"ξ' ∈ is_rreq ξ ⇒ rreqs ξ' = rreqs ξ"
"ξ' ∈ is_pkt ξ ⇒ rreqs ξ' = rreqs ξ"
"ξ' ∈ is_newpkt ξ ⇒ rreqs ξ' = rreqs ξ"
unfolding is_msg_defs
by (cases "msg ξ", clarsimp+)+

lemma is_msg_inv_store [simp]:
"ξ' ∈ is_rerr ξ ⇒ store ξ' = store ξ"
"ξ' ∈ is_rrep ξ ⇒ store ξ' = store ξ"
"ξ' ∈ is_rreq ξ ⇒ store ξ' = store ξ"
"ξ' ∈ is_pkt ξ ⇒ store ξ' = store ξ"
"ξ' ∈ is_newpkt ξ ⇒ store ξ' = store ξ"
unfolding is_msg_defs
by (cases "msg ξ", clarsimp+)+

lemma is_msg_inv_sip [simp]:
"ξ' ∈ is_pkt ξ ⇒ sip ξ' = sip ξ"
"ξ' ∈ is_newpkt ξ ⇒ sip ξ' = sip ξ"
unfolding is_msg_defs
by (cases "msg ξ", clarsimp+)+

4.3.3 The protocol process

datatype pseqp =
  PAodv
| PNewPkt
| PPkt
| PRreq
| PRrep
| PRerr

fun nat_of_seqp :: "pseqp ⇒ nat"
where
"nat_of_seqp PAodv = 1"
| "nat_of_seqp PPkt = 2"
| "nat_of_seqp PNewPkt = 3"
| "nat_of_seqp PRreq = 4"
| "nat_of_seqp PRrep = 5"
| "nat_of_seqp PRerr = 6"

instantiation "pseqp" :: ord
begin
definition less_eq_seqp [iff]: "l1 ≤ l2 = (nat_of_seqp l1 ≤ nat_of_seqp l2)"
definition less_seqp [iff]: "l1 < l2 = (nat_of_seqp l1 < nat_of_seqp l2)"
instance ..
end

abbreviation AODV
where
"AODV ≡ λ_. [clear_locals] call(PAodv)"

abbreviation PKT
where
"PKT args ≡
[ξ. let (data, dip, oip) = args ξ in
 (clear_locals ξ) () data := data, dip := dip, oip := oip ]
call(PPkt)"

abbreviation NEWPKT
where
"NEWPKT args ≡
[ξ. let (data, dip) = args ξ in
 (clear_locals ξ) () data := data, dip := dip ]
call(PNewPkt)"

abbreviation RREQ
where
"RREQ args ≡
[ξ. let (hops, rreqid, dip, dsn, dsk, oip, osn, sip, handled) = args ξ in
 (clear_locals ξ) () hops := hops, rreqid := rreqid, dip := dip,
 dsn := dsn, dsk := dsk, oip := oip,
 osn := osn, sip := sip, handled := handled ]
call(PRreq)"

abbreviation RREP
where
"RREP args ≡
[ξ. let (hops, dip, dsn, oip, sip) = args ξ in
 (clear_locals ξ) () hops := hops, dip := dip, dsn := dsn,
 oip := oip, sip := sip ]
call(PRrep)"

abbreviation RERR
where
"RERR args ≡
[ξ. let (dests, sip) = args ξ in
 (clear_locals ξ) () dests := dests, sip := sip ]
call(PRerr)"

fun Γ_AODV :: "(state, msg, pseqp, pseqp label) seqp_env"
where
"Γ_AODV PAodv = labelled PAodv (receive(λmsg' ξ. ξ () msg := msg')).
 (is_newpkt) NEWPKT(λξ. (data ξ, ip ξ))
 ⊕ (is_pkt) PKT(λξ. (data ξ, dip ξ, oip ξ))
 ⊕ (is_rreq)"
[\xi, \xi \vdash rt := update (rt \xi) (sip \xi) \sigma (0, unk, val, 1, sip \xi, \{\}) \}]
RREQ(\xi. (hops \xi, rreqid \xi, dip \xi, dsn \xi, oip \xi, osn \xi, sip \xi, handled \xi))

(\text{is_rrep})
[\xi, \xi \vdash rt := update (rt \xi) (sip \xi) \sigma (0, unk, val, 1, sip \xi, \{\}) \}]
RREP(\xi. (hops \xi, dip \xi, dsn \xi, oip \xi, sip \xi))

(\text{is_rerr})
[\xi, \xi \vdash rt := update (rt \xi) (sip \xi) \sigma (0, unk, val, 1, sip \xi, \{\}) \}]
RERR(\xi. (dests \xi, sip \xi))

(\text{λξ. } \{\xi \vdash \text{dip} := \text{dip} \} | \text{dip. } \text{dip} \in qD(\text{store} \xi) \cap vD(rt \xi) \})
[\xi, \xi \vdash \text{data} := \text{hd}(\text{seq}(\text{store} \xi, \text{dip} \xi)) \}]
unicast(\xi. \text{the} (\text{nhop} (rt \xi) (\text{dip} \xi)), \xi. \text{pkt}(\text{data} \xi, \text{dip} \xi, \text{ip} \xi)).
[\xi, \xi \vdash \text{store} := \text{the} (\text{dip} \xi) (\text{store} \xi) \}]
AODV()

(\text{λξ. } \{\xi \vdash \text{dests} := (\text{λrip. if} (\text{rip} \in vD (rt \xi) \land \text{nhop} (rt \xi) \text{rip} \neq \text{nhop} (rt \xi) (\text{dip} \xi))
\text {then Some} (\text{inc} (\text{sqn} (rt \xi) \text{rip})) \text {else None} \}) \}]
[\xi, \xi \vdash \text{rt} := \text{invalidate} (\text{rt} \xi) (\text{dests} \xi) \}]
[\xi, \xi \vdash \text{store} := \text{setRRF} (\text{store} \xi) (\text{dests} \xi) \}]
[\xi, \xi \vdash \text{pre} := \bigcup \{ \text{the} (\text{precs} (rt \xi) \text{rip}) \mid \text{rip. } \text{rip} \in \text{dom} (\text{dests} \xi) \} \}]
[\xi, \xi \vdash \text{dests} := (\text{λrip. if} ((\text{dests} \xi) \text{rip} \neq \text{None} \land \text{the} (\text{precs} (rt \xi) \text{rip}) \neq \{\})
\text {then} (\text{dests} \xi) \text{rip} \text {else None} \}) \}]
groupcast(\xi. \text{pre} \xi, \xi. \text{rerr}(\text{dests} \xi, \text{ip} \xi)). \text{AODV}()

(\text{λξ. } \{\xi \vdash \text{dip} := \text{dip} \} | \text{dip. } \text{dip} \in qD(\text{store} \xi) - vD(rt \xi) \land \text{the} (\sigma_{\text{p-flag}}(\text{store} \xi, \text{dip} \xi)) = \text{req} \})
[\xi, \xi \vdash \text{store} := \text{unsetRRF} (\text{store} \xi) (\text{dip} \xi) \}]
[\xi, \xi \vdash \text{sn} := \text{inc} (\text{sn} \xi) \}]
[\xi, \xi \vdash \text{rreqid} := \text{nrreqid} (\text{rreqs} \xi) (\text{ip} \xi) \}]
[\xi, \xi \vdash \text{rreqs} := \text{rreqs} \xi \cup \{(\text{ip} \xi, \text{rreqid} \xi) \}]
broadcast(\xi. \text{rreq}(0, \text{rreqid} \xi, \text{dip} \xi, \text{sqn} (rt \xi) (\text{dip} \xi), \text{sqnf} (rt \xi) (\text{dip} \xi), \text{ip} \xi, \text{sn} \xi, \text{ip} \xi, \text{False})). \text{AODV}();

"\text{Γ_{AODV}}. \text{PNewPkt} = \text{labelled PNewPkt} (\})
(\xi. \text{dip} \xi = \text{ip} \xi)
\text{deliver}(\xi. \text{data} \xi). \text{AODV}();

(\text{λξ. } \{\xi \vdash \text{store} := \text{add} (\text{data} \xi) (\text{dip} \xi) (\text{store} \xi) \})
\text{AODV}();"

"\text{Γ_{AODV}}. \text{PPkt} = \text{labelled PPkt} (\})
(\xi. \text{dip} \xi = \text{ip} \xi)
\text{deliver}(\xi. \text{data} \xi). \text{AODV}();

(\text{λξ. } \{\xi \vdash \text{store} := \text{add} (\text{data} \xi) (\text{dip} \xi) (\text{store} \xi) \})
\text{AODV}();"

(\xi. \text{dip} \xi \in vD (rt \xi))
unicast(\xi. \text{the} (\text{nhop} (rt \xi) (\text{dip} \xi)), \xi. \text{pkt}(\text{data} \xi, \text{dip} \xi, \text{oip} \xi)). \text{AODV}()

(\xi. \text{dip} \xi \neq \text{ip} \xi)
(\{\xi. \text{dip} \xi \in vD (rt \xi)\}
\text{unicast}(\xi. \text{the} (\text{nhop} (rt \xi) (\text{dip} \xi)), \xi. \text{pkt}(\text{data} \xi, \text{dip} \xi, \text{oip} \xi)). \text{AODV}()
"\Gamma_{AODV} \text{ PRreq } = \text{labeled PRreq (}
\langle \xi, (\text{oip } \xi, \text{rreqid } \xi) \in \text{rreqs } \rangle
\text{AODV(})
\text{\textcircled{1}} \text{\langle } \xi, (\text{oip } \xi, \text{rreqid } \xi) \notin \text{rreqs } \text{\rangle}
[\xi, \xi \mid \text{rt := update (rt } \xi) (\text{oip } \xi) (\text{dsn } \xi, \text{kno }, \text{val }, \text{hops } + 1, \text{sip } \xi, \{\}) \text{ })]
[\xi, \xi \mid \text{rreqs := rreqs } \xi \cup \{(\text{oip } \xi, \text{rreqid } \xi)\} \text{ }]$
\text{(}
\langle \xi, \text{handled } \xi = \text{False} \rangle
\text{)}
\text{(}
\langle \xi, \text{dip } \xi = \text{ip } \xi \rangle
[\xi, \xi \mid \text{sn := max (sn } \xi \text{ ) (dsn } \xi \text{ )}]$
\text{unicast(}\lambda \xi . \text{ the (nhop (rt } \xi) (\text{oip } \xi)), \lambda \xi . \text{ rrep(0, dip } \xi, \text{sn } \xi, \text{oip } \xi, \text{ip } \xi))$
\text{broadcast(}\lambda \xi . \text{ rreq(hops } + 1, \text{rreqid } \xi, \text{dip } \xi, \text{dsn } \xi, \text{dsk } \xi, \text{oip } \xi, \text{osn } \xi, \text{ip } \xi, \text{True})).$
\text{AODV(})
\text{>}[\xi, \xi \mid \text{dests := (}\lambda \text{rip}. \text{if (rip } \in \text{vd (rt } \xi) \text{ ) nhop (rt } \xi) \text{ rip = nhop (rt } \xi) (\text{oip } \xi))$
\text{then Some (inc (sqn (rt } \xi) \text{ rip))) else None} \text{ )}]$
[\xi, \xi \mid \text{rt := invalidate (rt } \xi) (\text{dests } \xi)]$
[\xi, \xi \mid \text{store := setRRF (store } \xi) (\text{dests } \xi)]$
[\xi, \xi \mid \text{pre := } \bigcup \{ \text{the (precs (rt } \xi) \text{ rip) } | \text{ rip, rip } \in \text{dom (dests } \xi) \text{ } \} \text{ }]$
[\xi, \xi \mid \text{dests := (}\lambda \text{rip}. \text{if ((dests } \xi) \text{ rip } \neq \text{None } \land \text{the (precs (rt } \xi) \text{ rip) } \neq \{\})$
\text{then (dests } \xi) \text{ rip else None}) \text{ )}]$
\text{groupcast(}\lambda \xi . \text{pre } \xi, \lambda \xi . \text{rerr(dests } \xi, \text{ip } \xi)) \text{.AODV(}]
\text{\textcircled{1}} \text{\langle } \xi, \text{dip } \xi \neq \text{ip } \xi \rangle
\text{(}
\langle \xi, \text{dip } \xi \in \text{vd (rt } \xi) \land \text{dsn } \xi \leq \text{sqn (rt } \xi) (\text{dip } \xi) \land \text{sqnf (rt } \xi) (\text{dip } \xi) = \text{kno})
[\xi, \xi \mid \text{rt := the (addpreRT (rt } \xi) (\text{dip } \xi) \{\text{sip } \xi\} \text{ )}]$
[\xi, \xi \mid \text{store := setRRF (store } \xi) (\text{dests } \xi)]$
[\xi, \xi \mid \text{pre := } \bigcup \{ \text{the (precs (rt } \xi) \text{ rip) } | \text{ rip, rip } \in \text{dom (dests } \xi) \text{ } \} \text{ }]$
[\xi, \xi \mid \text{dests := (}\lambda \text{rip}. \text{if ((dests } \xi) \text{ rip } \neq \text{None } \land \text{the (precs (rt } \xi) \text{ rip) } \neq \{\})$
\text{then (dests } \xi) \text{ rip else None}) \text{ )}]$
\text{groupcast(}\lambda \xi . \text{pre } \xi, \lambda \xi . \text{rerr(dests } \xi, \text{ip } \xi)) \text{.AODV(}]
\text{\textcircled{1}} \text{\langle } \xi, \text{dip } \xi \notin \text{vd (rt } \xi) \lor \text{sqn (rt } \xi) (\text{dip } \xi) < \text{dsn } \xi \lor \text{sqnf (rt } \xi) (\text{dip } \xi) = \text{unk}\rangle$
\text{broadcast(}\lambda \xi . \text{rreq(hops } + 1, \text{rreqid } \xi, \text{dip } \xi, \text{dsn } \xi, \text{dsk } \xi, \text{oip } \xi, \text{osn } \xi, \text{ip } \xi, \text{True})).$
\text{AODV(})$
\text{)}$
\text{)}$
\text{\textcircled{1}} \text{\langle } \xi, \text{handled } \xi = \text{True} \rangle$
\text{broadcast(}\lambda \xi . \text{rreq(hops } + 1, \text{rreqid } \xi, \text{dip } \xi, \text{dsn } \xi, \text{dsk } \xi, \text{oip } \xi, \text{osn } \xi, \text{ip } \xi, \text{True})).$
\text{AODV(})$
\text{)}$
\text{\textcircled{1}} \text{\langle } \xi, \text{rt } \xi \neq \text{update (rt } \xi) (\text{dip } \xi) (\text{dsn } \xi, \text{kno }, \text{val }, \text{hops } + 1, \text{sip } \xi, \{\}) \text{ ) }$
\text{(}
\langle \xi, \xi \mid \text{rt := update (rt } \xi) (\text{dip } \xi) (\text{dsn } \xi, \text{kno }, \text{val }, \text{hops } + 1, \text{sip } \xi, \{\}) \text{ })\text{ ]}
\text{(}
\langle \xi, \text{oip } \xi = \text{ip } \xi \rangle
\text{AODV(})$
\text{\textcircled{1}} \text{\langle } \xi, \text{oip } \xi \neq \text{ip } \xi \rangle
\text{(}
\langle \xi, \text{oip } \xi \in \text{vd (rt } \xi) \rangle
\text{[}\xi, \xi \mid \text{rt := the (addpreRT (rt } \xi) (\text{dip } \xi) \{\text{the (nhop (rt } \xi) (\text{oip } \xi))\} \text{ )}]\text{ ]}\text{ )}
lemma Γ. ξ (| rt := the (addpreRT (rt ξ) (the (nhop (rt ξ) (dip ξ))))
{the (nhop (rt ξ) (oip ξ))) |]
unicast(λξ. the (nhop (rt ξ) (oip ξ))), \(\lambda\). rrep(hops ξ + 1, dip ξ, dns ξ, oip ξ, ip ξ))}.
AODV()

> Γ. ξ (| dests := (λrip. if (rip \in vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (oip ξ))
then Some (inc (sqn (rt ξ) rip)) else None) |]
Γ. ξ (| store := setRRF (store ξ) (dests ξ) |]
Γ. ξ (| pre := \{ the (precs (rt ξ) rip) | rip. rip \in dom (dests ξ) \} |]
Γ. ξ (| dests := (λrip. if ((dests ξ) rip \neq None ∧ the (precs (rt ξ) rip) \neq {})
then (dests ξ) rip else None) |]
groupcast(ξ. pre ξ, ξ. rerr(dests ξ, ip ξ)).AODV()

ξ. oip ξ \notin vD (rt ξ))
AODV()

)"

"Γ_{AODV} \text{PRerr} = labelled \text{PRerr} \{
\begin{align*}
| & \text{dests} := (\lambda\text{rip}. \text{case} (\text{dests} \xi) \text{rip of None} \Rightarrow \text{None})
\text{ Some rsn} \Rightarrow \text{if} \text{rip} \in vD (rt \xi) \land \text{the} \text{nhop} (rt \xi) \text{rip} = \text{sip} \xi
\land \text{sqn} (rt \xi) \text{rip} < \text{rsn} \text{then} \text{Some} \text{rsn} \text{else} \text{None}) |] \\
| & \text{store} := \text{setRRF} (\text{store} \xi) (\text{dests} \xi) |]
| & \text{pre} := \{ \text{the} \text{precs} (\text{rt} \xi) \text{rip} \mid \text{rip} \text{. rip} \in \text{dom} (\text{dests} \xi) \} |]
| & \text{dests} := (\lambda\text{rip}. \text{if} ((\text{dests} \xi) \text{rip} \neq \text{None} \land \text{the} \text{precs} (\text{rt} \xi) \text{rip} \neq \{\})
\text{then} (\text{dests} \xi) \text{rip} \text{else} \text{None}) |]
\end{align*}
groupcast(\lambda\xi. \text{pre} \xi, \lambda\xi. \text{rerr}(\text{dests} \xi, \text{ip} \xi)). \text{AODV}()"

declare \(\Gamma_{AODV}.\text{simps} [\text{simp del, code del}]
\text{lemmas} \(\Gamma_{AODV}.\text{simps} [\text{simp}, \text{code}] = \Gamma_{AODV}.\text{simps} [\text{simplified}]
\n\text{fun} \\Gamma_{AODV}.\text{skeleton}
\text{where}
\begin{align*}
\Gamma_{AODV}.\text{skeleton} \text{PAodv} &= \text{seqp_skeleton} (\Gamma_{AODV} \text{PAodv}) \\
\Gamma_{AODV}.\text{skeleton} \text{PNewPkt} &= \text{seqp_skeleton} (\Gamma_{AODV} \text{PNewPkt}) \\
\Gamma_{AODV}.\text{skeleton} \text{PPkt} &= \text{seqp_skeleton} (\Gamma_{AODV} \text{PPkt}) \\
\Gamma_{AODV}.\text{skeleton} \text{PRreq} &= \text{seqp_skeleton} (\Gamma_{AODV} \text{PRreq}) \\
\Gamma_{AODV}.\text{skeleton} \text{PRrep} &= \text{seqp_skeleton} (\Gamma_{AODV} \text{PRrep}) \\
\Gamma_{AODV}.\text{skeleton} \text{PRerr} &= \text{seqp_skeleton} (\Gamma_{AODV} \text{PRerr})
\end{align*}
\n\text{lemma} \(\Gamma_{AODV}.\text{skeleton}_\text{wf} [\text{simp}]:
\begin{align*}
\text{wellformed} \Gamma_{AODV}.\text{skeleton}"
\text{proof (rule, intro allI)}
\text{fix} \text{pn \text{pn'}}
\text{show "call(pn') \notin \text{stermsl} (\Gamma_{AODV} \text{skeleton} pn)"}
\text{by (cases pn) simp_all}
\text{qed}
\end{align*}
\n\text{declare} \(\Gamma_{AODV}.\text{skeleton}_\text{simps} [\text{simp del, code del}]
\text{lemmas} \(\Gamma_{AODV}.\text{skeleton}_\text{simps} [\text{simp}, \text{code}]
\text{= \Gamma_{AODV}.\text{skeleton}_\text{simps} [\text{simplified} \Gamma_{AODV}.\text{simps seqp_skeleton.simps]}
\n\text{lemma} \text{aodv.proc_cases [dest]:}
\text{fixes} \text{p \text{pn}}
\text{shows "p \in \text{ctermsl} (\Gamma_{AODV} \text{pn}) \implies"
\begin{align*}
p \in \text{ctermsl} (\Gamma_{AODV} \text{PAodv}) \lor \\
p \in \text{ctermsl} (\Gamma_{AODV} \text{PNewPkt}) \lor \\
p \in \text{ctermsl} (\Gamma_{AODV} \text{PPkt}) \lor \\
p \in \text{ctermsl} (\Gamma_{AODV} \text{PRreq}) \lor
\end{align*}
\text{show "call(pn') \notin \text{stermsl} (\Gamma_{AODV} \text{pn})"}
\text{by (cases pn) simp_all}
\text{qed}
\[ p \in \text{ctermsl} (\Gamma_{AODV} \text{PRrep}) \lor \\
\quad p \in \text{ctermsl} (\Gamma_{AODV} \text{PRerr}) \]

using `assms`
by (cases \( p \)) simp_all

---

**Definition** \( \sigma_{AODV} \) :
\[ \text{"ip} \Rightarrow (\text{state} \times (\text{state}, \text{msg}, \text{pseqp}, \text{pseqp label}) \text{ seqp}) \text{ set} \]
where \( \sigma_{AODV} i \equiv \{(\text{aodv_init} i, \Gamma_{AODV} \text{PAodv})\} \)

**Abbreviation** \( \text{paodv} \) :
\[ \text{"ip} \Rightarrow (\text{state} \times (\text{state}, \text{msg}, \text{pseqp}, \text{pseqp label}) \text{ seqp}, \text{msg seq_action}) \text{ automaton} \]
where
\[ \text{paodv} i \equiv (| \text{init} = \sigma_{AODV} i, \text{trans} = \text{seqp sos} \Gamma_{AODV} |) \]

**Lemma** \( \text{aodv_trans} \) :
\[ \text{trans} \ (\text{paodv} i) = \text{seqp sos} \Gamma_{AODV} \]
by simp

**Lemma** \( \text{aodv_control_within} \) :
\[ \text{control_within} \ (\Gamma_{AODV} \text{init} \ (\text{paodv} i)) \]

unfolding \( \sigma_{AODV} \_\text{def} \) by (rule control_withinI) (auto simp del: \( \Gamma_{AODV} \_\text{simps} \))

---

**Lemma** \( \text{aodv_wf} \) :
\[ \text{wellformed} \ \Gamma_{AODV} \]
proof (rule, intro allI)
fix \( \text{pn} \ \text{pn'} \)
show \( \text{"call} (\text{pn'}) \notin \text{stermsl} (\Gamma_{AODV} \text{pn})" \)
by (cases \( \text{pn} \)) simp_all
qed

**Lemmas** \( \text{aodv_labels_not_empty [simp]} = \text{labels_not_empty [OF aodv_wf]} \)

**Lemma** \( \text{aodv_ex_label [intro]} \) :
\[ \exists \ l. \ l \in \text{labels} \ (\Gamma_{AODV} \ p) \]
by (metis aodv_labels_not_empty all_not_in_conv)

**Lemma** \( \text{aodv_ex_labelE [elim]} \) :
assumes \( \forall l \in \text{labels} \ (\Gamma_{AODV} \ p) \ L \ L \ p \)
and \( \exists p \ l. \ L \ L \ p \Longrightarrow Q \)
shows \( Q \)
using assms by (metis aodv_ex_label)

**Lemma** \( \text{aodv_simple_labels [simp]} \) :
\[ \text{simple_labels} \ (\Gamma_{AODV}) \]
proof
fix \( \text{pn} \ \text{p} \)
assume \( \text{"p}\in\text{subterms}(\Gamma_{AODV} \text{pn})" \)
thus \( \exists l. \ \text{labels} \ (\Gamma_{AODV} \ p) = \{l\} \)
by (cases \( \text{pn} \)) (simp_all cong: seqp_congs | elim disjE)+
qed

**Lemma** \( \sigma_{AODV} \_\text{labels [simp]} \) :
\[ (\xi, p) \in \sigma_{AODV} \Rightarrow \text{labels} \ (\Gamma_{AODV} \ p) \ = \ \{\text{PAodv}\_0\} \]
unfolding \( \sigma_{AODV} \_\text{def} \) by simp

**Lemma** \( \text{aodv_init_kD_empty [simp]} \) :
\[ (\xi, p) \in \sigma_{AODV} \Rightarrow \text{kD} \ (\text{rt} \ (\xi)) = \{\} \]
unfolding \( \sigma_{AODV} \_\text{def} \) kD_def by simp

**Lemma** \( \text{aodv_init_sip_not_ip [simp]} \) :
\[ \neg (\text{sip} \ (\text{aodv_init} i) = i) \]
by simp

**Lemma** \( \text{aodv_init_sip_not_ip'} [simp] \) :
assumes \( \"(\xi, p) \in \sigma_{AODV} \" \)
shows \( \text{"sip \ \xi \ \neq \ ip \ \xi"} \)
using assms unfolding \( \sigma_{AODV} \_\text{def} \) by simp

**Lemma** \( \text{aodv_init_sip_not_i [simp]} \) :
assumes \( \"(\xi, p) \in \sigma_{AODV} \" \)
shows \( \text{"sip \ \xi \ \neq \ i"} \)
using assms unfolding \( \sigma_{AODV} \_\text{def} \) by simp

356
lemma clear_locals_sip_not_ip':
assumes "ip ξ = i"
shows "¬(sip (clear_locals ξ) = i)"
using asms by auto
Stop the simplifier from descending into process terms.

declare seq_congs [cong]
Configure the main invariant tactic for AODV.

declare Γ_AODV.simps [cterm_env]
aodv_proc_cases [cterm_cases]
seq_invariant_ctermsI [OF aodv_wf aodv_control_within aodv_simple_labels aodv_trans,
cterm_intros]
seq_step_invariant_ctermsI [OF aodv_wf aodv_control_within aodv_simple_labels aodv_trans,
cterm_intros]
end

4.4 Invariant assumptions and properties

theory D_Aodv_Predicates
imports D_Aodv
begin
Definitions for expression assumptions on incoming messages and properties of outgoing messages.

abbreviation not_Pkt :: "msg ⇒ bool"
where "not_Pkt m ≡ case m of Pkt _ _ _ = False | _ ⇒ True"

definition msg_sender :: "msg ⇒ ip"
where "msg_sender m ≡ case m of Rreq _ _ _ _ _ _ _ ipc _ ⇒ ipc

| Rrep _ _ _ ipc ⇒ ipc
| Rerr _ ipc ⇒ ipc
| Pkt _ _ ipc ⇒ ipc"

lemma msg_sender_simps [simp]:
"∀ hops rreqid dip dsn dsk oip osn sip handled.
   msg_sender (Rreq hops rreqid dip dsn dsk oip osn sip handled) = sip"
"∀ hops dip dsn oip sip. msg_sender (Rrep hops dip dsn oip sip) = sip"
"∀ dests sip. msg_sender (Rerr dests sip) = sip"
"∀ d dip sip. msg_sender (Pkt d dip sip) = sip"
unfolding msg_sender_def by simp_all

definition msg_zhops :: "msg ⇒ bool"
where "msg_zhops m ≡ case m of
   Rreq hopsc _ dipc _ _ oipc _ _ sipc _ ⇒ hopsc = 0 → oipc = sipc
   | Rrep hopsc dipc _ _ sipc _ _ sipc ⇒ hopsc = 0 → dipc = sipc
   | _ ⇒ True"

lemma msg_zhops_simps [simp]:
"∀ hops rreqid dip dsn dsk oip osn sip handled.
   msg_zhops (Rreq hops rreqid dip dsn dsk oip osn sip handled) = (hops = 0 → oip = sip)"
"∀ hops dip dsn oip sip. msg_zhops (Rrep hops dip dsn oip sip) = (hops = 0 → dip = sip)"
"∀ dests sip. msg_zhops (Rerr dests sip) = True"
"∀ d dip. msg_zhops (Newpkt d dip) = True"
"∀ d dip sip. msg_zhops (Pkt d dip sip) = True"
unfolding msg_zhops_def by simp_all

definition rreq_rrep_sn :: "msg ⇒ bool"
where "rreq_rrep_sn m ≡ case m of
   Rreq _ _ _ _ _ _ _ osnc _ _ ⇒ osnc ≥ 1
   | Rrep _ _ dsnc _ _ ⇒ dsnc ≥ 1
   | _ ⇒ True"
lemma rreq_rrep_sn_simps [simp]:
"\(\forall\) hops rreqid dip dsn dsk oip osn sip handled.
\(\text{rreq\_rrep\_sn}\ (Rreq\ hops\ rreqid\ dip\ dsn\ osn\ sip\ handled) = (osn \geq 1)\)"
"\(\forall\) hops dip dsn osn sip. \(\text{rreq\_rrep\_sn}\ (Rrep\ hops\ dip\ dsn\ oip\ sip) = (dsn \geq 1)\)"
"\(\forall\) dests sip. \(\text{rreq\_rrep\_sn}\ (Rerr\ dests\ sip) = True\)"
"\(\forall\) d dip. \(\text{rreq\_rrep\_sn}\ (Newpkt\ d\ dip) = True\)"
"\(\forall\) d dip sip. \(\text{rreq\_rrep\_sn}\ (Pkt\ d\ dip\ sip) = True\)"

unfolding rreq_rrep_sn_def by simp_all

definition rreq_rrep_fresh :: "rt \Rightarrow msg \Rightarrow bool"
where "rreq_rrep_fresh\ crt\ m \equiv \text{case}\ m\ \text{of} Rreq\ hopsc\ _\ _\ _\ _\ oipc\ osnc\ ipcc\ _\ \Rightarrow (ipcc\ \neq\ oipc\ \rightarrow\ oipc\ \in kD(crt)\ \land\ (sqn\ crt\ oip\ >\ osn \lor\ (sqn\ crt\ oip\ =\ osn\ \land\ the\ (dhops\ crt\ oip)\ \leq\ hops)\ \land\ the\ (flag\ crt\ oip)\ =\ val)\)))
\text{| Rrep\ hopsc\ dipc\ dsnc\ _\ ipcc\ \Rightarrow (sip\ \neq\ dip\ \rightarrow\ dipc\ \in kD(crt)\ \land\ sqn\ crt\ dipc\ =\ dsnc\ \land\ the\ (dhops\ crt\ dipc)\ =\ hopsc\ \land\ the\ (flag\ crt\ dipc)\ =\ val)\)}
\text{| _\ \Rightarrow True}\"

lemma rreq_rrep_fresh [simp]:
"\(\forall\) hops rreqid dip dsn dsk oip osn sip handled.
\(\text{rreq\_rrep\_fresh}\ (Rreq\ hops\ rreqid\ dip\ dsn\ osn\ sip\ handled) = (sip \neq\ oip\ \rightarrow\ oip\ \in kD(crt)\ \land\ (sqn\ crt\ oip\ >\ osn \lor\ (sqn\ crt\ oip\ =\ osn\ \land\ the\ (dhops\ crt\ oip)\ \leq\ hops)\ \land\ the\ (flag\ crt\ oip)\ =\ val)))"
"\(\forall\) hops dip dsn osn sip. \(\text{rreq\_rrep\_fresh}\ (Rrep\ hops\ dip\ dsn\ oip\ sip) = (sip \neq\ dip\ \rightarrow\ dipc\ \in kD(crt)\ \land\ sqn\ crt\ dipc\ =\ dsnc\ \land\ the\ (dhops\ crt\ dipc)\ =\ hopsc\ \land\ the\ (flag\ crt\ dipc)\ =\ val)\))"
"\(\forall\) dests sip. \(\text{rreq\_rrep\_fresh}\ (Rerr\ dests\ sip) = True\)"
"\(\forall\) d dip. \(\text{rreq\_rrep\_fresh}\ (Newpkt\ d\ dip) = True\)"
"\(\forall\) d dip sip. \(\text{rreq\_rrep\_fresh}\ (Pkt\ d\ dip\ sip) = True\)"

unfolding rreq_rrep_fresh_def by simp_all

definition rerr_invalid :: "rt \Rightarrow msg \Rightarrow bool"
where "rerr_invalid\ crt\ m \equiv \text{case}\ m\ \text{of} Rerr\ destsc\ _\ \Rightarrow (\forall\ ripc\ \in\ \text{dom}(destsc). (ripc\ \in\ iD(crt)\ \land\ the\ (destsc\ ripc)\ =\ sqn\ crt\ ripc))\)
\text{| _\ \Rightarrow True}\"

lemma rerr_invalid [simp]:
"\(\forall\) hops rreqid dip dsn dsk oip osn sip handled.
\(\text{rerr\_invalid}\ (Rreq\ hops\ rreqid\ dip\ dsn\ osn\ sip\ handled) = True\)"
"\(\forall\) hops dip dsn osn sip. \(\text{rerr\_invalid}\ (Rrep\ hops\ dip\ dsn\ oip\ sip) = True\)"
"\(\forall\) dests sip. \(\text{rerr\_invalid}\ (Rerr\ dests\ sip) = (\forall\ ripc\ \in\ \text{dom}(dests). (ripc\ \in\ iD(crt)\ \land\ \text{the}\ (dests\ ripc)\ =\ sqn\ crt\ ripc))\)"
"\(\forall\) d dip. \(\text{rerr\_invalid}\ (Newpkt\ d\ dip) = True\)"
"\(\forall\) d dip sip. \(\text{rerr\_invalid}\ (Pkt\ d\ dip\ sip) = True\)"

unfolding rerr_invalid_def by simp_all

definition initmissing :: "(nat \Rightarrow\ state\ option) \times 'a \Rightarrow (nat \Rightarrow\ state) \times 'a"
where "initmissing\ \sigma = (\lambda i. \text{case}\ (\text{fst}\ \sigma)\ i\ \text{of}\ None\ \Rightarrow\ \text{aodv\_init}\ i\ |\ \text{Some}\ s\ \Rightarrow\ s,\ \text{snd}\ \sigma)"

lemma not_in_net_ips_fst_init_missing [simp]:
assumes "i \notin\ \text{net\_ips}\ \sigma"
sshows "\text{fst}\ (\text{initmissing}\ (\text{net\_map}\ \text{fst}\ \sigma))\ i = \text{aodv\_init}\ i"

358
using assms unfolding initmissing_def by simp

lemma fst_initmissing_netgmap_pair_fst [simp]:
  "fst (initmissing (netgmap (\(p, q). (fst (id p), snd (id p), q)) s))
  = fst (initmissing (netgmap fst s))"

unfolding initmissing_def by auto

We introduce a streamlined alternative to initmissing with netgmap to simplify invariant statements and thus facilitate their comprehension and presentation.

lemma fst_initmissing_netgmap_default_aodv_init_netlift:
  "fst (initmissing (netgmap fst s)) = default aodv_init (netlift fst s)"

unfolding initmissing_def default_def
by (simp add: fst_netgmap_netlift del: One_nat_def)

definition netglobal :: "((nat ⇒ state) ⇒ bool) ⇒ ((state × 'b) × 'c) net_state ⇒ bool"
where
"netglobal P ≡ (\s. P (default aodv_init (netlift fst s)))"

end

4.5 Quality relations between routes

theory D_Fresher
imports D_Aodv_Data
begin

4.5.1 Net sequence numbers

On individual routes

definition nsqn_r :: "r ⇒ sqn"
where
"nsqn, r ≡ if \(π₄(r) = val \lor \(π₂(r) = 0 \) then \(π₂(r)\) else (π₂(r) - 1)"

lemma nsqn_r_def':
  "nsqn, r = (if \(π₄(r) = inv \) then \(π₂(r)\) - 1 else \(π₂(r)\))"

unfolding nsqn_r_def by simp

lemma nsqn_r_zero [simp]:
  "\(dsn dsk flag hops nhip pre. nsqn_r (0, dsk, flag, hops, nhip, pre) = 0)"

unfolding nsqn_r_def by clarsimp

lemma nsqn_r_val [simp]:
  "\(dsn dsk hops nhip pre. nsqn_r (dsn, dsk, val, hops, nhip, pre) = dsn)"

unfolding nsqn_r_def by clarsimp

lemma nsqn_r_inv [simp]:
  "\(dsn dsk hops nhip pre. nsqn_r (dsn, dsk, inv, hops, nhip, pre) = dsn - 1)"

unfolding nsqn_r_def by clarsimp

lemma nsqn_r_lte_dsn [simp]:
  "\(dsn dsk flag hops nhip pre. nsqn_r (dsn, dsk, flag, hops, nhip, pre) ≤ dsn)"

unfolding nsqn_r_def by clarsimp

On routes in routing tables

definition nsqn :: "rt ⇒ ip ⇒ sqn"
where
"nsqn ≡ λrt dip. case σ_route(rt, dip) of None ⇒ 0 | Some r ⇒ nsqn_r(r)"

lemma nsqn_sqn_def:
lemma not_in_kD_nsqn [simp]:
assumes "dip ∉ kD(rt)"
shows "nsqn rt dip = 0"
using assms unfolding nsqn_def by simp

lemma kD_nsqn:
assumes "dip ∈ kD(rt)"
shows "nsqn rt dip = nsqn (the (σroute(rt, dip)))"
using assms [THEN kD_Some] unfolding nsqn_def by clarsimp

lemma nsqn_r_flag_pred [simp, intro]:
fixes dsn dsk flag hops nhip pre
assumes "P (nsqn_r (dsn, dsk, val, hops, nhip, pre))"
and "P (nsqn_r (dsn, dsk, inv, hops, nhip, pre))"
shows "P (nsqn_r (dsn, dsk, flag, hops, nhip, pre))"
using assms by (cases flag) auto

lemma nsqn_r_addpreRT_inv [simp]:
"∀rt dip npre dip'. dip ∈ kD(rt) =⇒ nsqn_r (the (the (addpreRT rt dip npre) dip')) = nsqn_r (the (rt dip'))"
unfolding addpreRT_def nsqn_r_def
by (frule kD_Some) (clarsimp split: option.split)

lemma sqn_nsqn:
"∀rt dip. sqn rt dip - 1 ≤ nsqn rt dip"
unfolding sqn_def nsqn_def by (clarsimp split: option.split)

lemma nsqn_sqn: "nsqn rt dip ≤ sqn rt dip"
unfolding sqn_def nsqn_def by (cases rt dip) auto

lemma val_nsqn_sqn [elim, simp]:
assumes "ip ∈ kD(rt)"
and "the (flag rt ip) = val"
shows "nsqn rt ip = sqn rt ip"
using assms unfolding nsqn_sqn_def by auto

lemma vD_nsqn_sqn [elim, simp]:
assumes "ip ∈ vD(rt)"
shows "nsqn rt ip = sqn rt ip"
proof -
from ⟨ip ∈ vD(rt)⟩ have "ip ∈ kD(rt)"
and "the (flag rt ip) = val" by auto
thus ?thesis ..
qed

lemma inv_nsqn_sqn [elim, simp]:
assumes "ip ∈ kD(rt)"
and "the (flag rt ip) = inv"
shows "nsqn rt ip = sqn rt ip - 1"
using assms unfolding nsqn_sqn_def by auto

lemma iD_nsqn_sqn [elim, simp]:
assumes "ip ∈ iD(rt)"
shows "nsqn rt ip = sqn rt ip - 1"
proof -
from ⟨ip ∈ iD(rt)⟩ have "ip ∈ kD(rt)"
and "the (flag rt ip) = inv" by auto
thus ?thesis ..
qed
lemma nsqn_update_changed_kno_val [simp]: \[ rt \neq update rt ip (dsn, kno, val, hops, nhip, \{} \] \[ \Rightarrow nsqn (update rt ip (dsn, kno, val, hops, nhip, \{})) ip = dsn \] 
using assms unfolding nsqn_def update_def 
by (clarsimp simp: kD_nsqn split: option.split_asm option.split split_if_asm) 
     (metis fun_upd_triv) 

lemma nsqn_addpreRT_inv [simp]: 
\[ rt \neq dip npre dip'. dip \in kD(rt) \Rightarrow nsqn (the (addpreRT rt dip npre)) dip' = nsqn rt dip' \] 
unfolding addpreRT_def nsqn_def nsqn_r_def 
by (frule kD_Some) (clarsimp split: option.split) 

lemma nsqn_update_other [simp]: 
fixes dsn dsk flag hops dip nhip pre rt ip 
assumes \[ dip \neq ip \] 
shows "nsqn (update rt ip (dsn, dsk, flag, hops, nhip, pre)) dip = nsqn rt dip" 
using assms unfolding nsqn_def 
by (clarsimp split: option.split) 

lemma nsqn_invalidate_eq: 
assumes \[ dip \in kD(rt) \] 
and \[ dests dip = Some rsn \] 
shows "nsqn (invalidate rt dests) dip = rsn - 1" 
using assms 
proof - 
from assms obtain dsk hops nhip pre 
where "invalidate rt dests dip = Some (rsn, dsk, inv, hops, nhip, pre)" 
unfolding invalidate_def 
by auto 
moreover from \[ dip \in kD(rt) \] have "dip \in kD(invalidate rt dests)" by simp 
ultimately show \[ \] 
using \[ dests dip = Some rsn \] by simp 
qed 

lemma nsqn_invalidate_other [simp]: 
assumes \[ dip \in kD(rt) \] 
and \[ dip \notin dom dests \] 
shows "nsqn (invalidate rt dests) dip = nsqn rt dip" 
using assms by (clarsimp simp add: kD_nsqn) 

4.5.2 Comparing routes

definition 
fresher :: "r \Rightarrow r \Rightarrow bool" ("(_/ ⊑_/ _)" [51, 51] 50) 
where 
"fresher r r' \equiv ((nsqn, r < nsqn, r') \lor (nsqn, r = nsqn, r' \land \pi_5(r) \geq \pi_5(r')))"

lemma fresherI1 [intro]: 
assumes "nsqn, r < nsqn, r'" 
shows "r \sqsubseteq r'" 
unfolding fresher_def using assms by simp 

lemma fresherI2 [intro]: 
assumes "nsqn, r = nsqn, r'" 
and \[ \pi_5(r) \geq \pi_5(r') \] 
shows "r \sqsubseteq r'" 
unfolding fresher_def using assms by simp 

lemma fresherI [intro]: 
assumes "(nsqn, r < nsqn, r') \lor (nsqn, r = nsqn, r' \land \pi_5(r) \geq \pi_5(r'))" 
shows "r \sqsubseteq r'" 
unfolding fresher_def using assms .
4.5.3 Comparing routing tables

definition
  rt_fresher :: "ip ⇒ rt ⇒ rt ⇒ bool"
where
  "rt_fresher ≡ λ dip rt1 rt2'. (the (σ_route(rt1, dip))) ⊑ (the (σ_route(rt2', dip)))"

abbreviation
  rt_fresher_syn :: "rt ⇒ ip ⇒ rt ⇒ bool" ("(_/ ⊑_/ _)" [51, 999, 51] 50)
where
  "rt1 ⊑_1 rt2 ≡ rt_fresher ip rt1 rt2"

lemma rt_fresherI1 [intro]:
  assumes "dip ∈ kD(rt1)"
  and "dip ∈ kD(rt2)"
  shows "(rt1 ⊑_1 rt2) = (nsqn rt1 dip < nsqn rt2 dip
            ∨ (nsqn rt1 dip = nsqn rt2 dip
            ∧ the (dhops rt1 dip) ≥ the (dhops rt2 dip)))"
  using assms unfolding rt_fresher_def fresher_def
  by (simp add: kD_nsqn proj5_eq_dhops)
and "nsqn rt1 dip < nsqn rt2 dip"
shows "rt1 ⊑_dip rt2"
unfolding rt_fresher_def2 [OF assms(1-2)] using assms(3) by simp

lemma rt_fresherI2 [intro]:
assumes "dip ∈ kD(rt1)"
and "dip ∈ kD(rt2)"
and "nsqn rt1 dip = nsqn rt2 dip"
and "the (dhops rt1 dip) ≥ the (dhops rt2 dip)"
shows "rt1 ⊑_dip rt2"
unfolding rt_fresher_def2 [OF assms(1-2)] using assms(3) by simp

lemma rt_fresherE [elim]:
assumes "rt1 ⊑_dip rt2"
and "dip ∈ kD(rt1)"
and "dip ∈ kD(rt2)"
and "\[ \[ nsqn rt1 dip < nsqn rt2 dip \] \implies P rt1 rt2 dip \]
and "\[ \[ nsqn rt1 dip = nsqn rt2 dip; \]
the (dhops rt1 dip) ≥ the (dhops rt2 dip) \] \implies P rt1 rt2 dip"
shows "P rt1 rt2 dip"
using assms(1) unfolding rt_fresher_def2 [OF assms(2-3)]
using assms(4-5) by auto

lemma rt_fresher_refl [simp]: "rt ⊑_dip rt"
unfolding rt_fresher_def by simp

lemma rt_fresher_trans [elim, trans]:
assumes "rt1 ⊑_dip rt2"
and "rt2 ⊑_dip rt3"
shows "rt1 ⊑_dip rt3"
using assms unfolding rt_fresher_def by auto

lemma rt_fresher_if_Some [intro!]:
assumes "the (rt dip) ⊑ r"
shows "rt ⊑_dip (λip. if ip = dip then Some r else rt ip)"
using assms unfolding rt_fresher_def by simp

definition rt_fresh_as :: "ip ⇒ rt ⇒ rt ⇒ bool"
where "rt_fresh_as ≡ λdip rt1 rt2. (rt1 ⊑_dip rt2) ∧ (rt2 ⊑_dip rt1)"

abbreviation rt_fresh_as_syn :: "rt ⇒ ip ⇒ rt ⇒ bool" ("(_/ ≈ _)") [51, 999, 51] 50
where "rt1 ≈_i rt2 ≡ rt_fresh_as i rt1 rt2"

lemma rt_fresh_as_refl [simp]: "∀rt dip. rt ≈_dip rt"
unfolding rt_fresh_as_def by simp

lemma rt_fresh_as_trans [simp, intro, trans]:
"∀rt1 rt2 rt3 dip. [ rt1 ≈_dip rt2; rt2 ≈_dip rt3 ] \implies rt1 ≈_dip rt3"
unfolding rt_fresh_as_def rt_fresher_def
by (metis (mono_tags) fresher_trans)

lemma rt_fresh_asI [intro!]:
assumes "rt1 ⊑_dip rt2"
and "rt2 ⊑_dip rt1"
shows "rt1 ≈_dip rt2"
using assms unfolding rt_fresh_as_def by simp

lemma rt_fresh_as_fresherI [intro]:
assumes "dip ∈ kD(rt1)"
and "dip ∈ kD(rt2)"
and "the (rt1 dip) ⊑ the (rt2 dip)"
and "the (rt2 dip) ⊑ the (rt1 dip)"
shows "rt1 ≈_{dip} rt2"

using assms unfolding rt_fresh_as_def
by (clarsimp dest!: single_rt_fresher)

lemma nsqn_rt_fresh_asI:
assumes "dip ∈ kD(rt)"
and "dip ∈ kD(rt')"
and "nsqn rt dip = nsqn rt' dip"
and "π₅(the (rt dip)) = π₅(the (rt' dip))"
shows "rt ≈_{dip} rt'"

proof
from assms(1-2,4) have dhops': "the (dhops rt' dip) ≤ the (dhops rt dip)"
  by (simp add: proj5_eq_dhops)
with assms(1-3) show "rt ⊑ dip rt'"
  by (rule rt_fresherI2)

next
from assms(1-2,4) have dhops: "the (dhops rt dip) ≤ the (dhops rt' dip)"
  by (simp add: proj5_eq_dhops)
with assms(2,1) assms(3) [symmetric] show "rt' ⊑ dip rt"
  by (rule rt_fresherI2)

qed

lemma rt_fresh_asE [elim]:
assumes "rt1 ≈_{dip} rt2"
and "[ rt1 ⊑_{dip} rt2; rt2 ⊑_{dip} rt1 ] ⇒ P rt1 rt2 dip"
shows "P rt1 rt2 dip"

using assms unfolding rt_fresh_as_def by simp

lemma rt_fresh_asD1 [dest]:
assumes "rt1 ≈_{dip} rt2"
shows "rt1 ⊑_{dip} rt2"

using assms unfolding rt_fresh_as_def by simp

lemma rt_fresh_asD2 [dest]:
assumes "rt1 ≈_{dip} rt2"
shows "rt2 ⊑_{dip} rt1"

using assms unfolding rt_fresh_as_def by simp

lemma rt_fresh_as_sym:
assumes "rt1 ≈_{dip} rt2"
shows "rt2 ≈_{dip} rt1"

using assms unfolding rt_fresh_as_def by simp

lemma not_rt_fresh_asI1 [intro]:
assumes "¬ (rt1 ⊑_{dip} rt2)"
shows "¬ (rt1 ≈_{dip} rt2)"

proof
  assume "rt1 ≈_{dip} rt2"
  hence "rt1 ⊑_{dip} rt2" ..
  with "¬ (rt1 ⊑_{dip} rt2)" show False ..

qed

lemma not_rt_fresh_asI2 [intro]:
assumes "¬ (rt2 ⊑_{dip} rt1)"
shows "¬ (rt1 ≈_{dip} rt2)"

proof
  assume "rt1 ≈_{dip} rt2"
  hence "rt2 ⊑_{dip} rt1" ..
  with "¬ (rt2 ⊑_{dip} rt1)" show False ..

qed

lemma not_single_rt_fresher [elim]:
assumes "¬ (the (rt1 ip) ⊑ the (rt2 ip))"
shows \(\neg (rt_1 \subseteq_{ip} rt_2)\)

proof

assume \(rt_1 \subseteq_{ip} rt_2\)

hence \(\neg (\text{the (rt}_1 \text{ ip}) \subseteq \text{the (rt}_2 \text{ ip)})\) ..

with \(\neg (\text{the (rt}_1 \text{ ip}) \subseteq \text{the (rt}_2 \text{ ip)})\) show False ..

qed

lemmas not_single_rt_fresh_asI1 [intro] = not_rt_fresh_asI1 [OF not_single_rt_fresher]
lemmas not_single_rt_fresh_asI2 [intro] = not_rt_fresh_asI2 [OF not_single_rt_fresher]

lemma not_rt_fresher_single [elim]:

assumes \(\neg (rt_1 \subseteq_{ip} rt_2)\)

shows \(\neg (\text{the (rt}_1 \text{ ip}) \subseteq \text{the (rt}_2 \text{ ip)})\)

proof

assume \(\text{the (rt}_1 \text{ ip}) \subseteq \text{the (rt}_2 \text{ ip)}\) ..

hence \(rt_1 \subseteq_{ip} rt_2\) ..

with \(\neg (rt_1 \subseteq_{ip} rt_2)\) show False ..

qed

lemma rt_fresh_as_nsqnr:

assumes \(dip \in kD(rt_1)\)

and \(dip \in kD(rt_2)\)

and \(rt_1 \approx_{ip} dip rt_2\)

shows \(\text{nsqn}_r (\text{the (rt}_2 \text{ dip)}) = \text{nsqn}_r (\text{the (rt}_1 \text{ dip)})\)

using assms(3) unfolding rt_fresher_def

by (auto simp: rt_fresher_def2 [OF \langle \dip \in kD(rt_1) \rangle \langle \dip \in kD(rt_2) \rangle] kD_nsqn [OF \langle \dip \in kD(rt_1) \rangle] kD_nsqn [OF \langle \dip \in kD(rt_2) \rangle])

lemma rt_fresher_mapupd [intro!]:

assumes \(dip \in kD(rt)\)

and \(\text{the (rt dip)} \subseteq r\)

shows \(rt \subseteq_{dip} rt(dip \mapsto r)\)

using assms unfolding rt_fresher_def by simp

lemma rt_fresher_map_update_other [intro!]:

assumes \(dip \in kD(rt)\)

and \(dip \neq ip\)

shows \(rt \subseteq_{dip} rt(ip \mapsto r)\)

using assms unfolding rt_fresher_def by simp

lemma rt_fresher_update_other [simp]:

assumes \(\text{inkD: dip} \in kD(rt)\)

and \(dip \neq ip\)

shows \(rt \subseteq_{dip} \text{update rt ip r}\)

using assms unfolding update_def by (clarsimp split: option.split) (fastforce)

theorem rt_fresher_update [simp]:

assumes \(dip \in kD(rt)\)

and \(\text{the (dhops rt dip)} \geq 1\)

and \(\text{update_arg_wf r}\)

shows \(rt \subseteq_{dip} \text{update rt ip r}\)

proof (cases \(dip = ip\))

assume \(dip = ip\) with \(dip \in kD(rt)\) show ?thesis

by (rule rt_fresher_update_other)

next

assume \(dip = ip\)

from \(dip \in kD(rt)\) obtain \(dsn_n, dsk_n, f_n, hops_n, nhip_n, pre_n\)

where \(\text{rtn [simp]: the (rt dip) = (dsn}_n, dsk}_n, f}_n, hops}_n, nhip}_n, pre}_n)\)

by (metis prod_cases6)

with \(\text{the (dhops rt dip)} \geq 1\) and \(dip \in kD(rt)\) have \(\text{hops}_n \geq 1\)
by (metis proj5_eq_dhops proj4_eq_flag [symmetric])
from ⟨dip∈kD(rt) ⟩ have [simp]: "sqn rt dip = dsn_n "
  and [simp]: "the (dhops rt dip) = hops_n "
  and [simp]: "the (flag rt dip) = f_n "
by (simp add: sqn_def proj5_eq_dhops [symmetric]
proj4_eq_flag [symmetric])+

from ⟨update_arg_wf r ⟩ have "(dsn_n , dsk_n , f_n , hops_n , nhip_n , pre_n ) ⊑ the ((update rt dip r) dip)"
proof (rule wf_r_cases)
  fix nhip pre
  from ⟨hops_n ≥ 1 ⟩ have "∀ pre'. (dsn_n , dsk_n , f_n , hops_n , nhip_n , pre_n )
    ⊑ (dsn_n , unk , val , Suc 0 , nhip , pre')"
    unfolding fresher_def sqn_def by (cases f_n ) auto
  thus "(dsn_n , dsk_n , f_n , hops_n , nhip_n , pre_n )
    ⊑ the (update rt dip (0 , unk , val , Suc 0 , nhip , pre) dip)"
    using ⟨dip∈kD(rt) ⟩ by - (rule update_cases_kD , simp_all)
next
  fix dsn :: sqn and hops nhip pre
  assume "0 < dsn"
  show "(dsn_n , dsk_n , f_n , hops_n , nhip_n , pre_n )
    ⊑ the (update rt dip (dsn , kno , val , hops , nhip , pre) dip)"
  proof (rule update_cases_kD [OF _ ⟨dip∈kD(rt) ⟩], simp_all add: ⟨0 < dsn ⟩)
    assume "dsn_n < dsn"
    thus "(dsn_n , dsk_n , f_n , hops_n , nhip_n , pre_n )
      ⊑ (dsn , kno , val , hops , nhip , pre ∪ pre_n )"
    unfolding fresher_def by auto
next
  assume "dsn_n = dsn"
  and "hops < hops_n "
  thus "(dsn , dsk_n , f_n , hops_n , nhip_n , pre_n )
    ⊑ (dsn , kno , val , hops , nhip , pre ∪ pre_n )"
  unfolding fresher_def nsqn_r_def by simp
next
  assume "dsn_n = dsn"
  with ⟨0 < dsn ⟩ have "(dsn , dsk_n , inv , hops_n , nhip_n , pre_n )
    ⊑ (dsn , kno , val , hops , nhip , pre ∪ pre_n )"
  unfolding fresher_def by simp
qed

theorem rt_fresher_invalidate [simp]:
  assumes "dip∈kD(rt) "
  and indests: "∀ rip∈dom(dests). rip∈vD(rt) ∧ sqn rt rip < the (dests rip)"
  shows "rt ⊑ dip invalidate rt dests"
proof (cases "dip∈dom(dests)")
  assume "dip∉dom(dests)"
  thus ?thesis using ⟨dip∈kD(rt) ⟩ by - (rule single_rt_fresher, simp)
next
  assume "dip∈dom(dests)"
  moreover with indests have "dip∈vD(rt)"
    and "sqn rt dip < the (dests dip)"
  by auto
  ultimately show ?thesis
  unfolding invalidate_def sqn_def
  by - (rule single_rt_fresher, auto simp: fresher_def)
qed
lemma nsqn_r_invalidate [simp]:
assumes "dip ∈ kD(rt)"
and "dip ∈ dom(dests)"
shows "nsqn_r (the (invalidate rt dests dip)) = the (dests dip) - 1"
using assms unfolding invalidate_def by auto

lemma rt_fresh_as_inc_invalidate [simp]:
assumes "dip ∈ kD(rt)"
and "∀ rip ∈ dom(dests). rip ∈ vD(rt) ∧ the (dests rip) = inc (sqn rt rip)"
shows "rt ≈ dip invalidate rt dests"
proof (cases "dip ∈ dom(dests)"
assume "dip /∈ dom(dests)"
with ⟨dip ∈ kD(rt)⟩ have "dip ∈ kD(invalidate rt dests)"
  by simp
with ⟨dip ∈ kD(rt)⟩ show ?thesis
  by rule (simp_all add: ⟨dip /∈ dom(dests)⟩)
next
assume "dip ∈ dom(dests)"
with assms(2)
have "dip ∈ vD(rt)" by auto
from ⟨dip ∈ vD(rt)⟩
with ⟨dip ∈ kD(rt)⟩ have "dip ∈ kD(invalidate rt dests)"
bysimp
moreover then have "dip ∈ kD(invalidate rt dests)"
ultimately show ?thesis
proof (rule nsqn_rt_fresh_asI)
from ⟨dip ∈ vD(rt)⟩
have "sqn rt dip = nsqn_r (the (invalidate rt dests dip))"
also have "sqn rt dip = nsqn_r (the (invalidate rt dests dip))"
proof -
  from ⟨dip ∈ kD(rt)⟩
  have "nsqn_r (the (invalidate rt dests dip)) = the (dests dip) - 1""
    using ⟨dip ∈ dom(dests)⟩ by (rule nsqn_r_invalidate)
  with ⟨the (dests dip) = inc (sqn rt dip)⟩
  show "sqn rt dip = nsqn_r (the (invalidate rt dests dip))"
bysimp
qed
also from ⟨dip ∈ kD(invalidate rt dests)⟩
  have "nsqn_r (the (invalidate rt dests dip)) = nsqn_r (invalidate rt dests dip)"
by (simp add: kD_nsqn)
finally show "nsqn rt dip = nsqn_r (invalidate rt dests dip)".
qed simp

lemmas rt_fresher_inc_invalidate [simp] = rt_fresh_as_inc_invalidate [THEN rt_fresh_asD1]

lemma rt_fresh_as_addpreRT [simp]:
assumes "ip ∈ kD(rt)"
shows "rt ≈ dip the (addpreRT rt ip npre)"
using assms [THEN kD_Some] by (auto simp: addpreRT_def)

lemmas rt_fresher_addpreRT [simp] = rt_fresh_as_addpreRT [THEN rt_fresh_asD1]

4.5.4 Strictly comparing routing tables

definition rt_strictly_fresher :: "ip ⇒ rt ⇒ rt ⇒ bool"
where
  "rt_strictly_fresher ≡ λdip rt1 rt2. (rt1 ⊑ dip rt2) ∧ ¬(rt2 ⊑ dip rt1)"
abbreviation
  rt_strictly_fresher_syn :: "rt ⇒ ip ⇒ rt ⇒ bool" ("_ ⊏ _ _") [51, 999, 51] 50
where
  "rt1 ⊏ ip rt2 ≡ rt_strictly_fresher i rt1 rt2"

lemma rt_strictly_fresher_def":
  "rt1 ⊏ rt2 = ((rt1 ⊑ rt2) ∧ ¬(rt2 ⊑ rt1))"
unfolding rt_strictly_fresher_def rt_fresh_as_def by auto

lemma rt_strictly_fresherI' [intro]:
assumes "rt1 ⊑_i rt2"
and "¬(rt2 ⊑_i rt1)"
shows "rt1 ⊏_i rt2"
using assms unfolding rt_strictly_fresher_def'' by simp

lemma rt_strictly_fresherE' [elim]:
assumes "rt1 ⊑_i rt2"
and "[ rt1 ⊑_i rt2; ¬(rt2 ⊑_i rt1) ] ⇒ P rt1 rt2 i"
shows "P rt1 rt2 i"
using assms unfolding rt_strictly_fresher_def'' by simp

lemma rt_strictly_fresherI [intro]:
assumes "rt1 ⊏_i rt2"
and "¬(rt1 ≈_i rt2)"
shows "rt1 ⊑_i rt2"
unfolding rt_strictly_fresher_def using assms ..

lemmas rt_strictly_fresher_singleI [elim] = rt_strictly_fresherI [OF single_rt_fresher]

lemma rt_strictly_fresherE [elim]:
assumes "rt1 ⊏_i rt2"
and "[ rt1 ⊑_i rt2; ¬(rt2 ⊑_i rt1) ] ⇒ P rt1 rt2 i"
shows "P rt1 rt2 i"
using assms(1) unfolding rt_strictly_fresher_def using assms(2)

lemma rt_strictly_fresher_def':
"rt1 ⊏_i rt2 =
(nsqn r (the (rt1 i)) < nsqn r (the (rt2 i))
 ∨ (nsqn r (the (rt1 i)) = nsqn r (the (rt2 i)) ∧ π5(the (rt1 i)) > π5(the (rt2 i))))"
unfolding rt_strictly_fresher_def rt_fresher_def fresher_def by auto

lemma rt_strictly_fresher_fresherD [dest]:
assumes "rt1 ⊏_dip rt2"
shows "the (rt1 dip) ⊑ the (rt2 dip)"
using assms unfolding rt_strictly_fresher_def rt_fresher_def by auto

lemma rt_strictly_fresher_not_fresh_asD [dest]:
assumes "rt1 ⊏_dip rt2"
shows "¬ rt1 ≈_dip rt2"
using assms unfolding rt_strictly_fresher_def by auto

lemma rt_strictly_fresher_trans [elim, trans]:
assumes "rt1 ⊏_dip rt2"
and "rt2 ⊏_dip rt3"
shows "rt1 ⊏_dip rt3"
using assms proof -
from rt1 ⊏_dip rt2 obtain "the (rt1 dip) ⊑ the (rt2 dip)" by auto
also from rt2 ⊏_dip rt3 obtain "the (rt2 dip) ⊑ the (rt3 dip)" by auto
finally have "the (rt1 dip) ⊑ the (rt3 dip)" .

moreover have "¬ (rt1 ≈_dip rt3)"
proof -
from rt1 ⊏_dip rt2 obtain "¬(the (rt2 dip) ⊑ the (rt1 dip))" by auto
also from rt2 ⊏_dip rt3 obtain "¬(the (rt3 dip) ⊑ the (rt2 dip))" by auto
finally have "¬(the (rt3 dip) ⊑ the (rt1 dip))" .
then thesis ..
qed
ultimately show "rt1 ⊏_dip rt3" ..
qed

lemma rt_strictly_fresher_irefl [simp]: "¬ (rt ⊏_dip rt)"
using assms unfolding rt_strictly_fresher_def
by clarsimp

368
lemma rt_fresher_trans_rt_strictly_fresher [elim, trans]:
assumes "rt1 ⊑ dip rt2"
and "rt2 ⊑ dip rt3"
shows "rt1 ⊏ dip rt3"
proof -
  from rt1 ⊑ dip rt2: have "rt1 ⊑ dip rt2"
  and "¬ (rt2 ⊏ dip rt1)"
  unfolding rt_strictly_fresher_def'' by auto
  from this(1) and rt2 ⊑ dip rt3: have "rt1 ⊏ dip rt3" ..
moreover from (¬ (rt2 ⊏ dip rt1)) have "¬ (rt3 ⊏ dip rt1)"
  proof (rule contrapos_nn)
    assume "rt3 ⊏ dip rt1"
    with rt2 ⊑ dip rt3: show "rt2 ⊏ dip rt1" ..
  qed
ultimately show "rt1 ⊏ dip rt3"
  unfolding rt_strictly_fresher_def'' by auto
qed

lemma rt_fresher_trans_rt_strictly_fresher' [elim, trans]:
assumes "rt1 ⊑ dip rt2"
and "rt2 ⊏ dip rt3"
shows "rt1 ⊏ dip rt3"
proof -
  from rt2 ⊏ dip rt3: have "rt2 ⊏ dip rt3"
  and "¬ (rt3 ⊑ dip rt2)"
  unfolding rt_strictly_fresher_def'' by auto
  from rt1 ⊑ dip rt2 and this(1) have "rt1 ⊏ dip rt3" ..
moreover from (¬ (rt3 ⊑ dip rt2)) have "¬ (rt3 ⊏ dip rt1)"
  proof (rule contrapos_nn)
    assume "rt3 ⊏ dip rt1"
    hence "rt2 ⊑ dip rt1" ..
    hence "nsqn rt2 dip ≤ nsqn rt1 dip"
    using ⟨dip ∈ kD(rt2)⟩ ⟨dip ∈ kD(rt1)⟩
    by (rule rt_fresher_imp_nsqn_le)
  qed
ultimately show "rt1 ⊏ dip rt3"
  unfolding rt_strictly_fresher_def'' by auto
qed

lemma rt_fresher_imp_nsqn_le:
assumes "rt1 ⊑ ip rt2"
and "ip ∈ kD rt1"
and "ip ∈ kD rt2"
shows "nsqn rt1 ip ≤ nsqn rt2 ip"
using assms(1)
by (auto simp add: rt_fresher_def2 [OF assms(2-3)])

lemma rt_strictly_fresher_ltI [intro]:
assumes "dip ∈ kD(rt1)"
and "dip ∈ kD(rt2)"
and "nsqn rt1 dip < nsqn rt2 dip"
shows "rt1 ⊏ dip rt2"
proof
  from assms show "rt1 ⊏ dip rt2" ..
next
  show "¬ (rt1 ≈ dip rt2)"
  proof
    assume "rt1 ≈ dip rt2"
    hence "rt2 ⊏ dip rt1" ..
    hence "nsqn rt2 dip ≤ nsqn rt1 dip"
    using ⟨dip ∈ kD(rt2)⟩ ⟨dip ∈ kD(rt1)⟩
    by (rule rt_fresher_imp_nsqn_le)
with \( \text{nsqn } \text{rt1 } \text{dip } \text{< nsqn } \text{rt2 } \text{dip} \) show "False"
by simp
qed
qed

lemma \( \text{rt_strictly_fresher_eqI} \) [intro]:
assumes "\( i \in \text{kD} \text{(rt1)} \)"
and "\( i \in \text{kD} \text{(rt2)} \)"
and "\( \text{nsqn } \text{rt1 } i = \text{nsqn } \text{rt2 } i \)"
and "\( \pi_5(\text{the } (\text{rt2 } i)) < \pi_5(\text{the } (\text{rt1 } i)) \)"
shows "\( \text{rt1 } \sqsubseteq_1 \text{rt2} \)"
using assms unfolding \( \text{rt_strictly_fresher_def}' \) by (auto simp add: \( \text{kD}_{\text{nsqn}} \))

lemma \( \text{invalidate_rtsf_left} \) [simp]:
\( \{ \text{dests dip rt rt'}. \text{dests dip = None } \rightarrow (\text{invalidate } \text{rt dests } \sqsubseteq_1 \text{dip rt'}) = (\text{rt } \sqsubseteq_1 \text{dip rt'}) \)"
unfolding \( \text{invalidate_def } \text{rt_strictly_fresher_def}' \) by (rule iffI) (auto split: option.split_asm)

lemma \( \text{vD}_{\text{invalidate } \text{rt_strictly_fresher}} \) [simp]:
assumes "\( \text{dip } \in \text{vD} (\text{invalidate } \text{rt1 dests}) \)"
shows "\( (\text{invalidate } \text{rt1 dests } \sqsubseteq_1 \text{dip rt2}) = (\text{rt1 } \sqsubseteq_1 \text{dip rt2}) \)"
proof (cases "\( \text{dip } \in \text{dom(dests)} \)")
assume "\( \text{dip } \notin \text{dom(dests)} \)"
moreover with \( \{ \text{dip } \in \text{vD} (\text{invalidate } \text{rt1 dests}) \)\) have "\( \text{dip } \in \text{vD} (\text{rt1}) \)"
unfolding \( \text{invalidate_def } \text{vD_def} \)
by clarsimp (metis (hide_lams, no_types) assms vD_Some vD_invalidate_vD_not_dests)
ultimately show \( ? \text{thesis} \) by simp
next
assume "\( \text{dip } \notin \text{dom(dests)} \)"
moreover with \( \{ \text{dip } \in \text{vD} (\text{invalidate } \text{rt1 dests}) \)\) have "\( \text{dip } \in \text{vD} (\text{rt1}) \)"
unfolding \( \text{vD_def} \)
by clarsimp (metis (hide_lams, no_types) assms vD_Some vD_invalidate_vD_not_dests)
ultimately show \( ? \text{thesis} \) by simp
qed

lemma \( \text{rt_strictly_fresher_update_other} \) [elim!]:
\( \{ \text{dip ip rt rt'}. \text{dip } \neq \text{ip}; \text{rt } \sqsubseteq_1 \text{dip rt'} \} \rightarrow (\text{update } \text{rt dip osn, kno, val, hops, nhip, {} } \sqsubseteq_1 \text{dip rt2}) \)" by clarsimp

lemma \( \text{addpreRT_strictly_fresher} \) [simp]:
assumes "\( \text{dip } \in \text{kD} (\text{rt}) \)"
shows "\( (\text{the } (\text{addpreRT } \text{rt dip npre}) ) \sqsubseteq_1 \text{ip rt2} = (\text{rt } \sqsubseteq_1 \text{ip rt2}) \)"
using assms unfolding \( \text{rt_strictly_fresher_def}' \) by clarsimp

lemma \( \text{lt_sqn_imp_update_strictly_fresher} \):
assumes "\( \text{dip } \in \text{vD} (\text{rt2 nhip}) \)"
and \( \} \) \( \text{rt } \neq \text{update } \text{rt dip osn, kno, val, hops, nhip, {}} \)"
shows "\( \text{update } \text{rt dip osn, kno, val, hops, nhip, {}} \sqsubseteq_1 \text{dip rt2 nhip} \)"
proof (rule disjI1)
from \( \} \) have "\( \text{nsqn } (\text{update } \text{rt dip osn, kno, val, hops, nhip, {}}) \) \text{dip } = \text{osn} \)"
by (rule nsqn_update_changed_kno_val)
with \( \{ \text{dip } \in \text{vD} (\text{rt2 nhip}) \)\)
have "\( \text{nsqn} \text{(rt2 nhip)} \) \text{dip } = \text{osn} \text{(update } \text{rt dip osn, kno, val, hops, nhip, {}}) \)"
by (simp add: \( \text{kD}_{\text{nsqn}} \))
also have "\( \text{osn } \sqsubseteq_1 \text{rt2 nhip} \) dip" by (rule *)
also have "\( \text{sqn} \text{(rt2 nhip)} \) dip = \text{nsqn} \text{(rt2 nhip)}\)" unfolding \( \text{nsqn}_{\text{def}} \) using \( \{ \text{dip } \in \text{vD} (\text{rt2 nhip}) \)\)
by - (metis \( \text{vD}_{\text{flag_val val proj2_eq_sqn proj4_eq_flag vD_iD_gives_kD(1)} \))
finally show "\( \text{nsqn} \text{(update } \text{rt dip osn, kno, val, hops, nhip, {}}) \) < \text{nsqn} \text{(rt2 nhip dip)} \)".
lemma dhops_le_hops_imp_update_strictly_fresher:
assumes "dip ∈ vD(rt2 nhip)"
and sqn: "sqn (rt2 nhip) dip = osn"
and hop: "the (dhops (rt2 nhip) dip) ≤ hops"
and "rt ≠ update rt dip (osn, kno, val, Suc hops, nhip, {})"
shows "update rt dip (osn, kno, val, Suc hops, nhip, {}) ⊏ dip rt2 nhip"
unfolding rt_strictly_fresher_def'
proof (rule disjI2, rule conjI)
from assms have "nsqn (update rt dip (osn, kno, val, Suc hops, nhip, {}) dip) = osn"
by (rule nsqn_update_changed_kno_val)
with ⟨dip ∈ vD(rt2 nhip)⟩
have "nsqn r (the (update rt dip (osn, kno, val, Suc hops, nhip, {}) dip)) = osn"
by (simp add: kD_nsqn)
also have "osn = sqn (rt2 nhip) dip" by (rule sqn [symmetric])
also have "sqn (rt2 nhip) dip = nsqn r (the (update rt dip (osn, kno, val, Suc hops, nhip, {}) dip))"
unfolding nsqn_r_def using ⟨dip ∈ vD(rt2 nhip)⟩
by (metis vD_flag_val proj2_eq_sqn proj4_eq_flag vD_iD_gives_kD(1))
finally show "nsqn r (the (update rt dip (osn, kno, val, Suc hops, nhip, {}) dip)) = nsqn r (the (rt2 nhip dip))".
next
have "the (dhops (rt2 nhip) dip) ≤ hops" by (rule hop)
also have "hops < hops + 1" by simp
also have "hops + 1 = the (dhops (update rt dip (osn, kno, val, Suc hops, nhip, {})) dip)"
using assms by simp
finally have "the (dhops (rt2 nhip) dip) < the (dhops (update rt dip (osn, kno, val, Suc hops, nhip, {})) dip)"
thus "π₅ (the (rt2 nhip dip)) < π₅ (the (update rt dip (osn, kno, val, Suc hops, nhip, {}) dip))"
using ⟨dip ∈ vD(rt2 nhip)⟩ by (simp add: proj5_eq_dhops)
qed
lemma nsqn_invalidate:
assumes "dip ∈ kD(rt)"
and "∀ip∈dom(dests). ip ∈ vD(rt) ∧ the (dests ip) = inc (sqn rt ip)"
shows "nsqn (invalidate rt dests) dip = nsqn rt dip"
proof
from assms have "rt ≈_{dip} invalidate rt dests" by (rule rt_fresh_as_inc_invalidate)
with ⟨dip ∈ kD(rt)⟩ ⟨dip ∈ kD(invalidate rt dests)⟩ show ?thesis
by (simp add: kD_nsqn del: invalidate_kD_inv)
(erule(2) rt_fresh_as_nsqnr)
qed

4.6 Invariant proofs on individual processes

theory D_Seq_Invariants
imports "."../AWN/Invariants" D_Aodv D_Aodv_Data D_Aodv_Predicates D_Fresher
begin
The proposition numbers are taken from the December 2013 version of the Fehnker et al technical report.

Proposition 7.2

lemma sequence_number_increases:
"paodv i |= onll Γ_{AODV} (λ((ξ, _), _, (ξ', _)). sn ξ ≤ sn ξ')"
by inv_cterms

lemma sequence_number_one_or_bigger:
"paodv i |= onll Γ_{AODV} (λ(ξ, _). 1 ≤ sn ξ)"
by (rule onll_step_to_invariantI [OF sequence_number_increases])
We can get rid of the onl/onll if desired...

**Lemma sequence_number_increases':**

\[ \text{paodv } i \models (\lambda((\xi, \_), \_), (\xi', \_)). \text{sn } \xi \leq \text{sn } \xi' \]

by (rule step_invariant_weakenE [OF sequence_number_increases]) (auto dest!: onllD)

**Lemma sequence_number_one_or_bigger':**

\[ \text{paodv } i \models (\lambda((\xi, \_)). 1 \leq \text{sn } \xi') \]

by (rule invariant_weakenE [OF sequence_number_one_or_bigger]) auto

**Lemma sip_in_kD:**

\[ \text{paodv } i \models \text{onl } \Gamma_{\text{AODV}} (\lambda(\xi, 1). 1 \in \{\text{PAodv-:7} \cup \{\text{PAodv-:5} \cup \{\text{PRrep-:0..PRrep-:1} \}
\cup \{\text{PRreq-:0..PRreq-:3}\} \rightarrow \text{sip } \xi \in kD (rt \xi))} \]

by inv_cterms

**Lemma rrep_1_update_changes:**

\[ \text{paodv } i \models \text{onl } \Gamma_{\text{AODV}} (\lambda(\xi, l). (l = \text{PRrep-:1} \rightarrow rt \xi \neq \text{update} (rt \xi) (dip \xi) (dsn \xi, kno, val, hops \xi + 1, sip \xi, {}))) \]

by inv_cterms

**Lemma addpreRT_partly_welldefined:**

\[ \text{paodv } i \models \text{onl } \Gamma_{\text{AODV}} (\lambda(\xi, l). (l \in \{\text{PRreq-:18..PRreq-:20} \cup \{\text{PRrep-:2..PRrep-:6}\} \rightarrow \text{dip } \xi \in kD (rt \xi)) \]
\[ \land (l = \text{PRreq-:3..PRreq-:19} \rightarrow \text{oip } \xi \in kD (rt \xi))) \]

by inv_cterms

**Proposition 7.38:**

**Lemma includes_nhip:**

\[ \text{paodv } i \models \text{onl } \Gamma_{\text{AODV}} (\lambda(\xi, 1). \forall dip \in kD(rt \xi). \text{the} (\text{nhop} (rt \xi) dip) \in kD(rt \xi)) \]

proof -

{ fix ip and \( \xi, \xi' :: \text{state} \)
assumption "\( \forall dip \in kD (rt \xi). \text{the} (\text{nhop} (rt \xi) dip) \in kD (rt \xi)" 
and "\( \xi' = \xi (| rt := \text{update} (rt \xi) ip (0, unk, val, Suc 0, ip, {}))\)"

hence "\( \forall dip \in kD (rt \xi). \)
\( \text{the} (\text{nhop} (rt \xi) ip (0, unk, val, Suc 0, ip, {})) dip) = ip \
\lor \text{the} (\text{nhop} (rt \xi) ip (0, unk, val, Suc 0, ip, {})) dip) \in kD (rt \xi)" 
by clarsimp (metis nhop_update_unk_val update_another)
}

note one_hop = this

{ fix ip and \( \xi, \xi' :: \text{state} \)
assumption "\( \forall dip \in kD (rt \xi). \text{the} (\text{nhop} (rt \xi) dip) \in kD (rt \xi)" 
and "\( \xi' = \xi (| rt := \text{update} (rt \xi) ip (sn, kno, val, Suc hops, sip, {}))\)"
and "\( \text{sip} \in kD (rt \xi)" 

hence "\( \text{the} (\text{nhop} (rt \xi) ip (sn, kno, val, Suc hops, sip, {})) ip) = ip \
\lor \text{the} (\text{nhop} (rt \xi) ip (sn, kno, val, Suc hops, sip, {})) ip) \in kD (rt \xi)) \]
\[ \land (l = \text{PRreq-:19} \rightarrow \text{oip } \xi \in kD (rt \xi)) \land 
\[ (l = \text{PRrep-:5} \rightarrow \text{dip } \xi \in kD (rt \xi)) \land 
\[ (l = \text{PRrep-:6} \rightarrow (\text{the} (\text{nhop} (rt \xi) dip))) \in kD (rt \xi))) \]

by (metis kD_update_unchanged nhop_update_changed update_another)
}

note nhip_is_sip = this

show ?thesis 
by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf sip_in_kD] 
onl_invariant_sterms [OF aodv_wf addpreRT_partly_welldefined] 
solve: one_hop nhip_is_sip)

qed

**Proposition 7.22:** needed in Proposition 7.4

**Lemma addpreRT_welldefined:**

\[ \text{paodv } i \models \text{onl } \Gamma_{\text{AODV}} (\lambda(\xi, 1). (l \in \{\text{PRreq-:18..PRreq-:20} \cup \{\text{PRrep-:2..PRrep-:6}\} \rightarrow \text{dip } \xi \in kD (rt \xi)) \]
\[ \land (l = \text{PRreq-:3..PRreq-:19} \rightarrow \text{oip } \xi \in kD (rt \xi))) \]

(is "\( _\models \text{onl } \Gamma_{\text{AODV}} ?P\)"
unfolding invariant_def)
proof
fix s
assume "s ∈ reachable (paodv i) TT"
then obtain ξ p where "s = (ξ, p)"
and "(ξ, p) ∈ reachable (paodv i) TT"
by (metis PairE)
have "onl Γ\text{AODV} ?P (ξ, p)"
proof (rule onlI)
fix l
assume "l ∈ labels Γ\text{AODV} p"
with ⟨(ξ, p) ∈ reachable (paodv i) TT⟩ have I1: "l ∈ {PRreq-:18..PRreq-:20} −→ dip ξ ∈ kD(rt ξ)"
and I2: "l = PRreq-:19 −→ oip ξ ∈ kD(rt ξ)"
and I3: "l ∈ {PRrep-:2..PRrep-:6} −→ dip ξ ∈ kD(rt ξ)"
by (auto dest!: invariantD [OF addpreRT_partly_welldefined])
moreover from ⟨(ξ, p) ∈ reachable (paodv i) TT⟩ ⟨l ∈ labels Γ\text{AODV} p⟩ and I3
ultimately have "l = PRrep-:6 −→ (\text{the (nhop (rt ξ) (dip ξ))) ∈ kD(rt ξ)}"
by (auto dest!: invariantD [OF includes_nhip])
ultimately show "?P (ξ, 1)"
by simp
qed
with ⟨s = (ξ, p)⟩ show "onl Γ\text{AODV} ?P s"
by simp
qed

Proposition 7.4

lemma known_destinations_increase:
"paodv i ||= A onll Γ\text{AODV} (λ((ξ, _), _, (ξ', _)). kD (rt ξ) ⊆ kD (rt ξ'))"
by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf addpreRT_welldefined]
simp add: subset_insertI)

Proposition 7.5

lemma rreqs_increase:
"paodv i ||= A onll Γ\text{AODV} (λ((ξ, _), _, (ξ', _)). rreqs ξ ⊆ rreqs ξ')"
by (inv_cterms simp add: subset_insertI)

lemma dests_bigger_than_sqn:
"paodv i ||= onl Γ\text{AODV} (λ(ξ, 1). l ∈ \{PAodv-:15..PAodv-:19\}
∪ \{PPkt-:7..PPkt-:11\}
∪ \{PRreq-:11..PRreq-:15\}
∪ \{PRreq-:24..PRreq-:28\}
∪ \{PRrep-:10..PRrep-:14\}
∪ \{PRerr-:1..PRerr-:5\}
−→ (∀ip∈dom(dests ξ). ip∈kD(rt ξ) ∧ sqn (rt ξ) ip ≤ the (dests ξ ip)))"
proof -
have sqninv:
"\(\land\) dests rt rsn ip.
[ ∨ ip∈dom(dests). ip∈kD(rt) ∧ sqn rt ip ≤ (dests ip); dests ip = Some rsn ]
⇒ sqn (invalidate rt dests) ip ≤ rsn"
by (rule sqn_invalidate_in_dests [THEN eq_imp_le], assumption) auto
have indests:
"\(\land\) dests rt rsn ip.
[ ∨ ip∈dom(dests). ip∈kD(rt) ∧ sqn rt ip ≤ (dests ip); dests ip = Some rsn ]
⇒ ip∈kD(rt) ∧ sqn rt ip ≤ rsn"
by (metis domI option.sel)
show ?thesis
by inv_cterms
  (clarsimp split: split_if_asm option.split_asm elim!: sqninv indests)+
qed

Proposition 7.6

lemma sqns_increase:
"paodv i ||= onl Γ\text{AODV} (λ((ξ, _), _, (ξ', _)). ∀ip. sqn (rt ξ) ip ≤ sqn (rt ξ') ip)"
proof -
{ fix ξ :: state
  assume *: "∀ ip∈dom(dests ξ). ip ∈ kD (rt ξ) ∧ sqn (rt ξ) ip ≤ the (dests ξ ip)
  have "∀ ip. sqn (rt ξ) ip ≤ sqn (invalidate (rt ξ) (dests ξ ip) ip"
  proof
  fix ip
  from * have "ip∉dom(dests ξ) ∨ sqn (rt ξ) ip ≤ the (dests ξ ip)" by simp
  thus "sqn (rt ξ) ip ≤ sqn (invalidate (rt ξ) (dests ξ ip)) ip"
  by (metis domI invalidate_sqn option.sel)
  qed
} note solve_invalidate = this
show ?thesis
by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf addpreRT_welldefined]
  onl_invariant_sterms [OF aodv_wf dests_bigger_than_sqn]
  simp add: solve_invalidate)
qed

Proposition 7.7

lemma ip_constant:
"paodv i ||= onl Γ AODV (λ(ξ, _). ip ξ = i)"
by (inv_cterms simp add: σ AODV _def)

Proposition 7.8

lemma sender_ip_valid':
"paodv i ||= A onll Γ AODV (λ((ξ, _), a, _). anycast (λm. not_Pkt m −→ msg_sender m = ip ξ) a)"
by inv_cterms

lemma sender_ip_valid:
"paodv i ||= A onll Γ AODV (λ((ξ, _), a, _). anycast (λm. not_Pkt m −→ msg_sender m = i) a)"
by (rule step_invariant_weaken_with_invariantE [OF ip_constant sender_ip_valid'])
(auto dest!: onlD onllD)

lemma received_msg_inv:
"paodv i ||= (recvmsg P →) onl Γ AODV (λ(ξ, l). l ∈ {PAodv-:1} −→ P (msg ξ))"
by inv_cterms

Proposition 7.9

lemma sip_not_ip':
"paodv i ||= (recvmsg (λm. not_Pkt m −→ msg_sender m ≠ i) →) onl Γ AODV (λ(ξ, _). sip ξ ≠ ip ξ)"
by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]
  onl_invariant_sterms [OF aodv_wf ip_constant [THEN invariant_restrict_inD]]
  simp add: clear_locals_sip_not_ip') clarsimp+

lemma sip_not_ip:
"paodv i ||= (recvmsg (λm. not_Pkt m −→ msg_sender m ≠ i) →) onl Γ AODV (λ(ξ, _). sip ξ ≠ i)"
by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]
  onl_invariant_sterms [OF aodv_wf ip_constant [THEN invariant_restrict_inD]]
  simp add: clear_locals_sip_not_ip') clarsimp+

Neither sip_not_ip' nor sip_not_ip is needed to show loop freedom.

Proposition 7.10

lemma hop_count_positive:
"paodv i ||= onl Γ AODV (λ(ξ, _). ∀ ip∈kD (rt ξ). the (dhops (rt ξ) ip) ≥ 1)"
by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf addpreRT_welldefined]) auto

lemma rreq_dip_in_vD_dip_eq_ip:
"paodv i ||= onl Γ AODV (λ(ξ, 1). l ∈ {PRreq-:18..PRreq-:21} −→ dip ξ ∈ vD(rt ξ))
  ∧ (l ∈ {PRreq-:6, PRreq-:7} −→ dip ξ = ip ξ)
  ∧ (l ∈ {PRreq-:17..PRreq-:21} −→ dip ξ ≠ ip ξ)"
proof (inv_cterms, elim conjE)
  fix l ξ pp p'
  assume "((ξ, pp) ∈ reachable (paodv i) TT"
and \( \{\text{PRreq-19}\} \\lambda_\xi. \xi(\text{rt} := \text{the (addpreRT (rt \_\xi) (oip \_\xi) \{the (nhop (rt \_\xi) (dip \_\xi)\}))}) \) \( p' \in \text{sterms } \Gamma_{AODV} pp \)
and \( l = \text{PRreq-19} \)
and \( \text{dip } \_\xi \in vD (rt \_\xi) \)
from this (1-3) have \( \text{oip } \_\xi \in kD (rt \_\xi) \)
by (auto dest: onl_invariant_sterms [OF aodv_wf addpreRT_welldefined, where l="PRreq-19"])
with \( \text{dip } \_\xi \in vD (rt \_\xi) \)
show \( \text{dip } \_\xi \in vD (\text{the (addpreRT (rt \_\xi) (oip \_\xi) \{the (nhop (rt \_\xi) (dip \_\xi)\}))}) \)
by simp
qed

Proposition 7.11

lemma anycast_msg_zhops:
"\( \lambda rreqid dip dsn ds k ip osn sip. \)
\( \text{paodv i } \models_A \text{ onl } \Gamma_{AODV} (\lambda(_, a, _). \text{anycast } \text{msg_zhops } a) \)"
proof (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf rreq_dip_in_vD_dip_eq_ip [THEN invariant_restrict_inD]]
onl_invariant_sterms [OF aodv_wf hop_count_positive [THEN invariant_restrict_inD]],
elim conjE)
fix \( l \_\xi a pp p' pp' \)
assume "(\_\xi, pp) \in \text{reachable (paodv i) } TT"
and "\( \{\text{PRreq-20}\} \\lambda_\xi. \text{the (nhop (rt \_\xi) (oip \_\xi))} \)
(\( \text{Rrep (the (dhops (rt \_\xi) (dip \_\xi))) (dip \_\xi) (sqn (rt \_\xi) (dip \_\xi)) (oip \_\xi) (ip \_\xi). \) \)
\( p' > pp' \in \text{sterms } \Gamma_{AODV} pp \)
and "l = \text{PRreq-20}"
and "a = \text{unicast (the (nhop (rt \_\xi) (oip \_\xi)))} \)
(\( \text{Rrep (the (dhops (rt \_\xi) (dip \_\xi))) (dip \_\xi) (sqn (rt \_\xi) (dip \_\xi)) (oip \_\xi) (ip \_\xi). \)"
and "\( \forall ip \in kD (rt \_\xi). \text{Suc 0 } \leq \text{the (dhops (rt \_\xi) ip)} \)"
and "\( \text{dip } \_\xi \in vD (rt \_\xi) \)"
from \( \text{dip } \_\xi \in vD (rt \_\xi) \) have "\( \text{dip } \_\xi \in kD (rt \_\xi) \)"
by (rule vD_iD_gives_kD(1))
with \* have "\( \text{Suc 0 } \leq \text{the (dhops (rt \_\xi) (dip \_\xi))} .. \) .
thus "\( 0 < \text{the (dhops (rt \_\xi) (dip \_\xi))} \)" by simp
qed

lemma hop_count_zero_oip_dip_sip:
"\( \text{paodv i } \models (\text{recvmsg } \text{msg_zhops } \rightarrow) \text{ onl } \Gamma_{AODV} (\lambda(_, l). \text{hop_count_zero_oip_dip_sip } a) \)"
by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]) auto

lemma osn_rreq:
"\( \text{paodv i } \models (\text{recvmsg } \text{rreq_rrep_sn } \rightarrow) \text{ onl } \Gamma_{AODV} (\lambda(_, l). \text{osn_rreq } a) \)"
by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]) clarsimp

lemma osn_rreq':
"\( \text{paodv i } \models (\text{recvmsg } (\lambda m. \text{rreq_rrep_sn } m \land \text{msg_zhops } m) \rightarrow) \text{ onl } \Gamma_{AODV} (\lambda(_, l). \text{osn_rreq'} a) \)"
proof (rule invariant_weakenE [OF osn_rreq])
fix a
assume "\( \text{recvmsg } (\lambda m. \text{rreq_rrep_sn } m \land \text{msg_zhops } m) a \)"
thus "\( \text{recvmsg } \text{rreq_rrep_sn } a \)"
by (cases a) simp_all
qed

lemma dsn_rrep:
"\( \text{paodv i } \models (\text{recvmsg } \text{rreq_rrep_sn } \rightarrow) \text{ onl } \Gamma_{AODV} (\lambda(_, l). \text{dsn_rrep } a) \)"
by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]) clarsimp
lemma dsn_rrep':
"paodv i |= (recvmsg (\lambda m. rreq_rrep_sn m \land msg_zhops m) \rightarrow) onl \Gamma_{AODV} (\lambda(\xi, 1).
1 \in \{PAodv-:6, PAodv-:7\} \cup \{PRrep-:n\{n. True\} \rightarrow 1 \leq dsn(\xi)"
proof (rule invariant_weakenE [OF dsn_rrep])
  fix a
  assume "recvmsg (\lambda m. rreq_rrep_sn m \land msg_zhops m) a"
  thus "recvmsg rreq_rrep_sn a"
  by (cases a) simp_all
qed

lemma hop_count_zero_oip_dip_sip':
"paodv i ||= (recvmsg (\lambda m. rreq_rrep_sn m \land msg_zhops m) \rightarrow) onl \Gamma_{AODV} (\lambda(\xi, l).
(l \in \{PAodv-:4..PAodv-:5\} \cup \{PRreq-:n\{n. True\} \rightarrow (hops(\xi) = 0 \rightarrow oip(\xi) = sip(\xi))\))
\land
(((l \in \{PAodv-:6..PAodv-:7\} \cup \{PRrep-:n\{n. True\} \rightarrow (hops(\xi) = 0 \rightarrow dip(\xi) = sip(\xi))))"
proof (rule invariant_weakenE [OF hop_count_zero_oip_dip_sip])
  fix a
  assume "recvmsg (\lambda m. rreq_rrep_sn m \land msg_zhops m) a"
  thus "recvmsg msg_zhops a"
  by (cases a) simp_all
qed

Proposition 7.12

lemma zero_seq_unk_hops_one':
"paodv i ||= (recvmsg (\lambda m. rreq_rrep_sn m \land msg_zhops m) \rightarrow) onl \Gamma_{AODV} (\lambda(\xi, _).
\forall dip: kD(rt(\xi)). (sqn(rt(\xi)) dip = 0 \rightarrow sqnf(rt(\xi)) dip = unk)
\land
(sqn(rt(\xi)) dip = unk \rightarrow the(dhops(rt(\xi)) dip) = 1)
\land
(the(dhops(rt(\xi)) dip) = 1 \rightarrow the(nhop(rt(\xi)) dip) = dip))"
proof -
{ fix dip and \xi :: state and P
  assume "sqn(invalidate(rt(\xi))(dests(\xi))) dip = 0"
  and all: "\forall ip. sqn(rt(\xi)) ip \leq sqn(invalidate(rt(\xi))(dests(\xi))) ip"
  and *: "sqn(rt(\xi)) dip = 0 \Longrightarrow P \xi dip"
  have "P \xi dip"
  proof -
    from all have "sqn(rt(\xi)) dip \leq sqn(invalidate(rt(\xi))(dests(\xi))) dip "..
    with \langle sqn(invalidate(rt(\xi))(dests(\xi))) dip = 0 \rangle have "sqn(rt(\xi)) dip = 0" by simp
    thus "P \xi dip" by (rule *)
  qed
}
note sqn_invalidate_zero [elim!] = this

{ fix dsn hops :: nat and sip oip rt and ip dip :: ip
  assume "\forall dip: kD(rt).
  (sqn rt dip = 0 \rightarrow \pi_3(the(rt dip)) = unk) \land
  (\pi_3(the(rt dip)) = unk \rightarrow the(dhops rt dip) = Suc 0) \land
  (the(dhops rt dip) = Suc 0 \rightarrow the(nhop rt dip) = dip)"

  and "hops = 0 \rightarrow sip = dip"
  and "Suc 0 \leq dsn"
  and "ip \neq dip \rightarrow ip: kD(rt)"
  hence "the(dhops(update rt dip(dsn, kno, val, Suc hops, sip, {})) ip) = Suc 0 \rightarrow
  the(nhop(update rt dip(dsn, kno, val, Suc hops, sip, {})) ip) = ip"
    by - (rule update_cases, auto simp add: sqn_def dest!: bepec)
} note prreq_ok1 [simp] = this

{ fix ip dsn hops sip oip rt dip
  assume "\forall dip: kD(rt).
  (sqn rt dip = 0 \rightarrow \pi_3(the(rt dip)) = unk) \land
  (\pi_3(the(rt dip)) = unk \rightarrow the(dhops rt dip) = Suc 0) \land
  (the(dhops rt dip) = Suc 0 \rightarrow the(nhop rt dip) = dip)"

  and "Suc 0 \leq dsn"
  and "ip \neq dip \rightarrow ip: kD(rt)"
  hence "\pi_3(the(update rt dip(dsn, kno, val, Suc hops, sip, {})) ip)) = unk \rightarrow
the (dhops (update rt dip (dsn, kno, val, Suc hops, sip, {})) ip) = Suc 0" 
by - (rule update_cases, auto simp add: sqn_def sqnf_def dest!: bspec)
} note prreq_ok2 [simp] = this

{ fix ip dsn hops sip oip rt dip 
assume \( \forall \text{ dip} \in \text{kD}(rt) \).
(\( \text{sqn} (\text{rt dip}) = 0 \longrightarrow \pi_3 (\text{the (rt dip)}) = \text{unk} \) \land 
(\( \pi_3 (\text{the (rt dip)}) = \text{unk} \longrightarrow \text{the (dhops rt dip)} = \text{Suc 0} \) \land 
(\( \text{the (dhops rt dip)} = \text{Suc 0} \longrightarrow \text{the (nhop rt dip)} = \text{dip} \))" 
and "Suc 0 \leq dsn"
and "ip \neq dip \longrightarrow \text{ip} \in \text{kD}(rt)"
hence "sqn (update rt dip (dsn, kno, val, Suc hops, sip, {})) ip = 0 \longrightarrow 
\( \pi_3 (\text{the (update rt dip (dsn, kno, val, Suc hops, sip, {})) ip}) = \text{unk} \)"
by - (rule update_cases, auto simp add: sqn_def dest!: bspec)
} note prreq_ok3 [simp] = this

{ fix rt sip 
assume \( \forall \text{ dip} \in \text{kD} rt. \)
(\( \text{sqn} (\text{update rt dip (0, unk, val, Suc 0, sip, {})) dip} = 0 \longrightarrow 
\( \pi_3 (\text{the (update rt dip (0, unk, val, Suc 0, sip, {})) dip}) = \text{unk} \land 
(\( \text{the (dhops (update rt dip (0, unk, val, Suc 0, sip, {})) dip)} = \text{Suc 0} \land 
(\( \text{the (nhop (update rt dip (0, unk, val, Suc 0, sip, {})) dip)} = \text{dip} \))" 
by - (rule update_cases, simp_all add: sqnf_def sqn_def)
} note prreq_ok4 [simp] = this

have prreq_ok5 [simp]: "\( \forall \text{ sip rt.} \)
\( \pi_3 (\text{the (update rt sip (0, unk, val, Suc 0, sip, {})) sip}) = \text{unk} \longrightarrow 
\text{the (dhops (update rt sip (0, unk, val, Suc 0, sip, {})) sip)} = \text{Suc 0} \)"
by (rule update_cases) simp_all

have prreq_ok6 [simp]: "\( \forall \text{ sip rt.} \)
\( \text{sqn (update rt sip (0, unk, val, Suc 0, sip, {})) sip} = 0 \longrightarrow 
\( \pi_3 (\text{the (update rt sip (0, unk, val, Suc 0, sip, {})) sip}) = \text{unk} \)"
by (rule update_cases) simp_all

show \(?\text{thesis}\) 
by (inv_cterms inv add: onl_invariant_sterms_TT [OF aodv_wf addpreRT_welldefined] 
onl_invariant_sterms [OF aodv_wf hop_count_zero_oip_dip_sip'] 
seq_step_invariant_sterms_TT [OF sqns_increase aodv_wf aodv_trans] 
onl_invariant_sterms [OF aodv_wf osn_rreq'] 
onl_invariant_sterms [OF aodv_wf dsn_rrep']) clarsimp+
qed

lemma zero_seq_unk_hops_one:
"paodv i ||= (recvmsg (\( \lambda m. \text{rreq_rrep_sn m} \land \text{msg_zhops m} \)) \rightarrow) onl \Gamma_{AODV} (\lambda (\xi, _).
\forall \text{ dip} \in \text{kD}(rt \xi). \ (\text{sqn (rt dip)} = 0 \longrightarrow \text{sqnf (rt dip)} = \text{unk} \land 
\text{the (dhops (rt dip))} = 1 \land 
\text{the (nhop (rt dip))} = \text{dip}))" 
by (rule invariant_weakenE [OF zero_seq_unk_hops_one']) auto

lemma kD_unk_or_atleast_one:
"paodv i ||= (recvmsg \text{rreq_rrep_sn} \rightarrow) onl \Gamma_{AODV} (\lambda (\xi, 1).
\forall \text{ dip} \in \text{kD}(rt \xi). \ \pi_3 (\text{the (rt dip)}) = \text{unk} \lor 1 \leq \pi_2 (\text{the (rt dip)}))" 
proof -
{ fix sip rt dsn1 dsn2 dsk1 dsk2 flag1 flag2 hops1 hops2 nhip1 nhip2 pre1 pre2 
assume "dsk1 = unk \lor Suc 0 \leq dsn2"
hence "\( \pi_3 (\text{the (update rt sip (dsn1, dsk1, flag1, hops1, nhip1, pre1) sip}) = \text{unk} \lor Suc 0 \leq \text{sqn (update rt sip (dsn2, dsk2, flag2, hops2, nhip2, pre2)) sip} \)"
unfolding update_def by (cases "dsk1 = unk") (clarsimp split: option.split)+
} note fromsip [simp] = this

{ fix dip sip rt dsn1 dsn2 dsk1 dsk2 flag1 flag2 hops1 hops2 nhip1 nhip2 pre1 pre2
  assume allkd: "∀ dip∈kD(rt). π₃(the (rt dip)) = unk ∨ Suc 0 ≤ sqn rt dip"
  and **: "dsk1 = unk ∨ Suc 0 ≤ dsn2"
  have "∀ dip∈kD(rt). π₃(the (update rt sip (dsn1, dsk1, flag1, hops1, nhip1, pre1) dip)) = unk
               ∨ Suc 0 ≤ sqn (update rt sip (dsn2, dsk2, flag2, hops2, nhip2, pre2)) dip"
  (is "∀ dip∈kD(rt). ?prop dip")
  proof
    fix dip
    assume "dip∈kD(rt)"
    thus "?prop dip"
    proof (cases "dip = sip")
      assume "dip = sip" with ** show ?thesis by simp
    next
      assume "dip ≠ sip" with ⟨dip∈kD(rt)⟩ allkd show ?thesis by simp
    qed
  qed
} note solve_update [simp] = this

{ fix dip dests
  assume *: "∀ ip∈dom(dests). ip∈kD(rt) ∧ sqn rt ip ≤ the (dests ip)"
  and **: "∀ ip∈kD(rt). π₃(the (rt ip)) = unk ∨ Suc 0 ≤ sqn rt ip"
  have "∀ dip∈kD(rt). π₃(the (rt dip)) = unk ∨ Suc 0 ≤ sqn (invalidate rt dests) dip"
  proof
    fix dip
    assume "dip∈kD(rt)"
    with ** have "π₃(the (rt dip)) = unk ∨ Suc 0 ≤ sqn rt dip" ..
    thus "π₃ (the (rt dip)) = unk ∨ Suc 0 ≤ sqn (invalidate rt dests) dip"
    proof
      assume "π₃(the (rt dip)) = unk" thus ?thesis ..
    next
      assume "Suc 0 ≤ sqn rt dip"
      have "Suc 0 ≤ sqn (invalidate rt dests) dip"
      proof (cases "dip∈dom(dests)"
        assume "dip∈dom(dests)"
        with * have "sqn rt dip ≤ the (dests dip)" by simp
        with ⟨Suc 0 ≤ sqn rt dip⟩ have "Suc 0 ≤ the (dests dip)" by simp
        with ⟨dip∈dom(dests)⟩ ⟨dip∈kD(rt)⟩ [THEN kD_Some] show ?thesis
        unfolding invalidate_def sqn_def by auto
      next
        assume "dip∉dom(dests)"
        with ⟨Suc 0 ≤ sqn rt dip⟩ ⟨dip∈kD(rt)⟩ [THEN kD_Some] show ?thesis
        unfolding invalidate_def sqn_def by auto
      qed
      thus ?thesis by (rule disjI2)
    qed
  qed
} note solve_invalidate [simp] = this

show ?thesis
  by (inv_terms inv add: onl_invariant_sterms TT [OF aodv_wf adpreRT_welldefined]
        onl_invariant_sterms [OF aodv_wf dests_biggerRT_welldefined]
        [THEN invariant_restrict_inD]
        onl_invariant_sterms [OF aodv_wf osn_rreq]
        onl_invariant_sterms [OF aodv_wf dsn_rrep]
        simp add: proj3_inv proj2_eq_sqn)
qed
Proposition 7.13

**lemma rreq_rrep_sn_any_step_invariant:**

"paodv i \models A (recvmsg rreq_rrep_sn \rightarrow) onll \Gamma_{AODV} (\lambda(\_a, \_a, \_a). anycast rreq_rrep_sn a)"

**proof** -

have sqnf_kno: "paodv i \models onl \Gamma_{AODV} (\lambda(\xi, 1).

\{(l \in \{PReq\:-18..PReq\:-20\} \rightarrow sqnf (rt \xi) (dip \xi) = k\no\})"

by \{inv \_cters inv add: onl \_invariant_\_sterms \_TT [OF aodv \_wf addpreRT \_welldefined]\}

show ?thesis

by \{inv \_cters inv add: onl \_invariant_\_sterms \_TT [OF aodv \_wf addpreRT \_welldefined]

onl \_invariant_\_sterms [OF aodv \_wf sequence_number \_one \_or \_bigger

[THEN invariant \_restrict \_inD]]

onl \_invariant_\_sterms [OF aodv \_wf k\D \_unk \_or \_atleast \_one]

onl \_invariant_\_sterms \_TT [OF aodv \_wf sqnf \_kno]

onl \_invariant_\_sterms [OF aodv \_wf osn \_rreq]

onl \_invariant_\_sterms [OF aodv \_wf dsn \_rrep]]

(auto simp: proj2 \_eq \_sqn)

qed

Proposition 7.14

**lemma rreq_rrep_fresh_any_step_invariant:**

"paodv i \models A onll \Gamma_{AODV} (\lambda(\xi, 1, a, \_a, \_\_a). anycast (rreq_rrep_fresh (rt \xi)) a)"

**proof** -

have rreq_oip: "paodv i \models onl \Gamma_{AODV} (\lambda(\xi, 1).

\{(l \in \{PReq\:-3..PReq\:-9\} \cup \{PReq\:-17, PReq\:-30, PReq\:-32\}

\rightarrow oip \xi \in k\D (rt \xi)

\wedge (sqn (rt \xi) (oip \xi) > (osn \xi))

\vee (sqn (rt \xi) (oip \xi) = (osn \xi))

\wedge (hops (rt \xi) (oip \xi)) \leq Suc (hops \xi)

\wedge (flag (rt \xi) (oip \xi) = val)))"}

proof

fix 1 \ l \ pp \ p'

assume "((\xi, pp) \in reachable (paodv i) \ TT"

and "{PReq\:-2..PReq\:-7}\[\lambda\xi. \xi[rt :=

update (rt \xi) (oip \xi) (osn \xi, k\no, val, Suc (hops \xi), sip \xi, \{\})]

\}\ p' \in \_sterms \Gamma_{AODV} pp"

and "1' = PReq\:-3"

show "osn \xi < sqn (update (rt \xi) (oip \xi) (osn \xi, k\no, val, Suc (hops \xi), sip \xi, \{\})) (oip \xi)

\vee (sqn (update (rt \xi) (oip \xi) (osn \xi, k\no, val, Suc (hops \xi), sip \xi, \{\})) (oip \xi) = osn \xi

\wedge (hops (rt \xi) (oip \xi)) \leq Suc (hops \xi)

\wedge (flag (rt \xi) (oip \xi)) (osn \xi, k\no, val, Suc (hops \xi), sip \xi, \{\})) (oip \xi) = val)"

unfolding update_def by \{clarsimp split: option.split

(metis linorder_neqE_nat not_less)

qed

have rrep_prrep: "paodv i \models onl \Gamma_{AODV} (\lambda(\xi, 1).

\{(l \in \{PReq\:-2..PReq\:-7\} \rightarrow (dip \xi \in k\D (rt \xi)

\wedge sqn (rt \xi) (dip \xi) = dsn \xi

\wedge (hops (rt \xi) (dip \xi)) = Suc (hops \xi)

\wedge (flag (rt \xi) (dip \xi)) = val

\wedge (nhop (rt \xi) (dip \xi)) \in k\D (rt \xi))))"

by \{inv \_cters inv add: onl \_invariant_\_sterms [OF aodv \_wf rrep \_1 \_update \_changes]

onl \_invariant_\_sterms [OF aodv \_wf sip \_in \_k\D]

have rreq_oip_kD: "paodv i \models onl \Gamma_{AODV} (\lambda(\xi, 1). (l \in \{PReq\:-3..PReq\:-28\} \rightarrow oip \xi \in k\D (rt \xi)))"

by \{inv \_cters inv add: onl \_invariant_\_sterms \_TT [OF aodv \_wf addpreRT \_welldefined]\}

have rreq_dip_kD_oip_sqn: "paodv i \models onl \Gamma_{AODV} (\lambda(\xi, 1).

\{(l \in \{PReq\:-18..PReq\:-21\}

\rightarrow (dip \xi \in k\D (rt \xi)

\wedge (sqn (rt \xi) (oip \xi) > (osn \xi))

\vee (sqn (rt \xi) (oip \xi) = (osn \xi))

\wedge (hops (rt \xi) (oip \xi)) \leq Suc (hops \xi)

\wedge (flag (rt \xi) (oip \xi) = val))))"
by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf rreq_oip]
onl_invariant_sterms [OF aodv_wf addpreRT_welldefined])

show ?thesis
by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf rreq_oip]
onl_invariant_sterms [OF aodv_wf rreq_dip_in_vD_dip_eq_ip]
onl_invariant_sterms [OF aodv_wf rrep_prrep]
onl_invariant_sterms [OF aodv_wf rreq_oip_kD]
onl_invariant_sterms [OF aodv_wf rreq_dip_kD_oip_sqn])

qed

Proposition 7.15

lemma rerr_invalid_any_step_invariant:
"paodv i \|=\ _\_ Γ_{AODV} (λ((ξ, _), a, _). anycast (rerr_invalid (rt ξ)) a)"
proof -
  have dests_inv: "paodv i \|=\
onl Γ_{AODV} (λ((ξ, 1), (l). (l ∈ {PAodv-:15, PPkt-:7, PRreq-:11, PRreq-:24, PRrep-:10, PRerr-:1}
    → (∀ip∈dom(dests ξ). ip∈vD(rt ξ)))
  ∧ (l ∈ {PAodv-:16..PAodv-:19} ∪ {PPkt-:8..PPkt-:11} ∪ {PRreq-:12..PRreq-:15} ∪ {PRreq-:25..PRreq-:28} ∪ {PRrep-:11..PRrep-:14} ∪ {PRerr-:2..PRerr-:5} → (∀ip∈dom(dests ξ). ip∈iD(rt ξ)
    ∧ the (dests ξ ip) = sqn (rt ξ) ip))
  ∧ (l = PPkt-:14 → dip ξ∈iD(rt ξ))))" by inv_cterms (clarsimp split: split_if_asm option.split_asm simp: domIff)+

show ?thesis
by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf dests_inv])

qed

Proposition 7.16

Some well-definedness obligations are irrelevant for the Isabelle development:

1. In each routing table there is at most one entry for each destination: guaranteed by type.

2. In each store of queued data packets there is at most one data queue for each destination: guaranteed by structure.

3. Whenever a set of pairs \((rip, rsn)\) is assigned to the variable \(dests\) of type \(ip → sqn\), or to the first argument of the function \(rerr\), this set is a partial function, i.e., there is at most one entry \((rip, rsn)\) for each destination \(rip\): guaranteed by type.

lemma dests_vD_inc_sqn:
"paodv i \|=\ _\_ Γ_{AODV} (λ(ξ, 1). (l ∈ {PAodv-:15, PPkt-:7, PRreq-:11, PRreq-:24, PRrep-:10} → (∀ip∈dom(dests ξ). ip∈vD(rt ξ) ∧ the (dests ξ ip) = inc (sqn (rt ξ) ip))
  ∧ (l = PRerr-:1 → (∀ip∈dom(dests ξ). ip∈vD(rt ξ) ∧ the (dests ξ ip) > sqn (rt ξ) ip)))"
by inv_cterms (clarsimp split: split_if_asm option.split_asm simp: domIff)+

proof (inv_cterms inv add:
onl_invariant_sterms [OF aodv_wf dests_vD_inc_sqn [THEN invariant_restrict_inD]]
onl_invariant_sterms [OF aodv_wf hop_count_positive [THEN invariant_restrict_inD]]
onl_invariant_sterms [OF aodv_wf osn_rreq]
onl_invariant_sterms [OF aodv_wf dsn_rrep]
onl_invariant_sterms [OF aodv_wf addpreRT_welldefined [THEN invariant_restrict_inD]])

Proposition 7.27

lemma route_tables_fresher:
"paodv i \|=\ _\_ Γ_{AODV} (λ((ξ, _), (ξ', _)). (∀dip∈kD(rt ξ). rt ξ ⊆ dip rt ξ'))"
proof (inv_cterms inv add:
onl_invariant_sterms [OF aodv_wf dests_vD_inc_sqn [THEN invariant_restrict_inD]]
onl_invariant_sterms [OF aodv_wf hop_count_positive [THEN invariant_restrict_inD]]
onl_invariant_sterms [OF aodv_wf osn_rreq]
onl_invariant_sterms [OF aodv_wf dsn_rrep]
onl_invariant_sterms [OF aodv_wf addpreRT_welldefined [THEN invariant_restrict_inD]]

380
4.7 The quality increases predicate

theory D_Quality_Increases
imports D_Aodv_Predicates D_Fresher
begin

definition quality_increases :: "state ⇒ bool" where "quality_increases ξ ξ' ≡ (∀ dip ∈ kD(rt ξ). dip ∈ kD(rt ξ') ∧ rt ξ ⊑ dip rt ξ') ∧ (∀ dip. sqn (rt ξ) dip ≤ sqn (rt ξ') dip)"

lemma quality_increasesI [intro!]:
  assumes "∀ dip. dip ∈ kD(rt ξ) ⇒ dip ∈ kD(rt ξ')" and "∀ dip. dip ∈ kD(rt ξ); dip ∈ kD(rt ξ') ⇒ rt ξ ⊑ dip rt ξ'"
  shows "quality_increases ξ ξ'"
unfolding quality_increases_def using assms by clarsimp

lemma quality_increasesE [elim]:
  fixes dip
  assumes "quality_increases ξ ξ'" and "dip ∈ kD(rt ξ)" and "[ dip ∈ kD(rt ξ'); rt ξ ⊑ dip rt ξ'; sqn (rt ξ) dip ≤ sqn (rt ξ') dip ] ⇒ R dip ξ'"
  shows "R dip ξ''"
using assms unfolding quality_increases_def by clarsimp

lemma quality_increases_rt_fresherD [dest]:
  fixes ip

assumes "quality_increases ξ ξ'" and "ip ∈ kD(rt ξ)"
shows "rt ξ ⊑p rt ξ'"
using assms by auto

lemma quality_increases_sqnE [elim]:
  fixes dip
  assumes "quality_increases ξ ξ'"
  and "sqn (rt ξ) dip ≤ sqn (rt ξ') dip ⊃ R dip ξ ξ'"
  shows "R dip ξ ξ'"
  using assms unfolding quality_increases_def by clarsimp

lemma quality_increases_refl [intro, simp]: "quality_increases ξ ξ"
  by rule simp_all

lemma strictly_fresher_quality_increases_right [elim]:
  fixes σ σ' dip
  assumes "rt (σ i) ⊏ dip rt (σ' nhip)"
  and qinc: "quality_increases (σ nhip) (σ' nhip)"
  and "dip ∈ kD(rt (σ nhip))"
  shows "rt (σ i) ⊏ dip rt (σ' nhip)"
  proof -
    from qinc have "rt (σ nhip) ⊑ dip rt (σ' nhip)" using ⟨dip ∈ kD(rt (σ nhip))⟩
    by auto
    with ⟨rt (σ i) ⊏ dip rt (σ nhip)⟩ show ?thesis ..
  qed

lemma kD_quality_increases [elim]:
  assumes "i ∈ kD(rt ξ)"
  and "quality_increases ξ ξ'"
  shows "i ∈ kD(rt ξ')"
  using assms by auto

lemma kD_nsqn_quality_increases [elim]:
  assumes "i ∈ kD(rt ξ)"
  and "quality_increases ξ ξ'"
  shows "i ∈ kD(rt ξ') ∧ nsqn (rt ξ) i ≤ nsqn (rt ξ') i"
  proof -
    from assms have "i ∈ kD(rt ξ')" ..
    moreover with assms have "rt ξ ⊑ i rt ξ'" by auto
    ultimately have "nsqn (rt ξ) i ≤ nsqn (rt ξ') i" using ⟨i ∈ kD(rt ξ)⟩ by (erule(2) rt_fresher_imp_nsqn_le)
    with ⟨i ∈ kD(rt ξ')⟩ show ?thesis ..
  qed

lemma nsqn_quality_increases [elim]:
  assumes "i ∈ kD(rt ξ)"
  and "quality_increases ξ ξ'"
  shows "nsqn (rt ξ) i ≤ nsqn (rt ξ') i"
  using assms by (rule kD_nsqn_quality_increases [THEN conjunct2])

lemma kD_nsqn_quality_increases_trans [elim]:
  assumes "i ∈ kD(rt ξ)"
  and "s ≤ nsqn (rt ξ) i"
  and "quality_increases ξ ξ'"
  shows "i ∈ kD(rt ξ') ∧ s ≤ nsqn (rt ξ') i"
  proof
    from ⟨i ∈ kD(rt ξ)⟩ and ⟨quality_increases ξ ξ'⟩ have "nsqn (rt ξ) i ≤ nsqn (rt ξ') i" ..
    next
    from ⟨i ∈ kD(rt ξ)⟩ and ⟨quality_increases ξ ξ'⟩ have "nsqn (rt ξ) i ≤ nsqn (rt ξ') i" ..
    with ⟨s ≤ nsqn (rt ξ) i⟩ show "s ≤ nsqn (rt ξ') i" by (rule le_trans)
  qed

lemma nsqn_quality_increases_nsqn_lt_lt [elim]:

382
assumes "i ∈ kD(rt ξ)"
and "quality_increases ξ ξ'"
and "s < nsqn (rt ξ) i"
shows "s < nsqn (rt ξ') i"

proof -
from assms(1-2) have "nsqn (rt ξ) i ≤ nsqn (rt ξ') i" ..
with "s < nsqn (rt ξ) i" show "s < nsqn (rt ξ') i" by simp
qed

lemma nsqn_quality_increases_dhops [elim]:
assumes "i ∈ kD(rt ξ)"
and "quality_increases ξ ξ'"
and "nsqn (rt ξ) i = nsqn (rt ξ') i"
shows "the (dhops (rt ξ) i) ≥ the (dhops (rt ξ') i)"
using assms unfolding quality_increases_def
by (clarsimp) (drule(1) bspec, clarsimp simp: rt_fresher_def2)

lemma nsqn_quality_increases_nsqn_eq_le [elim]:
assumes "i ∈ kD(rt ξ)"
and "quality_increases ξ ξ'"
and "s = nsqn (rt ξ) i"
shows "s < nsqn (rt ξ') i ∨ (s = nsqn (rt ξ') i ∧ the (dhops (rt ξ) i) ≥ the (dhops (rt ξ') i))"
using assms by (metis nat_less_le nsqn_quality_increases nsqn_quality_increases_dhops)

lemma quality_increases_rreq_rrep_props [elim]:
fixes sn ip hops sip
assumes qinc: "quality_increases (σ sip) (σ' sip)"
and "1 ≤ sn"
and "*: ip ∈ kD(rt (σ sip)) ∧ sn ≤ nsqn (rt (σ sip)) ip
∧ (nsqn (rt (σ sip)) ip = sn
→ (the (dhops (rt (σ sip)) ip) ≤ hops
∨ the (flag (rt (σ sip)) ip) = inv))"
shows "ip ∈ kD(rt (σ' sip)) ∧ sn ≤ nsqn (rt (σ' sip)) ip
∧ (nsqn (rt (σ' sip)) ip = sn
→ (the (dhops (rt (σ' sip)) ip) ≤ hops
∨ the (flag (rt (σ' sip)) ip) = inv))"
(is "_ ∧ ?nsqnafter")
proof -
from * obtain "ip ∈ kD(rt (σ sip))" and "sn ≤ nsqn (rt (σ sip)) ip" by auto
from ⟨quality_increases (σ sip) (σ' sip)⟩
have "snq (rt (σ sip)) ip ≤ nsqn (rt (σ' sip)) ip" ..
from ⟨quality_increases (σ sip) (σ' sip)⟩ and ⟨ip ∈ kD (rt (σ sip))⟩
have "ip ∈ kD (rt (σ' sip))" ..
from "sn ≤ nsqn (rt (σ sip)) ip" have ?nsqnafter
proof
assume "sn < nsqn (rt (σ sip)) ip"
also from ⟨ip ∈ kD(rt (σ sip))⟩ and ⟨quality_increases (σ sip) (σ' sip)⟩
have "... ≤ nsqn (rt (σ' sip)) ip" ..
finally have "sn < nsqn (rt (σ' sip)) ip" .
thus ?thesis by simp
next
assume "sn = nsqn (rt (σ sip)) ip"
with ⟨ip ∈ kD(rt (σ sip))⟩ and ⟨quality_increases (σ sip) (σ' sip)⟩
have "sn < nsqn (rt (σ' sip)) ip
∨ (sn = nsqn (rt (σ' sip)) ip
∧ the (dhops (rt (σ' sip)) ip) ≤ the (dhops (rt (σ sip)) ip))" ..
hence "sn < nsqn (rt (σ' sip)) ip
∨ (nsqn (rt (σ' sip)) ip = sn ∧ (the (dhops (rt (σ' sip)) ip) ≤ hops
∨ the (flag (rt (σ' sip)) ip) = inv))"
proof
assume "sn < nsqn (rt (σ sip)) ip" thus ?thesis ..
next
assume "sn = nsqn (rt (σ' sip)) ip ∧ the (dhops (rt (σ sip)) ip) ≥ the (dhops (rt (σ' sip)) ip)"

hence "sn = nsqn (rt (σ' sip)) ip" and "the (dhops (rt (σ' sip)) ip) ≤ the (dhops (rt (σ sip)) ip)" by auto

from * and ⟨sn = nsqn (rt (σ sip)) ip⟩ have "the (dhops (rt (σ sip)) ip) ≤ hops ∨ the (flag (rt (σ sip)) ip) = inv"

by simp
thus ?thesis

proof
assume "the (dhops (rt (σ sip)) ip) ≤ hops"
with ⟨the (dhops (rt (σ' sip)) ip) ≤ the (dhops (rt (σ sip)) ip)⟩ have "the (dhops (rt (σ' sip)) ip) ≤ hops" by simp

next
assume "the (flag (rt (σ sip)) ip) = inv"
with ⟨ip ∈ kD (rt (σ' sip))⟩ have "nsqn (rt (σ' sip)) ip = sqn (rt (σ sip)) ip - 1" ..

with ⟨sn ≥ 1⟩ and ⟨sn = nsqn (rt (σ sip)) ip⟩ have "sqn (rt (σ sip)) ip > 1" by simp

with ⟨sqn (rt (σ sip)) ip > 1⟩ and ⟨nsqn (rt (σ' sip)) ip = sn⟩ show ?thesis by simp

next
assume "ip ∈ DV (rt (σ' sip))"

hence "the (flag (rt (σ' sip)) ip) = inv" ..

with ⟨nsqn (rt (σ' sip)) ip = sn⟩ have "nsqn (rt (σ' sip)) ip ≥ sqn (rt (σ sip)) ip" by simp

with ⟨nsqn (rt (σ' sip)) ip > 1⟩ have "nsqn (rt (σ' sip)) ip > sqn (rt (σ sip)) ip - 1" by simp

with ⟨nsqn (rt (σ' sip)) ip = sqn (rt (σ sip)) ip - 1⟩ have "nsqn (rt (σ' sip)) ip > nsqn (rt (σ sip)) ip" by simp

with ⟨sn = nsqn (rt (σ sip)) ip⟩ have "nsqn (rt (σ' sip)) ip > sn" by simp
thus ?thesis ..

qed

lemma quality_increases_rreq_rrep_props':
fixes sn ip hops sip
assumes "∀ j. quality_increases (σ j) (σ' j)"
and "1 ≤ sn"
and *: "ip ∈ kD (rt (σ sip)) ∧ sn ≤ nsqn (rt (σ sip)) ip ∧ (nsqn (rt (σ sip)) ip = sn → (the (dhops (rt (σ sip)) ip) ≤ hops ∨ the (flag (rt (σ sip)) ip) = inv))"

shows "ip ∈ kD (rt (σ' sip)) ∧ sn ≤ nsqn (rt (σ' sip)) ip ∧ (nsqn (rt (σ' sip)) ip = sn → (the (dhops (rt (σ' sip)) ip) ≤ hops ∨ the (flag (rt (σ' sip)) ip) = inv))"

proof -
from assms(1) have "quality_increases (σ sip) (σ' sip)" ..
thus ?thesis using assms(2-3) by (rule quality_increases_rreq_rrep_props)
lemma rteq_quality_increases:
  assumes "∀ j. j ≠ i → quality_increases (σ j) (σ' j)"
  and "rt (σ' i) = rt (σ i)"
  shows "∀ j. quality_increases (σ j) (σ' j)"
  using assms by clarsimp (metis order_refl quality_increasesI rt_fresher_refl)

definition msg_fresh :: "((ip ⇒ state) ⇒ msg) ⇒ bool"
  where "msg_fresh σ m ≡
    case m of
      Rreq hops rreqid dip dsn oip osn sip handled ↦
        (osn ≥ 1 ∧ (sip ≠ oip → oip∈kD(rt (σ sip))) ∧ nsqn (rt (σ sip)) oip ≥ osn
        ∧ (nsqn (rt (σ sip)) oip = osn
        → (hops ≥ the (dhops (rt (σ sip)) oip)
            ∨ the (flag (rt (σ sip)) oip) = inv)))
      Rrep hops dip dsn oip sip ⇒
        (dsn ≥ 1 ∧ (sip ≠ dip → dip∈kD(rt (σ sip))) ∧ nsqn (rt (σ sip)) dip ≥ dsn
        ∧ (nsqn (rt (σ sip)) dip = dsn
        → (hops ≥ the (dhops (rt (σ sip)) dip)
            ∨ the (flag (rt (σ sip)) dip) = inv)))
      Rerr destsc sipc ⇒
        (∀ ripc∈dom(destsc). (ripc∈kD(rt (σ sipc))
          ∧ the (destsc ripc) - 1 ≤ nsqn (rt (σ sipc)) ripc))
      _ ⇒ True"

lemma msg_fresh [simp]:
  "∀ hops rreqid dip dsn oip osn sip handled.
    msg_fresh σ (Rreq hops rreqid dip dsn oip osn sip handled) =
      (osn ≥ 1 ∧ (sip ≠ oip → oip∈kD(rt (σ sip)))
       ∧ nsqn (rt (σ sip)) oip ≥ osn
       ∧ (nsqn (rt (σ sip)) oip = osn
       → (hops ≥ the (dhops (rt (σ sip)) oip)
           ∨ the (flag (rt (σ sip)) oip) = inv))))"

lemma msg_fresh_inc_sn [simp, elim]:
  "∀ hops dip dsn oip sip.
    msg_fresh σ (Rreq hops dip dsn oip sip) =
      (dsn ≥ 1 ∧ (sip ≠ dip → dip∈kD(rt (σ sip)))
       ∧ nsqn (rt (σ sip)) dip ≥ dsn
       ∧ (nsqn (rt (σ sip)) dip = dsn
       → (hops ≥ the (dhops (rt (σ sip)) dip)
           ∨ the (flag (rt (σ sip)) dip) = inv))))"

lemma msg_fresh_def by simp_all

unfolding lemma msg_fresh_def [simp, elim]:
  "msg_fresh σ m ⇒ rreq_rrep_sn m"
  by (cases m) simp_all

lemma recv_msg_fresh_inc_sn [simp, elim]:
  "orecvmsg (msg_fresh) σ m ⇒ recvmsg rreq_rrep_sn m"
  by (cases m) simp_all

lemma rreq_nsqn_is_fresh [simp]:
  fixes σ msg hops rreqid dip dsn oip osn sip handled
  assumes "rreq_rrep_sn (rt (σ sip)) (Rreq hops rreqid dip dsn oip osn sip handled)"
  and "rreq_sn (Rreq hops rreqid dip dsn oip osn sip handled)"
  shows "msg_fresh σ (Rreq hops rreqid dip dsn oip osn sip handled)"
  (is "msg_fresh σ ?msg")
  proof -
    let ?rt = "rt (σ sip)"
    from assms(2) have "1 ≤ osn" by simp
    thus ?thesis
    unfolding msg_fresh_def
    385
proof (simp only: msg.case, intro conjI impI)
  assume "sip ≠ oip"
  with assms(1) show "oip ∈ kD(?rt)" by simp
next
  assume "sip ≠ oip"
  and "nsqn ?rt oip = osn"
  show "the (dhops ?rt oip) ≤ hops ∨ the (flag ?rt oip) = inv"
  proof (cases "oip ∈ vD(?rt)"
  assume "oip ∈ vD(?rt)"
  hence "nsqn ?rt oip = sqn ?rt oip" ..
  with ⟨nsqn ?rt oip = osn⟩ have "sqn ?rt oip = osn" by simp
  with assms(1) and ⟨sip ≠ oip⟩ have "the (dhops ?rt oip) ≤ hops"
    by simp
  thus ?thesis ..
next
  assume "oip ∈ vD(?rt)"
  moreover from assms(1) and ⟨sip ≠ oip⟩ have "oip ∈ kD(?rt)" by simp
  ultimately have "oip ∈ iD(?rt)" by auto
  hence "the (flag ?rt oip) = inv" ..
  thus ?thesis ..
qed
next
  assume "sip ≠ oip"
  with assms(1) have "osn ≤ sqn ?rt oip" by auto
  thus "osn ≤ nsqn (rt (σ sip)) oip" (is "msg_fresh ?msg")
  proof -
    let ?rt = "rt (σ sip)"
    from assms have "sip ≠ dip → dip ∈ kD(?rt) ∧ sqn ?rt dip = dsn ∧ the (flag ?rt dip) = val"
      by simp
    hence "sip ≠ dip → dip ∈ kD(?rt) ∧ nsqn ?rt dip ≥ dsn"
      by clarsimp
    with assms show "msg_fresh ?msg"
      by clarsimp
  qed
  qed simp

lemma rrep_nsqn_is_fresh [simp]:
  fixes σ msg hops dip dsn oip sip
  assumes "rreq_rrep_fresh (rt (σ sip)) (Rrep hops dip dsn oip sip)"
    and "rreq_rrep_sn (Rrep hops dip dsn oip sip)"
  shows "msg_fresh σ (Rrep hops dip dsn oip sip)"
    (is "msg_fresh σ ?msg")
  proof -
    let ?rt = "rt (σ sip)"
    from assms have "sip ≠ dip → dip ∈ kD(?rt) ∧ sqn ?rt dip = dsn ∧ the (flag ?rt dip) = val"
      by simp
    hence "sip ≠ dip → dip ∈ kD(?rt) ∧ nsqn ?rt dip ≥ dsn"
      by clarsimp
    with assms show "msg_fresh σ ?msg"
      by clarsimp
  qed

lemma rerr_nsqn_is_fresh [simp]:
  fixes σ msg dests sip
  assumes "rerr_invalid (rt (σ sip)) (Rerr dests sip)"
  shows "msg_fresh σ (Rerr dests sip)"
    (is "msg_fresh σ ?msg")
  proof -
let \( \texttt{rt} = "rt (\sigma \texttt{sip})" \)

from \( \texttt{assms} \) have \( "(\forall \texttt{rip} \in \text{dom(dests)}. (\texttt{rip} \in \text{I}(\texttt{rt} (\sigma \texttt{sip}))) \wedge \text{the} (\text{dests} \texttt{rip}) = \text{sqn} (\texttt{rt} (\sigma \texttt{sip})) \texttt{rip})" \)

by clarsimp

have \( "(\forall \texttt{rip} \in \text{dom(dests)}. (\texttt{rip} \in \text{K}(\texttt{rt} (\sigma \texttt{sip}))) \wedge \text{the} (\text{dests} \texttt{rip}) - 1 \leq \text{nsqn} (\texttt{rt} (\sigma \texttt{sip})) \texttt{rip})" \)

proof
fix \( \texttt{rip} \)
assume "\( \texttt{rip} \in \text{dom dests} \)"
with \( \ast \) have "\( \texttt{rip} \in \text{I}(\texttt{rt} (\sigma \texttt{sip})) \)" and "\( \text{the} (\text{dests} \texttt{rip}) = \text{sqn} (\texttt{rt} (\sigma \texttt{sip})) \texttt{rip} \)"
by auto
from this(2)
have "\( \text{the} (\text{dests} \texttt{rip}) - 1 = \text{sqn} (\texttt{rt} (\sigma \texttt{sip})) \texttt{rip} - 1 \)"
by simp
also have "... \leq \text{nsqn} (\texttt{rt} (\sigma \texttt{sip})) \texttt{rip} \)" by (rule \texttt{sqn_nsqn})
finally have "\( \text{the} (\text{dests} \texttt{rip}) - 1 \leq \text{nsqn} (\texttt{rt} (\sigma \texttt{sip})) \texttt{rip} \)".

with \( \texttt{rip} \in \text{I}(\texttt{rt} (\sigma \texttt{sip})) \)
show "\( \texttt{rip} \in \text{K}(\texttt{rt} (\sigma \texttt{sip})) \wedge \text{the} (\text{dests} \texttt{rip}) - 1 \leq \text{nsqn} (\texttt{rt} (\sigma \texttt{sip})) \texttt{rip} \)"
by clarsimp
qed

thus "\texttt{msg_fresh} (\sigma \ ?\texttt{msg})" by simp
qed

lemma quality_increases_msg_fresh [elim]:
assumes qinc: "(\forall j. \text{quality_increases} (\sigma j) (\sigma' j))"
and "\texttt{msg_fresh} (\sigma m)"
shows "\texttt{msg_fresh} (\sigma' m)"
using \( \texttt{assms}(2) \)
proof (cases \( m \))
fix \( \texttt{hops} \ \texttt{rreqid} \ \texttt{dsn} \ \texttt{dsk} \ \texttt{oip} \ \texttt{osn} \ \texttt{sip} \ \texttt{handled} \)
assume \[\texttt{simp}]: "\( m = \texttt{Rreq} \texttt{hops} \texttt{rreqid} \texttt{dsn} \texttt{dsk} \texttt{oip} \texttt{osn} \texttt{sip} \texttt{handled} \)"
and "\texttt{msg_fresh} (\sigma m)"
then have "\( \texttt{osn} \geq 1 \)" and "\( \texttt{sip} = \texttt{oip} \lor (\texttt{oip} \in \text{K}(\texttt{rt} (\sigma \texttt{sip}))) \wedge \text{osn} \leq \text{nsqn} (\texttt{rt} (\sigma \texttt{sip})) \texttt{oip} \wedge (\text{nsqn} (\texttt{rt} (\sigma \texttt{sip})) \texttt{oip} = \text{osn} \rightarrow (\text{the} (\text{dhop}(\text{rt} (\sigma \texttt{sip})) \texttt{oip}) \leq \texttt{hops} \lor \text{the} (\text{flag} (\text{rt} (\sigma \texttt{sip})) \texttt{oip}) = \text{inv}))""
by auto
from this(2)
show \?thesis
proof
assume "\( \texttt{sip} = \texttt{oip} \)" with \( \texttt{osn} \geq 1 \) show \?thesis by simp
next
assume "\( \texttt{oip} \in \text{K}(\texttt{rt} (\sigma \texttt{sip})) \) \wedge \text{osn} \leq \text{nsqn} (\texttt{rt} (\sigma \texttt{sip})) \texttt{oip} \wedge (\text{nsqn} (\texttt{rt} (\sigma \texttt{sip})) \texttt{oip} = \text{osn} \rightarrow (\text{the} (\text{dhop}(\text{rt} (\sigma \texttt{sip})) \texttt{oip}) \leq \texttt{hops} \lor \text{the} (\text{flag} (\text{rt} (\sigma \texttt{sip})) \texttt{oip}) = \text{inv}))""
moreover from \( \texttt{qinc} \) have "\texttt{quality_increases} (\sigma \texttt{sip}) (\sigma' \texttt{sip})" ..
ultimately have "\( \texttt{oip} \in \text{K}(\texttt{rt} (\sigma' \texttt{sip})) \) \wedge \text{osn} \leq \text{nsqn} (\texttt{rt} (\sigma' \texttt{sip})) \texttt{oip} \wedge (\text{nsqn} (\texttt{rt} (\sigma' \texttt{sip})) \texttt{oip} = \text{osn} \rightarrow (\text{the} (\text{dhop}(\text{rt} (\sigma' \texttt{sip})) \texttt{oip}) \leq \texttt{hops} \lor \text{the} (\text{flag} (\text{rt} (\sigma' \texttt{sip})) \texttt{oip}) = \text{inv}))""
using \( \texttt{osn} \geq 1 \) by (rule \texttt{quality_increases_rreq_rrep_props [rotated 2]})
with \( \texttt{osn} \geq 1 \) show "\texttt{msg_fresh} (\sigma' \texttt{m})" by clarsimp
qed

next
fix \( \texttt{hops} \ \texttt{dip} \ \texttt{dsn} \ \texttt{oip} \ \texttt{sip} \)
assume \[\texttt{simp}]: "\( m = \texttt{Rrep} \texttt{hops} \texttt{dip} \texttt{dsn} \texttt{oip} \texttt{sip} \)"
and "\texttt{msg_fresh} (\sigma m)"
then have "\( \texttt{dsn} \geq 1 \)" and "\( \texttt{sip} = \texttt{dip} \lor (\texttt{dip} \in \text{K}(\texttt{rt} (\sigma \texttt{sip})) \wedge \text{dsn} \leq \text{nsqn} (\texttt{rt} (\sigma \texttt{sip})) \texttt{dip} \wedge (\text{nsqn} (\texttt{rt} (\sigma \texttt{sip})) \texttt{dip} = \text{dsn} \rightarrow (\text{the} (\text{dhop}(\text{rt} (\sigma \texttt{sip})) \texttt{dip}) \leq \texttt{hops} \lor \text{the} (\text{flag} (\text{rt} (\sigma \texttt{sip})) \texttt{dip}) = \text{inv}))""
by auto

from this(2) show "?thesis"

proof
  assume "sip = dip" with \( dsn \geq 1 \) show \( ?thesis \) by simp

next
  assume "dip \in kD(rt (\sigma \ sip)) \land dsn \leq nsqn (rt (\sigma \ sip)) dip \\
    \land (nsqn (rt (\sigma \ sip)) dip = dsn \\
    \rightarrow (the (dhops (rt (\sigma \ sip)) dip) \leq hops \\
    \lor the (flag (rt (\sigma \ sip)) dip) = inv))"

moreover from qinc have "quality_increases (\sigma \ sip) (\sigma' \ sip)" ..
ultimately have "dip \in kD(rt (\sigma' \ sip)) \land dsn \leq nsqn (rt (\sigma' \ sip)) dip \\
    \land (nsqn (rt (\sigma' \ sip)) dip = dsn \\
    \rightarrow (the (dhops (rt (\sigma' \ sip)) dip) \leq hops \\
    \lor the (flag (rt (\sigma' \ sip)) dip) = inv))"

using \( dsn \geq 1 \) by (rule quality_increases_rreq_rrep_props [rotated 2])
with \( dsn \geq 1 \) show "msg_fresh \sigma' m"
  by clarsimp

qed

next

fix dests sip
  assume [simp]: "m = Rerr dests sip"
  and "msg_fresh \sigma m"
  then have *: "\( \forall rip \in \text{dom(dests)}. \ rip \in kD(rt (\sigma \ sip)) \\
    \land the (dests rip) - 1 \leq nsqn (rt (\sigma \ sip)) rip \)"
    by simp

have "\( \forall rip \in \text{dom(dests)}. \ rip \in kD(rt (\sigma' \ sip)) \\
    \land the (dests rip) - 1 \leq nsqn (rt (\sigma' \ sip)) rip \)" ..
  by (drule(1) bspec, clarsimp)+

moreover from qinc have "quality_increases (\sigma \ sip) (\sigma' \ sip)" by simp
ultimately show "\( \forall rip \in \text{dom(dests)}. \ rip \in kD(rt (\sigma' \ sip)) \\
    \land the (dests rip) - 1 \leq nsqn (rt (\sigma' \ sip)) rip \)" ..
  by simp

thus ?thesis by simp

qed simp_all

end

4.8 The ‘open’ AODV model

theory D_OAodv
imports D_Aodv ../../../AWN/OAWN_SOS_Labels ../../../AWN/OAWN_Convert
begin

Definitions for stating and proving global network properties over individual processes.

definition \( \sigma_{AODV}' \) :: "((ip \Rightarrow state) \times ((state, msg, pseqp, pseqp label) seqp)) set"
where "\( \sigma_{AODV}' \equiv \{\lambda i. \ aodv_INIT \ i, \ \Gamma_{AODV} PAodv\}\)"

abbreviation opaodv :: "ip \Rightarrow ((ip \Rightarrow state) \times (state, msg, pseqp, pseqp label) seqp, msg seq_action) automaton"
where "opaodv \ i \equiv \{(iop => state) \times (state, msg, pseqp, pseqp label) seqp, msg seq_action\} automaton"

lemma initiali_aodv [intro!, simp]: "initiali i (init (opaodv i)) (init (paodv i))"
  unfolding \( \sigma_{AODV}' \_def = \sigma_{AODV}' \_def \) by rule simp_all

lemma oadv_control_within [simp]: "control_within \( \Gamma_{AODV} \) (init (opaodv i))"
  unfolding \( \sigma_{AODV}' \_def = \sigma_{AODV}' \_def \) by (rule control_withinI) (auto simp del: \( \Gamma_{AODV}' \_sims)"

lemma \( \sigma_{AODV}' \_labels \) [simp]: "(\sigma, p) \in \sigma_{AODV}' \quad \Rightarrow \quad \text{labels} \( \Gamma_{AODV} \) p = \{\text{PAodv} : 0\}"
  unfolding \( \sigma_{AODV}' \_def = \sigma_{AODV}' \_def \) by simp

lemma oadv_init_kD_empty [simp]:
unfolding $\sigma_AODV'$ by simp

lemma oaodv_init_vD_empty [simp]:
"$(\sigma, p) \in \sigma_AODV' \implies vD (rt (\sigma \ i)) = {}$"
unfolding $\sigma_AODV'$ by simp

lemma oaodv_trans: "trans (opaodv i) = oseqp_sos $\Gamma_AODV$ i"
by simp

declare oseq_invariant_ctermsI [OF aodv_wf aoadv_control_within aoadv_simple_labels aoadv_trans, cterms_intros]
oseq_step_invariant_ctermsI [OF aodv_wf aoadv_control_within aoadv_simple_labels aoadv_trans, cterms_intros]

end

4.9 Global invariant proofs over sequential processes

theory D_Global_Invariants
imports D_Seq_Invariants
D_Aodv_Predicates
D_Fresher
D_Quality_Increases
"../../../AWN/OAWN_Convert"
D_OAodv
begin

lemma other_quality_increases [elim]:
assumes "other quality_increases I $\sigma$ $\sigma'$"
shows "$\forall j. quality_increases ($\sigma$ j) ($\sigma'$ j)"
using assms by (rule, clarsimp) (metis quality_increases_refl)

lemma weaken_otherwith [elim]:
fixes m
assumes "otherwith P I (orecvmsg Q) $\sigma$ $\sigma'$ a"
and weakenP: "$\forall \sigma m. P \sigma m \implies P' \sigma m" 
and weakenQ: "$\forall \sigma m. Q \sigma m \implies Q' \sigma m"
shows "otherwith P' I (orecvmsg Q') $\sigma$ $\sigma'$ a"
proof
fix j
assume "j /\ I"
with * have "P ($\sigma$ j) ($\sigma'$ j)" by auto
thus "P' ($\sigma$ j) ($\sigma'$ j)" by (rule weakenP)
next
from * have "orecvmsg Q a" by auto
thus "orecvmsg Q' a" 
by rule (erule weakenQ)
qed

lemma oreceived_msg_inv:
assumes other: "$\forall \sigma $\sigma' m. [P \sigma m; other Q {i} \sigma \sigma'] \Rightarrow P' \sigma m" 
and local: "$\forall \sigma m. P \sigma m \Rightarrow P (\sigma (i := \sigma i(msg := m))) m"
shows "opaodv i = (otherwith Q {i} (orecvmsg P), other Q {i} ->) \Gamma_AODV (\lambda (\sigma, l). l \in \{PAodv-:1\} \Rightarrow P \sigma (msg (\sigma i)))" 
proof (inv_cterms, intro impl)
fix $\sigma \gamma l$
assume "l = PAodv-:1 \Rightarrow P \sigma (msg (\sigma i))"
and "l = PAodv-:1" 
and "other Q {i} $\sigma$ $\sigma'$"
from this(1-2) have "P $\sigma$ (msg (\sigma i))" ..
hence "P $\sigma'$ (msg (\sigma i))" using (other Q {i} i $\sigma$ $\sigma'$) 
by (rule other)
moreover from (other Q {i} i $\sigma$ $\sigma'$) have "$\sigma'$ i = $\sigma$ i" ..
ultimately show "P $\sigma'$ (msg (\sigma i))" by simp
next

fix σ' msg
assume "otherwith Q {i} (orecmmsg P) σ σ' (receive msg)"
and "σ' i = σ i(msg := msg)"
from this(1) have "P σ msg"
and "∀ j. j ≠ i → Q (σ j) (σ' j)" by auto
from this(1) have "P (σ i := σ i(msg := msg)) msg" by (rule local)
thus "P σ' msg"
proof (rule other)
from ⟨σ' i = σ i(msg := msg)⟩ and ⟨∀ j. j ≠ i → Q (σ j) (σ' j)⟩
show "other Q {i} (σ i := σ i(msg := msg)) σ'" by - (rule otherI, auto)
qed

qed

(Equivalent to) Proposition 7.27

lemma local_quality_increases:
"paodv i =⇒ (reccmsg rreq_rrep_sn →) onll Γ_AODV (λ((ξ, _, _), (ξ', _)). quality_increases ξ ξ')"
proof (rule step_invariantI)
fix s a s'
assume sr: "s ∈ reachable (paodv i) (reccmsg rreq_rrep_sn)"
and tr: "(s, a, s') ∈ trans (paodv i)"
and rm: "reccmsg rreq_rrep_sn a"
from sr have srTT: "s ∈ reachable (paodv i) TT" ..
from route_tables_fresher sr tr rm
have "onll Γ_AODV (λ((ξ, _, _), (ξ', _)). ∀ dip∈kD (rt ξ). rt ξ ⊑ dip (rt ξ')) (s, a, s')" by (rule step_invariantD)
moreover from known_destinations_increase srTT tr TT_True
have "onll Γ_AODV (λ((ξ, _, _), (ξ', _)). kD (rt ξ) ⊑ kD (rt ξ')) (s, a, s')" by (rule step_invariantD)
moreover from sqns_increase srTT tr TT_True
have "onll Γ_AODV (λ((ξ, _, _), (ξ', _)). ∀ ip. sqn (rt ξ) ip ≤ sqn (rt ξ') ip) (s, a, s')" by (rule step_invariantD)
ultimately show "onll Γ_AODV (λ((ξ, _, _), (ξ', _)). quality_increases ξ ξ') (s, a, s')" unfolding onll_def by auto
qed

lemmas olocal_quality_increases =
open_seq_step_invariant [OF local_quality_increases initiali_aodv oaodv_trans aodv_trans, simplified seqll_onll_swap]

lemma equality_increases:
"opaodv i =⇒ (otherwith quality_increases {i} (orecmmsg (λ_. rreq_rrep_sn)), other quality_increases {i} (i) →)
onll Γ_AODV (λ((ξ, _, _), (ξ', _)). ∀ j. quality_increases (σ j) (σ' j))"
(is "_ =⇒ (?S, _ →) _")
proof (rule onll_ostep_invariantI, simp)
fix σ p l a σ' p' l' 
assume or: "("σ, p) ∈ reachable (opaodv i) ?S (other quality_increases {i})"
and 11: "l ∈ labels Γ_AODV p"
and ?S σ σ' a"
and tr: "("(σ, p), a, (σ', p')\) ∈ oseqp sos Γ_AODV i"
and 11': "1' ∈ labels Γ_AODV p'"
from this(1-3) have "orecmmsg (λ_. rreq_rrep_sn) σ a" by (auto dest!: reachable_weakenE [where QS="act (reccmsg rreq_rrep_sn)"
and QU="other quality_increases {i}"]
other_with_actionD)
with or have orw: "("σ, p) ∈ reachable (opaodv i) (act (reccmsg rreq_rrep_sn))
(other quality_increases {i})" by - (erule reachable_weakenE, auto)
with \( tr \) 11 11' and \( \text{recvmsg} \ (\lambda \_ . \text{rreq_rrep_sn}) \) \( \sigma \) \( a \), have "quality_increases \( (\sigma \ i) \ (\sigma' \ i)\)"

by - (drule \( \text{onll} \_ \text{ostep_invariantD} \) [OF \( \text{oLocal} \_ \text{quality_increases} \)], \( \text{auto simp: seqll_def} \)

with \( \exists \) \( \sigma \) \( \sigma' \) \( a \) show "\( \forall \ j. \) quality_increases \( (\sigma \ j) \ (\sigma' \ j)\)"

by (auto dest!: \( \text{otherwith} \_ \text{syncD} \))

\( \text{qed} \)

lemma \( \text{rreq_rrep} \_ \text{nsqf} \_ \text{fresh} \_ \text{any_step_invariant} \):
"\( \text{opaodv} \ \text{i} \:=\text{A} \) \( \text{act} \ \text{(recvmsg} \ \text{rreq_rrep_sn}) \) \( \text{other} \ \text{A} \) \( \{i\} \rightarrow \)

\( \text{onll} \ \Gamma \_ \text{AODV} \ (\lambda \ ((\sigma, \_), \ a, \ _). \) \text{anycast} \ (\text{msgfresh} \ \sigma) \) \( a \)"

proof (rule \( \text{ostep_invariantI} \), \( \text{simp del: act_simp} \))

fix \( \sigma \) \( a \) \( \sigma' \) \( p' \)
assume or: "\( (\sigma, \ p) \) \in \text{oreachable} (\text{opaodv} \ \text{i}) \ (\text{act} \ \text{(recvmsg} \ \text{rreq_rrep_sn})) \ (\text{other} \ \text{A} \) \( \{i\} \)"

and recv: "\( \text{act} \ \text{(recvmsg} \ \text{rreq_rrep_sn}) \ \sigma \ \sigma' \ a \)"

obtain \( l \) \( l' \)
where "\( l \in \text{labels} \ \Gamma \_ \text{AODV} \ p \)"
and "\( l' \in \text{labels} \ \Gamma \_ \text{AODV} \ p' \)"

by (metis aodv_ex_label)

from "\( l \in \text{labels} \ \Gamma \_ \text{AODV} \ p \)" and "\( l' \in \text{labels} \ \Gamma \_ \text{AODV} \ p' \)"

have tr: "\( l \in \text{labels} \ \Gamma \_ \text{AODV} \ p \)" by simp

have "\( \text{anycast} \ (\text{rreq_rrep_fresh} \ (\text{rt} \ (\sigma \ i))) \) \( a \)"

proof -

have "\( \text{opaodv} \ \text{i} \:=\text{A} \) \( \text{act} \ \text{(recvmsg} \ \text{rreq_rrep_sn}) \) \( \text{other} \ \text{A} \) \( \{i\} \rightarrow \)

\( \text{onll} \ \Gamma \_ \text{AODV} \ (\text{seqll} \ i \ (\lambda ((\xi, \_), \ a, \ _). \) \text{anycast} \ (\text{msgfresh} \ \sigma) \) \( a \)\)"

by (rule \( \text{ostep_invariant_weakenE} \) [OF \( \text{open_seq_step_invariant} \) [OF \( \text{rreq_rrep_fresh} \_ \text{any_step_invariant} \ \text{initiali} \_ \text{aodv}, \)

simplified \( \text{seqll_onll_swap} \)])\) \( \text{auto} \)

hence "\( \text{onll} \ \Gamma \_ \text{AODV} \ (\text{seqll} \ i \ (\lambda ((\xi, \_), \ a, \ _). \) \text{anycast} \ (\text{rreq_rrep_fresh} \ (\text{rt} \ (\xi))) \) \( a \))\)"

by (erule(4) \( \text{ostep_invariantE} \))

thus \( \text{thesis} \) using "\( l \in \text{labels} \ \Gamma \_ \text{AODV} \ p \)" and "\( l' \in \text{labels} \ \Gamma \_ \text{AODV} \ p' \)" by auto

\( \text{qed} \)

moreover have "\( \text{anycast} \ (\text{rerr} \_ \text{invalid} \ (\text{rt} \ (\sigma \ i))) \) \( a \)"

proof -

have "\( \text{opaodv} \ \text{i} \:=\text{A} \) \( \text{act} \ \text{(recvmsg} \ \text{rreq_rrep_sn}) \) \( \text{other} \ \text{A} \) \( \{i\} \rightarrow \)

\( \text{onll} \ \Gamma \_ \text{AODV} \ (\text{seqll} \ i \ (\lambda ((\xi, \_), \ a, \ _). \) \text{anycast} \ (\text{rerr} \_ \text{invalid} \ (\text{rt} \ (\xi))) \) \( a \)\)"

by (rule \( \text{ostep_invariant_weakenE} \) [OF \( \text{open_seq_step_invariant} \) [OF \( \text{rerr} \_ \text{invalid} \_ \text{any_step_invariant} \ \text{initiali} \_ \text{aodv}, \)

simplified \( \text{seqll_onll_swap} \)])\) \( \text{auto} \)

hence "\( \text{onll} \ \Gamma \_ \text{AODV} \ (\text{seqll} \ i \ (\lambda ((\xi, \_), \ a, \ _). \) \text{anycast} \ (\text{rerr} \_ \text{invalid} \ (\text{rt} \ (\xi))) \) \( a \))\)"

by (erule(4) \( \text{ostep_invariantE} \))

thus \( \text{thesis} \) using "\( l \in \text{labels} \ \Gamma \_ \text{AODV} \ p \)" and "\( l' \in \text{labels} \ \Gamma \_ \text{AODV} \ p' \)" by auto

\( \text{qed} \)

moreover have "\( \text{anycast} \ (\text{rreq} \_ \text{rrep} \_ \text{sn} \ a) \)"

proof -

from or tr recv

have "\( \text{onll} \ \Gamma \_ \text{AODV} \ (\text{seqll} \ i \ (\lambda ((\xi, \_, \ _). \) \text{anycast} \ (\text{rreq} \_ \text{rrep} \_ \text{sn} \ a)))\)"

by (erule(4) \( \text{ostep_invariantE} \) [OF \( \text{open_seq_step_invariant} \) [OF \( \text{rreq} \_ \text{rrep} \_ \text{sn} \_ \text{any_step_invariant} \ \text{initiali} \_ \text{aodv}, \)

\( \text{aodv} \_ \text{trans} \ \text{aodv} \_ \text{trans}, \)

simplified \( \text{seqll_onll_swap} \)])\) \( \text{auto} \)

thus \( \text{thesis} \) using "\( l \in \text{labels} \ \Gamma \_ \text{AODV} \ p \)" and "\( l' \in \text{labels} \ \Gamma \_ \text{AODV} \ p' \)" by auto

\( \text{qed} \)

moreover have "\( \text{anycast} \ (\lambda \ m. \ \text{not} \_ \text{Pkt} \ m \rightarrow \ \text{msg} \_ \text{sender} \ m \:=\text{i} \) \( a \)"

proof -

have "\( \text{opaodv} \ \text{i} \:=\text{A} \) \( \text{act} \ \text{(recvmsg} \ \text{rreq_rrep_sn}) \) \( \text{other} \ \text{A} \) \( \{i\} \rightarrow \)

\( \text{onll} \ \Gamma \_ \text{AODV} \ (\text{seqll} \ i \ (\lambda ((\xi, \_, \ _). \) \text{anycast} \ (\lambda \ m. \ \text{not} \_ \text{Pkt} \ m \rightarrow \ \text{msg} \_ \text{sender} \ m \:=\text{i}) \) \( a \)\)"

by (rule \( \text{ostep_invariant_weakenE} \) [OF \( \text{open_seq_step_invariant} \) [OF \( \text{rreq} \_ \text{rrep} \_ \text{sn} \_ \text{any_step_invariant} \ \text{initiali} \_ \text{aodv}, \)

\( \text{aodv} \_ \text{trans} \ \text{aodv} \_ \text{trans}, \)

simplified \( \text{seqll_onll_swap} \)])\) \( \text{auto} \)

thus \( \text{thesis} \) using "\( l \in \text{labels} \ \Gamma \_ \text{AODV} \ p \)" and "\( l' \in \text{labels} \ \Gamma \_ \text{AODV} \ p' \)" by auto

\( \text{qed} \)
ultimately have "anycast (msg_fresh σ) a"
by (simp_all add: anycast_def
del: msg_fresh
split: seq.action.split_asm msg.split_asm) simp_all
by auto
qed

lemma oreceived_rreq_rrep_nsqn_fresh_inv:
"opaodv i \|= (otherwith quality_increases \{i\} (orecvmsg msg_fresh),
other quality_increases \{i\} \rightarrow)
onll Γ_AODV (λ((σ, _), a, _). anycast (msg_fresh σ) a) ((σ, p), a, (σ', p'))"
proof (rule oreceived_msg_inv)
fix σ σ' m
assume *: "msg_fresh σ m" and "other quality_increases \{i\} σ σ'"
from this(2) have "∀j. quality_increases (σ j) (σ' j)" ..
thus "msg_fresh σ' m" using * ..
next
fix σ m
assume "msg_fresh σ m"
thus "msg_fresh (σ i := σ i (msg := m)) m"
proof (cases m)
fix dests sip
assume "m = Rerr dests sip"
with ⟨msg_fresh σ m⟩ show ?thesis by auto
qed auto
qed

lemma oquality_increases_nsqn_fresh:
"opaodv i \|= A (otherwith quality_increases \{i\} (orecvmsg msg_fresh),
other quality_increases \{i\} \rightarrow)
onll Γ_AODV (λ((σ, _), _, (σ', _)). ∀j. quality_increases (σ j) (σ' j))"
by (rule ostep_invariant_weakenE [OF oquality_increases]) auto

lemma oosn_rreq:
"opaodv i \|= (otherwith quality_increases \{i\} (orecvmsg msg_fresh),
other quality_increases \{i\} \rightarrow)
onl Γ_AODV (seq i (λ(ξ, l). l ∈ \{PAodv-\:4, PAodv-\:5\} ∪ \{PRreq-n l n. True\} \rightarrow 1 ≤ osn ξ))"
by (rule oinvariant_weakenE [OF open_seq_invariant [OF osn_rreq initiali_aodv]])
(auto simp: seql_onl_swap)

lemma rreq_sip:
"opaodv i \|= (otherwith quality_increases \{i\} (orecvmsg msg_fresh),
other quality_increases \{i\} \rightarrow)
onl Γ_AODV (λ(ξ, l). l ∈ \{PAodv-\:4, PAodv-\:5, PRreq-\:0, PRreq-\:2\} ∧ sip (σ i) \neq oip (σ i))
→ oip (σ i) ∈ kD(rt (σ (sip (σ i))))
∧ nsqn (rt (σ (sip (σ i)))) (oip (σ i)) ≥ osn (σ i)
∧ (nsqn (rt (σ (sip (σ i)))) (oip (σ i)) = osn (σ i))
→ (hops (σ i) ≥ the (dhops (rt (σ (sip (σ i)))) (oip (σ i))))
∨ the (flag (rt (σ (sip (σ i)))) (oip (σ i)) = inv))"
(is "_ \|= (?S, ?U \rightarrow _) ")
proof (inv_cterms inv add: oseq_step_invariant_sterms [OF oquality_increases_nsqn_fresh oadv_wf oadodv_trans]
onl oinvariant_sterms [OF aodv_wf oreceived_rreq_rrep_nsqn_fresh_inv]
onl oinvariant_sterms [OF aodv_wf oosn_rreq]
simp add: seqlsimp
lemma odsn_rrep:

"(sigma, p) in reachable (opaodv i) ?S ?U"
and "1 in labels Gamma_AODV p"
and pre:

"(l = PAodv-:4 ∨ l = PAodv-:5 ∨ l = PRreq-:0 ∨ l = PRreq-:2) ∧ sip (sigma i) ≠ oip (sigma i)
→ oip (sigma i) ∈ kD (rt (sigma (sip (sigma i))))
∧ osn (sigma i) ≤ nsqn (rt (sigma (sip (sigma i)))) (oip (sigma i))
∧ (nsqn (rt (sigma (sip (sigma i)))) (oip (sigma i))) = osn (sigma i)
→ the (dhops (rt (sigma (sip (sigma i)))) (oip (sigma i))) ≤ hops (sigma i)
∧ \(\flag{}\) (rt (sigma (sip (sigma i)))) (oip (sigma i)) = inv"
and "other quality_increases {i} sigma sigma"
and hyp: "(l = PAodv-:4 ∨ l = PAodv-:5 ∨ l = PRreq-:0 ∨ l = PRreq-:2) ∧ sip (sigma i) ≠ oip (sigma i)"
is "?labels ∧ sip (sigma i) ≠ oip (sigma i)"
from (4) have "sigma i = sigma i" ...
with hyp have hyp': "?labels ∧ sip (sigma i) ≠ oip (sigma i)" by simp

show "oip (sigma i) ∈ kD (rt (sigma (sip (sigma i))))
∧ osn (sigma i) ≤ nsqn (rt (sigma (sip (sigma i)))) (oip (sigma i))
∧ (nsqn (rt (sigma (sip (sigma i)))) (oip (sigma i))) = osn (sigma i)
→ the (dhops (rt (sigma (sip (sigma i)))) (oip (sigma i))) ≤ hops (sigma i)
∧ \(\flag{}\) (rt (sigma (sip (sigma i)))) (oip (sigma i)) = inv"

proof (cases "sip (sigma i) = i")
assume "sip (sigma i) ≠ i"
from (other quality_increases {i} sigma sigma)
have "quality_increases (sigma (sip (sigma i))) (sigma (sip (sigma i)))"
by (rule otherE) (clarsimp simp: simp)
moreover from (sigma, p) in reachable (opaodv i) ?S ?U \(\{l \in \text{labels Gamma_AODV p}\\) and hyp
have "1 ≤ osn (sigma i)"
by (auto dest!: onl_o invariant_weakenD \[of \text{onsn req}\]
simp add: seqsimp (sigma i = sigma i))
moreover from :sip (sigma i) ≠ i hyp' and pre
have "oip (sigma i) ∈ kD (rt (sigma (sip (sigma i))))
∧ osn (sigma i) ≤ nsqn (rt (sigma (sip (sigma i)))) (oip (sigma i))
∧ (nsqn (rt (sigma (sip (sigma i)))) (oip (sigma i))) = osn (sigma i)
→ the (dhops (rt (sigma (sip (sigma i)))) (oip (sigma i))) ≤ hops (sigma i)
∧ \(\flag{}\) (rt (sigma (sip (sigma i)))) (oip (sigma i)) = inv"
by (auto simp: (sigma i = sigma i))
ultimately show ?thesis
by (rule quality_increases req req props)
next
assume "sip (sigma i) = i" thus ?thesis
using (sigma i = sigma i) hyp and pre by auto
qed

lemmas rrep: 

"opaodv i = (otherwith quality_increases {i} (orecmsg msg_fresh),
other quality_increases {i} →)
onl Gamma_AODV (seql \{\lambda (c_1, l) . l \in \{PAodv-:6, PAodv-:7\} \cup \{PRrep-:n|n. True\} \→ 1 \leq dsn \xi\})"
by (rule o invariant_weakenE \[of \text{open seq invariant [of dsn rep initiali_aodv]}\]
(auto simp: seql_onl_swap)

lemma odsn_rrep:

"opaodv i = (otherwith quality_increases {i} (orecmsgmsg fresh),
other quality_increases {i} →)
onl Gamma_AODV (\lambda (sigma_1). (l ∈ \{PAodv-:6, PAodv-:7, PRrep-:0, PRrep-:1\} ∧ sip (sigma i) ≠ dip (sigma i))
→ dip (sigma i) ∈ kD(rt (sigma (sip (sigma i))))
∧ nsqn (rt (sigma (sip (sigma i)))) (dip (sigma i)) ≥ dsn (sigma i)
∧ (nsqn (rt (sigma (sip (sigma i)))) (dip (sigma i))) = dsn (sigma i)
→ (hops (sigma i) ≥ the (dhops (rt (sigma (sip (sigma i)))) (dip (sigma i))))
∧ \(\flag{}\) (rt (sigma (sip (sigma i)))) (dip (sigma i)) = inv)
(is "\(\_\) \→ (?S, \_ U) \→ \")

proof (inv cterms inv add: oseq_step_invariant_sterms \[of \text{quality increases nsqn fresh aodv wf}\]
\begin{verbatim}
lemma rerr_sip:
  "oapovd i \models (otherwith quality_increases \{i\} (oercvmsg msg_fresh),
  other_quality_increases \{i\} \rightarrow
  onl \Gamma_{AODV} (\lambda(\sigma, l).
    l \in \{PAodv-:8, PAodv-:9, PRerr-:0, PRerr-:1\}
    \rightarrow (\forall ripc\in dom(dests (\sigma i)).
      ripc\in kD(rt (\sigma (sip (\sigma i))))
      \land the (dests (\sigma i) ripc) - 1 \leq nsqn (rt (\sigma (sip (\sigma i)))) ripc))" 
(is "\_ \models (\_?S, \_?U \rightarrow \_?)")

proof -
  \{ fix dests rip sip rsn and \sigma \sigma' :: "ip \Rightarrow state"
  assume qinc: "\forall j quality_increases (\sigma j) (\sigma' j)"
  and *: "\forall ripc\in dom dests. rip \in kD(rt (\sigma sip))
    \land the (dests rip) - 1 \leq nsqn (rt (\sigma sip)) rip"
  and "dests rip = Some rsn"
  from this(3) have "rip\in dom dests" by auto
  with * and (dests rip = Some rsn) have "rip\in kD(rt (\sigma sip))"
\end{verbatim}
and "rsn - 1 ≤ nsqn (rt (σ sip)) rip"

by (auto dest!: bspec)
from qinc have "quality_increases (σ sip) (σ' sip)"
have "rip ∈ kD(rt (σ sip)) ∧ rsn - 1 ≤ nsqn (rt (σ sip)) rip"
proof
  from ⟨rip∈kD(rt (σ sip))⟩ and ⟨quality_increases (σ sip) (σ' sip)⟩
  show "rip ∈ kD(rt (σ sip))"
next
  from ⟨rip∈kD(rt (σ sip))⟩ and ⟨quality_increases (σ sip) (σ' sip)⟩
  have "nsqn (rt (σ sip)) rip ≤ nsqn (rt (σ sip)) rip" ..
  with ⟨rsn - 1 ≤ nsqn (rt (σ sip)) rip⟩ show "rsn - 1 ≤ nsqn (rt (σ sip)) rip"
  by (rule le_trans)
qed

} note partial = this

show ?thesis
by (inv_cterms inv add: oseq_step_invariant_sterms [OF equality_increases_nsqn_fresh adov_wf
oaodv_trans]
onl_only_invariant_sterms [OF adov_wf oreceived_rreq_rrep_nsqn_fresh_inv]
other_quality_increases other_localD
simp del: One_nat_def, intro conjI)
(clarsimp simp del: One_nat_def split: split_if_asm option.split_asm, erule(2) partial)+

qed

lemma prerr_guard: "paodv i \parallel
  onl Γ AODV (λ(ξ, l). (l = PRerr−:1 −→
    (∀ ip ∈ dom(dests ξ). ip∈vD(rt ξ)
      ∧ the (nhop (rt ξ) ip) = sip ξ
      ∧ sqn (rt ξ) ip < the (dests ξ ip))))"
by (inv_cterms) (clarsimp split: option.split_asm split_if_asm)

lemmas oaddpreRT_welldefined =
  open_seq_invariant [OF addpreRT_welldefined initiali_aodv oaodv_trans adov_trans,
simplified seql_onl_swap,
THEN oinvariant_anyact]

lemmas odests_vD_inc_sqn =
  open_seq_invariant [OF odests_vD_inc_sqn initiali_aodv oaodv_trans adov_trans,
simplified seql_onl_swap,
THEN oinvariant_anyact]

lemmas oprerr_guard =
  open_seq_invariant [OF prerr_guard initiali_aodv aoaddpreRT_welldefined adov_trans,adov_trans,
simplified seql_onl_swap,
THEN oinvariant_anyact]

Proposition 7.28

lemma seq_compare_next_hop':
  "opaodv i \parallel (otherwith quality_increases {i} (orecvmsg msg_fresh),
other quality_increases {i} \rightarrow) onl Γ AODV (λ(σ, _).
∀ dip. let nhip = the (nhop (rt σ i)) dip
  in dip ∈ kD(rt σ i)) ∧ nhip ≠ dip −→
dip ∈ kD(rt σ nhip)) ∧ nsqn (rt σ i) dip ≤ nsqn (rt σ nhip) dip"
(is "_ \parallel (?S, ?U \rightarrow) _")
proof -

{ fix nhop and σ σ' :: "ip ⇒ state"
  assume pre: "∀ dip∈kD(rt σ i). nhop dip ≠ dip −→
dip∈kD(rt σ (nhip dip))) ∧ nsqn (rt σ i) dip ≤ nsqn (rt σ (nhip dip)) dip"
  and qinc: "∀ j. quality_increases (σ j) (σ' j)"
  have "∀ dip∈kD(rt σ i). nhop dip ≠ dip −→
dip∈kD(rt σ' (nhip dip))) ∧ nsqn (rt σ i) dip ≤ nsqn (rt σ' (nhip dip)) dip"
proof (intro ballI impI)
  fix dip
assume "dip∈kD(rt (σ i))"
and "nhop dip ≠ dip"
with pre have "dip∈kD(rt (σ (nhop dip)))"
and "nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (nhop dip))) dip"
by auto
from qinc have qinc_nhop: "quality_increases (σ (nhop dip)) (σ' (nhop dip))" ..
with "dip∈kD(rt (σ (nhop dip)))" have "dip∈kD (rt (σ' (nhop dip)))" ..
moreover have "nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (nhop dip))) dip"
proof -
from "dip∈kD(rt (σ (nhop dip)))" qinc_nhop
have "nsqn (rt (σ (nhop dip))) dip ≤ nsqn (rt (σ' (nhop dip))) dip" ..
with "nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (nhop dip))) dip" show ?thesis
by simp
qed
ultimately show "dip∈kD(rt (σ' (nhop dip)))"
and "nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (nhop dip))) dip" ..
qed
}

{ fix nhop and σ σ' :: "ip ⇒ state"
assume pre: "∀ dip∈kD(rt (σ i)). nhop dip ≠ dip → dip∈kD(rt (σ (nhop dip)))"
and ndest: "∀ ripc∈dom (dests (σ i)). ripc ∈ kD (rt (σ (sip (σ i))))
∧ the (dests (σ i)) (ripc) - 1 ≤ nsqn (rt (σ (sip (σ i))))" ripc"
and issip: "∀ ip∈dom (dests (σ i)). nhop ip = sip (σ i)"
and qinc: "∀ j. quality_increases (σ j) (σ' j)"
have "∀ dip∈kD(rt (σ i)). nhop dip ≠ dip → dip∈kD (rt (σ' (nhop dip)))"
∧ nsqn (invalidate (rt (σ i)) (dests (σ i))) dip ≤ nsqn (rt (σ' (nhop dip))) dip"
proof (intro ballI impI)
fix dip
assume "dip∈kD(rt (σ i))"
and "nhop dip ≠ dip"
with pre and qinc have "dip∈kD(rt (σ' (nhop dip)))"
and "nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (nhop dip))) dip"
by (auto dest!: basic)
have "nsqn (invalidate (rt (σ i)) (dests (σ i))) dip ≤ nsqn (rt (σ' (nhop dip))) dip"
proof (cases (dip∈dom (dests (σ i))))
assume "dip∈dom (dests (σ i))"
with "dip∈kD(rt (σ i)))": obtain dsn where "dests (σ i) dip = Some dsn"
by auto
with "dip∈kD(rt (σ i)))": have "nsqn (invalidate (rt (σ i)) (dests (σ i))) dip = dsn - 1"
by (rule nsqn_invalidate_eq)
moreover have "dsn - 1 ≤ nsqn (rt (σ' (nhop dip))) dip"
proof -
from "dests (σ i) dip = Some dsn": have "the (dests (σ i) dip = dsn)" by simp
with ndest and "dip∈dom (dests (σ i))": have "dip ∈ kD (rt (σ (sip (σ i))))"
"dsn - 1 ≤ nsqn (rt (σ (sip (σ i))) dip"
by auto
moreover from issip and "dip∈dom (dests (σ i))": have "nhop dip = sip (σ i)" ..
ultimately have "dip ∈ kD (rt (σ (nhop dip)))"
and "dsn - 1 ≤ nsqn (rt (σ (nhop dip))) dip" by auto
with qinc show "dsn - 1 ≤ nsqn (rt (σ' (nhop dip))) dip" by simp
(extra of merge kD_nsqn_quality_increases_trans)
qed
ultimately show ?thesis by simp
proof
fix dip
assume "dip\in kD(rt (\sigma (i)))"
with a1 and a2
have "\forall dip\in kD(rt (\sigma (i))). the (nhop (rt (\sigma (i))) dip) \neq dip \\
\rightarrow dip\in kD(rt (\sigma (the (nhop (rt (\sigma (i))) dip)))) \\
\land nsqn (rt (\sigma (i))) dip \leq nsqn (rt (\sigma (the (nhop (rt (\sigma (i))) dip)))) dip"
by (drule(1) basic, auto)
thus "?P dip" by (cases "dip = sip (\sigma (i))") auto
qed

{ fix \sigma' :: "ip \Rightarrow state"
assume a1: "\forall dip\in kD(rt (\sigma (i))). the (nhop (rt (\sigma (i))) dip) \neq dip \\
\rightarrow dip\in kD(rt (\sigma' (the (nhop (rt (\sigma (i))) dip)))) \\
\land nsqn (rt (\sigma (i))) dip \leq nsqn (rt (\sigma' (the (nhop (rt (\sigma (i))) dip)))) dip"
and a2: "\forall j. quality_increases (\sigma j) (\sigma' j)"
have "\forall dip\in kD(rt (\sigma (i))). the (nhop (update (rt (\sigma (i))) (sip (\sigma (i)))) \\
\rightarrow dip\in kD(rt (\sigma' (the (nhop (update (rt (\sigma (i))) (sip (\sigma (i))))) \\
\rightarrow dip)))) \\
\land nsqn (update (rt (\sigma (i))) (sip (\sigma (i)))) (0, unk, val, Suc 0, sip (\sigma (i)), {}) dip \\
\leq nsqn (rt (\sigma' (the (nhop (update (rt (\sigma (i))) (sip (\sigma (i))))) \\
\rightarrow dip)))) dip"
(is "\forall dip\in kD(rt (\sigma (i))). ?P dip")
proof
fix dip
assume "dip\in kD(rt (\sigma (i)))"
with a1 and a2
have "the (nhop (rt (\sigma (i))) dip) \neq dip \\
\rightarrow dip\in kD(rt (\sigma' (the (nhop (rt (\sigma (i))) dip)))) \\
\land nsqn (rt (\sigma (i))) dip \leq nsqn (rt (\sigma' (the (nhop (rt (\sigma (i))) dip)))) dip"
by (rule basic)
thus "?P dip" by (cases "dip = sip (\sigma (i))") auto
qed

from pre and qinc
have pre': "\forall dip\in kD(rt (\sigma (i))). the (nhop (rt (\sigma (i))) dip) \neq dip \\
\rightarrow dip\in kD(rt (\sigma' (the (nhop (rt (\sigma (i))) dip)))) \\
\land nsqn (rt (\sigma (i))) dip \leq nsqn (rt (\sigma' (the (nhop (rt (\sigma (i))) dip)))) dip"
by (rule basic)
have "(the (nhop (update (rt (\sigma (i))) oip (osn, kno, val, Suc hops, sip, {})) oip) \neq oip \\
\rightarrow oip\in kD(rt (\sigma' (the (nhop (update (rt (\sigma (i))) oip \\
\rightarrow oip, kno, val, Suc hops, sip, {})))) oip)) \\
\land nsqn (update (rt (\sigma (i))) oip (osn, kno, val, Suc hops, sip, {})) oip \\
\leq nsqn (rt (\sigma' (the (nhop (update (rt (\sigma (i))) oip \\
\rightarrow oip, kno, val, Suc hops, sip, {})))) oip)" \\
(is "?nhop_not_oip \rightarrow ?oip_in_kD \land ?nsqn_le_nsqn")
proof (rule, split update_rt_split_asm)
assume rtnot: "rt (\sigma (i)) \neq update (rt (\sigma (i))) oip (osn, kno, val, Suc hops, sip, {})" \\
and "the (nhop (rt (\sigma (i))) oip) \neq oip"
with pre' show "?oip_in_kD \land ?nsqn_le_nsqn" by auto
next
assume rtnot: "rt (\sigma (i)) \neq update (rt (\sigma (i))) oip (osn, kno, val, Suc hops, sip, {})"
and notoip: ?nhop_not_oip
with * qinc have ?oip_in_kD
by (clarsimp_elim!: kD_quality_increases)
moreover with * pre qinc rtnot notoip have ?nsqn_le_nsqn
by simp (metis kD_nsqn_quality_increases_trans)
ultimately show "?oip_in_kD ∧ ?nsqn_le_nsqn" ..
qed
}

note update1 = this

{ fix σ σ' oip sip osn hops
assume pre: "∀ dip∈kD (rt (σ i)). the (nhop (rt (σ i)) dip) ≠ dip
→ dip∈kD( rt (σ (the (nhop (rt (σ i)) dip))))
∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (the (nhop (rt (σ i)) dip)))) dip"
and qinc: "∀ j. quality_increases (σ j) (σ' j)"
and *: "sip ≠ oip → oip∈kD(rt (σ sip))
∧ osn ≤ nsqn (rt (σ sip)) oip
∧ (nsqn (rt (σ sip)) oip = osn
→ the (dhops (rt (σ sip)) oip) ≤ hops
∨ the (flag (rt (σ sip)) oip) = inv)"

from pre and qinc
have pre': "∀ dip∈kD (rt (σ i)). the (nhop (rt (σ i)) dip) ≠ dip
→ dip∈kD( rt (σ' (the (nhop (rt (σ i)) dip))))
∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (the (nhop (rt (σ i)) dip)))) dip"
by (rule basic)

have "∀ dip∈kD(rt (σ i)).
the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})) dip) ≠ dip
→ dip∈kD( rt (σ' (the (nhop (update (rt (σ i)) oip
(osn, kno, val, Suc hops, sip, {})) dip))))
∧ nsqn (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})) dip
≤ nsqn (rt (σ' (the (nhop (update (rt (σ i)) oip
(osn, kno, val, Suc hops, sip, {})) dip)))) dip"
(is "∀ dip∈kD(rt (σ i)). _ → ?dip_in_kD dip ∧ ?nsqn_le_nsqn dip")

proof (intro ballI impI, split update_rt_split_asm)

fix dip
assume "dip∈kD(rt (σ i))"
and "the (nhop (rt (σ i)) dip) ≠ dip"
and "rt (σ i) = update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})"
with pre' show "?dip_in_kD dip ∧ ?nsqn_le_nsqn dip" by simp

next

fix dip
assume "dip∈kD(rt (σ i))"
and notdip: "the (nhop (update (rt (σ i)) oip
(osn, kno, val, Suc hops, sip, {})) dip) ≠ dip"
and rtnot: "rt (σ i) ≠ update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})"
show "?dip_in_kD dip ∧ ?nsqn_le_nsqn dip" proof (cases "dip = oip")
assume "dip ≠ oip"
with pre' (dip∈kD(rt (σ i))) notdip
show ?thesis by clarsimp

next

assume "dip = oip"
with rtnot qinc (dip∈kD(rt (σ i))) notdip *
have "?dip_in_kD dip"
by simp (metis kD_quality_increases)

moreover from (dip = oip) rtnot qinc (dip∈kD(rt (σ i))) notdip *
have "?nsqn_le_nsqn dip" by simp (metis kD_nsqn_quality_increases_trans)
ultimately show ?thesis ..
qed
}

note update2 = this

have "opaodv i = (?S, ?U →) onl Γ_AODV (λ(σ, _).
∀ dip ∈ kD(rt (σ i)). the (nhop (rt (σ i)) dip) ≠ dip
→ dip ∈ kD(rt (σ (the (nhop (rt (σ i)) dip))))
∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (the (nhop (rt (σ i)) dip)))) dip)"
by (inv_cterms inv add: oseq_step_invariant_sterms [OF oquality_increases_nsqn_fresh aodv_wf oaoeq_trans]
onl_oinvariant_sterms [OF adv_wf oaddpreRT_welldefined]
onl_oinvariant_sterms [OF adv_wf odests_vD_inc_sqn]
onl_oinvariant_sterms [OF adv_wf oprerr_guard]
onl_oinvariant_sterms [OF adv_wf rreq_sip]
onl_oinvariant_sterms [OF adv_wf rrep_sip]
onl_oinvariant_sterms [OF adv_wf rerr_sip]
other_quality_increases
other_localD

solve: basic basic_prerr
simp add: seqlsimp nsqn_invalidate nhop_update_sip
simp del: One_nat_def)
(rule conjI, erule(2) update1, erule(2) update2)+

thus thesis unfolding Let_def by auto
qed

Proposition 7.30
lemmas okD_unk_or_atleast_one =
open_seq_invariant [OF kD_unk_or_atleast_one initiali_aodv,
simplified seql_onl_swap]
lemmas ozero_seq_unk_hops_one =
open_seq_invariant [OF zero_seq_unk_hops_one initiali_aodv,
simplified seql_onl_swap]

lemma oreachable_fresh_okD_unk_or_atleast_one:
  fixes dip
  assumes "((\sigma, p) \in oreachable (opaodv i))
  (otherwith (op=) {i} (orecvmsg (\lambda\sigma m. msg_fresh \sigma m 
\land msg_zhops m)))
  (other_quality_increases {i})"
  and "dip\in kD(rt (\sigma i))"
  shows "\pi_3 the (rt (\sigma i) dip) = unk \lor 1 \leq \pi_2 the (rt (\sigma i) dip)"
(is "?P dip")
proof -
  have "\exists l. l \in labels \Gamma_{AODV} p" by (metis aodv_ex_label)
  with assms(1) have "\forall dip\in kD (rt (\sigma i)). ?P dip"
    by - (drule oinvariant_weakenD [OF okD_unk_or_atleast_one [OF oaodv_trans aodv_trans]],
      auto dest!: onlD otherwith_actionD onlD simp: seqlsimp)
  with ⟨dip\in kD(rt (\sigma i))⟩ show thesis by simp
qed

lemma oreachable_fresh_ozero_seq_unk_hops_one:
  fixes dip
  assumes "((\sigma, p) \in oreachable (opaodv i))
  (otherwith (op=) {i} (orecvmsg (\lambda\sigma m. msg_fresh \sigma m 
\land msg_zhops m)))
  (other_quality_increases {i})"
  and "dip\in kD(rt (\sigma i))"
  shows "sqn (rt (\sigma i) dip) = 0 \rightarrow
  sqnf (rt (\sigma i) dip) = unk
  \land the (dhops (rt (\sigma i)) dip) = 1
  \land the (nhop (rt (\sigma i) dip) = dip"
(is "?P dip")
proof -
  have "\exists l. l \in labels \Gamma_{AODV} p" by (metis aodv_ex_label)
  with assms(1) have "\forall dip\in kD (rt (\sigma i)). ?P dip"
    by - (drule oinvariant_weakenD [OF ozero_seq_unk_hops_one [OF oaodv_trans aodv_trans]],
      auto dest!: onlD otherwith_actionD onlD simp: seqlsimp)
  with ⟨dip\in kD(rt (\sigma i))⟩ show thesis by simp
qed

lemma seq_nhop_quality_increases':
  shows "opaodv i \models (otherwith (op=) {i}"
(orecvmmsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
other quality_increases (i) →
onl Γ_{AODV} (λ(σ, _). ∀ dip. let nhip = the (nhop (rt (σ i))) dip
in dip ∈ vD (rt (σ i)) ∨ vD (rt (σ nhip))
∧ nhip ≠ dip
→ (rt (σ i)) ⊑ dip (rt (σ nhip)))

(is "_ = (?S i, _ →) _")

proof -

have weaken:
"∀ p Q R P. p ⊨ (otherwith quality_increases I (orecvmmsg Q), other quality_increases I →) P
⇒ p ⊨ (otherwith (op=) I (orecvmmsg (λσ m. Q σ m ∧ R σ m)), other quality_increases I →) P"
by auto

{ fix i a and σ σ': "ip ⇒ state"

assume a1: "∀ dip. dip∈vD(rt (σ i))
∧ dip∈vD(rt (σ (the (nhop (rt (σ i))) dip)))
∧ (the (nhop (rt (σ i)) dip)) ≠ dip
→ rt (σ i) ⊑ dip rt (σ (the (nhop (rt (σ i)) dip)))"

and ow: "?S i σ σ' a"

have "∀ dip. dip∈vD(rt (σ i))
∧ dip∈vD rt (σ' (the (nhop (rt (σ i)) dip)))
∧ (the (nhop (rt (σ i)) dip)) ≠ dip
→ rt (σ i) ⊑ dip rt (σ' (the (nhop (rt (σ i)) dip)))"

proof clarify

fix dip

assume a2: "dip∈vD(rt (σ i))"

and a3: "dip∈vD rt (σ' (the (nhop (rt (σ i)) dip)))"

and a4: "(the (nhop (rt (σ i)) dip)) ≠ dip"

from ow have "∀ j. j ≠ i → σ j = σ' j" by auto

show "rt (σ i) ⊑ dip rt (σ' (the (nhop (rt (σ i)) dip)))" by auto

proof (cases "(the (nhop (rt (σ i)) dip)) = i")

assume "(the (nhop (rt (σ i)) dip)) = i"

with dip ∈ vD(rt (σ i)) have "dip ∈ vD rt (σ (the (nhop (rt (σ i)) dip)))" by simp

with a1 a2 a4 have "rt (σ i) ⊑ dip rt (σ (the (nhop (rt (σ i)) dip)))" by simp

hence False by simp

thus ?thesis .

next

assume "(the (nhop (rt (σ i)) dip)) ≠ i"

with ∀ j. j ≠ i → σ j = σ' j

have *: σ' (the (nhop (rt (σ i)) dip)) = σ' (the (nhop (rt (σ i)) dip))" by simp

with dip∈vD rt (σ' (the (nhop (rt (σ i)) dip))).

have "dip∈vD rt (σ (the (nhop (rt (σ i)) dip)))" by simp

with a1 a2 a4 have "rt (σ i) ⊑ dip rt (σ (the (nhop (rt (σ i)) dip)))" by simp

with * show ?thesis by simp

qed

} note basic = this

{ fix σ a dip sip i

assume a1: "∀ dip. dip∈vD(rt (σ i))
∧ dip∈vD(rt (σ (the (nhop (rt (σ i)) dip))))
∧ the (nhop (rt (σ i)) dip) ≠ dip
→ rt (σ i) ⊑ dip rt (σ (the (nhop (rt (σ i)) dip)))"

and ow: "?S i σ σ' a"

have "∀ dip. dip∈vD(update (rt (σ i)) sip 0, unk, val, Suc 0, sip, {}))
∧ dip∈vD rt (σ' (the (nhop (update (rt (σ i)) sip 0, unk, val, Suc 0, sip, {})) dip)))
∧ the (nhop (update (rt (σ i)) sip 0, unk, val, Suc 0, sip, {})) dip) ≠ dip
→ update (rt (σ i)) sip 0, unk, val, Suc 0, sip, {}
∧ dip rt (σ' (the (nhop (update (rt (σ i)) sip 0, unk, val, Suc 0, sip, {})) dip)))"

proof clarify

fix dip

assume a2: "dip∈vD (update (rt (σ i)) sip 0, unk, val, Suc 0, sip, {}))"

and a3: "dip∈vD(rt (σ' (the (nhop
(update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {})) dip))")
and a4: "the (nhop (update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {})) dip) ≠ dip"

show "update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {})
   ⊏_dip rt (σ' (the (nhop (update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {})) dip)))"

proof (cases "dip = sip")
assume "dip = sip"
with (the (nhop (update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {})) dip) ≠ dip)
have False by simp
thus ?thesis ..
next
assume [simp]: "dip ≠ sip"
from a2 have "dip∈vD(rt (σ i)) ∨ dip = sip"
   by (rule vD_update_val)
with [dip ≠ sip] have "dip∈vD(rt (σ i))" by simp
moreover from a3 have "dip∈vD(rt (σ' (the (nhop (rt (σ i)) dip))))" by simp
moreover from a4 have "the (nhop (rt (σ i)) dip) ≠ dip" by simp
ultimately have "rt (σ i) ⊏_dip rt (σ' (the (nhop (rt (σ i)) dip)))"
   using a1 ow by - (drule(1) basic, simp)
with [dip ≠ sip] show ?thesis
   by - (erule rt_strictly_fresher_update_other, simp)
qed
qed

} note update_0_unk = this

{ fix σ a σ' nhop
assume pre: "∀ dip. dip∈vD(rt (σ i)) ∧ dip∈vD(rt (σ (nhop dip))) ∧ nhop dip ≠ dip
   → rt (σ i) ⊏_dip rt (σ (nhop dip))"
and ow: "?S i σ σ' a"
have "∀ dip. dip ∈ vD (invalidate (rt (σ i)) (dests (σ i)))
   ∧ dip ∈ vD (rt (σ' (nhop dip))) ∧ nhop dip ≠ dip
   → rt (σ i) ⊏_dip rt (σ' (nhop dip))"

proof clarify
fix dip
assume "dip∈vD(invalidate (rt (σ i)) (dests (σ i)))"
  and "dip∈vD(rt (σ' (nhop dip)))"
  and "nhop dip ≠ dip"
from this(1) have "dip∈vD (rt (σ i))"
   by (clarsimp dest!: vD_invalidate_vD_not_dests)
moreover from ow have "∀j. j ≠ i → σ j = σ' j" by auto
ultimately have "rt (σ i) ⊏_dip rt (σ (nhop dip))"
   using pre [dip ∈ vD (rt (σ' (nhop dip))) ∧ nhop dip ≠ dip]
   by metis
with (∀j. j ≠ i → σ j = σ' j) show "rt (σ i) ⊏_dip rt (σ' (nhop dip))"
   by (metis rt_strictly_fresher_irefl)
qed

} note invalidate = this

{ fix σ a σ' dip oip osn sip hops i
assume pre: "∀ dip. dip ∈ vD (rt (σ i))
   ∧ dip ∈ vD (rt (σ (the (nhop (rt (σ i)) dip))))
   ∧ the (nhop (rt (σ i)) dip) ≠ dip
   → rt (σ i) ⊏_dip rt (σ (the (nhop (rt (σ i)) dip))))"
and ow: "?S i σ σ' a"
and "Suc 0 ≤ osn"

and a6: "sip ≠ oip → oip ∈ kD (rt (σ sip))
   ∧ osn ≤ nsqn (rt (σ sip)) oip
   ∧ (nsqn (rt (σ sip)) oip = osn
      → the (dhop (rt (σ sip)) oip) ≤ hops
      ∨ the (flag (rt (σ sip)) oip) = inv)"

and after: "σ' i = σ i (rt := update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {}))"

have "∀ dip. dip ∈ vD (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {}))
   ∧ dip ∈ vD (rt (σ' (the (nhop (update (rt (σ i)) oip
   (osn, kno, val, Suc hops, sip, {})) dip))))
   ∧ the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})) dip) ≠ dip
update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})

rt (σ' (the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})) dip)))

proof clarify

fix dip

assume a2: "dip∈vD(update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {}))"

and a3: "dip∈vD(rt (σ' (the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})) dip))))"

and a4: "the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})) dip) ≠ dip"

from ow have a5: "∀ j. j ≠ i −→ σ j = σ' j" by auto

show "update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})

⊏ dip

rt (σ' (the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})) dip)))" (is "?rt1 ⊏ dip ?rt2 dip")

proof (cases "?rt1 = rt (σ i)"

assume nochange [simp]:

"update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {}) = rt (σ i)"

from after have "σ' i = σ i" by simp

with a5 have "∀ j. σ j = σ' j" by metis

from a2 have "dip∈vD (rt (σ i))" by simp

moreover from a3 have "dip∈vD (rt (σ (the (nhop (update (rt (σ i)) dip))))" using nochange and ∀ j. σ j = σ' j by clarsimp

moreover from a4 have "the (nhop (rt (σ i)) dip) ≠ dip" by simp

ultimately have "rt (σ i) ⊏ dip rt (σ (the (nhop (rt (σ i)) dip)))" using pre by simp

hence "rt (σ i) ⊏ dip rt (σ' (the (nhop (rt (σ i)) dip)))"

using ⟨ ∀ j. σ j = σ' j ⟩ by simp

thus "?thesis" by simp

next

assume change: "?rt1 ≠ rt (σ i)"

from after a2 have "dip∈kD (rt (σ' i))" by auto

show ?thesis

proof (cases "dip = oip"

assume "dip ≠ oip"

with a2 have "dip∈vD (rt (σ i))" by auto

moreover with a3 a5 after and ⟨ dip ≠ oip ⟩

have "dip∈vD (rt (σ (the (nhop (rt (σ i)) dip))))" by simp metis

moreover from a4 and ⟨ dip ≠ oip ⟩ have "the (nhop (rt (σ i)) dip) ≠ dip" by simp

ultimately have "rt (σ i) ⊏ dip rt (σ (the (nhop (rt (σ i)) dip)))" using pre by simp

with after and a5 and ⟨ dip ≠ oip ⟩ show ?thesis by simp (metis rt_strictly_fresher_update_other rt_strictly_fresher_irefl)

next

assume "dip = oip"

with a4 and change have "sip ≠ oip" by simp

with a6 have "oip∈kD (rt (σ sip))" and "osn ≤ nsqn (rt (σ sip)) oip" by auto

from a3 change ⟨ dip = oip ⟩ have "oip∈vD (rt (σ' sip))" by simp
def

hence "the (flag (rt (σ' sip)) oip) = val" by simp

from ⟨ oip∈kD (rt (σ sip)) ⟩ have "osn < nsqn (rt (σ' sip)) oip ∨ (osn = nsqn (rt (σ' sip)) oip ∧ the (dhops (rt (σ' sip)) oip) ≤ hops)"

proof

assume "oip∈vD (rt (σ sip))"
hence "the (flag (rt (σ sip)) oip) = val" by simp
with a5 (sip ≠ oip) have "nsqn (rt (σ sip)) oip = osn →
  the (dhops (rt (σ sip)) oip) ≤ hops"
  by simp
  show ?thesis
proof (cases "sip = i")
  assume "sip ≠ i"
  with a5 have "σ sip = σ' sip" by simp
  with (osn ≤ nsqn (rt (σ sip)) oip)
    and nsqn (rt (σ sip)) oip = osn → the (dhops (rt (σ sip)) oip) ≤ hops
    show ?thesis by auto
next
  — alternative to using sip_not_ip
  assume [simp]: "sip = i"
  have "?rt1 = rt (σ i)"
    proof (rule update_cases_kD, simp_all)
      from ⟨Suc 0 ≤ osn⟩ show "0 < osn" by simp
    next
      from ⟨oip ∈ vD (rt (σ sip))⟩
        and ⟨sip = i⟩ show "oip ∈ kD (rt (σ i))" by simp
    next
      assume "sqn (rt (σ i)) oip < osn"
      also from ⟨osn ≤ nsqn (rt (σ sip)) oip⟩
        and nsqn (rt (σ sip)) oip = osn → the (dhops (rt (σ sip)) oip) ≤ hops
        have "the (dhops (rt (σ sip)) oip) ≤ hops" by simp
      finally have "sqn (rt (σ i)) oip < sqn (rt (σ i)) oip" .
      hence False by simp
      thus "(λa. if a = oip
          then Some (osn, kno, val, Suc hops, i, π7 (the (rt (σ i) oip)))
          else rt (σ i) a) = rt (σ i)" ..
    next
      assume "sqn (rt (σ i)) oip = osn"
      and "Suc hops < the (dhops (rt (σ i)) oip)"
      from this(1) and ⟨oip ∈ vD (rt (σ sip))⟩ have "nsqn (rt (σ i)) oip = osn"
        by simp
      with ⟨nsqn (rt (σ sip)) oip = osn → the (dhops (rt (σ sip)) oip) ≤ hops⟩
        and "the (dhops (rt (σ i)) oip) ≤ hops" by simp
      with ⟨Suc hops < the (dhops (rt (σ i)) oip)⟩ have False by simp
      thus "(λa. if a = oip
          then Some (osn, kno, val, Suc hops, i, π7 (the (rt (σ i) oip)))
          else rt (σ i) a) = rt (σ i)" ..
    next
      assume "the (flag (rt (σ i)) oip) = inv"
      with ⟨the (flag (rt (σ sip)) oip) = val⟩ have False by simp
      thus "(λa. if a = oip
          then Some (osn, kno, val, Suc hops, i, π7 (the (rt (σ i) oip)))
          else rt (σ i) a) = rt (σ i)" ..
    next
      from ⟨oip ∈ kD (rt (σ sip))⟩
        show "(λa. if a = oip then Some (the (rt (σ i) oip)) else rt (σ i) a) = rt (σ i)"
          by (auto dest!: kD_Some)
      qed
      with change have False ..
      thus ?thesis ..
    qed
next
  assume "oip ∈ iD (rt (σ sip))"
  with ⟨the (flag (rt (σ sip)) oip) = val⟩ and a5 have "sip = i"
    by (metis f.distinct(1) iD_flag_is_inv)
  from ⟨oip ∈ iD (rt (σ sip))⟩ have "the (flag (rt (σ sip)) oip) = inv" by auto
  with ⟨sip = i; Suc 0 ≤ osn⟩ change after ⟨oip ∈ kD (rt (σ sip))⟩
    have "nsqn (rt (σ sip)) oip < nsqn (rt (σ’ sip)) oip"
    unfolding update_def
by (clarsimp split: option.split_asm split_if_asm)
(auto simp: sqn_def)
with 
\text{osn} \leq \text{nsqn} (\text{rt} (\sigma \ (\sigma') \ \text{sip})) \ \text{oip} \ \text{have} \ "\text{osn} < \text{nsqn} (\text{rt} (\sigma' \ \text{sip})) \ \text{oip}"
by simp
thus \ ?thesis ..
qed
thus \ ?thesis
proof
  assume \ osnltn: "\text{osn} < \text{nsqn} (\text{rt} (\sigma' \ \text{sip})) \ \text{oip}"

  from \ \langle \text{dip} \in kD(\text{rt} (\sigma' \ i)) \rangle \ \text{and} \ \langle \text{dip} = \text{oip} \rangle 

  have "\text{dip} \in kD(?\text{rt1})" 
  by simp
  moreover from \ a3 \ have "\text{dip} \in kD(?\text{rt2} \ \text{dip})" 
  by simp
  moreover have "\text{nsqn} ?\text{rt1} \ \text{dip} < \text{nsqn} (?\text{rt2} \ \text{dip}) \ \text{dip}"

  proof
    have "\text{nsqn} ?\text{rt1} \ \text{oip} = \text{osn}"
      by (simp add: \langle \text{dip} = \text{oip} \rangle \ \text{nsqn_update_changed_kno_val} \ [OF \ change \ \text{THEN} \ \text{not_sym}] )
    also have "\ldots < \text{nsqn} (\text{rt} (\sigma' \ \text{sip})) \ \text{oip}" 
      using \ osnltn .
    also have "\ldots = \text{nsqn} (?\text{rt2} \ \text{oip}) \ \text{oip}" 
      by (simp add: change)
    finally show \ ?thesis
      using \ \langle \text{dip} = \text{oip} \rangle 
      by simp
  qed

  ultimately show \ ?thesis
  by (rule rt_strictly_fresher_ltI)
next
assume \ osneq: "\text{osn} = \text{nsqn} (\text{rt} (\sigma' \ \text{sip})) \ \text{oip} \land \ (\text{dhops} (\text{rt} (\sigma' \ \text{sip})) \ \text{oip}) \leq \text{hops}"

have "\text{oip} \in kD(?\text{rt1})" 
  by simp

  moreover from \ a3 \ have "\text{oip} \in kD(?\text{rt2} \ \text{oip})" 
  by simp

  moreover have "\text{nsqn} ?\text{rt1} \ \text{oip} = \text{nsqn} (?\text{rt2} \ \text{oip}) \ \text{oip}"

  proof

    from \ osneq 

    have "\text{osn} = \text{nsqn} (\text{rt} (\sigma' \ \text{sip})) \ \text{oip}" ..
    also have "\text{osn} = \text{nsqn} ?\text{rt1} \ \text{oip}" 
      by (simp add: \langle \text{dip} = \text{oip} \rangle \ \text{nsqn_update_changed_kno_val} \ [OF \ change \ \text{THEN} \ \text{not_sym}] )
    also have "\text{nsqn} (\text{rt} (\sigma' \ \text{sip})) \ \text{oip} = \text{nsqn} (?\text{rt2} \ \text{oip}) \ \text{oip}"
      by (simp add: change)
    finally show \ ?thesis .
  qed

  moreover have "{\pi_5}(\text{the} (?\text{rt2} \ \text{oip} \ \text{oip})) < {\pi_5}(\text{the} (?\text{rt1} \ \text{oip}))"

  proof

    from \ osneq 

    have "\text{the} (\text{dhops} (\text{rt} (\sigma' \ \text{sip})) \ \text{oip}) \leq \text{hops}" ..
    moreover from \ \langle \text{oip} \in vD(\text{rt} (\sigma' \ \text{sip})) \rangle 

    have "\text{oip} \in kD(\text{rt} (\sigma' \ \text{sip}))" 
      by auto
    ultimately have "{\pi_5}(\text{the} (\text{rt} (\sigma' \ \text{sip}) \ \text{oip})) \leq \text{hops}"
      by (auto simp add: \text{proj5_eq_dhops})
    also from \ \text{change after} \ have "\text{hops} < {\pi_5}(\text{the} (\sigma' \ \text{i} \ \text{oip}))"
      by (simp add: \text{proj5_eq_dhops} \ \text{metis} \ \text{dhops_update_changed_lessI})
    finally have "{\pi_5}(\text{the} (\sigma' \ \text{sip} \ \text{oip})) < {\pi_5}(\text{the} (\sigma' \ \text{i} \ \text{oip}))" .

    with \ \text{change after show} \ ?thesis 
    by simp
  qed

  ultimately have "?\text{rt1} \sqsubseteq _{\text{oip}} ?\text{rt2} \ \text{oip}"

    by (rule rt_strictly_fresher_eqI)
with \ \langle \text{dip} = \text{oip} \rangle 

  show \ ?thesis 
  by simp
  qed
  qed
  qed
\}

note \ rreq_rrep_update = this

have "\text{opaodv} i \models (\text{otherwith} (\text{opm}) \ \{i\}) \ (\text{orecvmsg} (\lambda \sigma \ m. \ \text{msg\_fresh} \ \sigma \ m \\
  \land \ \text{msg\_zhops} \ m), \ 
  \text{other quality increases} \ \{i\} \ \rightarrow)
\ \text{onl} \ \Gamma_{AODV} \ 
(\lambda (\sigma, _). \ \forall \text{dip}. \ \\text{dip} \in vD (\text{rt} \ (\sigma \ i)) \ \cap \ vD (\text{rt} \ (\sigma \ (\text{the} \ (\text{nhop} \ (\text{rt} \ (\sigma \ i) \ \text{dip})))))"

404
∧ the (nhop (rt (σ i)) dip) ≠ dip
→ rt (σ i) □ dip rt (σ (the (nhop (rt (σ i)) dip))))"

proof (inv_cterms inv add: onl_oinvariant_sterms [OF aodv_wf rreq_sip [THEN weaken]]
onl_oinvariant_sterms [OF aodv_wf rrep_sip [THEN weaken]]
onl_oinvariant_sterms [OF aodv_wf rerr_sip [THEN weaken]]
onl_oinvariant_sterms [OF aodv_wf oosn_rreq [THEN weaken]]
onl_oinvariant_sterms [OF aodv_wf odsn_rrep [THEN weaken]]
onl_oinvariant_sterms [OF aodv_wf oaddpreRT_welldefined]
solve: basic update_0_unk invalidate rreq_rrep_update
simp add: seqlsimp)

fix σ σ' p l
assume or: "(σ, p) ∈ reachable (opaodv i) (?S i) (other quality_increases {i})"
and "other quality_increases {i} σ σ'"
and ll: "l ∈ labels Γ AODV p"
and pre: "∀ dip. dip ∈ vD (rt (σ i)) ∧ dip ∈ vD(rt (σ i)) ∧ the (nhop (rt (σ i)) dip) ≠ dip
→ rt (σ i) □ dip rt (σ (the (nhop (rt (σ i)) dip))))"

from this(1-2)
have or': "(σ', p) ∈ reachable (opaodv i) (?S i) (other quality_increases {i})"
by - (rule oreachable_other')

from or and ll have next_hop: "∀ dip. let nhip = the (nhop (rt (σ i)) dip)
in dip ∈ kD(rt (σ i)) ∧ nhip ≠ dip
→ dip ∈ kD(rt (σ nhip))
∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ nhip)) dip"
by (auto dest!: onl_oinvariant_weakenD [OF seq_compare_next_hop'])

from or and ll have unk_hops_one: "∀ dip ∈ kD (rt (σ i)). sqn (rt (σ i)) dip = 0
→ sqnf (rt (σ i)) dip = unk
∧ the (dhops (rt (σ i)) dip) = 1
∧ the (nhop (rt (σ i)) dip) = dip"
by (auto dest!: onl_oinvariant_weakenD [OF ozero_seq_unk_hops_one [OF oaodv_trans aodv_trans]]
otherwith_actionD
simp: seqlsimp)

from (other quality_increases {i} σ σ') have "σ' i = σ i" by auto
hence "other quality_increases {i} (σ' i)" by auto
with (other quality_increases {i} σ σ') have "∀ j. quality_increases (σ j) (σ' j)"
by - (erule otherE, metis singleton_iff)

show "∀ dip. dip ∈ vD (rt (σ' i))
∧ dip ∈ vD (rt (σ' (the (nhop (rt (σ' i)) dip))))
∧ the (nhop (rt (σ' i)) dip) ≠ dip
→ rt (σ' i) □ dip rt (σ' (the (nhop (rt (σ' i)) dip))))"

proof clarify
fix dip
assume "dip ∈ vD(rt (σ' i))"
and "dip ∈ vD(rt (σ' (the (nhop (rt (σ' i)) dip))))"
and "the (nhop (rt (σ' i)) dip) ≠ dip"
from this(1) and (σ' i = σ i) have "dip ∈ vD(rt (σ i))"
and "dip ∈ kD(rt (σ i))"
by auto
from (the (nhop (rt (σ' i)) dip) ≠ dip) and (σ' i = σ i)
have "the (nhop (rt (σ i)) dip) ≠ dip" (is "?nhip ≠ ") by simp
with "dip ∈ kD(rt (σ i))" and next_hop have "dip ∈ kD(rt (σ (?nhip)))"
and nsqns: "nsqn (rt (σ i)) dip ≤ nsqn (rt (σ ?nhip)) dip"
by (auto simp: Let_def)

have "0 < sqn (rt (σ i)) dip"
proof (rule neq0_conv [THEN iffD1, OF notI])
assume "sqn (rt (σ i)) dip = 0"
with ⟨dip ∈ kD(rt (σ i))⟩ and unk_hops_one
have "?nhip = dip" by simp
with (?nhip ≠ dip) show False ..
qed
also have "... = nsqn (rt (σ i)) dip"
by (rule vD_nsqn_sqn [OF ⟨dip ∈ vD(rt (σ i))⟩, THEN sym])
also have "... ≤ nsqn (rt (?nhip)) dip"
by (rule nsqns)
also have "... ≤ sqn (rt (?nhip)) dip"
by (rule nsqn_sqn)
finally have "0 < sqn (rt (?nhip)) dip".

have "rt (σ i) ⊑ dip rt (σ' ?nhip)"
proof (cases "dip ∈ vD(rt (σ i))")
assume "dip ∈ vD(rt (σ ?nhip))"
with ⟨dip ∈ vD(rt (σ i))⟩ and (?nhip ≠ dip)
have "rt (σ i) ⊑ dip rt (σ ?nhip)" by auto
moreover from ∀ j. quality_increases (σ j) (σ j')
have "quality_increases (σ ?nhip) (σ' ?nhip)" ..
ultimately show ?thesis
using ⟨dip ∈ kD(rt (σ ?nhip))⟩
by (rule strictly_fresher_quality_increases_right)
next
assume "dip ∈ vD(rt (σ ?nhip))"
with ⟨dip ∈ vD(rt (σ i))⟩ and ⟨?nhip ≠ dip⟩
have "rt (σ i) ⊑ dip rt (σ ?nhip)" by auto
hence "the (flag (rt (σ i)) dip) = inv" by auto
have "nsqn (rt (σ i)) dip ≤ nsqn (rt (σ ?nhip)) dip"
by (rule nsqns)
also from ⟨dip ∈ vD(rt (σ ?nhip))⟩
have "... = sqn (rt (σ ?nhip)) dip - 1" ..
also have "... < sqn (rt (σ' ?nhip)) dip"
proof -
from ∀ j. quality_increases (σ j) (σ j')
have "quality_increases (σ ?nhip) (σ' ?nhip)" ..
hence "∀ ip. sqn (rt (σ ?nhip)) ip ≤ sqn (rt (σ' ?nhip)) ip" by auto
hence "sqn (rt (σ ?nhip)) dip ≤ sqn (rt (σ' ?nhip)) dip" ..
with ⟨0 < sqn (rt (σ ?nhip)) dip⟩ show ?thesis by auto
qed
also have "... = nsqn (rt (σ ?nhip)) dip"
proof (rule vD_nsqn_sqn [THEN sym])
from ⟨dip ∈ vD(rt (σ' (the (nhop (rt (σ i)) dip))))⟩ and ⟨σ' i = σ i⟩
show "dip ∈ vD(rt (σ' ?nhip))" by simp
qed
finally have "nsqn (rt (σ i)) dip < nsqn (rt (σ' ?nhip)) dip" .

moreover from ⟨dip ∈ vD(rt (σ' (the (nhop (rt (σ i)) dip))))⟩ and ⟨σ' i = σ i⟩
have "dip ∈ kD(rt (σ' ?nhip))" by auto
ultimately show "rt (σ i) ⊑ dip rt (σ' ?nhip)"
using ⟨dip ∈ kD(rt (σ i))⟩ by - (rule rt_strictly_fresher_ltI)
qed
with ⟨σ' i = σ i⟩ show "rt (σ i) ⊑ dip rt (σ' (the (nhop (rt (σ i)) dip)))" by simp
qed
qed
thus ?thesis unfolding Let_def .
qed

lemma seq_compare_next_hop:
  fixes w shows "opadv i |=} (otherwith (op=) {i} (orecvmsg msg_fresh),
  other quality_increases {i} →)
global (λσ. ∀ dip. let nhip = the (nhop (rt (σ i)) dip)
in dip ∈ kD(rt (σ i)) ∧ nhip ≠ dip →
dip ∈ kD(rt (σ nhip)) ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ nhip)) dip"
by (rule oinvariant_weakenE [OF seq_compare_next_hop']) (auto dest!: onlD)

lemma seq_nhop_quality_increases:
shows "opaodv i | (otherwith (op=) {i}
(orecvmsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
other quality_increases {i} →
global (λσ. ∀ dip. let nhip = the (nhop (rt (σ i)) dip)
in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
→ (rt (σ i)) ⊏ dip (rt (σ nhip)))"
by (rule oinvariant_weakenE [OF seq_nhop_quality_increases']) (auto dest!: onlD)
end

4.10 Routing graphs and loop freedom

theory D_Loop_Freedom
imports D_Aodv_Predicates D_Fresher
begin

Define the central theorem that relates an invariant over network states to the absence of loops in the associate routing graph.

definition
rt_graph :: "("ip ⇒ state) ⇒ ip ⇒ ip rel"
where
"rt_graph σ = (λdip.
{(ip, ip') | ip ip' dsn dsk hops pre.
ip ≠ dip ∧ rt (σ ip) dip = Some (dsn, dsk, val, hops, ip', pre))"

Given the state of a network σ, a routing graph for a given destination ip address dip abstracts the details of routing tables into nodes (ip addresses) and vertices (valid routes between ip addresses).

lemma rt_graphE [elim]:
fixes n dip ip ip'
assumes "(ip, ip') ∈ rt_graph σ dip"
shows "ip ≠ dip ∧ (∃ r. rt (σ ip) = r
∧ (∃ dsn dsk hops pre. r dip = Some (dsn, dsk, val, hops, ip', pre)))"
using assms unfolding rt_graph_def by auto

lemma rt_graph_vD [dest]:
"∀ ip ip' σ dip. (ip, ip') ∈ rt_graph σ dip → dip ∈ vD(rt (σ ip))"
unfolding rt_graph_def vD_def by auto

lemma rt_graph_vD_trans [dest]:
"∀ ip ip' σ dip. (ip, ip') ∈ (rt_graph σ dip)+ → dip ∈ vD(rt (σ ip))"
by (erule converse_tranclE) auto

lemma rt_graph_not_dip [dest]:
"∀ ip ip' σ dip. (ip, ip') ∈ rt_graph σ dip → ip ≠ dip"
unfolding rt_graph_def by auto

lemma rt_graph_not_dip_trans [dest]:
"∀ ip ip' σ dip. (ip, ip') ∈ (rt_graph σ dip)+ → ip ≠ dip"
by (erule converse_tranclE) auto

NB: the property below cannot be lifted to the transitive closure

lemma rt_graph_nhip_is_nhop [dest]:
"∀ ip ip' σ dip. (ip, ip') ∈ rt_graph σ dip → ip = the (nhop (rt (σ ip)) dip)"
unfolding rt_graph_def by auto

theorem inv_to_loop_freedom:
assumes "∀ i dip. let nhip = the (nhop (rt (σ i)) dip)"
\[
\begin{align*}
\text{in dip} & \in vD (rt (\sigma i)) \cap vD (rt (\sigma \text{nhip})) \land \text{nhip} \neq \text{dip} \\
& \quad \rightarrow (rt (\sigma i)) \sqsubseteq_{\text{dip}} (rt (\sigma \text{nhip}))
\end{align*}
\]

shows "\(\forall\text{dip}. \text{irrefl} ((rt_graph \sigma \text{dip})^+)\"

using \text{assms proof (intro allI)}

fix \(\sigma : \text{"ip }\Rightarrow\text{ state}\) and dip

assume inv: "\(\forall\text{ip dip. let nhip = the (nhop (rt (\sigma \text{ip})) dip) in dip \in vD(rt (\sigma \text{ip})) \land vD(rt (\sigma \text{nhip})) \land \text{nhip} \neq \text{dip} \rightarrow rt (\sigma \text{ip}) \sqsubseteq_{\text{dip}} rt (\sigma \text{nhip})\)"

\{ fix ip ip' \\
assume "(ip, ip') \in (rt_graph \sigma \text{dip})^+
\land \ "\text{dip} \in vD(rt (\sigma \text{ip}'))"
\land \ "ip' \neq \text{dip}"
\hence "rt (\sigma \text{ip}) \sqsubseteq_{\text{dip}} rt (\sigma \text{ip}')"

proof induction

fix nhip

assume "(ip, nhip) \in rt_graph \sigma \text{dip}"
\land \ "\text{dip} \in vD(rt (\sigma \text{nhip}))"
\land \ "\text{nhip} \neq \text{dip}"

from ⟨(ip, nhip) ∈ rt_graph σ dip⟩ have "dip ∈ vD(rt (σ ip))"
\land \ "nhip = the (nhop (rt (σ ip)) dip)"

by auto

from ⟨dip ∈ vD(rt (σ ip))⟩ and ⟨dip ∈ vD(rt (σ nhip))⟩
have "dip ∈ vD(rt (σ ip)) \land vD(rt (σ nhip))" ..
with \(\text{nhip} = \text{the (nhop (rt (\sigma \text{nhip})) dip)}\)
\land \ "\text{nhip} \neq \text{dip}"
\land inv

show "rt (\sigma \text{ip}) \sqsubseteq_{\text{dip}} rt (\sigma \text{nhip})"

by (clarsimp simp: Let_def)

next

fix nhip nhip'

assume "(ip, nhip) \in rt_graph \sigma \text{dip}"
\land \ "(nhip, nhip') \in rt_graph \sigma \text{dip}"
\land \ IH: "\[ dip \in vD(rt (\sigma \text{nhip})) \land \text{nhip} \neq \text{dip} \] \implies rt (\sigma \text{ip}) \sqsubseteq_{\text{dip}} rt (\sigma \text{nhip})"
\land \ "dip ∈ vD(rt (\sigma \text{nhip}')"
\land \ "\text{nhip'} \neq \text{dip}"

from ⟨(nhip, nhip') ∈ rt_graph σ dip⟩ have 1: "dip ∈ vD(rt (σ nhip))" ..
\land \ 2: "\text{nhip} \neq \text{dip}"
\land \ "\text{nhip'} = \text{the (nhop (rt (\sigma \text{nhip})) dip)}" ..

by auto

from 1 2 have "rt (\sigma \text{ip}) \sqsubseteq_{\text{dip}} rt (\sigma \text{nhip})" by (rule IH)
also have "rt (\sigma \text{nhip}) \sqsubseteq_{\text{dip}} rt (\sigma \text{nhip}')"

proof -

from "dip ∈ vD(rt (\sigma \text{nhip}))" and "dip ∈ vD(rt (\sigma \text{nhip}'))" have "dip ∈ vD(rt (\sigma \text{nhip})) \land vD(rt (\sigma \text{nhip}'))" ..
\land \ "\text{nhip'} \neq \text{dip}"
\land \ "\text{nhip'} = \text{the (nhop (rt (\sigma \text{nhip})) dip)}" 
\land inv

show "rt (\sigma \text{nhip}) \sqsubseteq_{\text{dip}} rt (\sigma \text{nhip}')"

by (clarsimp simp: Let_def)

qed

finally show "rt (\sigma \text{ip}) \sqsubseteq_{\text{dip}} rt (\sigma \text{nhip}')" .

qed } note fresher = this

show "irrefl ((rt_graph \sigma \text{dip})^+)"

unfolding irrefl_def proof (intro allI notI)

fix ip

assume "(ip, ip) \in (rt_graph \sigma \text{dip})^+
\land \ "\text{dip} \in vD(rt (\sigma \text{ip}))"
\land \ "ip \neq \text{dip}"

by auto

ultimately have "rt (\sigma \text{ip}) \sqsubseteq_{\text{dip}} rt (\sigma \text{ip})" by (rule fresher)
thus False by simp

qed
4.11 Lift and transfer invariants to show loop freedom

theory D_Aodv_Loop_Freedom
imports ../../../AWN/Qmsg_Lifting D_Global_Invariants D_Loop_Freedom
begin

4.11.1 Lift to parallel processes with queues

lemma par_step_no_change_on_send_or_receive:
  fixes σ s a σ' s'
  assumes "((σ, s), a, (σ', s')) ∈ oparp_sos i (oseqp_sos ΓAODV i) (seqp_sos ΓQMSG)"
  and "a ≠ τ"
  shows "σ' i = σ i"
using assms by (rule qmsg_no_change_on_send_or_receive)

lemma par_nhop_quality_increases:
  shows "opaodv i ⟨⟨ i qmsg |≡ (otherwith (op=) {i} (orecmsg (λσ. msg_fresh σ m ∧ msg_zhops m)), other quality_increases {i} →)
                           global (λσ. ∀ dip. let nhip = the (nhop (rt (σ i)) dip)
                                         in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
                                         → (rt (σ i)) ⊏ dip (rt (σ nhip)))⟩
                 ⟩ (other with quality_increases {i} (orecvmsg (λ_. rreq_rrep_sn)) σ σ' a)"
proof (rule lift_into_qmsg [OF seq_nhop_quality_increases])
  show "opaodv i |≡ (otherwith (op=) {i} (orecmsg (λσ. msg_fresh σ m ∧ msg_zhops m)), other quality_increases {i} →)
                     global (λσ. ∀ dip. let nhip = the (nhop (rt (σ i)) dip)
                                 in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
                                 → (rt (σ i)) ⊏ dip (rt (σ nhip)))" by (cases t) (clarsimp dest!: onllD, metis aodv_ex_label)
next
  fix t :: "(((nat ⇒ state) × (state, msg, pseqp, pseqp label) seqp), msg seq_action) transition"
  assume "onll ΓAODV (λ((σ, _), _, (σ', _)). quality_increases (σ j) (σ' j)) t" thus "quality_increases (fst (fst t) i) (fst (snd (snd t)) i)" by (erule weaken_otherwith, auto)
  thus ?thesis by (erule ostep_invariant_weakenE [OF olocal_quality_increases], simp_all)
qed

lemma par_rreq_rrep_sn_quality_increases:
  "opaodv i ⟨⟨ i qmsg |≡ (otherwith (op=) {i} (orecmsg (λ_. orecvmsg (λ_. rreq_rrep_sn) σ), other ((λ_ _). True) {i} →)
                      globala (λ(σ, _, σ'). quality_increases (σ i) (σ' i)))⟩
     ⟩ (other with quality_increases {i} (orecvmsg (λ_. rreq_rrep_sn)) σ σ' a)"
proof -
  have "opaodv i |≡ (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ), other ((λ_ _). True) {i} →)
                        globala (λ(σ, _, σ'). quality_increases (σ i) (σ' i))" by (rule ostep_invariant_weakenE [OF olocal_quality_increases])
  hence "opaodv i ⟨⟨ i qmsg |≡ (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ), other ((λ_ _). True) {i} →)
                      globala (λ(σ, _, σ'). quality_increases (σ i) (σ' i))⟩
               ⟩ (other with quality_increases {i} (orecvmsg (λ_. rreq_rrep_sn)) σ σ' a)" by (rule lift_step_into_qmsg_statelessassm) simp_all
  thus ?thesis by rule auto
qed

lemma par_rreq_rrep_nsqn_fresh_any_step:
  "opaodv i ⟨⟨ i qmsg |≡ (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ), other ((λ_ _). True) {i} →)
                      globala (λ(σ, _, σ'). anycast (msg_fresh σ) a)⟩
     ⟩ (other with quality_increases {i} (orecvmsg (λ_. rreq_rrep_sn)) σ σ' a)"
proof -

have "opaodv i \models (\lambda \sigma . (orecvmsg (\lambda _. rreq_rrep_sn)) \sigma), other (\lambda _. True) \{i\} \rightarrow  
globala (\lambda(\sigma, a, \sigma'). anycast (msg_fresh \sigma) a)"

proof (rule ostep_invariant_weakenE [OF rreq_rrep_msg_fresh_any_step_invariant])
  fix t
  assume "onll \Gamma_{AODV} (\lambda((\sigma, _), a, _). anycast (msg_fresh \sigma) a) t"
  thus "globala (\lambda(\sigma, a, \sigma'). anycast (msg_fresh \sigma) a) t"
    by (cases t) (clarsimp dest!: onllD, metis aodv_ex_label)
qed auto

hence "opaodv i (\{i qmsg \models (\lambda \sigma . (orecvmsg (\lambda_. rreq_rrep_sn)) \sigma), other (\lambda _. True) \{i\} )  
globala (\lambda(\sigma, a, _). anycast (msg_fresh \sigma) a)"
by (rule lift_step_into_qmsg_statelessasm) simp_all
thus ?thesis by rule auto

qed

---

4.11.2 Lift to nodes

**Lemma node_step_no_change_on_send_or_receive:**

assumes "\{(\sigma, NodeS i P R), a, (\sigma', NodeS i' P' R')\} \in onode_sos  
(oparp_sos i (oseqp_sos \Gamma_{AODV} i) (seqp_sos \Gamma_{QMSG}))"

and "a \neq \tau"

shows "\sigma' = \sigma i"

using assms
by (cases a) (auto elim!: par_step_no_change_on_send_or_receive)

**Lemma node_nhop_quality_increases:**

shows "\{(i : opaodv i (\{i qmsg \models R\}, o |)= A (\lambda \sigma . oarrivemsg (\lambda_. rreq_rrep_sn)) \sigma), other (\lambda _. True) \{i\} \rightarrow  
globala (\lambda(\sigma, _). quality_increases (\sigma i) (\sigma' i))"
by (rule node_lift_step_statelessasm [OF par_rreq_rrep_sn_quality_increases]) simp

**Lemma node_rreq_rrep_nsqn_fresh_any_step:**

shows "\{(i : opaodv i (\{i qmsg \models R\}, o |)= A (\lambda \sigma . oarrivemsg (\lambda_. rreq_rrep_sn)) \sigma), other (\lambda _. True) \{i\} \rightarrow  
globala (\lambda(\sigma, _). quality_increases (\sigma i) (\sigma' i))"
by (rule node_lift_step_statelessasm [OF par_rreq_rrep_sn_quality_increases]) simp

**Lemma node_quality_increases:**

shows "\{(i : opaodv i (\{i qmsg \models R\}, o |)= A (\lambda \sigma . oarrivemsg (\lambda_. rreq_rrep_sn)) \sigma), other (\lambda _. True) \{i\} \rightarrow  
globala (\lambda(\sigma, _). quality_increases (\sigma i) (\sigma' i))"
by (rule node_lift_step_statelessasm [OF par_rreq_rrep_sn_quality_increases]) simp

**Lemma par_anycast_msg_zhops:**

shows "oppaoDV i (\{i qmsg \models (\lambda \sigma . orecvmsg (\lambda_. rreq_rrep_sn)) \sigma), other (\lambda _. True) \{i\} \rightarrow  
globala (\lambda(\sigma, a, _). anycast msg_zhops a)"
by (rule lift_step_into_qmsg_statelessasm) simp_all
thus ?thesis by rule auto

qed
lemma node_anycast_msg_zhops:
  shows "(i : opaodv i (i qmsg : R) false) \rightarrow 
    (\lambda \sigma_. oarrivemsg (\lambda . rreq_rrep_sn) \sigma), 
    other (\lambda _. True) {i} \rightarrow 
    globala (\lambda (sigma, a, s'). castmsg (msg_fresh sigma) a)"
by (rule node_lift_anycast_statelessassm [OF par_rreq_rrep_nsqn_fresh_any_step])

lemma node_silent_change_only:
  shows "(i : opaodv i (i qmsg : R) false) \rightarrow 
    (\lambda \sigma_. oarrivemsg (\lambda . rreq_rrep_sn) \sigma), 
    other (\lambda _. True) {i} \rightarrow 
    globala (\lambda (sigma, a, s'). castmsg msg_zhops a)"
by (rule node_lift_anycast_statelessassm [OF par_anycast_msg_zhops])

lemma node_silent_change:
  shows "(i : opaodv i (i qmsg : R) false) \rightarrow 
    (\lambda \sigma_. oarrivemsg (\lambda . rreq_rrep_sn) \sigma), 
    other (\lambda _. True) {i} \rightarrow 
    globala (\lambda (sigma, a, s'). castmsg msg_zhops a)"
proof (rule ostep_invariantI, simp (no_asm), rule impI)
  have "\forall i R 
    (\exists (sigma, a, s'). a \neq \tau \rightarrow \sigma' = \sigma i)"
  and tr: "((\sigma, \zeta), a, (\sigma', \zeta')) \in trans ((i : opaodv i (i qmsg : R) false))"
  and "a \neq \tau_n"
  from or obtain p R where "\zeta = NodeS i p R"
  by (drule node_net_state, metis)
  with tr have "((\sigma, \zeta), a, (\sigma', \zeta')) \in onode_sos (oparp_sos i (trans (opaodv i))) (trans qmsg))"
  by simp
  thus "\sigma' = \sigma i" using (a \neq \tau_n)
  by (cases rule: onode_sos_cases)
      (auto elim: qmsg_no_change_on_send_recieve)
qed

4.11.3 Lift to partial networks

lemma arrive_rreq_rrep_nsqn_fresh_inc_sn [simp]:
  assumes "oarrivemsg (\lambda . m. msg_fresh sigma m \land P sigma m) sigma m"
  shows "oarrivemsg (\lambda . rreq_rrep_sn) sigma m"
  using assms by (cases m) auto

lemma opnet_nhopt_quality_increases:
  shows "opnet (\lambda i. opaodv i (i qmsg : R) false) \rightarrow 
    (\forall qmsg : R. \forall i \in net_tree_ips p \land dip.
      let nhip = the (nhop (rt (\sigma i)) dip)
      in dip \in VD (rt (\sigma i) \land dip (rt (\sigma nhip)) \land nhip \neq dip)
      \rightarrow (rt (\sigma i) \subseteq dip (rt (\sigma nhip)))"
proof (rule pnet_lift [OF node_nhopt_quality_increases])
  have "(i : opaodv i (i qmsg : R) false) \rightarrow 
    (\lambda \sigma_. oarrivemsg (\lambda . rreq_rrep_sn) \sigma), 
    other (\lambda _. True) {i} \rightarrow 
    globala (\lambda (sigma, a, s'). castmsg (msg_fresh sigma) a)"
  proof (rule ostep_invariantI, simp (no_asm))
    have "(i : opaodv i (i qmsg : R) false) \rightarrow 
      (\lambda \sigma_. oarrivemsg (\lambda . rreq_rrep_sn) \sigma), 
      other (\lambda _. True) {i} \rightarrow 
      globala (\lambda (sigma, a, s'). castmsg (msg_fresh sigma) a)"
  proof (rule ostep_invariantD [OF node_rreq_rrep_nsqn_fresh_any_step])
    moreover from or tr have "castmsg (msg_zhops a) a"
    by (auto dest!: ostep_invariantD [OF node_anycast_msg_zhops])
    ultimately show "castmsg (\lambda m. msg_fresh sigma m \land msg_zhops m) a"
by (case_tac a) auto

qed

thus "\langle i : opaodv i \langle\langle i \ qmsg \ : R \rangle_\rangle \sigma, other \ quality_increases \ \{i\} \rightarrow \ \rangle \ globala \ (\lambda(\sigma, a, \_)).
castmsg \ (\lambda m. \ msg_fresh \sigma m \wedge \msg_zhops m) \ a\)"

by rule auto

next

fix \ i R\n
show "\langle i : opaodv i \langle\langle i \ qmsg \ : R \rangle_\rangle \sigma, other \ quality_increases \ \{i\} \rightarrow \ \rangle \ globala \ (\lambda(\sigma, a, \sigma').
a = \tau \wedge (\forall d. \ a \neq i:deliver(d)) \rightarrow \ \sigma \ i = \sigma' \ i)"

by (rule ostep_invariant_weakenE [OF node_silent_change_only]) auto

next

fix \ i R\n
show "\langle i : opaodv i \langle\langle i \ qmsg \ : R \rangle_\rangle \sigma, other \ quality_increases \ \{i\} \rightarrow \ \rangle \ globala \ (\lambda(\sigma, a, \sigma').
a = \tau \vee (\exists d. \ a = i:deliver(d)) \rightarrow \ quality_increases \ (\sigma \ i) \ (\sigma' \ i))"

by (rule ostep_invariant_weakenE [OF node_quality_increases]) auto

qed simp_all

4.11.4 Lift to closed networks

lemma onet_nhop_quality_increases:

shows "oclosed (\langle i. \ opaodv i \langle\langle i \ qmsg \ : p \rangle_\rangle \ |\ = \ (otherwith (\op=) (\net_tree_ips p) inoclosed, \?U \rightarrow \ ?inv"

proof (rule inclosed_closed)

from opnet_nhop_quality_increases

show "opnet (\langle i. \ opaodv i \langle\langle i \ qmsg \ : p \rangle_\rangle \ |\ = (otherwith (\op=) (\net_tree_ips p) inoclosed, \?U \rightarrow \ ?inv"

proof (rule oinvariant_weakenE)

fix \ \sigma \ \sigma':: \ "ip \ \Rightarrow \ state" \ and \ \a:: \ "msg \ node_action"

assume "otherwith (\op=) (\net_tree_ips p) inoclosed \ \sigma \ \sigma' \ \a"

thus "otherwith (\op=) (\net_tree_ips p)

(oarrivemsg \ (\lambda \sigma m. \ msg_fresh \sigma m \wedge \msg_zhops m)) \ \sigma \ \sigma' \ \a"

proof (rule otherwithEI)

fix \ \sigma:: \ "ip \ \Rightarrow \ state" \ and \ \a:: \ "msg \ node_action"

assume "inoclosed \ \sigma \ \a"

thus "oarrivemsg \ (\lambda \sigma m. \ msg_fresh \sigma m \wedge \msg_zhops m) \ \sigma \ \a"

proof (cases \ \a)

fix \ \i i ni ms

assume "a = ii\ni:arrive(ms)"

moreover with \inoclosed \ \sigma \ \a\ load \ d \ di \ where \ "ms = newpkt(d, \ di)"

by (cases \ \ms) auto

ultimately show \ ?thesis \ by \ simp

qed simp_all

qed

4.11.5 Transfer into the standard model

interpretation aodv_openproc: openproc paodv opaodv id

where "aodv_openproc.initmissing = initmissing"

proof -

show "openproc paodv opaodv id"

proof unfold_locales
fix i :: ip
have "\{ (σ, ζ). (σ i, ζ) ∈ σ AODV i \ & \ (∀ j \neq i \ → \ σ j \in \text{fst} \ ' \ σ AODV j) \} \subseteq σ AODV'"
  unfolding σ AODV_def σ AODV'_def
proof (rule equalityD1)
  show "\{ f. \{ (σ, ζ). (σ i, ζ) ∈ \{ (f i, p) \} \ & \ (∀ j \neq i \ → \ σ j \in \{ (f j, p) \}) \} = \{ (f, p) \}"
    by (rule set_eqI) auto
qed

thus "\{ (σ, ζ) | σ ζ s. s ∈ init (paodv i) \ & \ (σ i, ζ) = id s \ & \ (∀ j \neq i \ → \ σ j \in \text{fst o id} ' \ init (paodv j)) \} \subseteq \text{init (opaodv i)}"
by simp

next
  show "∀ j. init (paodv j) \neq {}"
  unfolding σ AODV_def
  by simp

next
  fix i s a s' σ σ'
  assume "σ i = fst (id s)"
  and "σ' i = fst (id s')"
  and "(s, a, s') ∈ \text{trans (paodv i)}"
  then obtain q q' where "s = (σ i, q)"
  and "s' = (σ' i, q')"
  and "((σ i, q), a, (σ' i, q')) ∈ \text{trans (paodv i)}"
  by (cases s, cases s') auto

  from this(3) have ""(σ, q, a, (σ', q')) ∈ \text{trans (opaodv i)}"
    by simp (rule open_seqp_action [OF aodv_wf])

  with \( s = (σ i, q) \) and \( s' = (σ' i, q') \)
  show ""(σ, snd (id s)), a, (σ', snd (id s')) \in \text{trans (opaodv i)}"
    by simp
qed

interpretation aodv_openproc_par_qmsg: openproc_parq paodv opaodv id qmsg
where "\text{aodv_openproc_par_qmsg.netglobal = netglobal}"
  and "\text{aodv_openproc_par_qmsg.initmissing = initmissing}"
proof -
  show "\text{openproc_parq paodv opaodv id qmsg}"
    by (unfold_locales) simp
then interpret opq: openproc_parq paodv opaodv id qmsg .

have im: "\{ i. \text{openproc.initmissing (λ i. paodv i) (λ p, q). (fst (id p), snd (id p), q)) \} σ = \text{initmissing } σ"
  unfolding opq.initmissing_def opq.someinit_def initmissing_def
  unfolding σ AODV_def σ QMSG_def by (clarsimp cong: option.case_cong)
thus "\text{openproc.initmissing (λ i. paodv i) (λ p, q). (fst (id p), snd (id p), q)} = \text{initmissing}"
  by (rule ext)

have "\{ P σ. \text{openproc.netglobal (λ i. paodv i) (λ p, q). (fst (id p), snd (id p), q)} P σ = \text{netglobal } P σ\}"
  unfolding opq.netglobal_def netglobal_def opq.initmissing_def initmissing_def
  unfolding σ AODV_def σ QMSG_def
  by (clarsimp cong: option.case_cong)
  simp del: One_nat_def
  simp add: fst_initmissing_netmap_default_aodv_init_netlift [symmetric, unfolded initmissing_def])
thus "\text{openproc.netglobal (λ i. paodv i) (λ p, q). (fst (id p), snd (id p), q)} = \text{netglobal}"
  by auto
lemma net_nhop_quality_increases:
assumes "wf_net_tree n"
shows "closed (pnet (λi. paodv i ⟨⟨ qmsg⟩⟩ n) |= netglobal
(λ. ∀ i dip. let nhip = the (nhop (rt (σ i))) dip
in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
→ (rt (σ i)) ⊑ dip (rt (σ nhip)))"
(is "_ |= netglobal (λσ. ∀ i. ?inv σ i")
proof -
from "wf_net_tree n"
have proto: "closed (pnet (λi. paodv i ⟨⟨ qmsg⟩⟩ n) |= netglobal (λσ. ∀ i∈net_tree_ips n. ∀ dip.
let nhip = the (nhop (rt (σ i))) dip
in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
→ (rt (σ i)) ⊑ dip (rt (σ nhip)))"
by (rule aodv_openproc_par_qmsg.close_opnet [OF _ onet_nhop_quality_increases])
show ?thesis
unfolding invariant_def opnet_sos.opnet_tau1
proof (rule, simp only: aodv_openproc_par_qmsg.netglobalsimp
fst_initmissing_netgmap_pair_fst, rule allI)
fix σ i
assume sr: "σ ∈ reachable (closed (pnet (λi. paodv i ⟨⟨ qmsg⟩⟩ n)) TT"
 hence "∀ i∈net_tree_ips n. ?inv (fst (initmissing (netgmap fst σ))) i"
by (drule invariantD [OF proto],
simp only: aodv_openproc_par_qmsg.netglobalsimp
fst_initmissing_netgmap_pair_fst)
thus "?inv (fst (initmissing (netgmap fst σ))) i"
proof (cases "i∈net_tree_ips n")
assume "i∉net_tree_ips n"
from sr have "σ ∈ reachable (pnet (λi. paodv i ⟨⟨ qmsg⟩⟩ n) TT" ..
 hence "net_ips σ = net_tree_ips n" ..
with "i∉net_tree_ips n" have "i∉net_ips σ" by simp
hence "(fst (initmissing (netgmap fst σ))) i = aodv_init i"
by simp
thus ?thesis by simp
qed metis
qed

4.11.6 Loop freedom of AODV

theorem aodv_loop_freedom:
assumes "wf_net_tree n"
shows "closed (pnet (λi. paodv i ⟨⟨ qmsg⟩⟩ n) |= netglobal (λσ. ∀ dip. irrefl ((rt_graph σ dip)⁺))"
using assms by (rule aodv_openproc_par_qmsg.netglobal_weakenE
[OF net_nhop_quality_increases inv_to_loop_freedom])
end
Chapter 5

Variants A–D: All proposed modifications

This model combines the changes proposed in each of the individual variant models.

5.1 Predicates and functions used in the AODV model

theory E_Aodv_Data
imports E_All_ABCD
begin

5.1.1 Sequence Numbers

Sequence numbers approximate the relative freshness of routing information.

definition inc :: "sqn ⇒ sqn"
  where "inc sn ≡ if sn = 0 then sn else sn + 1"

lemma less_than_inc [simp]: "x ≤ inc x"
  unfolding inc_def by simp

lemma inc_minus_suc_0 [simp]:
  "inc x - Suc 0 = x"
  unfolding inc_def by simp

lemma inc_never_one' [simp, intro]: "inc x ≠ Suc 0"
  unfolding inc_def by simp

lemma inc_never_one [simp, intro]: "inc x ≠ 1"
  by simp

5.1.2 Modelling Routes

A route is a t-tuple, (dsn, dsk, flag, hops, nhip) where dsn is the ‘destination sequence number’, dsk is the ‘destination-sequence-number status’, flag is the route status, hops is the number of hops to the destination, and nhip is the next hop toward the destination.

type_synonym r = "sqn × k × f × nat × ip"

definition proj2 :: "r ⇒ sqn" ("π₂")
  where "π₂ ≡ λ(dsn, _, _, _, _). dsn"

definition proj3 :: "r ⇒ k" ("π₃")
  where "π₃ ≡ λ(_, dsk, _, _, _). dsk"

definition proj4 :: "r ⇒ f" ("π₄")
  where "π₄ ≡ λ(_, _, flag, _, _). flag"

definition proj5 :: "r ⇒ nat" ("π₅")
  where "π₅ ≡ λ(_, _, _, hops, _). hops"

definition proj6 :: "r ⇒ ip" ("π₆")
where \( \pi_6 \equiv \lambda(\_, \_, \_, \_, \_, \text{nhip}). \text{nhip} \)

lemma \( \text{projs [simp]} \):
- \( \pi_2(\text{dsn}, \text{dsk}, \text{flag}, \text{hops}, \text{nhip}) = \text{dsn} \)
- \( \pi_3(\text{dsn}, \text{dsk}, \text{flag}, \text{hops}, \text{nhip}) = \text{dsk} \)
- \( \pi_4(\text{dsn}, \text{dsk}, \text{flag}, \text{hops}, \text{nhip}) = \text{flag} \)
- \( \pi_5(\text{dsn}, \text{dsk}, \text{flag}, \text{hops}, \text{nhip}) = \text{hops} \)
- \( \pi_6(\text{dsn}, \text{dsk}, \text{flag}, \text{hops}, \text{nhip}) = \text{nhip} \)
  by (clarsimp simp: proj2_def proj3_def proj4_def proj5_def proj6_def)+

lemma \( \text{proj3_pred [intro]} \):
- \( \pi_3(x) \rightarrow P \)
  by (rule \text{k.induct})

lemma \( \text{proj4_pred [intro]} \):
- \( \pi_4(x) \rightarrow P \)
  by (rule \text{f.induct})

lemma \( \text{proj6_pair_snd [simp]} \):
  \( \pi_6(\text{dsn'}, \text{snd}(r)) = \pi_6(r) \)
  by (cases \( r \)) simp

5.1.3 Routing Tables

Routing tables map ip addresses to route entries.

type synonym \( \text{rt} = \text{"ip \rightarrow r"} \)

syntax
- \( _\Sigma \text{route} :: \text{"rt} \Rightarrow \text{ip} \Rightarrow \text{r} \) (\( \sigma \text{route}'(\_, \_)' \))

translations
- \( \sigma \text{route}(\text{rt}, \text{dip}) \Rightarrow \text{"rt dip} \)

definition \( \text{sqn} :: \text{"rt} \Rightarrow \text{ip} \Rightarrow \text{sqn} \)
  where \( \text{sqn rt dip} \equiv \text{case } \sigma \text{route}(\text{rt}, \text{dip}) \text{ of Some } \pi_2(\text{r}) \mid \text{None } \Rightarrow \text{0} \)

definition \( \text{sqnf} :: \text{"rt} \Rightarrow \text{ip} \Rightarrow \text{k} \)
  where \( \text{sqnf rt dip} \equiv \text{case } \sigma \text{route}(\text{rt}, \text{dip}) \text{ of Some } \pi_3(\text{r}) \mid \text{None } \Rightarrow \text{unk} \)

abbreviation \( \text{flag} :: \text{"rt} \Rightarrow \text{ip} \Rightarrow \text{f} \)
  where \( \text{flag rt dip} \equiv \text{map_option } \pi_4(\sigma \text{route}(\text{rt}, \text{dip})) \)

abbreviation \( \text{dhops} :: \text{"rt} \Rightarrow \text{ip} \Rightarrow \text{nat} \)
  where \( \text{dhops rt dip} \equiv \text{map_option } \pi_5(\sigma \text{route}(\text{rt}, \text{dip})) \)

abbreviation \( \text{nhop} :: \text{"rt} \Rightarrow \text{ip} \Rightarrow \text{ip} \)
  where \( \text{nhop rt dip} \equiv \text{map_option } \pi_6(\sigma \text{route}(\text{rt}, \text{dip})) \)

definition \( \text{vD} :: \text{"rt} \Rightarrow \text{ip set} \)
  where \( \text{vD rt} \equiv \{\text{dip}. \text{flag rt dip } = \text{Some val}\} \)

definition \( \text{iD} :: \text{"rt} \Rightarrow \text{ip set} \)
  where \( \text{iD rt} \equiv \{\text{dip}. \text{flag rt dip } = \text{Some inv}\} \)

definition \( \text{kD} :: \text{"rt} \Rightarrow \text{ip set} \)
  where \( \text{kD rt} \equiv \{\text{dip}. \text{rt dip } \neq \text{None}\} \)

lemma \( \text{kD_is_vD_and_iD} :: \text{"kD rt } = \text{vD rt} \cup \text{iD rt} \)
  unfolding \( \text{kD_def vD_def iD_def by auto} \)

lemma \( \text{vD_iD_gives_kD [simp]} \):
- \( \forall \text{rt}. \text{ip } \in \text{vD rt } \Rightarrow \text{ip } \in \text{kD rt} \)
- \( \forall \text{rt}. \text{ip } \in \text{iD rt } \Rightarrow \text{ip } \in \text{kD rt} \)
  unfolding \( \text{kD_is_vD_and_iD by simp_all} \)
lemma kD_Some [dest]:
  fixes dip rt
assumes "dip ∈ kD rt"
shows "∃dsn dsk flag hops nhip.
  σroute(rt, dip) = Some (dsn, dsk, flag, hops, nhip)"
using assms unfolding kD_def by simp

lemma kD_None [dest]:
  fixes dip rt
assumes "dip ∉ kD rt"
shows "σroute(rt, dip) = None"
using assms unfolding kD_def
by (metis (mono_tags) mem_Collect_eq)

lemma vD_Some [dest]:
  fixes dip rt
assumes "dip ∈ vD rt"
shows "∃dsn dsk hops nhip.
  σroute(rt, dip) = Some (dsn, dsk, val, hops, nhip)"
using assms unfolding vD_def by simp

lemma vD_empty [simp]: "vD Map.empty = {}"
unfolding vD_def by simp

lemma iD_Some [dest]:
  fixes dip rt
assumes "dip ∈ iD rt"
shows "∃dsn dsk hops nhip.
  σroute(rt, dip) = Some (dsn, dsk, inv, hops, nhip)"
using assms unfolding iD_def by simp

lemma val_is_vD [elim]:
  fixes ip rt
assumes "ip ∈ kD(rt)"
  and "the (flag rt ip) = val"
shows "ip ∈ vD(rt)"
using assms unfolding vD_def by auto

lemma inv_is_iD [elim]:
  fixes ip rt
assumes "ip ∈ kD(rt)"
  and "the (flag rt ip) = inv"
shows "ip ∈ iD(rt)"
using assms unfolding iD_def by auto

lemma iD_flag_is_inv [elim, simp]:
  fixes ip rt
assumes "ip ∈ iD(rt)"
shows "the (flag rt ip) = inv"
proof -
  from ⟨ip ∈ iD(rt)⟩ have "ip ∈ kD(rt)" by auto
  with assms show ?thesis unfolding iD_def by auto
  qed

lemma kD_but_not_vD_is_iD [elim]:
  fixes ip rt
assumes "ip ∈ kD(rt)"
  and "ip ∉ vD(rt)"
shows "ip ∈ iD(rt)"
proof -
  from ⟨ip ∈ kD(rt)⟩ obtain dsn dsk f hops nhop
    where rtip: "rt ip = Some (dsn, dsk, f, hops, nhop)"
    by (metis kD_Some)
from \( \langle \text{ip} \notin vD(rt) \rangle \) have "\( f \neq \text{val} \)

proof (rule contrapos_nn)
    assume "\( f = \text{val} \)
    with \( \text{rtip} \) have "\( \text{the (flag rt ip)} = \text{val} \)" by simp
    with \( \langle \text{ip} \in kD(rt) \rangle \) show "\( \text{ip} \in vD(rt) \)" ..
    qed
with \( \text{rtip} \) have "\( \text{the (flag rt ip)} = \text{inv} \)" by simp
    with \( \langle \text{ip} \in kD(rt) \rangle \) show "\( \text{ip} \in iD(rt) \)" ..
    qed

lemma vD_or_iD [elim]:
    fixes \( \text{ip} \ \text{rt} \)
    assumes  "\( \text{ip} \in kD(rt) \)"
and  "\( \text{ip} \in vD(rt) \) \implies \( \text{P rt ip} \)"
and  "\( \text{ip} \in iD(rt) \) \implies \( \text{P rt ip} \)
    shows  "\( \text{P rt ip} \)"
proof -
    from \( \langle \text{ip} \in kD(rt) \rangle \) have "\( \text{ip} \in vD(rt) \cup iD(rt) \)"
        by (simp add: kD_is_vD_and_iD)
    thus ?thesis by (auto elim: assms(2-3))
    qed

lemma proj5_eq_dhops: "\( \cdot \cdot \cdot \)
    unfolding sqn_def by (drule kD_Some) clarsimp

lemma proj4_eq_flag: "\( \cdot \cdot \cdot \)
    unfolding sqn_def by (drule kD_Some) clarsimp

lemma proj2_eq_sqn: "\( \cdot \cdot \cdot \)
    unfolding sqn_def by (drule kD_Some) clarsimp

lemma kD_sqnf_is_proj3 [simp]:
    "\( \cdot \cdot \cdot \)
    unfolding sqnf_def by auto

lemma vD_flag_val [simp]:
    "\( \cdot \cdot \cdot \)
    unfolding vD_def byclarsimp

lemma kD_update [simp]:
    "\( \cdot \cdot \cdot \)
    unfolding kD_def by auto

lemma ip_equal_or_known [elim]:
    fixes \( \text{rt} \ \text{ip} \ \text{ip}' \)
    assumes  "\( \text{ip} = \text{ip}' \lor \text{ip} \in kD(rt) \)"
and  "\( \text{ip} = \text{ip}' \implies \text{P rt ip ip}' \)"
and  "\( \lceil \text{ip} \neq \text{ip}'; \text{ip} \in kD(rt) \rceil \implies \text{P rt ip ip}' \)"
    shows  "\( \text{P rt ip ip}' \)"
    using assms by auto

5.1.4 Updating Routing Tables

Routing table entries are modified through explicit functions. The properties of these functions are important in invariant proofs.

Updating route entries

lemma in_kD_case [simp]:
    fixes \( \text{dip} \ \text{rt} \)
    assumes  "\( \text{dip} \in kD(rt) \)"
shows "(case rt dip of None ⇒ en | Some r ⇒ es r) = es (the (rt dip))" using assms [THEN kD_Some] by auto

lemma not_in_kD_case [simp]:
  fixes dip rt
  assumes "dip ∉ kD(rt)"
  shows "(case rt dip of None ⇒ en | Some r ⇒ es r) = en"
  using assms [THEN kD_None] by auto

lemma rt_Some_sqn [dest]:
  fixes rt and ip dsn dsk flag hops nhip
  assumes "rt ip = Some (dsn, dsk, flag, hops, nhip)"
  shows "sqn rt ip = dsn"
  unfolding sqn_def using assms by simp

lemma not_kD_sqn [simp]:
  fixes dip rt
  assumes "dip ∉ kD(rt)"
  shows "sqn rt dip = 0"
  using assms unfolding sqn_def by simp

definition update_arg_wf :: "r ⇒ bool"
  where "update_arg_wf r ≡ π 4 (r) = val ∧
  (π 2 (r) = 0) = (π 3 (r) = unk) ∧
  (π 3 (r) = unk −→ π 5 (r) = 1)"

lemma update_arg_wf_gives_cases:
  "∀ r. update_arg_wf r =⇒ (π 2 (r) = 0) = (π 3 (r) = unk)"
  unfolding update_arg_wf_def by simp

lemma update_arg_wf_tuples [simp]:
  "∀ nhip. update_arg_wf (0, unk, val, Suc 0, nhip)"
  "∀ n hops nhip. update_arg_wf (Suc n, kno, val, hops, nhip)"
  unfolding update_arg_wf_def by auto

lemma update_arg_wf_tuples' [elim]:
  "∀ n hops nhip. Suc 0 ≤ n ⇒ update_arg_wf (n, kno, val, hops, nhip)"
  unfolding update_arg_wf_def by auto

lemma wf_r_cases [intro]:
  fixes P r
  assumes "update_arg_wf r"
  and c1: "∀ nhip. P (0, unk, val, Suc 0, nhip)"
  and c2: "∀ dsn hops nhip. dsn > 0 ⇒ P (dsn, kno, val, hops, nhip)"
  shows "P r"
proof -
  obtain dsn dsk flag hops nhip
  where *: "r = (dsn, dsk, flag, hops, nhip)" by (cases r)
  with update_arg_wf_r have wf1: "flag = val"
      and wf2: "dsn = 0 = (dsk = unk)"
      and wf3: "dsk = unk −→ (hops = 1)"
    unfolding update_arg_wf_def by auto
  have "P (dsn, dsk, flag, hops, nhip)"
proof (cases dsk)
  assume "dsk = unk"
  moreover with wf2 wf3 have "dsn = 0" and "hops = Suc 0" by auto
  ultimately show ?thesis using (flag = val) by simp (rule c1)
next
  assume "dsk = kno"
  moreover with wf2 have "dsn > 0" by simp
  ultimately show ?thesis using (flag = val) by simp (rule c2)
qed
with * show "P r" by simp
definition update :: "rt ⇒ ip ⇒ r ⇒ rt"
where
"update rt ip r ≡
  case σ_route(rt, ip) of
  None ⇒ rt (ip ↦→ r)
  | Some s ⇒
    if π_2(s) < π_2(r) then rt (ip ↦→ r)
    else if π_2(s) = π_2(r) ∧ (π_5(s) > π_5(r) ∨ π_4(s) = inv)
      then rt (ip ↦→ r)
    else if π_3(r) = unk
      then rt (ip ↦→ (π_3(s), snd(r)))
    else rt (ip ↦→ s)"

lemma update_simps [simp]:
fixes r s nrt nr' ns rt ip
defines "s ≡ the σ_route(rt, ip)"
and "nr' ≡ (π_2(s), π_3(r), π_4(r), π_5(r), π_6(r))"
shows
  "[ [ ip /∈ kD(rt) ] ] ⇒ update rt ip r = rt (ip ↦→ r)"
  "[ [ ip ∈ kD(rt); sqn rt ip < π_2(r) ] ] ⇒ update rt ip r = rt (ip ↦→ r)"
  "[ [ ip ∈ kD(rt); sqn rt ip = π_2(r);
       the (dhops rt ip) > π_5(r) ] ] ⇒ update rt ip r = rt (ip ↦→ r)"
  "[ [ ip ∈ kD(rt); π_3(r) = unk; (π_2(r) = 0) = (π_3(r) = unk) ] ] ⇒ update rt ip r = rt (ip ↦→ nr')"
  "[ [ ip ∈ kD(rt); sqn rt ip ≥ π_2(r); π_3(r) = kno;
      sqn rt ip = π_2(r) ⇒ the (dhops rt ip) ≤ π_5(r) ∧ the (flag rt ip) = val ] ] ⇒ update rt ip r = rt (ip ↦→ s)"

proof -
  assume "ip /∈ kD(rt)"
  hence "σ_route(rt, ip) = None" ..
  thus "update rt ip r = rt (ip ↦→ r)"
    unfolding update_def by simp
next
  assume "ip ∈ kD(rt)"
  and "sqn rt ip < π_2(r)"
  from this(1) obtain dsn dsk fl hops nhip
    where "rt ip = Some (dsn, dsk, fl, hops, nhip)"
    by (metis kD_Some)
  with ⟨sqn rt ip < π_2(r)⟩ show "update rt ip r = rt (ip ↦→ r)"
    unfolding update_def s_def by auto
next
  assume "ip ∈ kD(rt)"
  and "sqn rt ip = π_2(r)"
  and "the (dhops rt ip) > π_5(r)"
  from this(1) obtain dsn dsk fl hops nhip
    where "rt ip = Some (dsn, dsk, fl, hops, nhip)"
    by (metis kD_Some)
  with ⟨sqn rt ip = π_2(r)⟩ and ⟨the (dhops rt ip) > π_5(r)⟩
    show "update rt ip r = rt (ip ↦→ r)"
    unfolding update_def s_def by auto
next
  assume "ip ∈ kD(rt)"
  and "sqn rt ip = π_2(r)"
  and "flag rt ip = Some inv"
  from this(1) obtain dsn dsk fl hops nhip
    where "rt ip = Some (dsn, dsk, fl, hops, nhip)"
    by (metis kD_Some)
  with ⟨sqn rt ip = π_2(r)⟩ and ⟨flag rt ip = Some inv⟩
    show "update rt ip r = rt (ip ↦→ r)"
    unfolding update_def s_def by auto
next
assume "ip ∈ kD(rt)"
  and "π₃(r) = unk"
  and "(π₂(r) = 0) = (π₃(r) = unk)"
from this(1) obtain dsn dsk fl hops nhip
  where "rt ip = Some (dsn, dsk, fl, hops, nhip)"
  by (metis kD_Some)
with (π₂(r) = 0) = (π₃(r) = unk) and (π₃(r) = unk)
show "update rt ip r = rt (ip 🜢 nr)"
  unfolding update_def nr'_def s_def
by (cases r) simp
next
assume "ip ∈ kD(rt)"
  and otherassms: "sqn rt ip ≥ π₂(r)"
  "π₃(r) = kno"
  "sqn rt ip = π₂(r) =⇒ the (dhops rt ip) ≤ π₅(r) ∧ the (flag rt ip) = val"
from this(1) obtain dsn dsk fl hops nhip
  where "rt ip = Some (dsn, dsk, fl, hops, nhip)"
  by (metis kD_Some)
with otherassms show "update rt ip r = rt (ip 🜢 s)"
  unfolding update_def s_def by auto
qed

lemma update_cases [elim]:
assumes "(π₂(r) = 0) = (π₃(r) = unk)"
  and c1: "[ [ ip /∈ kD(rt) ] =⇒ P (rt (ip 🜢 r)) ]"
  and c2: "[ [ ip ∈ kD(rt); sqn rt ip < π₂(r) ]
              =⇒ P (rt (ip 🜢 r )) ]"
  and c3: "[ [ ip ∈ kD(rt); sqn rt ip = π₂(r); the (dhops rt ip) > π₅(r) ]
              =⇒ P (rt (ip 🜢 r )) ]"
  and c4: "[ [ ip ∈ kD(rt); sqn rt ip = π₂(r); the (flag rt ip) = inv ]
              =⇒ P (rt (ip 🜢 r )) ]"
  and c5: "[ [ ip ∈ kD(rt); π₃(r) = unk ]
              =⇒ P (rt (ip 🜢 (π₃(the σroute(rt, ip)), π₃(r),
                                           π₄(r), π₅(r), π₆(r)))) )]"
  and c6: "[ [ ip ∈ kD(rt); sqn rt ip ≥ π₂(r); π₃(r) = kno; sqn rt ip = π₂(r) =⇒ the (dhops rt ip) ≤ π₅(r) ∧ the (flag rt ip) = val ]
              =⇒ P (rt (ip 🜢 the σroute(rt, ip))) ]"
shows "(P (update rt ip r))"
proof (cases "ip ∈ kD(rt)"
  assume "ip /∈ kD(rt)"
with c1 show ?thesis
  by simp
next
assume "ip ∈ kD(rt)"
moreover then obtain dsn dsk fl hops nhip
  where rteq: "rt ip = Some (dsn, dsk, fl, hops, nhip)"
  by (metis kD_Some)
moreover obtain dsn' dsk' fl' hops' nhip'
  where req: "r = (dsn', dsk', fl', hops', nhip')"
  by (cases r) metis
ultimately show ?thesis
using (π₂(r) = 0) = (π₃(r) = unk)
  c2 [OF (ip∈kD(rt))]
  c3 [OF (ip∈kD(rt))]
  c4 [OF (ip∈kD(rt))]
  c5 [OF (ip∈kD(rt))]
  c6 [OF (ip∈kD(rt))]
  unfolding update_def sqn_def by auto
qed

lemma update_cases_kD:
assumes "(π₂(r) = 0) = (π₃(r) = unk)"
  and "ip ∈ kD(rt)"
and c2: "sqn rt ip < π₂(r) ⇒ P (rt (ip ↦→ r))"
and c3: "[sqn rt ip = π₂(r); the (dhops rt ip) > π₅(r)]
⇒ P (rt (ip ↦→ r))"
and c4: "[sqn rt ip = π₂(r); the (flag rt ip) = inv]
⇒ P (rt (ip ↦→ r))"
and c5: "π₃(r) = unk ⇒ P (rt (ip ↦→ (π₂(the σroute(rt, ip)), π₃(r), π₄(r), π₅(r), π₆(r))))"
and c6: "[sqn rt ip ≥ π₂(r); π₃(r) = kno;
sqn rt ip = π₂(r) ⇒ the (dhops rt ip) ≤ π₅(r) ∧ the (flag rt ip) = val]
⇒ P (rt (ip ↦→ the σroute(rt, ip)))"

shows "(P (update rt ip r))"
using assms(1) proof (rule update_cases)
assume "sqn rt ip < π₂(r)"
thus "P (rt(ip ↦→ r))" by (rule c2)
next
assume "sqn rt ip = π₂(r)"
and "the (dhops rt ip) > π₅(r)"
thus "P (rt(ip ↦→ r))" by (rule c3)
next
assume "sqn rt ip = π₂(r)"
and "the (flag rt ip) = inv"
thus "P (rt(ip ↦→ r))" by (rule c4)
next
assume "π₃(r) = unk"
thus "P (rt (ip ↦→ (π₂(the σroute(rt, ip)), π₃(r), π₄(r), π₅(r), π₆(r))))"
by (rule c5)
next
assume "sqn rt ip ≥ π₂(r)"
and "π₃(r) = kno"
and "sqn rt ip = π₂(r) ⇒ the (dhops rt ip) ≤ π₅(r) ∧ the (flag rt ip) = val"
thus "P (rt (ip ↦→ the (rt ip))))"
by (rule c6)
qed (simp add: ⟨ip ∈ kD(rt)⟩)

lemma in_kD_after_update [simp]:
fixes rt nipdsn dsk flag hops nhip pre
shows "kD (update rt nip (dsn, dsk, flag, hops, nhip)) = insert nip (kD rt)"
unfolding update_def
by (cases "rt nip") auto

lemma nhop_of_update [simp]:
fixes rt dipdsn dsk flag hops nhip
assumes "rt ≠ update rt dip (dsn, dsk, flag, hops, nhip)"
says "the (nhop (update rt dip (dsn, dsk, flag, hops, nhip)) dip) = nhip"
proof -
from assms
have update_neq: "∀ v. rt dip = Some v ⇒
update rt dip (dsn, dsk, flag, hops, nhip)
≠ rt(dip ↦→ (the (rt dip)))"
by auto
show ?thesis
proof (cases "rt dip = None")
assume "rt dip = None"
thus "?thesis" unfolding update_def by clarsimp
next
assume "rt dip ≠ None"
then obtain v where "rt dip = Some v" by (metis not_None_eq)
with update_neq [OF this] show ?thesis
  unfolding update_def by auto
qed
qed
lemma sqn_if_updated:
  fixes rip v rt ip
  shows "sqn (λx. if x = rip then Some v else rt x) ip = (if ip = rip then π_2(v) else sqn rt ip)"
  unfolding sqn_def by simp

lemma update_sqn [simp]:
  fixes rt dip rip dsn dsk flag hops nhip
  assumes "(dsn = 0) = (dsk = unk)"
  shows "sqn rt dip ≤ sqn (update rt rip (dsn, dsk, val, hops, nhip)) dip"
  proof (rule update_cases)
    show "(π_2 (dsn, dsk, val, hops, nhip) = 0) = (π_3 (dsn, dsk, val, hops, nhip) = unk)"
      by simp (rule assms)
  qed (clarsimp simp: sqn_if_updated sqn_def)+

lemma update_bigger [simp]:
  fixes rt ip ip' dsn dsk flag hops nhip
  assumes "1 ≤ hops"
  shows "sqn rt ip ≤ sqn (update rt ip' (dsn, dsk, flag, hops, nhip)) ip"
  using assms unfolding update_def sqn_def
  by (clarsimp split: option.split) auto

lemma dhops_update [intro]:
  fixes rt dsn dsk flag hops ip rip nhip
  assumes ex: "∀ip∈kD rt. the (dhops rt ip) ≥ 1"
  and ip: "(ip = rip ∧ Suc 0 ≤ hops) ∨ (ip ≠ rip ∧ ip∈kD rt)"
  shows "Suc 0 ≤ the (dhops (update rt rip (dsn, dsk, flag, hops, nhip)) ip)"
  using ip proof
    assume "ip = rip ∧ Suc 0 ≤ hops" thus ?thesis
    unfolding update_def using ex
    by (cases "rip ∈ kD rt") (drule(1) bspec, auto)
  next
    assume "ip ≠ rip ∧ ip∈kD rt" thus ?thesis
    using ex unfolding update_def
    by (cases "rip∈kD rt") auto
  qed

lemma update_another [simp]:
  fixes dip ip rt dsn dsk flag hops nhip
  assumes "ip ≠ dip"
  shows "(update rt dip (dsn, dsk, flag, hops, nhip)) ip = rt ip"
  using assms unfolding update_def
  by (clarsimp split: option.split)

lemma nhop_update_another [simp]:
  fixes dip ip rt dsn dsk flag hops nhip
  assumes "ip ≠ dip"
  shows "nhop (update rt dip (dsn, dsk, flag, hops, nhip)) ip = nhop rt ip"
  using assms unfolding update_def
  by (clarsimp split: option.split)

lemma dhops_update_another [simp]:
  fixes dip ip rt dsn dsk flag hops nhip
  assumes "ip ≠ dip"
  shows "dhops (update rt dip (dsn, dsk, flag, hops, nhip)) ip = dhops rt ip"
  using assms unfolding update_def
  by (clarsimp split: option.split)

lemma sqn_update_same [simp]:
  "∀rt ip dsn dsk flag hops nhip. sqn (rt(ip → v)) ip = π_2(v)"
  unfolding sqn_def by simp

lemma dhops_update_changed [simp]:
  fixes rt dip osn hops nhip
assumes "rt ≠ update rt dip (osn, kno, val, hops, nhip)"
  shows "the (dhops (update rt dip (osn, kno, val, hops, nhip)) dip) = hops"
using assms unfolding update_def
by (clarsimp split: option.split_asm option.split split_if_asm) auto

lemma nhop_update_unk_val [simp]:
  "\rt dip ip dsn hops.
   the (nhop (update rt dip (dsn, unk, val, hops, ip)) dip) = ip"
unfolding update_def by (clarsimp split: option.split)

lemma nhop_update_changed [simp]:
  fixes rt dip dsn dsk flg hops sip
assumes "update rt dip (dsn, dsk, flg, hops, sip) ≠ rt"
  shows "the (nhop (update rt dip (dsn, dsk, flg, hops, sip)) dip) = sip"
using assms unfolding update_def
by (clarsimp split: option.splits split_if_asm) auto

lemma update_rt_split_asm:
  "\rt ip dsn dsk flag hops sip.
   P (update rt ip (dsn, dsk, flag, hops, sip)) =
   (¬(rt = update rt ip (dsn, dsk, flag, hops, sip) ∧ ¬P rt
    ∨ rt ≠ update rt ip (dsn, dsk, flag, hops, sip)
    ∧ ¬P (update rt ip (dsn, dsk, flag, hops, sip))))"
by auto

lemma sqn_update [simp]: "\rt dip dsn flg hops sip.
  rt ≠ update rt dip (dsn, kno, flg, hops, sip) ⟹ sqn (update rt dip (dsn, kno, flg, hops, sip)) dip = dsn"
unfolding update_def by (clarsimp split: option.split split_if_asm) auto

lemma sqnf_update [simp]: "\rt dip dsn dsk flg hops sip.
  rt ≠ update rt dip (dsn, dsk, flg, hops, sip) ⟹ sqnf (update rt dip (dsn, dsk, flg, hops, sip)) dip = dsk"
unfolding update_def sqnf_def
by (clarsimp split: option.splits split_if_asm) auto

lemma update_kno_dsn_greater_zero:
  "\rt ip dsn hops. 1 ≤ dsn ⟹ 1 ≤ (sqn (update rt ip (dsn, kno, val, hops, ip)) dip)"
unfolding update_def
by (clarsimp split: option.splits)

lemma proj3_update [simp]: "\rt dip dsn dsk flg hops sip.
  rt ≠ update rt dip (dsn, dsk, flg, hops, sip) ⟹ π₃(the (update rt dip (dsn, dsk, flg, hops, sip)) dip) = dsk"
unfolding update_def sqnf_def
by (clarsimp split: option.splits split_if_asm) auto

lemma nhop_update_changed_kno_val [simp]: "\rt ip dsn dsk hops nhip.
  rt ≠ update rt ip (dsn, kno, val, hops, nhip) ⟹ the (nhop (update rt ip (dsn, kno, val, hops, nhip)) ip) = nhip"
using assms unfolding update_def
by (clarsimp split: option.split_asm option.split split_if_asm) auto

lemma flag_update [simp]: "\rt dip dsn flg hops sip.
  rt ≠ update rt dip (dsn, kno, flg, hops, sip) ⟹ the (flag (update rt dip (dsn, kno, flg, hops, sip)) dip) = flg"
unfolding update_def
by (clarsimp split: option.split split_if_asm) auto

lemma the_flag_Some [dest!]:
  fixes ip rt
assumes "the (flag rt ip) = x"
  and "ip ∈ kD rt"

424
shows "flag rt ip = Some x"
using assms by auto

lemma kD_update_unchanged [dest]:
  fixes rt dip dsn dsk flag hops nhip
assumes "rt = update rt dip (dsn, dsk, flag, hops, nhip)"
  shows "dip∈kD(rt)"
proof -
  have "dip∈kD(update rt dip (dsn, dsk, flag, hops, nhip))" by simp
  with assms show ?thesis by simp
qed

lemma nhop_update [simp]: "∀rt dip dsn dsk flg hops sip.
  rt ≠ update rt dip (dsn, dsk, flg, hops, sip)
  ⇒ the (nhop (update rt dip (dsn, dsk, flg, hops, sip)) dip) = sip"
unfolding update_def sqnf_def
by (clarsimp split: option.splits split_if_asm) auto

lemma sqn_update_another [simp]:
  fixes dip ip rt dsn dsk flag hops nhip
  assumes "ip ≠ dip"
  shows "sqn (update rt dip (dsn, dsk, flag, hops, nhip)) ip = sqn rt ip"
using assms unfolding update_def sqn_def
by (clarsimp split: option.splits) auto

lemma sqnf_update_another [simp]:
  fixes dip ip rt dsn dsk flag hops nhip
  assumes "ip ≠ dip"
  shows "sqnf (update rt dip (dsn, dsk, flag, hops, nhip)) ip = sqnf rt ip"
using assms unfolding update_def sqnf_def
by (clarsimp split: option.splits) auto

lemma vD_update_val [dest]:
  "∀dip rt dip' dsn dsk hops nhip.
  dip ∈ vD(update rt dip' (dsn, dsk, val, hops, nhip))" ⇒ (dip∈vD(rt) ∨ dip=dip')"
unfolding update_def vD_def by (clarsimp split: option.splits_split_if_asm)

Invalidating route entries

definition invalidate :: "rt ⇒ (ip ⇀ sqn) ⇒ rt"
where "invalidate rt dests ≡
λip. case (rt ip, dests ip) of
  (None, _) ⇒ None
  | (Some s, None) ⇒ Some s
  | (Some (_, dsk, _, hops, nhip), Some rsn) ⇒
    Some (rsn, dsk, inv, hops, nhip)"

lemma proj3_invalidate [simp]:
  "∀dip. π3(the ((invalidate rt dests) dip)) = π3(the (rt dip))"
unfolding invalidate_def by (clarsimp split: option.split)

lemma proj5_invalidate [simp]:
  "∀dip. π5(the ((invalidate rt dests) dip)) = π5(the (rt dip))"
unfolding invalidate_def by (clarsimp split: option.split)

lemma proj6_invalidate [simp]:
  "∀dip. π6(the ((invalidate rt dests) dip)) = π6(the (rt dip))"
unfolding invalidate_def by (clarsimp split: option.split)

5.1.5 Route Requests

lemma invalidate_kD_inv [simp]:
  "∀rt dests. kD (invalidate rt dests) = kD rt"
unfolding invalidate_def kD_def
by (simp split: option.split)
lemma invalidate_sqn:
  fixes rt dip dests
  assumes "∀ rsn. dests dip = Some rsn ⟹ sqn rt dip ≤ rsn"
  shows "sqn rt dip ≤ sqn (invalidate rt dests) dip"
  proof (cases "dip ∉ kD(rt)"
    assume "¬ dip ∉ kD(rt)"
    hence "dip ∈ kD(rt)" by simp
    then obtain dsn dsk flag hops nhip where "rt dip = Some (dsn, dsk, flag, hops, nhip)"
      by (metis kD_Some)
    with assms show "sqn rt dip ≤ sqn (invalidate rt dests) dip"
      by (cases "dests dip") (auto simp add: invalidate_def sqn_def)
  qed simp

lemma sqn_invalidate_in_dests [simp]:
  fixes dests ipa rsn rt
  assumes "dests ipa = Some rsn"
  and "ipa ∈ kD(rt)"
  shows "sqn (invalidate rt dests) ipa = rsn"
  unfolding invalidate_def sqn_def
  using assms(1) assms(2) [THEN kD_Some]
  by clarsimp

lemma dhops_invalidate [simp]:
  "∀ dip. the (dhops (invalidate rt dests) dip) = the (dhops rt dip)"
  unfolding invalidate_def
  by (clarsimp split: option.split)

lemma sqnf_invalidate [simp]:
  "∀ dip. sqnf (invalidate (rt ξ) (dests ξ)) dip = sqnf (rt ξ) dip"
  unfolding sqnf_def invalidate_def
  by (clarsimp split: option.split)

lemma nhop_invalidate [simp]:
  "∀ dip. the (nhop (invalidate (rt ξ) (dests ξ)) dip) = the (nhop (rt ξ) dip)"
  unfolding invalidate_def
  by (clarsimp split: option.split)

lemma invalidate_other [simp]:
  fixes rt dests dip dsn dsk flag hops nhip pre
  assumes "invalidate rt dests dip = Some (dsn, dsk, flag, hops, nhip)"
  shows "dsn = (case dests dip of None ⇒ π₂ (the (rt dip)) | Some rsn ⇒ rsn) ∧ dsk = π₃ (the (rt dip))"
∧ flag = (if dests dip = None then π₄(the (rt dip)) else inv)
∧ hops = π₅(π₄(the (rt dip)))
∧ nhip = π₆(π₄(the (rt dip)))

using assms unfolding invalidate_def
by (cases "rt dip", clarsimp, cases "dests dip") auto

lemma proj3_inv: ""/\ dip rt dests. dip ∈ kD (rt)
           → π₃(the (invalidate rt dests dip)) = π₃(the (rt dip))"
by (clarsimp simp: invalidate_def kD_def split: option.split)

lemma dests_iD_invalidate [simp]:
assumes "dests ip = Some rsn"
  and "ip ∈ kD(rt)"
shows "ip ∈ iD(invalidate rt dests)"
using assms(1) assms(2) [THEN kD_Some]
  unfolding invalidate_def iD_def
by (clarsimp split: option.split)

5.1.6 Queued Packets

Functions for sending data packets.

type synonym store = "ip ↦ (p × data list)"

definition sigma_queue :: "store ⇒ ip ⇒ data list" ("σ_queue('_, _)")
  where "σ_queue(store, dip) ≡ case store dip of None ⇒ [] | Some (p, q) ⇒ q"

definition qD :: "store ⇒ ip set"
  where "qD ≡ dom"

definition add :: "data ⇒ ip ⇒ store ⇒ store"
  where "add d dip store ≡ case store dip of
            None ⇒ store (dip := None)
            | Some (p, q) ⇒ store (dip := (p, q @ [d]))"
3. $\sigma_{p-flag}(store, dip)$ is partial.

4. The routing table ($rt$) is modelled as a map ($ip \Rightarrow r$ option) rather than a set of 7-tuples, likewise, the $r$ is a 6-tuple rather than a 7-tuple, i.e., the destination ip-address ($dip$) is taken from the argument to the function, rather than a part of the result. Well-definedness then follows from the structure of the type and more related facts are available automatically, rather than having to be acquired through tedious proofs.

5. Similar remarks hold for the dests mapping passed to invalidate, and $store$.

end

5.2 AODV protocol messages

theory E_Aodv_Message
imports E_All_ABCD
begin

datatype msg =
  Rreq nat ip sqn k ip sqn ip bool
| Rrep nat ip sqn ip ip
| Rerr "ip → sqn" ip
| Newpkt data ip
| Pkt data ip ip

instantiation msg :: msg
begin
  definition newpkt_def [simp]: "newpkt ≡ λ(d, dip). Newpkt d dip"
  definition eq_newpkt_def: "eq_newpkt m ≡ case m of Newpkt d dip ⇒ True | _ ⇒ False"

  instance by intro_classes (simp add: eq_newpkt_def)
end

The $msg$ type models the different messages used within AODV. The instantiation as a $msg$ is a technicality due to the special treatment of $newpkt$ messages in the AWN SOS rules. This use of classes allows a clean separation of the AWN-specific definitions and these AODV-specific definitions.

definition rreq :: "nat × ip × sqn × k × ip × sqn × ip × bool ⇒ msg"
where "rreq ≡ λ(hops, dip, dsn, dsk, oip, osn, sip, handled). Rreq hops dip dsn dsk oip osn sip handled"
lemma rreq_simp [simp]:
"rreq(hops, dip, dsn, dsk, oip, osn, sip, handled) = Rreq hops dip dsn dsk oip osn sip handled"
unfolding rreq_def by simp

definition rrep :: "nat × ip × sqn × ip × ip ⇒ msg"
where "rrep ≡ λ(hops, dip, dsn, oip, sip). Rrep hops dip dsn oip sip"
lemma rrep_simp [simp]:
"rrep(hops, dip, dsn, oip, sip) = Rrep hops dip dsn oip sip"
unfolding rrep_def by simp

definition rerr :: "(ip → sqn) × ip ⇒ msg"
where "rerr ≡ λ(dests, sip). Rerr dests sip"
lemma rerr_simp [simp]:
"rerr(dests, sip) = Rerr dests sip"
unfolding rerr_def by simp

lemma not_eq_newpkt_rreq [simp]: "¬eq_newpkt (Rreq hops dip dsn dsk oip osn sip handled)"
unfolding eq_newpkt_def by simp

lemma not_eq_newpkt_rrep [simp]: "¬eq_newpkt (Rrep hops dip dsn oip sip)"
unfolding eq_newpkt_def by simp
lemma not_eq_newpkt_rerr [simp]: "¬eq_newpkt (Rerr dests sip)"
  unfolding eq_newpkt_def by simp

lemma not_eq_newpkt_pkt [simp]: "¬eq_newpkt (Pkt d dip sip)"
  unfolding eq_newpkt_def by simp

definition pkt :: "data × ip × ip ⇒ msg"
  where "pkt ≡ λ(d, dip, sip). Pkt d dip sip"

lemma pkt_simp [simp]:
  "pkt(d, dip, sip) = Pkt d dip sip"
  unfolding pkt_def by simp

end

5.3 The AODV protocol

theory E_Aodv
 imports E_Aodv_Data E_Aodv_Message
  "../../../AWN/AWN_SOS_Labels" "../../../AWN/AWN_Invariants"

begin

5.3.1 Data state

record state =
  ip :: "ip"
  sn :: "sqn"
  rt :: "rt"
  rreqs :: "(ip × sqn) set"
  store :: "store"
  msg :: "msg"
  data :: "data"
  dests :: "ip ⇒ sqn"
  dip :: "ip"
  oip :: "ip"
  hops :: "nat"
  dsn :: "sqn"
  dsk :: "k"
  osn :: "sqn"
  sip :: "ip"
  handled:: "bool"

abbreviation aodv_init :: "ip ⇒ state"
  where "aodv_init i ≡ |
    ip = i,
    sn = 1,
    rt = empty,
    rreqs = {},
    store = empty,
    msg = (SOME x. True),
    data = (SOME x. True),
    dests = (SOME x. True),
    dip = (SOME x. True),
    oip = (SOME x. True),
    hops = (SOME x. True),
    dsn = (SOME x. True),
    dsk = (SOME x. True),
    osn = (SOME x. True),
    sip = (SOME x. x ≠ i),
    handled= (SOME x. True)"
lemma some_neq_not_eq [simp]: "\((\text{SOME } x :: \text{nat} \ \cdot \ x \neq 1) = 1\)"
by (subst some_eq_ex) (metis zero_neq_numeral)

definition clear_locals :: "state \Rightarrow\ state"
where "clear_locals \xi = \xi (\langle msg := (\text{SOME } x. \text{True}),
data := (\text{SOME } x. \text{True}),
dests := (\text{SOME } x. \text{True}),
dip := (\text{SOME } x. \text{True}),
oip := (\text{SOME } x. \text{True}),
dsn := (\text{SOME } x. \text{True}),
dsk := (\text{SOME } x. \text{True}),
osn := (\text{SOME } x. \text{True}),
sip := (\text{SOME } x. x \neq ip \xi),
handled:= (\text{SOME } x. \text{True})\rangle)"

lemma clear_locals_sip_not_ip [simp]: "\neg (sip (clear_locals \xi) = ip \xi)"
unfolding clear_locals_def by simp

lemma clear_locals_but_notGlobals [simp]:
"ip (clear_locals \xi) = ip \xi"
"sn (clear_locals \xi) = sn \xi"
"rt (clear_locals \xi) = rt \xi"
"rreqs (clear_locals \xi) = rreqs \xi"
"store (clear_locals \xi) = store \xi"
unfolding clear_locals_def by auto

5.3.2 Auxilliary message handling definitions

definition is_newpkt
where "is_newpkt \xi \equiv \text{case } msg \xi \text{ of }
\langle \text{Newpkt data'} dip' \rangle \Rightarrow \{ \xi (\langle data := data', dip := dip' \rangle) \}
| _ \Rightarrow \{\}"

definition is_pkt
where "is_pkt \xi \equiv \text{case } msg \xi \text{ of }
\langle \text{Pkt data'} dip' oip' \rangle \Rightarrow \{ \xi (\langle data := data', dip := dip', oip := oip' \rangle) \}
| _ \Rightarrow \{\}"

definition is_rreq
where "is_rreq \xi \equiv \text{case } msg \xi \text{ of }
\langle \text{Rreq hops'} dip' dsn' dsk' osn' sip' handled' \rangle \Rightarrow
\{ \xi (\langle hops := hops', dip := dip', dsn := dsn',
dsk := dsk', oip := oip', osn := osn', sip := sip',
handled := handled' \rangle) \}
| _ \Rightarrow \{\}"

lemma is_rreq_asm [dest!]:
assumes "\xi' \in is_rreq \xi"
shows "(\exists \text{hops'} dip' dsn' dsk' oip' osn' sip' handled').
\xi' = \xi (\langle \text{Rreq hops'} dip' dsn' dsk' oip' osn' sip' handled' \land
\xi' = \xi (\langle \text{hops := hops', dip := dip', dsn := dsn',
dsk := dsk', oip := oip', osn := osn', sip := sip',
handled := handled' \rangle)\)"
using assms unfolding is_rreq_def
by (cases "msg \xi") simp_all

definition is_rrep
where "is_rrep \xi \equiv \text{case } msg \xi \text{ of}
Rrep hops' dip' dsn' oip' sip' ⇒
| _ ⇒ { }"
lemma is_msg_inv_store [simp]:
  "ξ' ∈ is_rerr ξ ⇒ store ξ' = store ξ"
  "ξ' ∈ is_rrep ξ ⇒ store ξ' = store ξ"
  "ξ' ∈ is_rreq ξ ⇒ store ξ' = store ξ"
  "ξ' ∈ is_pkt ξ ⇒ store ξ' = store ξ"
  "ξ' ∈ is_newpkt ξ ⇒ store ξ' = store ξ"

unfolding is_msg_defs
by (cases "msg ξ", clarsimp+)+

lemma is_msg_inv_sip [simp]:
  "ξ' ∈ is_pkt ξ ⇒ sip ξ' = sip ξ"
  "ξ' ∈ is_newpkt ξ ⇒ sip ξ' = sip ξ"

unfolding is_msg_defs
by (cases "msg ξ", clarsimp+)+

5.3.3 The protocol process

datatype pseqp =
  PAodv
| PNewPkt
| PPkt
| PRreq
| PRrep
| PRerr

fun nat_of_seqp :: "pseqp ⇒ nat"
where
  "nat_of_seqp PAodv = 1"
| "nat_of_seqp PPkt = 2"
| "nat_of_seqp PNewPkt = 3"
| "nat_of_seqp PRreq = 4"
| "nat_of_seqp PRrep = 5"
| "nat_of_seqp PRerr = 6"

instantiation "pseqp" :: ord
begin
definition less_eq_seqp [iff]: "l1 ≤ l2 = (nat_of_seqp l1 ≤ nat_of_seqp l2)"
definition less_seqp [iff]: "l1 < l2 = (nat_of_seqp l1 < nat_of_seqp l2)"
instance ..
end

abbreviation AODV
where
  "AODV ≡ λ. [clear_locals] call(PAodv)"

abbreviation PKT
where
  "PKT args ≡
  [ξ. let (data, dip, oip) = args ξ in
   (clear_locals ξ) () data := data, dip := dip, oip := oip ]
call(PPkt)"

abbreviation NEWPKT
where
  "NEWPKT args ≡
  [ξ. let (data, dip) = args ξ in
   (clear_locals ξ) () data := data, dip := dip ]
call(PNewPkt)"

abbreviation RREQ
where
  "RREQ args ≡
  [ξ. let (hops, dip, dsn, dsk, oip, osn, sip, handled) = args ξ in
   (clear_locals ξ) () hops := hops, dip := dip,
abbreviation RREP
where
"RREP args ≡
[ξ. let (hops, dip, dsn, oip, sip) = args ξ in
  (clear_locals ξ) () hops := hops, dip := dip, dsn := dsn,
  oip := oip, sip := sip ]
call(PRreq)"

abbreviation RERR
where
"RERR args ≡
[ξ. let (dests, sip) = args ξ in
  (clear_locals ξ) () dests := dests, sip := sip ]
call(PRrep)"

fun ΓAODV :: "((state, msg, pseqp, pseqp label) seqp_env"
where
"ΓAODV PAodv = labelled PAodv |
  receive(λmsg' ξ. ξ (λmsg := msg'))
  ( λis_newpkt) NEWPKT(λξ. (data ξ, ip ξ))
  λ(is_pkt) PKT(λξ. (data ξ, dip ξ, oip ξ))
  λ(is_rreq) [
    [ξ. ξ (λrt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ) )]
      RREQ(λξ. ⟨hops ξ, dip ξ, dsn ξ, oip ξ, osn ξ, sip ξ, handled ξ⟩)
  ]
  λ(is_rrep) [
    [ξ. ξ (λrt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ) )]
      RREP(λξ. ⟨hops ξ, dip ξ, dsn ξ, oip ξ, sip ξ⟩)
  ]
  λ(is_rerr) [
    [ξ. ξ (λrt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ) )]
      RERR(λξ. ⟨dests ξ, sip ξ⟩)
  ]
  λ(is_unicRreq) [
    λξ. { λξ (dip := dip) | / dip. dip ∈ qD(store ξ) ∩ vD(rt ξ) }
    λξ. { λξ (data := hd(queue(store ξ, dip ξ))) |}
    unicast(λξ. λdests (nhop (rt ξ) (dip ξ)), λξ. pkt(data ξ, dip ξ, ip ξ)).
    λξ. { λξ (store := (drop (dip ξ) (store ξ)) |)}
    AODV()
  ]
  λ(is_unicRrep) [
    λξ. { λξ (dip := dip) | / dip. dip ∈ qD(store ξ) ∩ vD(rt ξ) ∧ the (σp_flag(store ξ, dip)) = req }
    λξ. { λξ (data := add (data ξ) (dip ξ) (store ξ)) |}
    λξ. { sn := inc (sn ξ) |}
    λξ. { rreqs := rreqs ξ ∪ {(ip ξ, sn ξ)} |}
    broadcast(λξ. rreq(0, dip ξ, snq (rt ξ) (dip ξ), sqnf (rt ξ) (dip ξ), ip ξ, sn ξ, ip ξ, False)). AODV()"
]"
\( \langle \xi. \text{dip } \xi \in \text{Vd (rt } \xi) \rangle \\
\quad \text{unicast}(\lambda x. \text{the (nhop (rt } \xi) (\text{dip } \xi)), \lambda x. \text{pkt(data } \xi, \text{dip } \xi, \text{oip } \xi)). \text{AODV()} \)

\( \langle \xi. \xi \in \text{AODV } \rangle \\
\quad \text{dests := (} \lambda \text{rip. if (rip } \in \text{Vd (rt } \xi) \wedge \text{nhop (rt } \xi) \text{rip = nhop (rt } \xi) (\text{dip } \xi)) \text{ then Some (inc (sqn (rt } \xi) \text{ rip))) else None) } \rangle] \\
\quad \text{broadcast(} \lambda x. \text{rerr(dests } \xi, \text{ip } \xi)). \text{AODV()} \\
\langle \xi. \text{dip } \xi \notin \text{Vd (rt } \xi) \rangle \\
\quad \text{broadcast(} \lambda x. \text{rerr(} \text{dip } \xi \rightarrow \text{sqn (rt } \xi) (\text{dip } \xi), \text{ip } \xi)). \text{AODV()} \\
\langle \xi, \text{handled } \xi = \text{False} \rangle \\
\quad \langle \xi. \text{dip } \xi = \text{ip } \xi \rangle \\
\quad \langle \xi. \xi \in \text{sn := max (sn } \xi) (\text{dsn } \xi) \rangle \\
\quad \text{unicast}(\lambda x. \text{the (nhop (rt } \xi) (\text{oip } \xi)), \lambda x. \text{rrep(0, dip } \xi, \text{sn } \xi, \text{oip } \xi, \text{ip } \xi)). \\
\quad \text{broadcast(} \lambda x. \text{rreq(hops } \xi + 1, \text{dip } \xi, \text{dsn } \xi, \text{dsk } \xi, \text{oip } \xi, \text{osn } \xi, \text{ip } \xi, \text{True})). \text{AODV()} \\
\langle \xi. \xi \notin \text{ip } \xi \rangle \\
\quad \langle \xi. \text{dip } \xi \in \text{Vd (rt } \xi) \wedge \text{dsn } \xi \leq \text{sqn (rt } \xi) (\text{dip } \xi) \wedge \text{sqnf (rt } \xi) (\text{dip } \xi) = \text{kno) \\
\quad \text{unicast}(\lambda x. \text{the (nhop (rt } \xi) (\text{oip } \xi)), \lambda x. \text{rrep(the (dhops (rt } \xi) (\text{dip } \xi), \text{dip } \xi, \text{sqn (rt } \xi) (\text{dip } \xi), \text{dip } \xi, \text{ip } \xi)). \\
\quad \text{broadcast(} \lambda x. \text{rreq(hops } \xi + 1, \text{dip } \xi, \text{dsn } \xi, \text{dsk } \xi, \text{oip } \xi, \text{osn } \xi, \text{ip } \xi, \text{True})). \text{AODV()} \\
\langle \xi. \xi \notin \text{ip } \xi \rangle \\
\quad \langle \xi. \text{dip } \xi \notin \text{Vd (rt } \xi) \vee \text{sqn (rt } \xi) (\text{dip } \xi) < \text{dsn } \xi \vee \text{sqnf (rt } \xi) (\text{dip } \xi) = \text{unk}) \\
\quad \text{broadcast(} \lambda x. \text{rreq(hops } \xi + 1, \text{dip } \xi, \text{max (sqn (rt } \xi) (\text{dip } \xi) (\text{dsn } \xi), \text{dsk } \xi, \text{oip } \xi, \text{osn } \xi, \text{ip } \xi, \text{False})). \text{AODV()} \\
\rangle \\
\langle \xi, \text{handled } \xi = \text{True} \rangle \\
\quad \text{broadcast(} \lambda x. \text{rreq(hops } \xi + 1, \text{dip } \xi, \text{dsn } \xi, \text{dsk } \xi, \text{oip } \xi, \text{osn } \xi, \text{ip } \xi, \text{True})). \text{AODV()} \rangle$
\[
\begin{align*}
&\langle \xi. \text{oip }\xi = \text{ip }\xi \rangle \\
&AODV() \\
&+ \langle \xi. \text{oip }\xi \neq \text{ip }\xi \rangle \\
&\quad (\langle \xi. \text{oip }\xi \in \text{vD (rt }\xi) \land \text{dip }\xi \in \text{vD (rt }\xi)\rangle \\
&\quad \quad \text{unicast}(\lambda \xi. \text{the (nhop (rt }\xi) (oip }\xi)), \lambda \xi. \text{rrep}(\text{the (dhips (rt }\xi) (dip }\xi)), \text{dip }\xi, \text{sqn (rt }\xi) (\text{dip }\xi), \text{oip }\xi, \text{ip }\xi)). \\
&AODV() \\
&> \langle \xi. \text{oip }\xi \rangle \\
&\quad \text{dests} := (\lambda \text{rip. if (rip }\in \text{vD (rt }\xi) \land \text{nhop (rt }\xi) \text{rip} = \text{nhop (rt }\xi) (\text{oip }\xi)) \\
&\quad \quad \text{then Some (inc (sqn (rt }\xi) \text{rip})) else None) \rangle \\
&\langle \xi. \text{rt} := \text{invalidate (rt }\xi) (\text{dests }\xi) \rangle \\
&\langle \xi. \text{store} := \text{setRRF (store }\xi) (\text{dests }\xi) \rangle \\
&\quad \text{broadcast}(\lambda \xi. \text{rerr(dests }\xi, \text{ip }\xi)). \quad \text{AODV()} \\
&\quad + \langle \xi. \text{oip }\xi \neq \text{vD (rt }\xi) \lor \text{dip }\xi \neq \text{vD (rt }\xi) \rangle \\
&AODV()
\end{align*}
\]

"\(\Gamma_{AODV} P_{\text{PRerr}} = \text{labelled P}_{\text{PRerr}} \{
\langle \xi. dests := (\lambda \text{rip. case (dests }\xi) \text{rip} \text{ of None }\Rightarrow \text{None}
\quad \text{if rip }\in \text{vD (rt }\xi) \land \text{the (nhop (rt }\xi) \text{rip} = \text{sip }\xi
\quad \quad \land \text{sqn (rt }\xi) \text{rip} < \text{rsn then Some rsn else None}) \rangle
\langle \xi. \text{rt} := \text{invalidate (rt }\xi) (\text{dests }\xi) \rangle
\langle \xi. \text{store} := \text{setRRF (store }\xi) (\text{dests }\xi) \rangle
\quad \text{broadcast}(\lambda \xi. \text{rerr(dests }\xi, \text{ip }\xi)). \quad \text{AODV()} \\
\langle \xi. dests }\xi = \text{Map.empty} \rangle
\}
\)"

\begin{verbatim}
declare \Gamma_{AODV}.simps [simp del, code del]
lemmas \Gamma_{AODV}.simps [simp, code] = \Gamma_{AODV}.simps [simplified]
end
\end{verbatim}

fun \Gamma_{AODV}_skeleton where
  "\Gamma_{AODV}_skeleton P_{\text{Aodv}} = \text{seqp_skeleton (}\Gamma_{AODV} P_{\text{Aodv}})"
| "\Gamma_{AODV}_skeleton P_{\text{NewPkt}} = \text{seqp_skeleton (}\Gamma_{AODV} P_{\text{NewPkt}})"
| "\Gamma_{AODV}_skeleton P_{\text{Pkt}} = \text{seqp_skeleton (}\Gamma_{AODV} P_{\text{Pkt}})"
| "\Gamma_{AODV}_skeleton P_{\text{PRreq}} = \text{seqp_skeleton (}\Gamma_{AODV} P_{\text{PRreq}})"
| "\Gamma_{AODV}_skeleton P_{\text{PRrep}} = \text{seqp_skeleton (}\Gamma_{AODV} P_{\text{PRrep}})"
| "\Gamma_{AODV}_skeleton P_{\text{PRerr}} = \text{seqp_skeleton (}\Gamma_{AODV} P_{\text{PRerr}})"

lemma \Gamma_{AODV}_skeleton_wf [simp]:
  "\text{wellformed }\Gamma_{AODV}_skeleton"
proof (rule, intro allI)
  fix pn pn'
  show "\text{call(pn') }\notin \text{ctermsl (}\Gamma_{AODV}_skeleton pn)"
  by (cases pn) simp_all
qed

declare \Gamma_{AODV}_skeleton.simps [simp del, code del]
lemmas \Gamma_{AODV}_skeleton.simps [simp, code] = \Gamma_{AODV}_skeleton.simps [simplified \Gamma_{AODV}.simps seqp_skeleton.simps]

lemma aodv_proc_cases [dest]:
  fixes pn
  shows "\text{p }\in \text{ctermsl (}\Gamma_{AODV} pn) \Rightarrow
  \quad \text{(p }\in \text{ctermsl (}\Gamma_{AODV} P_{\text{Aodv}}) \lor
  \quad \text{p }\in \text{ctermsl (}\Gamma_{AODV} P_{\text{NewPkt}}) \lor
  \quad \text{p }\in \text{ctermsl (}\Gamma_{AODV} P_{\text{Pkt}}) \lor
  \quad \text{p }\in \text{ctermsl (}\Gamma_{AODV} P_{\text{PRreq}}) \lor
  \quad \text{p }\in \text{ctermsl (}\Gamma_{AODV} P_{\text{PRrep}}) \lor
  \quad \text{p }\in \text{ctermsl (}\Gamma_{AODV} P_{\text{PRerr}}) \lor
  \quad \text{p }\in \text{ctermsl (}\Gamma_{AODV} \text{PRerr})\)"
\[ p \in \text{ctermsl} (\Gamma_{AODV} \text{ PPkt}) \lor p \in \text{ctermsl} (\Gamma_{AODV} \text{ PRreq}) \lor p \in \text{ctermsl} (\Gamma_{AODV} \text{ PRrep}) \lor p \in \text{ctermsl} (\Gamma_{AODV} \text{ PRerr}) \]

using assms
by (cases pn) simp_all

**Definition** \( \sigma_{AODV} : \) "\( \text{ip} \Rightarrow (\text{state} \times (\text{state}, \text{msg}, \text{pseqp}, \text{pseqp label}) \text{ seqp}) \text{ set} \)"
where "\( \sigma_{AODV} i \equiv \{(aodv_init i, \Gamma_{AODV} \text{ PAodv})\} \)"

**Abbreviation** \( \text{paodv} : \) "\( \text{ip} \Rightarrow (\text{state} \times (\text{state}, \text{msg}, \text{pseqp}, \text{pseqp label}) \text{ seqp}, \text{msg seq_action}) \text{ automaton} \)"
where "\( \text{paodv} i \equiv (| \text{init} = \sigma_{AODV} i, \text{trans} = \text{seqp sos} \Gamma_{AODV} |) \)"

**Lemma** \( \text{aodv_trans} : \) "\( \text{trans (paodv i) = seqp sos} \Gamma_{AODV} \)"
by simp

**Lemma** \( \text{aodv_control_within [simp]} : \) "\( \text{control_within} \Gamma_{AODV} \text{ (init (paodv i))} \)"
unfolding \( \sigma_{AODV} \text{ def} \) by (rule control_withinI) (auto simp del: \( \Gamma_{AODV} \text{ simps} \))

**Lemma** \( \text{aodv_wf [simp]} : \) "\( \text{wellformed} \Gamma_{AODV} \)"
proof (rule, intro allI)
fix \( \text{pn} \text{ pn'} \)
show "\( \text{call(pn')} \not\in \text{stermsl} (\Gamma_{AODV} \text{ pn}) \)"
by (cases pn) simp_all
qed

**Lemmas** \( \text{aodv_labels_not_empty [simp] = labels_not_empty [OF aodv_wf]} \)

**Lemma** \( \text{aodv_ex_label [intro]} : \) "\( \exists l. l \in \text{labels} \Gamma_{AODV} p \)"
by (metis aodv_labels_not_empty all_not_in_conv)

**Lemma** \( \text{aodv_ex_labelE [elim]} : \) "\( \forall l \in \text{labels} \Gamma_{AODV} p. P \ \\text{1} \ p \)"
and "\( \exists p \text{ 1.} \ P \ \text{1} \ p \ \Rightarrow \ Q \)"
shows "\( Q \)"
using assms by (metis aodv_ex_label)

**Lemma** \( \text{aodv_simple_labels [simp]} : \) "\( \text{simple_labels} \Gamma_{AODV} \)"
proof
fix \( \text{pn} \text{ p} \)
assume "\( \text{p}\in\text{subterms}(\Gamma_{AODV} \text{ pn}) \)"
thus "\( \exists l. \text{ labels} \Gamma_{AODV} p = \{l\} \)"
by (cases pn) (simp_all cong: seqp_congs | elim disjE)+
qed

**Lemma** \( \sigma_{AODV} \text{ labels [simp]} : \) "\( (\xi, p) \in \sigma_{AODV} i \Rightarrow \text{ labels} \Gamma_{AODV} p = \{\text{PAodv-:0}\} \)"
unfolding \( \sigma_{AODV} \text{ def} \) by simp

**Lemma** \( \text{aodv_init_kD_empty [simp]} : \) "\( (\xi, p) \in \sigma_{AODV} i \Rightarrow kD (\text{rt} \xi) = \{\} \)"
unfolding \( \sigma_{AODV} \text{ def} \) kD_def by simp

**Lemma** \( \text{aodv_init_sip_not_ip [simp]} : \) "\( \neg (\text{sip (aodv_init i) = i}) \)"
by simp

**Lemma** \( \text{aodv_init_sip_not_ip' [simp]} : \) "\( \neg (\text{sip (aodv_init i) = i}) \)"
assumes "\( (\xi, p) \in \sigma_{AODV} i \)"
shows "\( \text{sip} \xi \not= \text{ip} \xi \)"
using assms unfolding \( \sigma_{AODV} \text{ def} \) by simp

**Lemma** \( \text{aodv_init_sip_not_i [simp]} : \) "\( (\xi, p) \in \sigma_{AODV} i \)"
assumes "\( (\xi, p) \in \sigma_{AODV} i \)"
shows "sip ≠ i"
using assms unfolding σ_{AODV-def} by simp

lemma clear_locals_sip_not_ip':
  assumes "ip = i"
  shows "¬(sip (clear_locals ip) = i)"
using assms by auto

Stop the simplifier from descending into process terms.
declare seq_congs [cong]

Configure the main invariant tactic for AODV.
declare Γ_{AODV-simps} [cterm_env]
  aodv_proc_cases [ctermsl_cases]
  seq_invariant_ctermsI [OF aodv_wf aodv_control_within aodv_simple_labels aodv_trans, cterms_intros]
  seq_step_invariant_ctermsI [OF aodv_wf aodv_control_within aodv_simple_labels aodv_trans, cterms_intros]
end

5.4 Invariant assumptions and properties

theory E_Aodv_Predicates
imports E_Aodv
begin

Definitions for expression assumptions on incoming messages and properties of outgoing messages.

abbreviation not_Pkt :: "msg ⇒ bool"
where "not_Pkt m ≡ case m of Pkt _ _ _ ⇒ False | _ ⇒ True"
definition msg_sender :: "msg ⇒ ip"
where "msg_sender m ≡ case m of Rreq _ _ _ _ _ _ ipc _ ⇒ ipc
  | Rrep _ _ _ _ ipc ⇒ ipc
  | Rerr _ ipc ⇒ ipc
  | Pkt _ _ ipc ⇒ ipc"
lemma msg_sender_simps [simp]:
  "⋀hops dip dsn dsk oip osn sip handled.
  msg_sender (Rreq hops dip dsn dsk oip osn sip handled) = sip"
  "⋀hops dip dsn oip sip. msg_sender (Rrep hops dip dsn oip sip) = sip"
  "⋀dests sip. msg_sender (Rerr dests sip) = sip"
  "⋀d dip sip. msg_sender (Pkt d dip sip) = sip"
unfolding msg_sender_def by simp_all

definition msg_zhops :: "msg ⇒ bool"
where "msg_zhops m ≡ case m of
  Rreq hopsc dipc _ _ oipc _ _ oipc _ ⇒ hopsc = 0 → oipc = sipc
  | Rrep hopsc dipc _ _ sipc ⇒ hopsc = 0 → dipc = sipc
  | _ ⇒ True"
lemma msg_zhops_simps [simp]:
  "⋀hops dip dsn dsk oip osn sip handled.
  msg_zhops (Rreq hops dip dsn dsk oip osn sip handled) = (hops = 0 → oip = sip)"
  "⋀hops dip dsn oip sip. msg_zhops (Rrep hops dip dsn oip sip) = (hops = 0 → dip = sip)"
  "⋀dests sip. msg_zhops (Rerr dests sip) = True"
  "⋀d dip. msg_zhops (Newpkt d dip) = True"
  "⋀d dip sip. msg_zhops (Pkt d dip sip) = True"
unfolding msg_zhops_def by simp_all

definition rreq_rrep_sn :: "msg ⇒ bool"
where "rreq_rrep_sn m ≡ case m of Rreq _ _ _ _ _ _ osnc _ _ ⇒ osnc ≥ 1"
lemma rreq_rrep_sn_simps [simp]:
  "\(\forall \text{hops dip dsn dsk oip osn sip handled.}
  \text{rreq_rrep_sn (Rreq hops dip dsn dsk oip osn sip handled)} = (osn \geq 1)"\n  "\(\forall \text{hops dip dsn oip sip. rreq_rrep_sn (Rreq hops dip dsn oip sip)} = (dsn \geq 1)"\n  "\(\forall \text{dests sip. rreq_rrep_sn (Rerr dests sip)} = True"\n  "\(\forall \text{d dip. rreq_rrep_sn (Newpkt d dip)} = True"\n  "\(\forall \text{d dip sip. rreq_rrep_sn (Pkt d dip sip)} = True"\n
unfolding rreq_rrep_sn_def by simp_all

definition rreq_rrep_fresh :: "rt \Rightarrow msg \Rightarrow bool"
where "rreq_rrep_fresh crt m ≡ case m of
  \text{Rreq hopsc _ _ _ oipc osnc ipcc _} \Rightarrow
  \text{(ipcc \neq oipc \land sqn crt oipc > osnc \lor
    (sqn crt oipc = osnc \land the (dhops crt oipc) \leq hopsc \land
    the (flag crt oipc) = val))}\n  | \text{Rrep hopsc dipc dsnc _ ipcc} \Rightarrow
  \text{(ipcc \neq dipc \land sqn crt dipc = dsnc \land
    the (dhops crt dipc) = hopsc \land
    the (flag crt dipc) = val)}\n  | _ \Rightarrow True"

lemma rreq_rrep_fresh [simp]:
  "\(\forall \text{hops dip dsn dsk oip osn sip handled.}
  \text{rreq_rrep_fresh (Rreq hops dip dsn dsk oip osn sip handled)} = (sip \neq oip \land
    \text{dip \in kD(crt)} \lor
    \text{sqn crt oip > osn} \lor
    \text{sqn crt oip = osnc} \land
    \text{the (dhops crt oipc) \leq hopsc} \land
    \text{the (flag crt oipc) = val})"\n  "\(\forall \text{hops dip dsn oip sip. rreq_rrep_fresh (Rreq hops dip dsn oip sip)} = (sip \neq dip \land
    \text{dip \in kD(crt)} \land
    \text{sqn crt dip = dsnc} \land
    \text{the (dhops.crt dipc) = hopsc} \land
    \text{the (flag crt dipc) = val})"\n  "\(\forall \text{dests sip. rreq_rrep_fresh (Rerr dests sip)} = True"\n  "\(\forall \text{d dip. rreq_rrep_fresh (Newpkt d dip)} = True"\n  "\(\forall \text{d dip sip. rreq_rrep_fresh (Pkt d dip sip)} = True"\n
unfolding rreq_rrep_fresh_def by simp_all

definition rerr_invalid :: "rt \Rightarrow msg \Rightarrow bool"
where "rerr_invalid crt m ≡ case m of
  \text{Rerr destsc _} \Rightarrow (\forall ripc \in \text{dom(destsc)}.\n    \text{ripc \in iD(crt) \land
    the (destsc ripc) = sqn crt ripc})\n  | _ \Rightarrow True"

lemma rerr_invalid [simp]:
  "\(\forall \text{hops dip dsn dsk oip osn sip handled.}
  \text{rerr_invalid (Rreq hops dip dsn dsk oip osn sip handled)} = True"\n  "\(\forall \text{hops dip dsn oip sip. rerr_invalid (Rreq hops dip dsn oip sip)} = True"\n  "\(\forall \text{dests sip. rerr_invalid (Rerr dests sip) = (\forall ripc \in \text{dom(dests)}.\n    ripc \in iD(crt) \land
    the (dests ripc) = sqn crt ripc)"\n  "\(\forall \text{d dip. rerr_invalid (Newpkt d dip)} = True"\n  "\(\forall \text{d dip sip. rerr_invalid (Pkt d dip sip)} = True"\n
unfolding rerr_invalid_def by simp_all

definition initmissing :: "(nat \Rightarrow state option) \times 'a \Rightarrow (nat \Rightarrow state) \times 'a"
where
  "initmissing σ = (\lambda i. case (fst σ) i of None ⇒ aodv_init i | Some s ⇒ s, snd σ)"

lemma not_in_net_ips_fst_init_missing [simp]:
assumes "i \notin \text{net_ips} σ"

shows "\text{fst (initmissing (netgmap \text{fst} σ))} i = \text{aodv_init} i"

using assms unfolding initmissing_def by simp

lemma \text{fst_initmissing_netgmap_pair_fst [simp]}:
"\text{fst (initmissing (netgmap (λ(p, q). (\text{fst (id p)}, \text{snd (id p)}, q)) s))}
= \text{fst (initmissing (netgmap \text{fst} s))}"

unfolding initmissing_def by auto

We introduce a streamlined alternative to \text{initmissing} with \text{netgmap} to simplify invariant statements and thus facilitate their comprehension and presentation.

lemma \text{fst_initmissing_netgmap_default_aodv_init_netlift}:
"\text{fst (initmissing (netgmap \text{fst} s))} = \text{default aodv_init (netlift \text{fst} s)}"

unfolding initmissing_def default_def
by (simp add: \text{fst_netgmap_netlift del: One_nat_def})

definition \text{netglobal} :: "((nat ⇒ \text{state}) ⇒ \text{bool}) ⇒ ((\text{state} × 'b) × 'c) \text{net_state} ⇒ \text{bool}"
where
"\text{netglobal} \text{P} ≡ (λs. \text{P (default aodv_init (netlift \text{fst} s))})"

end

5.5 Quality relations between routes

theory \text{E_Fresher}
imports \text{E_Aodv_Data}
begin

5.5.1 Net sequence numbers

On individual routes

definition \text{nsqn_r} :: "r ⇒ sqn"
where
"\text{nsqn_r} r ≡ \text{if } π_4(r) = \text{val} \lor π_2(r) = 0 \text{ then } π_2(r) \text{ else } (π_2(r) - 1)"

lemma \text{nsqnr_def'}:
"\text{nsqn_r} r = (\text{if } π_4(r) = \text{inv then } π_2(r) - 1 \text{ else } π_2(r))"

unfolding \text{nsqn_r_def} by simp

lemma \text{nsqn_r_zero [simp]}:
"\text{∀dsn dsk \text{flag hops nhip. nsqn_r (0, dsk, \text{flag, hops, nhip}) = 0}"

unfolding \text{nsqn_r_def} by clarsimp

lemma \text{nsqn_r_val [simp]}:
"\text{∀dsn dsk \text{hops nhip. nsqn_r (dsn, dsk, val, hops, nhip) = dsn}"

unfolding \text{nsqn_r_def} by clarsimp

lemma \text{nsqn_r_inv [simp]}:
"\text{∀dsn dsk \text{hops nhip. nsqn_r (dsn, dsk, inv, hops, nhip) = dsn - 1}"

unfolding \text{nsqn_r_def} by clarsimp

lemma \text{nsqn_r_lte_dsn [simp]}:
"\text{∀dsn dsk \text{flag hops nhip. nsqn_r (dsn, dsk, flag, hops, nhip) ≤ dsn}"

unfolding \text{nsqn_r_def} by clarsimp

On routes in routing tables

definition \text{nsqn} :: "rt ⇒ ip ⇒ sqn"
where
"\text{nsqn} ≡ \text{λrt dip. case } σ_{\text{route}}(rt, dip) \text{ of } \text{None ⇒ 0 | Some } r ⇒ \text{nsqn_r(r)}"
lemma nsqn_sqn_def:
"∀ rt dip. nsqn rt dip = (if flag rt dip = Some val ∨ sqn rt dip = 0
  then sqn rt dip else sqn rt dip - 1)"
unfolding nsqn_def sqn_def by (clarsimp split: option.split) auto

lemma not_in_kD_nsqn [simp]:
assumes "dip /∈ kD(rt)"
shows "nsqn rt dip = 0"
using assms unfolding nsqn_def by simp

lemma kD_nsqn:
assumes "dip /∈ kD(rt)"
shows "nsqn rt dip = nsqn r (the (σroute(rt, dip)))"
using assms [THEN kD_Some] unfolding nsqn_def by clarsimp

lemma nsqr_r_flag_pred [simp, intro]:
fixes dsn dsk flag hops nhip pre
assumes "P (nsqn r (dsn, dsk, val, hops, nhip))"
  and "P (nsqn r (dsn, dsk, inv, hops, nhip))"
shows "P (nsqn r (dsn, dsk, flag, hops, nhip))"
using assms by (cases flag) auto

lemma sqn_nsqn:
"∀ rt dip. sqn rt dip - 1 ≤ nsqn rt dip"
unfolding sqn_def nsqn_def by (clarsimp split: option.split)

lemma nsqn_sqn: "nsqn rt dip ≤ sqn rt dip"
unfolding sqn_def nsqn_def by (cases "rt dip") auto

lemma val_nsqn_sqn [elim, simp]:
assumes "ip /∈ kD(rt)"
  and "the (flag rt ip) = val"
shows "nsqn rt ip = sqn rt ip"
using assms unfolding nsqn_sqn_def by auto

lemma vD_nsqn_sqn [elim, simp]:
assumes "ip /∈ vD(rt)"
shows "nsqn rt ip = sqn rt ip - 1"
proof -
  from ⟨ip /∈ vD(rt)⟩ have "ip /∈ kD(rt)"
    and "the (flag rt ip) = val" by auto
  thus ?thesis ..
qed

lemma inv_nsqn_sqn [elim, simp]:
assumes "ip /∈ kD(rt)"
  and "the (flag rt ip) = inv"
shows "nsqn rt ip = sqn rt ip - 1"
using assms unfolding nsqn_sqn_def by auto

lemma iD_nsqn_sqn [elim, simp]:
assumes "ip /∈ iD(rt)"
shows "nsqn rt ip = sqn rt ip - 1"
proof -
  from ⟨ip /∈ iD(rt)⟩ have "ip /∈ kD(rt)"
    and "the (flag rt ip) = inv" by auto
  thus ?thesis ..
qed

lemma nsqn_update_changed_kno_val [simp]: "∀ rt ip dsn dsk hops nhip. rt ≠ update rt ip (dsn, kno, val, hops, nhip)
  ⇒ nsqn (update rt ip (dsn, kno, val, hops, nhip)) ip = dsn"
using assms unfolding nsqn_r_def update_def

440
by (clarsimp simp: kD_nsqn split: option.split_asm option.split split_if_asm)
  (metis fun_upd_triv)

lemma nsqn_update_other [simp]:
  fixes dsn dsk flag hops dip nhip pre rt ip
  assumes "dip ≠ ip"
  shows "nsqn (update rt ip (dsn, dsk, flag, hops, nhip)) dip = nsqn rt dip"
  using assms unfolding nsqn_def
  by (clarsimp split: option.split)

lemma nsqn_invalidate_eq:
  assumes "dip ∈ kD(rt)"
  and "dests dip = Some rsn"
  shows "nsqn (invalidate rt dests) dip = rsn - 1"
  using assms proof
    from assms obtain dsk hops nhip
      where "invalidate rt dests dip = Some (rsn, dsk, inv, hops, nhip)"
      unfolding invalidate_def
      by auto
    moreover from ⟨dip ∈ kD(rt)⟩ have "dip ∈ kD(invalidate rt dests)" by simp
    ultimately show ?thesis
    using ⟨dests dip = Some rsn⟩ by simp
  qed

lemma nsqn_invalidate_other [simp]:
  assumes "dip ∈ kD(rt)"
  and "dip /∈ dom dests"
  shows "nsqn (invalidate rt dests) dip = nsqn rt dip"
  using assms by (clarsimp simp add: kD_nsqn)

5.5.2 Comparing routes

definition fresher :: "r ⇒ r ⇒ bool" ("(_/⊑_/)"
where
  "fresher r r' ≡ ((nsqn, r < nsqn, r') ∨ (nsqn, r = nsqn, r' ∧ π5(r) ≥ π5(r')))"

lemma fresherI1 [intro]:
  assumes "nsqn, r < nsqn, r'"
  shows "r ⊑ r'"
  unfolding fresher_def using assms by simp

lemma fresherI2 [intro]:
  assumes "nsqn, r = nsqn, r'" and "π5(r) ≥ π5(r')"
  shows "r ⊑ r'"
  unfolding fresher_def using assms by simp

lemma fresherI [intro]:
  assumes "(nsqn, r < nsqn, r') ∨ (nsqn, r = nsqn, r' ∧ π5(r) ≥ π5(r'))"
  shows "r ⊑ r'"
  unfolding fresher_def using assms .

lemma fresherE [elim]:
  assumes "r ⊑ r'"
  and "nsqn, r < nsqn, r' ⟹ P r r'"
  and "nsqn, r = nsqn, r' ∧ π5(r) ≥ π5(r') ⟹ P r r'"
  shows "P r r'"
  using assms unfolding fresher_def by auto

lemma fresher_refl [simp]: "r ⊑ r"
  unfolding fresher_def by simp
lemma freshness_trans [elim, trans]:
  "[ x ⊑ y; y ⊑ z ] ⇒ x ⊑ z"
unfolding freshness_def by auto

lemma not_freshness_trans [elim, trans]:
  "[ ¬(x ⊑ y); ¬(z ⊑ x) ] ⇒ ¬(z ⊑ y)"
unfolding freshness_def by auto

lemma freshness_dsn_flag_hops_const [simp]:
  fixes dsn dsk dsk' flag hops nhip nhip'
  shows "(dsn, dsk, flag, hops, nhip) ⊑ (dsn, dsk', flag, hops, nhip')"
unfolding freshness_def by (cases flag) simp_all

5.5.3 Comparing routing tables

definition rt_fresher :: "ip ⇒ rt ⇒ rt ⇒ bool"
where
  "rt_fresher ≡ λdip rt rt'. (the (σ_route(rt, dip))) ⊑ (the (σ_route(rt', dip)))"

abbreviation rt_fresher_syn :: "rt ⇒ ip ⇒ rt ⇒ bool" ("(_/ ⊑ _)") [51, 999, 51] 50
where
  "rt1 ⊑ i rt2 ≡ rt_fresher i rt1 rt2"

lemma rt_fresher_def':
  "(rt1 ⊑ i rt2) = (nsqn rt1 dip < nsqn rt2 dip ∨
  nsqn rt1 dip = nsqn rt2 dip ∧ π5 rt1 dip ≤ π5 rt2 dip)"
unfolding rt_fresher_def freshness_def by (rule refl)

lemma single_rt_fresher [intro]:
  assumes "the (rt1 ip) ⊑ the (rt2 ip)"
  shows "rt1 ⊑ ip rt2"
using assms unfolding rt_fresher_def .

lemma rt_fresher_single [intro]:
  assumes "rt1 ⊑ ip rt2"
  shows "the (rt1 ip) ⊑ the (rt2 ip)"
using assms unfolding rt_fresher_def .

lemma rt_fresher_def2:
  assumes "dip ∈ kD(rt1)"
  and "dip ∈ kD(rt2)"
  shows "(rt1 ⊑ dip rt2) = (nsqn rt1 dip < nsqn rt2 dip
  ∨ (nsqn rt1 dip = nsqn rt2 dip ∧ the (dhops rt1 dip) ≥ the (dhops rt2 dip)))"
using assms unfolding rt_fresher_def freshness_def by (simp add: kD_nsqn proj5_eq_dhops)

lemma rt_fresherI1 [intro]:
  assumes "dip ∈ kD(rt1)"
  and "dip ∈ kD(rt2)"
  and "nsqn rt1 dip < nsqn rt2 dip"
  shows "rt1 ⊑ dip rt2"
unfolding rt_fresher_def2 [OF assms(1-2)] using assms(3) by simp

lemma rt_fresherI2 [intro]:
  assumes "dip ∈ kD(rt1)"
  and "dip ∈ kD(rt2)"
  and "nsqn rt1 dip = nsqn rt2 dip"
  and "the (dhops rt1 dip) ≥ the (dhops rt2 dip)"
  shows "rt1 ⊑ dip rt2"
unfolding rt_fresher_def2 [OF assms(1-2)] using assms(3-4) by simp

lemma rt_fresherE [elim]:
assumes "rt1 ⊑ dip rt2"
and "dip ∈ kD(rt1)"
and "dip ∈ kD(rt2)"
and "[ nsqn rt1 dip < nsqn rt2 dip ] ⇒ P rt1 rt2 dip"
and "[ nsqn rt1 dip = nsqn rt2 dip;
the (dhops rt1 dip) ≥ the (dhops rt2 dip) ] ⇒ P rt1 rt2 dip"

shows "P rt1 rt2 dip"
using assms(1) unfolding rt_fresher_def2 [OF assms(2-3)]
using assms(4-5) by auto

lemma rt_fresher_refl [simp]: "rt ⊑ dip rt"
unfolding rt_fresher_def by simp

lemma rt_fresher_trans [elim, trans]:
assumes "rt1 ⊑ dip rt2"
and "rt2 ⊑ dip rt3"
shows "rt1 ⊑ dip rt3"
using assms unfolding rt_fresher_def by auto

lemma rt_fresher_if_Some [intro!]:
assumes "the (rt dip) ⊑ r"
shows "rt ⊑ dip (λip. if ip = dip then Some r else rt ip)"
using assms unfolding rt_fresher_def by simp

definition rt_fresh_as :: "ip ⇒ rt ⇒ rt ⇒ bool"
where "rt_fresh_as ≡ λdip rt1 rt2. (rt1 ⊑ dip rt2) ∧ (rt2 ⊑ dip rt1)"

abbreviation rt_fresh_as_syn :: "rt ⇒ ip ⇒ rt ⇒ bool" ("(_/≈_)
where "rt1 ≈ rt2 ≡ rt_fresh_as i rt1 rt2"

lemma rt_fresh_as_refl [simp]: "⋀rt dip. rt ≈ dip rt"
unfolding rt_fresh_as_def by simp

lemma rt_fresh_as_trans [simp, intro, trans]:
"⋀rt1 rt2 rt3 dip. [ rt1 ≈ dip rt2; rt2 ≈ dip rt3 ] ⇒ rt1 ≈ dip rt3"
unfolding rt_fresh_as_def rt_fresher_def
by (metis (mono_tags) fresher_trans)

lemma rt_fresh_asI [intro!]:
assumes "rt1 ⊑ dip rt2"
and "rt2 ⊑ dip rt1"
shows "rt1 ≈ dip rt2"
using assms unfolding rt_fresh_as_def by simp

lemma rt_fresh_as_fresherI [intro]:
assumes "dip ∈ kD(rt1)"
and "dip ∈ kD(rt2)"
and "the (rt1 dip) ⊑ the (rt2 dip)"
and "the (rt2 dip) ⊑ the (rt1 dip)"
shows "rt1 ≈ dip rt2"
using assms unfolding rt_fresh_as_def
by (clarsimp dest!: single_rt_fresher)

lemma nsqn_rt_fresh_asI:
assumes "dip ∈ kD(rt)"
and "dip ∈ kD(rt')"
and "nsqn rt dip = nsqn rt' dip"
and "π₅(the (rt dip)) = π₅(the (rt' dip))"
shows "rt ≈ dip rt'"
proof
from assms(1-2,4) have dhops': "the (dhops rt' dip) ≤ the (dhops rt dip)"

443
by (simp add: proj5_eq_dhops)
with assms(1-3) show "rt \sqsubseteq dip rt'"
by (rule rt_fresherI2)

next
from assms(1-2,4) have dhops: "the (dhops rt dip) \leq the (dhops rt' dip)"
by (simp add: proj5_eq_dhops)
with assms(2,1) assms(3) [symmetric] show "rt' \sqsubseteq dip rt"
by (rule rt_fresherI2)

qed

lemma rt_fresh_asE [elim]:
assumes "rt1 \approx dip rt2"
and "[ rt1 \sqsubseteq dip rt2; rt2 \sqsubseteq dip rt1 ] \implies P rt1 rt2 dip"
shows "P rt1 rt2 dip"
using assms unfolding rt_fresh_as_def by simp

lemma rt_fresh_asD1 [dest]:
assumes "rt1 \approx dip rt2"
shows "rt1 \sqsubseteq dip rt2"
using assms unfolding rt_fresh_as_def by simp

lemma rt_fresh_asD2 [dest]:
assumes "rt1 \approx dip rt2"
shows "rt2 \sqsubseteq dip rt1"
using assms unfolding rt_fresh_as_def by simp

lemma rt_fresh_as_sym:
assumes "rt1 \approx dip rt2"
shows "rt2 \approx dip rt1"
using assms unfolding rt_fresh_as_def by simp

lemma not_rt_fresh_asI1 [intro]:
assumes "\neg (rt1 \sqsubseteq dip rt2)"
shows "\neg (rt1 \approx dip rt2)"
proof
assume "rt1 \approx dip rt2"
hence "rt1 \sqsubseteq dip rt2" ..
with \neg (rt1 \sqsubseteq dip rt2) show False ..
qed

lemma not_rt_fresh_asI2 [intro]:
assumes "\neg (rt2 \sqsubseteq dip rt1)"
shows "\neg (rt1 \approx dip rt2)"
proof
assume "rt1 \approx dip rt2"
hence "rt2 \sqsubseteq dip rt1" ..
with \neg (rt2 \sqsubseteq dip rt1) show False ..
qed

lemma not_single_rt_fresher [elim]:
assumes "\neg ((the (rt1 ip) \sqsubseteq the (rt2 ip)))"
shows "\neg (rt1 \sqsubseteq ip rt2)"
proof
assume "rt1 \sqsubseteq ip rt2"
hence "the (rt1 ip) \sqsubseteq the (rt2 ip)" ..
with \neg (the (rt1 ip) \sqsubseteq the (rt2 ip)) show False ..
qed

lemmas not_single_rt_fresh_asI1 [intro] = not_rt_fresh_asI1 [OF not_single_rt_fresher]
lemmas not_single_rt_fresh_asI2 [intro] = not_rt_fresh_asI2 [OF not_single_rt_fresher]

lemma not_rt_fresher_single [elim]:
assumes "\neg (the (rt1 ip) \sqsubseteq the (rt2 ip))"
shows "\neg (rt1 \sqsubseteq dip rt2)"
proof
assume "the (rt1 ip) ⊑ the (rt2 ip)"
hence "rt1 ⊑ ip rt2" ..
with ¬(rt1 ⊑ ip rt2) show False ..
qed

lemma rt_fresh_as_nsqn:
assumes "dip ∈ kD(rt1)"
and "dip ∈ kD(rt2)"
and "rt1 ≈ dip rt2"
shows "nsqn_r (the (rt2 dip)) = nsqn_r (the (rt1 dip))"
using assms(3) unfolding rt_fresh_as_def
by (auto simp: rt_fresher_def2 [OF ⟨dip ∈ kD(rt1)⟩ ⟨dip ∈ kD(rt2)⟩] kD_nsqn [OF ⟨dip ∈ kD(rt1)⟩] kD_nsqn [OF ⟨dip ∈ kD(rt2)⟩])

lemma rt_fresher_mapupd [intro!]:
assumes "dip ∈ kD(rt)"
and "the (rt dip) ⊑ r"
shows "rt ⊑ dip rt(dip ↦ r)"
using assms unfolding rt_fresher_def by simp

lemma rt_fresher_map_update_other [intro!]:
assumes "dip ∈ kD(rt)"
and "dip ≠ ip"
shows "rt ⊑ dip rt(ip ↦ r)"
using assms unfolding rt_fresher_def by simp

lemma rt_fresher_update_other [simp]:
assumes inkD: "dip ∈ kD(rt)"
and "dip ≠ ip"
shows "rt ⊑ dip update rt ip r"
using assms unfolding update_def by (clarsimp split: option.split) (fastforce)

theorem rt_fresher_update [simp]:
assumes "dip ∈ kD(rt)"
and "the (dhops rt dip) ≥ 1"
and "update_arg_wf r"
shows "rt ⊑ dip update rt dip r"
proof (cases "dip = ip")
  assume "dip ≠ ip" with ⟨dip ∈ kD(rt)⟩ show ?thesis
    by (rule rt_fresher_update_other)
next
  assume "dip = ip"
from ⟨dip ∈ kD(rt)⟩ obtain dsn n dsk n f n hops n nhip n
where rt [simp]: "the (rt dip) = (dsn n, dsk n, f n, hops n, nhip n)"
by (metis prod_cases5)
with ⟨the (dhops rt dip) ≥ 1⟩ and ⟨dip ∈ kD(rt)⟩ have "hops n ≥ 1"
by (metis proj5_eq_dhops projs(4))
from ⟨dip ∈ kD(rt)⟩ rt have [simp]: "sqn rt dip = dsn n"
  and [simp]: "the (dhops rt dip) = hops n"
  and [simp]: "the (flag rt dip) = f n"
by (simp add: sqn_def proj5_eq_dhops [symmetric]
  proj4_eq_flag [symmetric])

from ⟨update_arg_wf r⟩ have "(dsn n, dsk n, f n, hops n, nhip n)
  ⊑ the ((update rt dip r) dip)"
proof (rule wf_r_cases)
  fix nhip
  from ⟨hops n ≥ 1⟩ have "(dsn n, dsk n, f n, hops n, nhip n)
  ⊑ (dsn n, unk, val, Suc 0, nhip)"
 
unfolding fresher_def sqn_def by (cases \( f_n \)) auto
thus "(dsn_n, dsk_n, f_n, hops_n, nhip_n) ⊑ the (update rt dip (0, unk, val, Suc 0, nhip) dip)"
using \( \langle \text{dip} \in kD(rt) \rangle \) by - (rule update_cases_kD, simp_all)

next
fix dsn :: sqn and hops nhip
assume "0 < dsn"
show "(dsn_n, dsk_n, f_n, hops_n, nhip_n) ⊑ (dsn, dsk_n, f_n, hops_n, nhip)"
unfolding fresher_def by auto

next
assume "dsn_n = dsn"
and "hops < hops_n"
thus "(dsn, dsk_n, f_n, hops_n, nhip_n) ⊑ (dsn, dsk_n, f_n, hops_n, nhip)"
unfolding fresher_def nsqn_r_def by simp

next
assume "dsn_n = dsn"
with \( \langle 0 < dsn \rangle \) show "(dsn_n, dsk_n, inv, hops_n, nhip_n) ⊑ (dsn, dsk_n, inv, hops_n, nhip)"
unfolding fresher_def by simp

qed

hence "rt ⊑ dip update rt dip r"
by - (rule single_rt_fresher, simp)

qed

theorem rt_fresher_invalidate [simp]:
assumes "dip \in kD(rt)"
and indests: "\( \forall \text{rip} \in \text{dom(dests)}. \text{rip} \in vD(rt) \wedge \text{sqn rt rip} < \text{the (dests rip)} \)"
shows "rt \( \in dip \text{ invalidate rt dests} )"
proof (cases "dip \in \text{dom(dests)}")
assume "dip \in \text{dom(dests)}"
thus \(?thesis \) using \( \langle dip \in kD(rt) \rangle \)
by - (rule single_rt_fresher, simp)

next
assume "dip \in \text{dom(dests)}"
moreover with indests have "dip \in vD(rt)"
and "sqn rt dip < the (dests dip)"
by auto
ultimately show \(?thesis \)
unfolding invalidate_def sqn_def
by - (rule single_rt_fresher, auto simp: fresher_def)

qed

lemma nsqn_r_invalidate [simp]:
assumes "dip \in kD(rt)"
and "dip \in \text{dom(dests)}"
shows "nsqn (the (invalidate rt dests dip)) = the (dests dip) - 1"
using assms unfolding invalidate_def by auto

lemma rt_fresh_as_inc_invalidate [simp]:
assumes "dip \in kD(rt)"
and "\( \forall \text{rip} \in \text{dom(dests)}. \text{rip} \in vD(rt) \wedge \text{the (dests rip)} = \text{inc (sqn rt rip)} \)"
shows "rt \( \approx dip \text{ invalidate rt dests} )"
proof (cases "dip \in \text{dom(dests)}")
assume "dip \in \text{dom(dests)}"
with \( \langle dip \in kD(rt) \rangle \) have "dip \in kD(invalidate rt dests)"


by simp

with \( \langle \text{dip} \in \text{kD}(\text{rt}) \rangle \) show \(?\text{thesis}\)
  by rule \( (\text{simp\_all\ add: dip\in\text{dom(dests)})} \)

next

assume \( \text{dip} \in \text{dom(dests)} \)

with \( \text{assms}(2) \) have \( \text{dip} \in \text{vD}(\text{rt}) \)
  and \( \text{the (dests dip)} = \text{inc (sqn rt dip)} \)
  by auto

from \( \text{dip} \in \text{vD}(\text{rt}) \) have \( \text{dip} \in \text{kD}(\text{rt}) \)
by simp

moreover then have \( \text{dip} \in \text{kD} (\text{invalidate rt dests}) \)
by simp

ultimately show \(?\text{thesis}\)
  proof (rule nsqn\_rt\_fresh\_asI)
from \( \langle \text{dip} \in \text{vD}(\text{rt}) \rangle \) have \( \text{nsqn rt dip} = \text{sqn rt dip} \)
by simp

also have \( \text{sqn rt dip} = \text{nsqn r (the (invalidate rt dests dip))} \)
proof
- from \( \langle \text{dip} \in \text{kD(\text{rt})} \rangle \) have \( \text{nsqn} r (\text{the (invalidate rt dests dip)}) = \text{the (\text{dests dip})} - 1 \)
  using \( \langle \text{dip} \in \text{dom(dests)} \rangle \) by (rule nsqn\_invalidate)

  with \( \langle \text{the (dests dip)} = \text{inc (sqn rt dip)} \rangle \)
  show \( \text{sqn rt dip} = \text{nsqn} r (\text{the (invalidate rt dests dip)}) \)
  by simp

  qed

also from \( \langle \text{dip} \in \text{kD(\text{invalidate rt dests})} \rangle \)
  have \( \text{nsqn} r (\text{the (invalidate rt dests dip)}) = \text{nsqn (invalidate rt dests) dip} \)
  by (simp add: kD\_nsqn)

finally show \( \text{nsqn rt dip} = \text{nsqn (invalidate rt dests) dip} \)
  qed

qed

lemmas \( \text{rt\_fresher\_inc\_invalidate [simp]} = \text{rt\_fresh\_as\_inc\_invalidate [THEN rt\_fresh\_asD1]} \)

5.5.4 Strictly comparing routing tables

definition \( \text{rt\_strictly\_fresher :} \text{ "ip \Rightarrow rt \Rightarrow rt \Rightarrow bool"} \)
  where
  \( \text{rt\_strictly\_fresher} \equiv \lambda \text{dip rt1 rt2. (rt1} \sqsubseteq_{\text{dip}} \text{rt2). (rt1} \sqsubseteq_{\text{dip}} \text{rt2)} \wedge \neg (\text{rt1} \approx_{\text{dip}} \text{rt2}) \)

abbreviation
  \( \text{rt\_strictly\_fresher\_syn :} \text{ "rt \Rightarrow ip \Rightarrow rt \Rightarrow bool" ("(_/ \sqsubseteq_{\_} \_)" [51, 999, 51] 50)} \)
  where
  \( \text{rt1} \sqsubseteq_{\_} \text{rt2} \equiv \text{rt\_strictly\_fresher i rt1 rt2} \)

lemma \( \text{rt\_strictly\_fresher\_def'} : \)
  \( \text{rt1} \sqsubseteq_{\_} \text{rt2} = ((\text{rt1} \sqsubseteq_{\_} \text{rt2}) \wedge \neg (\text{rt2} \sqsubseteq_{\_} \text{rt1})) \)
  unfolding rt\_strictly\_fresher\_def rt\_fresh\_as\_def by auto

lemma \( \text{rt\_strictly\_fresherI'} \ [intro] : \)
  assumes \( \text{rt1} \sqsubseteq_{\_} \text{rt2} \)
  and \( \neg (\text{rt2} \sqsubseteq_{\_} \text{rt1}) \)
  shows \( \text{rt1} \sqsubseteq_{\_} \text{rt2} \)
  using \( \text{assms unfolding rt\_strictly\_fresher\_def'} \) by simp

lemma \( \text{rt\_strictly\_fresherE'} \ [elim] : \)
  assumes \( \text{rt1} \sqsubseteq_{\_} \text{rt2} \)
  and \( \neg (\text{rt1} \approx_{\_} \text{rt2}) \)
  shows \( \text{rt1} \sqsubseteq_{\_} \text{rt2} \)
  using \( \text{assms unfolding rt\_strictly\_fresher\_def'} \) by simp

lemma \( \text{rt\_strictly\_fresherI} \ [intro] : \)
  assumes \( \text{rt1} \sqsubseteq_{\_} \text{rt2} \)
  and \( \neg (\text{rt1} \approx_{\_} \text{rt2}) \)
  shows \( \text{rt1} \sqsubseteq_{\_} \text{rt2} \)
  unfolding rt\_strictly\_fresher\_def using \( \text{assms} .. \)

lemmas \( \text{rt\_strictly\_fresher\_singleI} \ [elim] = \text{rt\_strictly\_fresherI} \ [OF single\_rt\_fresher] \)

lemma \( \text{rt\_strictly\_fresherE} \ [elim] : \)

447
assumes "rt1 ⊑ i rt2"
and "[ rt1 ⊑ i rt2; ¬(rt1 ≈ i rt2) ] ⊢ P rt1 rt2 i"
shows "P rt1 rt2 i"
using assms(1) unfolding rt_strictly_fresher_def
by rule (erule(1) assms(2))

lemma rt_strictly_fresher_def':
"rt1 ⊑ i rt2 =
(naq_r (the (rt1 i)) < naq_r (the (rt2 i))
 ∨ (naq_r (the (rt1 i)) = naq_r (the (rt2 i)) ∧ π₅(the (rt1 i)) > π₅(the (rt2 i))))"
unfolding rt_strictly_fresher_def'' rt_fresher_def fresher_def by auto

lemma rt_strictly_fresher_fresherD [dest]:
assumes "rt1 ⊏ dip rt2"
shows "the (rt1 dip) ⊑ the (rt2 dip)"
using assms unfolding rt_strictly_fresher_def rt_fresher_def by auto

lemma rt_strictly_fresher_not_fresh_asD [dest]:
assumes "rt1 ⊏ dip rt2"
shows "¬ rt1 ≈ dip rt2"
using assms unfolding rt_strictly_fresher_def by auto

lemma rt_strictly_fresher_trans [elim, trans]:
assumes "rt1 ⊏ dip rt2"
and "rt2 ⊑ dip rt3"
shows "rt1 ⊏ dip rt3"
using assms proof -
from rt1 ⊏ dip rt2: obtain "the (rt1 dip) ⊑ the (rt2 dip)" by auto
also from rt2 ⊑ dip rt3: obtain "the (rt2 dip) ⊑ the (rt3 dip)" by auto
finally have "the (rt1 dip) ⊑ the (rt3 dip)".
moreover have "¬ (rt1 ≈ dip rt3)"
proof -
from rt1 ⊏ dip rt2: obtain "¬(the (rt2 dip) ⊑ the (rt1 dip))" by auto
also from rt2 ⊏ dip rt3: obtain "¬(the (rt3 dip) ⊑ the (rt2 dip))" by auto
finally have "¬(the (rt3 dip) ⊑ the (rt1 dip))".
thus ?thesis ..
qed ultimately show "rt1 ⊏ dip rt3" ..
qed

lemma rt_strictly_fresher_irefl [simp]: "¬ (rt ⊏ dip rt)"
using assms unfolding rt_strictly_fresher_def by clarsimp

lemma rt_fresher_trans_rt_strictly_fresher [elim, trans]:
assumes "rt1 ⊏ dip rt2"
and "rt2 ⊏ dip rt3"
shows "rt1 ⊏ dip rt3"
proof -
from rt1 ⊏ dip rt2: have "rt1 ⊏ dip rt2"
and "¬(rt2 ⊏ dip rt1)"
unfolding rt_strictly_fresher_def'' by auto
from this(1) and rt2 ⊏ dip rt3: have "rt1 ⊏ dip rt3" ..
moreover from ¬(rt2 ⊏ dip rt1) have "¬(rt3 ⊏ dip rt1)"
proof (rule contrapos_nm)
assume "rt3 ⊏ dip rt1"
with rt2 ⊏ dip rt3 show "rt2 ⊏ dip rt1" ..
qed ultimately show "rt1 ⊏ dip rt3"
unfolding rt_strictly_fresher_def'' by auto
qed

448
lemma rt_fresher_trans_rt_strictly_fresher' [elim, trans]:
  assumes "rt1 ⊑ dip rt2"
  and "rt2 ⊑ dip rt3"
  shows "rt1 ⊑ dip rt3"
proof -
  from rt2 ⊑ dip rt3 have "rt2 ⊑ dip rt3"
    and "¬ (rt3 ⊑ dip rt2)"
  unfolding rt_strictly_fresher_def'' by auto
  moreover from (¬(rt3 ⊑ dip rt2)) have "¬ (rt3 ⊑ dip rt1)"
  proof (rule contrapos_nn)
    assume "rt3 ≈ dip rt1"
    hence "rt2 ⊑ dip rt1" ..
  qed
  ultimately show "rt1 ⊑ dip rt3"
  unfolding rt_strictly_fresher_def'' by auto
  qed

lemma rt_fresher_imp_nsqn_le:
  assumes "rt1 ⊑ ip rt2"
  and "ip ∈ kD rt1"
  and "ip ∈ kD rt2"
  shows "nsqn rt1 ip ≤ nsqn rt2 ip"
  using assms(1)
  by (auto simp add: rt_fresher_def2 [OF assms(2-3)])

lemma rt_strictly_fresher_ltI [intro]:
  assumes "dip ∈ kD(rt1)"
  and "dip ∈ kD(rt2)"
  and "nsqn rt1 dip < nsqn rt2 dip"
  shows "rt1 ⊑ dip rt2"
proof
  from assms show "¬(rt1 ≈ dip rt2)" ..
  next
  show "¬(rt1 ≈ dip rt2)"
  proof
    assume "rt1 ≈ dip rt2"
    hence "rt2 ⊑ dip rt1" ..
    hence "nsqn rt2 dip ≤ nsqn rt1 dip"
      using (dip ∈ kD(rt2)) (dip ∈ kD(rt1))
      by (rule rt_fresher_imp_nsqn_le)
    with (nsqn rt1 dip < nsqn rt2 dip) show "False"
      by simp
  qed
  qed

lemma rt_strictly_fresher_eqI [intro]:
  assumes "i ∈ kD(rt1)"
  and "i ∈ kD(rt2)"
  and "nsqn rt1 i = nsqn rt2 i"
  and "π5(the (rt2 i)) < π5(the (rt1 i))"
  shows "rt1 ⊑ i rt2"
using assms unfolding rt_strictly_fresher_def' by (auto simp add: kD_nsqn)

lemma invalidate_rtsf_left [simp]:
  "\(\text{dests dip rt rt'}. \text{dests dip = None} \implies (\text{invalidate rt dests \(\subseteq dip rt\')} = (rt \subseteq dip rt'))" 
unfolding invalidate_def rt_strictly_fresher_def'
  by (rule iffI) (auto split: option.split_asm)

lemma vD_invalidate_rt_strictly_fresher [simp]:
  assumes "dip ∈ vD(invalidate rt1 dests)"
shows 
\[(\text{invalidate } rt1 \text{ dests } \sqsubseteq_{\text{dip}} rt2) = (rt1 \sqsubseteq_{\text{dip}} rt2)\]

proof (cases "dip \in \text{dom(dests)}")
assume "dip \in \text{dom(dests)}"

hence "dip \notin \text{vD(invalidate } rt1 \text{ dests)}"

unfolding invalidate_def vD_def
by clarsimp (metis assms option.simps(3) vD_invalidate_vD_not_dests)

with \[(dip \in \text{vD(invalidate } rt1 \text{ dests)}): \text{show } ?\text{thesis by simp}\]

next
assume "dip \notin \text{dom(dests)}"

hence "dests dip = None" by auto

moreover with \[(dip \in \text{vD(invalidate } rt1 \text{ dests)}): \text{have } ?\text{thesis by simp}\]

ultimately show ?thesis unfolding invalidate_def rt_strictly_fresher_def' by auto

qed

lemma rt_strictly_fresher_update_other [elim!]:
\[\forall \text{ dip } \text{ ip } \text{ rt } \text{ r } \text{ rt'}. \ [ \text{ dip } \neq \text{ ip } \land \text{ rt } \sqsubseteq_{\text{dip}} \text{ rt'} ] \Rightarrow \text{ update rt ip r } \sqsubseteq_{\text{dip}} \text{ rt'}\]

unfolding rt_strictly_fresher_def' by clarsimp

lemma lt_sqn_imp_update_strictly_fresher:
assumes \[\text{dip } \in \text{vD (rt2 nhip)}\]
and \[\text{osn } < \text{sqn (rt2 nhip) dip}\]
and \[\text{rt } \neq \text{update rt dip (osn, kno, val, hops, nhip)}\]

shows "update rt dip (osn, kno, val, hops, nhip) \sqsubseteq_{\text{dip}} \text{rt2 nhip}"

unfolding rt_strictly_fresher_def'
proof (rule disjI1)
from ** have "\text{nsqn (update rt dip (osn, kno, val, hops, nhip) dip = osn)"
by (rule nsqn_update_changed_kno_val)
with \[(\text{dip }\in \text{vD (rt2 nhip)}): \text{have } \not\exists \text{ dip } \in \text{vD (rt2 nhip)}\]

hence "\text{nsqn, (the (update rt dip (osn, kno, val, hops, nhip) dip) = osn)}"
by (simp add: kD_nsqn)
also have "\text{osn } < \text{sqn (rt2 nhip) dip}" by (rule *)
also have "\text{sqn (rt2 nhip) dip = nsqn, (the (rt2 nhip dip))}"

unfolding nsqn_def using (dip \in \text{vD (rt2 nhip)});
by - (metis vD_flag_val proj2_eq_sqn proj4_eq_flag vD_iD_gives_kD(1))
finally show "\text{nsqn, (the (update rt dip (osn, kno, val, hops, nhip) dip) < nsqn, (the (rt2 nhip dip))}"

qed

lemma dhops_le_hops_imp_update_strictly_fresher:
assumes \[\text{dip } \in \text{vD (rt2 nhip)}\]
and \[\text{sqn (rt2 nhip) dip = osn}\]
and \[\text{the (dhops (rt2 nhip) dip)} \leq \text{hops}\]
and **: \[\text{rt } \neq \text{update rt dip (osn, kno, val, Suc hops, nhip)}\]

shows "update rt dip (osn, kno, val, Suc hops, nhip) \sqsubseteq_{\text{dip}} \text{rt2 nhip}"

unfolding rt_strictly_fresher_def'
proof (rule disjI2, rule conjI)
from ** have "\text{nsqn (update rt dip (osn, kno, val, Suc hops, nhip) dip = osn)"
by (rule nsqn_update_changed_kno_val)
with \[(\text{dip }\in \text{vD (rt2 nhip)}): \text{have } \not\exists \text{ dip } \in \text{vD (rt2 nhip)}\]

hence "\text{nsqn, (the (update rt dip (osn, kno, val, Suc hops, nhip) dip)) = osn)}"
by (simp add: kD_nsqn)
also have "\text{osn = sqn (rt2 nhip) dip) = osn)" by (rule sqn [symmetric])
also have "\text{sqn (rt2 nhip) dip = nsqn, (the (rt2 nhip dip))}"

unfolding nsqn_def using (dip \in \text{vD (rt2 nhip)});
by - (metis vD_flag_val proj2_eq_sqn proj4_eq_flag vD_iD_gives_kD(1))
finally show "\text{nsqn, (the (update rt dip (osn, kno, val, Suc hops, nhip) dip) = nsqn, (the (rt2 nhip dip))}"

next
have "\text{the (dhops (rt2 nhip) dip) \leq hops)}" by (rule hop)
also have "\text{hops < hops + 1) by simp}
also have "\text{hops + 1 = the (dhops (update rt dip (osn, kno, val, Suc hops, nhip) dip)"

450
using ** by simp
finally have "the (dhops (rt2 nhip) dip)
< the (dhops (update rt dip (osn, kno, val, Suc hops, nhip)) dip)"
thus "π₅ (the (rt2 nhip dip)) < π₅ (the (update rt dip (osn, kno, val, Suc hops, nhip) dip))"
using (dip ∈ vD(rt2 nhip)) by (simp add: proj5_eq_dhops)
qed

lemma nsqn_invalidate:
assumes "dip ∈ kD(rt)"
and "∀ ip∈dom(dests). ip ∈ vD(rt) ∧ the (dests ip) = inc (sqn rt ip)"
shows "nsqn (invalidate rt dests) dip = nsqn rt dip"
proof -
from ⟨dip ∈ kD(rt)⟩ have "dip ∈ kD(invalidate rt dests)" by simp
from assms have "rt ≈ dip invalidate rt dests" by (rule rt_fresh_as_inc_invalidate)
with ⟨dip ∈ kD(rt)⟩ ⟨dip ∈ kD(invalidate rt dests)⟩ show ?thesis by (simp add: kD_nsqn del: invalidate_kD_inv)
(erule(2) rt_fresh_as_nsqnr)
qed

end

5.6 Invariant proofs on individual processes

theory E_Seq_Invariants
imports "../../../AWN/Invariants" E_Aodv E_Aodv_Data E_Aodv_Predicates E_Fresher
begin

The proposition numbers are taken from the December 2013 version of the Fehnker et al technical report.

Proposition 7.2
lemma sequence_number_increases:
"paodv i |=A onll Γ_AODV (λ((ξ, _), _, (ξ', _)). sn ξ ≤ sn ξ')"
by inv_cterms

lemma sequence_number_one_or_bigger:
"paodv i |= onl Γ_AODV (λ(ξ, _). 1 ≤ sn ξ)"
by (rule onll_step_to_invariantI [OF sequence_number_increases])
(auto simp: σ_AODV_def)

We can get rid of the onl/onll if desired...

lemma sequence_number_increases':
"paodv i |=A (λ((ξ, _), _, (ξ', _)). sn ξ ≤ sn ξ')"
by (rule step_invariant_weakenE [OF sequence_number_increases])
(auto dest!: onllD)

lemma sequence_number_one_or_bigger':
"paodv i |= (λ(ξ, _). 1 ≤ sn ξ)"
by (rule invariant_weakenE [OF sequence_number_one_or_bigger])

lemma sip_in_kD:
"paodv i |= onl Γ_AODV (λ(ξ, 1). l ∈ {{PAodv-:7} ∪ {PAodv-:5} ∪ {PRrep-:0..PRrep-:4} ∪ {PRreq-:0..PRreq-:3}} → sip ξ ∈ kD (rt ξ))"
by inv_cterms

Proposition 7.38
lemma includes_nhip:
"paodv i |= onl Γ_AODV (λ(ξ, 1). ∀ dip∈kD(rt ξ). the (nhop (rt ξ) dip)∈kD(rt ξ))"
proof -
{ fix ip and ξ ξ' :: state
  assume "∀ dip∈kD (rt ξ). the (nhop (rt ξ) dip) ∈ kD (rt ξ)"
  and "ξ' = ξ[rt := update (rt ξ) ip (0, unk, val, Suc 0, ip)]"
  hence "∀ dip∈kD (rt ξ).".
}
the (nhop (update (rt ξ) ip (0, unk, val, Suc 0, ip)) dip) = ip \\
∨ the (nhop (update (rt ξ) ip (0, unk, val, Suc 0, ip)) dip) ∈ kD (rt ξ)"
by clarsimp (metis nhop_update_unk_val update_another)

} note one_hop = this

fix ip sn hops and ξ ξ' :: state
assume "∀ dip ∈ kD (rt ξ). the (nhop (rt ξ) dip) ∈ kD (rt ξ)"
and "ξ' = ξ (| rt := update (rt ξ) ip (sn, kno, val, Suc hops, sip))"
and "sip ∈ kD (rt ξ)"
hence "(the (nhop (update (rt ξ) ip (sn, kno, val, Suc hops, sip)) ip) = ip \\
∨ the (nhop (update (rt ξ) ip (sn, kno, val, Suc hops, sip)) ip) ∈ kD (rt ξ))"
by (metis kD_update_unchanged nhop_update_changed update_another)

} note nhip_is_sip = this

show ?thesis by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf sip_in_kD]
solve: one_hop nhip_is_sip)

qed

Proposition 7.4

lemma known_destinations_increase:
"paodv i ⪰ onll Γ AODV (λ((ξ, _), _, (ξ', _)). kD (rt ξ) ⊆ kD (rt ξ'))"
by (inv_cterms simp add: subset_insertI)

Proposition 7.5

lemma rreqs_increase:
"paodv i ⪰ onll Γ AODV (λ((ξ, _), _, (ξ', _)). rreqs ξ ⊆ rreqs ξ')"
by (inv_cterms simp add: subset_insertI)

lemma dests_bigger_than_sqn:
"paodv i ⪰ onll Γ AODV (λ(ξ, l). l ∈ {PAodv-:15..PAodv-:17} \\
∪ {PPkt-:7..PPkt-:9} \\
∪ {PRreq-:11..PRreq-:13} \\
∪ {PRreq-:20..PRreq-:22} \\
∪ {PRrep-:7..PRrep-:9} \\
∪ {PRerr-:1..PRerr-:4} ∪ {PRerr-:6} \\
→ (∀ ip∈dom(dests ξ). ip∈kD(rt ξ) ∧ sqn (rt ξ) ip ≤ the (dests ξ ip)))"
proof -
  have sqninv:
    "∀ dests rt rsn ip. 
    [ ∀ ip∈dom(dests). ip∈kD(rt) ∧ sqn rt ip ≤ the (dests ip); dests ip = Some rsn ] 
    → sqn (invalidate rt dests) ip ≤ rsn"
  by (rule sqn_invalidate_in_dests [THEN eq_imp_le], assumption) auto
  have indests:
    "∀ dests rt rsn ip. 
    [ ∀ ip∈dom(dests). ip∈kD(rt) ∧ sqn rt ip ≤ the (dests ip); dests ip = Some rsn ] 
    → ip∈kD(rt) ∧ sqn rt ip ≤ rsn"
  by (metis domI option.sel)
  show ?thesis by
  inv_cterms
    (clarsimp split: split_if_asm option.split_asm 
     elim!: sqninv indests)+

qed

Proposition 7.6

lemma sqns_increase:
"paodv i ⪰ onll Γ AODV (λ((ξ, _), _, (ξ', _)). ∀ ip. sqn (rt ξ) ip ≤ sqn (rt ξ') ip)"
proof -
  fix ξ :: state
  assume *: "∀ ip∈dom(dests ξ). ip ∈ kD (rt ξ) ∧ sqn (rt ξ) ip ≤ the (dests ξ ip)"
  have "∀ ip. sqn (rt ξ) ip ≤ sqn (invalidate (rt ξ) (dests ξ)) ip"
  proof
    fix ip

452
\[ \text{from * have } \text{"ip} \not\in \text{dom(dests } \xi) \lor \text{sqn (rt } \xi) \text{ ip } \leq \text{ the (dests } \xi \text{ ip)" by simp} \]
\[ \text{thus } \text{"sqn (rt } \xi) \text{ ip } \leq \text{ sqn (invalidate (rt } \xi) \text{ (dests } \xi)) \text{ ip" by (metis domI invalidate_sqn option.sel)} \]
\[ \text{qed} \]
}\]

\begin{enumerate}
\item note solve_invalidate = this
\item show \text{?thesis}
\item by (inv \_cterm inv add: onl\_invariant\_sterms \[ OF aodv\_wf dests\_bigger\_than\_sqn \]
  \[ simp add: solve_invalidate \]
\item qed
\end{enumerate}

Proposition 7.7

\textbf{lemma ip\_constant:}
\[
\text{"paodv i } \vdash \text{ onl } \Gamma_{\text{AODV}} (\lambda(\xi, _). \text{ ip } \xi = i)"
\]
by (inv \_cterm simp add: \[ \sigma_{\text{AODV}} \_def \]
\[ \]

Proposition 7.8

\textbf{lemma sender\_ip\_valid':}
\[
\text{"paodv i } \vdash \text{ A onll } \Gamma_{\text{AODV}} (\lambda((\xi, _), a, _). \text{ ancast (\lambda.m. not\_Pkt m } \rightarrow \text{ msg\_sender m } = \text{ ip } \xi) \ a)"
\]
by inv \_cterm

\textbf{lemma sender\_ip\_valid:}
\[
\text{"paodv i } \vdash \text{ A onll } \Gamma_{\text{AODV}} (\lambda((\xi, _), a, _). \text{ ancast (\lambda.m. not\_Pkt m } \rightarrow \text{ msg\_sender m } = \text{ i) } a)"
\]
by (rule step\_invariant\_weaken\_with\_invariantE \[ OF ip\_constant sender\_ip\_valid' \]
\[ (auto dest!: onl\_D onll\_D) \]
\[ \]

\textbf{lemma received\_msg\_inv:}
\[
\text{"paodv i } \vdash \text{ (recvmsg } \rightarrow) \text{ onl } \Gamma_{\text{AODV}} (\lambda(\xi, 1). \text{ l } \in \{\text{PAodv-:1}\} \rightarrow \text{ P (msg } \xi)"
\]
by inv \_cterm

Proposition 7.9

\textbf{lemma sip\_not\_ip':}
\[
\text{"paodv i } \vdash \text{ (recvmsg (\lambda.m. not\_Pkt m } \rightarrow \text{ msg\_sender m } \neq \text{ i) } \rightarrow) \text{ onl } \Gamma_{\text{AODV}} (\lambda(\xi, _). \text{ sip } \xi \neq \text{ ip } \xi)"
\]
by (inv \_cterm inv add: onl\_invariant\_sterms \[ OF aodv\_wf received\_msg\_inv \]
\[ onl\_invariant\_sterms \[ OF aodv\_wf ip\_constant \[ THEN invariant\_restrict\_inD \]
\[ simp add: clear\_locals\_sip\_not\_ip' \] clarsimp+ \]
\[ \]

\textbf{lemma sip\_not\_ip:}
\[
\text{"paodv i } \vdash \text{ (recvmsg (\lambda.m. not\_Pkt m } \rightarrow \text{ msg\_sender m } \neq \text{ i) } \rightarrow) \text{ onl } \Gamma_{\text{AODV}} (\lambda(\xi, _). \text{ sip } \xi \neq \text{ i)"
\]
by (inv \_cterm inv add: onl\_invariant\_sterms \[ OF aodv\_wf ip\_constant \[ THEN invariant\_restrict\_inD \]
\[ simp add: clear\_locals\_sip\_not\_ip' \] clarsimp+ \]
\[ \]

Neither \text{sip\_not\_ip'} \ nor \text{sip\_not\_ip} is needed to show loop freedom.

Proposition 7.10

\textbf{lemma hop\_count\_positive:}
\[
\text{"paodv i } \vdash \text{ onl } \Gamma_{\text{AODV}} (\lambda(\xi, _). \text{ \forall ip } \in \text{kD (rt } \xi) \text{. the (dhops (rt } \xi) \text{ ip)} } \geq 1)"
\]
by (inv \_cterm auto)

\textbf{lemma rreq\_dip\_in\_vD\_dip\_eq\_ip:}
\[
\text{"paodv i } \vdash \text{ onl } \Gamma_{\text{AODV}} (\lambda(\xi, 1). \text{ l } \in \{\text{PRreq-:16..PRreq-:17}\} \rightarrow \text{ dip } \xi \in \text{vD(rt } \xi)\)
\[ \wedge \text{ (l } \in \{\text{PRreq-:6, PRreq-:7}\} \rightarrow \text{ dip } \xi = \text{ ip } \xi\)
\[ \wedge \text{ (l } \in \{\text{PRreq-:15..PRreq-:17}\} \rightarrow \text{ dip } \xi \neq \text{ ip } \xi)"
\]
by inv \_cterm

\textbf{lemma rrep\_dip\_in\_vD:}
\[
\text{"paodv i } \vdash \text{ onl } \Gamma_{\text{AODV}} (\lambda(\xi, 1). \text{ l } \in \{\text{PRrep-:4}\} \rightarrow \text{ dip } \xi \in \text{vD(rt } \xi)\)"
\]
by inv \_cterm

Proposition 7.11

\textbf{lemma anycast\_msg\_zhops:}
\[
\text{"\lambda rreqid dip dsn dsk oip osn sip.}
\text{paodv i } \vdash \text{ onl } \Gamma_{\text{AODV}} (\lambda(\_, a, _). \text{ ancast msg\_zhops a)"
\]
proof (inv_cterms inv add:
  onl_invariant_sterms [OF aodv_wf rreq_dip_in_vD_dip_eq_ip]
onl_invariant_sterms [OF aodv_wf rrep_dip_in_vD]
onl_invariant_sterms [OF aodv_wf hop_count_positive],
elim conjE)
fix l ξ a pp p' pp'
assume "("ξ, pp) ∈ reachable (paodv i) TT"
and "(PRreq-:16)unicast(λξ. the (nhop (rt ξ) (oip ξ)),
  λξ. Rrep (the (dhops (rt ξ) (dip ξ))) (dip ξ) (sqn (rt ξ) (dip ξ)) (oip ξ) (ip ξ)).
p' ⩾ pp' ∈ sterms Γ_AODV pp"
and "l = PRreq-:16"
and "a = unicast (the (nhop (rt ξ) (oip ξ)))
  (Rrep (the (dhops (rt ξ) (dip ξ))) (dip ξ) (sqn (rt ξ) (dip ξ)) (oip ξ) (ip ξ))"
and *: "∀ip∈kD (rt ξ). Suc 0 ≤ the (dhops (rt ξ) ip)"
and "dip ξ ∈ vD (rt ξ)"
from dip ξ ∈ vD (rt ξ) have "dip ξ ∈ kD (rt ξ)"
by (rule vD_iD_gives_kD(1))
with * have "Suc 0 ≤ the (dhops (rt ξ) (dip ξ))" ..
thus "0 < the (dhops (rt ξ) (dip ξ))" by simp
next
fix l ξ a pp p' pp'
assume "("ξ, pp) ∈ reachable (paodv i) TT"
and "(PRrep-:4)unicast(λξ. the (nhop (rt ξ) (oip ξ)),
  λξ. Rrep (the (dhops (rt ξ) (dip ξ))) (dip ξ) (sqn (rt ξ) (dip ξ)) (oip ξ) (ip ξ)).
p' ⩾ pp' ∈ sterms Γ_AODV pp"
and "l = PRrep-:4"
and "a = unicast (the (nhop (rt ξ) (oip ξ)))
  (Rrep (the (dhops (rt ξ) (dip ξ))) (dip ξ) (sqn (rt ξ) (dip ξ)) (oip ξ) (ip ξ))"
and *: "∀ip∈kD (rt ξ). Suc 0 ≤ the (dhops (rt ξ) ip)"
and "dip ξ ∈ vD (rt ξ)"
from dip ξ ∈ vD (rt ξ) have "dip ξ ∈ kD (rt ξ)"
by (rule vD_iD_gives_kD(1))
with * have "Suc 0 ≤ the (dhops (rt ξ) (dip ξ))" ..
thus "the (dhops (rt ξ) (dip ξ)) = 0 → dip ξ = ip ξ" by auto
qed

lemma hop_count_zero_oip_dip_sip:
"paodv i |= (recvmsg msg_zhops →) onl Γ_AODV (λ(ξ, 1).
  (l∈{PAodv-:4..PAodv-:5} ∪ {PRreq-:n|n. True} →
  (hops ξ = 0 → oip ξ = sip ξ))
  ∧
  ((l∈{PAodv-:6..PAodv-:7} ∪ {PRrep-:n|n. True} →
  (hops ξ = 0 → dip ξ = sip ξ))))"
by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]) auto

lemma osn_rreq:
"paodv i |= (recvmsg rreq_rrep_sn →) onl Γ_AODV (λ(ξ, 1).
  1 ∈ {PAodv-:4, PAodv-:5} ∪ {PRreq-:n|n. True} → 1 ≤ osn ξ)"
by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]) clarsimp

lemma osn_rreq':
"paodv i |= (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) →) onl Γ_AODV (λ(ξ, 1).
  1 ∈ {PAodv-:4, PAodv-:5} ∪ {PRreq-:n|n. True} → 1 ≤ osn ξ)"
proof (rule invariant_weakenE [OF osn_rreq])
fix a
assume "recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) a"
thus "recvmsg rreq_rrep_sn a"
  by (cases a) simp_all
qed

lemma dsn_rrep:
"paodv i |= (recvmsg rreq_rrep_sn →) onl Γ_AODV (λ(ξ, 1).
  1 ∈ {PAodv-:6, PAodv-:7} ∪ {PRrep-:n|n. True} → 1 ≤ dsn ξ)"

454
by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]) clarsimp

lemma dsn_rrep':
"paodv i |= (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) →) onl Γ_AODV (λ(ξ, l).
1 ∈ {PAodv−:6, PAodv−:7} ∪ {PRrep−:n|n. True} → 1 ≤ dsn ξ)"

proof (rule invariant_weakenE [OF dsn_rrep])
fix a
assume "recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) a"
thus "recvmsg rreq_rrep_sn a"
by (cases a) simp_all
qed

lemma hop_count_zero_oip_dip_sip':
"paodv i ||= (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) →) onl Γ_AODV (λ(ξ, l).
(l ∈ {PAodv−:4..PAodv−:5} ∪ {PRreq−:n|n. True} → hops ξ = 0 → oip ξ = sip ξ)
∧ ((l ∈ {PAodv−:6..PAodv−:7} ∪ {PRrep−:n|n. True} →
(hops ξ = 0 → dip ξ = sip ξ))))"

proof (rule invariant_weakenE [OF hop_count_zero_oip_dip_sip])
fix a
assume "recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) a"
thus "recvmsg msg_zhops a"
by (cases a) simp_all
qed

Proposition 7.12
lemma zero_seq_unk_hops_one':
"paodv i ||= (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) →) onl Γ_AODV (λ(ξ, l).
∀ dip ∈ kD(rt ξ). (sqn (rt ξ) dip = 0 → sqnf (rt ξ) dip = unk)
∧ (sqnf (rt ξ) dip = unk → the (dhops (rt ξ) dip) = 1)
∧ (the (dhops (rt ξ) dip) = 1 → the (nhop (rt ξ) dip) = dip))"

proof -
{ fix dip and ξ :: state and P
assume "sqn (invalidate (rt ξ) (dests ξ)) dip = 0"
and all: "∀ ip. sqn (rt ξ) ip ≤ sqn (invalidate (rt ξ) (dests ξ)) ip"
and *: "sqn (rt ξ) dip = 0 ⟹ P ξ dip"

have "P ξ dip"
proof -
from all have "sqn (rt ξ) dip ≤ sqn (invalidate (rt ξ) (dests ξ)) dip" ..
with ⟨sqn (invalidate (rt ξ) (dests ξ)) dip = 0 ⟹ P ξ dip = 0 ⟹ P ξ dip ⟹ (rule *)

thus "P ξ dip" by (rule *)
}

note sqn_invalidate_zero [elim!] = this

{ fix dsn hops :: nat and sip oip rt and ip dip :: ip
assume "∀ dip ∈ kD(rt).
(sqn rt dip = 0 ⟹ π3(the (rt dip)) = unk) ∧
(π3(the (rt dip)) = unk ⟹ the (dhops rt dip) = Suc 0) ∧
(the (dhops rt dip) = Suc 0 ⟹ the (nhop rt dip) = dip)"

and "hops = 0 → sip = dip"
and "Suc 0 = dsn"
and "ip ≠ dip → dip ∈ kD(rt)"

hence "the (dhops (update rt dip (dsn, kno, val, Suc hops, sip)) ip) = Suc 0 →
the (nhop (update rt dip (dsn, kno, val, Suc hops, sip)) ip) = ip"
proof - (rule update_cases, auto simp add: sqn_def dest!: bepec)
}

note prreq_ok1 [simp] = this

{ fix ip dsn hops sip oip rt dip
assume "∀ dip ∈ kD(rt).
(sqn rt dip = 0 ⟹ π3(the (rt dip)) = unk) ∧
(π3(the (rt dip)) = unk ⟹ the (dhops rt dip) = Suc 0) ∧
(the (dhops rt dip) = Suc 0 ⟹ the (nhop rt dip) = dip)"

and "Suc 0 = dsn"

and "ip ≠ dip → ip ∈ kD(rt)"

hence "π₃(the (update rt dip (dsn, kno, val, Suc hops, sip) ip)) = unk →
  the (dhops (update rt dip (dsn, kno, val, Suc hops, sip)) ip) = Suc 0"

by (rule update_cases, auto simp add: sqn_def sqnf_def dest!: bspec)

} note prreq-ok2 [simp] = this

{ fix ip dsn hops sip oip rt dip

  assume "∀ dip ∈ kD(rt).
  (sqn rt dip = 0 → π₃(the (rt dip)) = unk) ∧
  (π₃(the (rt dip)) = unk → the (dhops rt dip) = Suc 0) ∧
  (the (dhops rt dip) = Suc 0 → the (nhop rt dip) = dip)"

  and "Suc 0 ≤ dsn"

  and "ip ≠ dip → ip ∈ kD(rt)"

  hence "sqn (update rt dip (dsn, kno, val, Suc hops, sip)) ip = 0 →
  π₃(the (update rt dip (dsn, kno, val, Suc hops, sip) ip)) = unk"

  by (rule update_cases, auto simp add: sqn_def sqnf_def dest!: bspec)

} note prreq-ok3 [simp] = this

{ fix rt sip

  assume "∀ dip ∈ kD rt.
  (sqn rt dip = 0 → π₃(the (rt dip)) = unk) ∧
  (π₃(the (rt dip)) = unk → the (dhops rt dip) = Suc 0) ∧
  (the (dhops rt dip) = Suc 0 → the (nhop rt dip) = dip)"

  hence "∀ dip ∈ kD rt.
  (sqn (update rt sip (0, unk, val, Suc 0, sip)) dip = 0 →
  π₃(the (update rt sip (0, unk, val, Suc 0, sip) dip)) = unk)
  ∧ (π₃(the (update rt sip (0, unk, val, Suc 0, sip) dip)) = unk →
  the (dhops (update rt sip (0, unk, val, Suc 0, sip)) dip) = Suc 0)
  ∧ (the (dhops (update rt sip (0, unk, val, Suc 0, sip) dip)) = Suc 0 →
  the (nhop (update rt sip (0, unk, val, Suc 0, sip) dip)) = dip)"

  by (rule update_cases) simp_all

} note prreq-ok4 [simp] = this

have prreq-ok5 [simp]: "∀ sip rt.
  π₃(the (update rt sip (0, unk, val, Suc 0, sip) sip)) = unk →
  the (dhops (update rt sip (0, unk, val, Suc 0, sip) sip)) = Suc 0"

  by (rule update_cases) simp_all

have prreq-ok6 [simp]: "∀ sip rt.
  sqn (update rt sip (0, unk, val, Suc 0, sip)) sip = 0 →
  π₃(the (update rt sip (0, unk, val, Suc 0, sip) sip)) = unk"

  by (rule update_cases) simp_all

show ?thesis

  by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf hop_count_zero_oip_dip_sip']
  seq_step_invariant_sterms_TT [OF sqns_increase aodv_wf aodv_trans]
  onl_invariant_sterms [OF aodv_wf osn_rreq']
  onl_invariant_sterms [OF aodv_wf dsn_rrep']) clarsimp+

qed

lemma zero_seq_unk_hops_one:

  "paodv i |= (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) → onl Γ_AODV (λ(ξ, _).
  ∀ dip ∈ kD(rt ξ). (sqn (rt ξ) dip = 0 → (sqnf (rt ξ) dip = unk
  ∧ the (dhops (rt ξ) dip) = 1
  ∧ the (nhop (rt ξ) dip) = dip))))"

  by (rule invariant_weakenE [OF zero_seq_unk_hops_one']) auto

lemma kD_unk_or_atleast_one:

  "paodv i |= (recvmsg rreq_rrep_sn →) onl Γ_AODV (λ(ξ, 1).
  ∀ dip ∈ kD(rt ξ). π₃(the (rt ξ dip)) = unk ∨ 1 ≤ π₂(the (rt ξ dip)))"

proof -

  { fix sip rt dsn1 dsn2 dsk1 dsk2 flag1 flag2 hops1 hops2 nhip1 nhip2

    assume "dsk1 = unk ∨ Suc 0 ≤ dsn2"

    hence "π₃(the (update rt sip (dsn1, dsk1, flag1, hops1, nhip1, nhip2) sip)) = unk


456
Suc 0 ≤ sqn (update rt sip (dsn1, dsk1, flag1, hops1, nhip1)) sip

unfolding update_def by (cases "dsk1 = unk") (clarsimp split: option.split)+

note fromsip [simp] = this

\{ fix dip sip rt dsn1 dsn2 dsk1 dsk2 flag1 flag2 hops1 hops2 nhip1 nhip2
  assume allkd: "∀ dip ∈ kD(rt). π₃ (the (rt dip)) = unk ∨ Suc 0 ≤ sqn rt dip"
  and **: "dsk1 = unk ∨ Suc 0 ≤ dsn2"
  have "∀ dip ∈ kD(rt). π₃ (the (update rt sip (dsn1, dsk1, flag1, hops1, nhip1) dip)) = unk
  ∨ Suc 0 ≤ sqn (update rt sip (dsn2, dsk2, flag2, hops2, nhip2)) dip"
  (is "∀ dip ∈ kD(rt). ?prop dip")
\} note solve_update [simp] = this

\{ fix dip rt dests
  assume *: "∀ ip ∈ dom(dests). ip ∈ kD(rt) ∧ sqn rt ip ≤ the (dests ip)"
  and **: "∀ ip ∈ kD(rt). π₃ (the (rt ip)) = unk ∨ Suc 0 ≤ sqn rt ip"
  have "∀ dip ∈ kD(rt). π₃ (the (rt dip)) = unk ∨ Suc 0 ≤ sqn (invalidate rt dests dip)"
\} note solve_invalidate [simp] = this

show ?thesis

by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf dests_bigger_than_sqn
  [THEN invariant_restrict_inD]]
onl_invariant_sterms [OF aodv_wf osn_rreq]
onl_invariant_sterms [OF aodv_wf dsn_rrep]
simp add: proj3_inv proj2_eq_sqn)

done
Proposition 7.13

lemma rreq_rrep_sn_any_step_invariant:
"paodv i |=_A (recvmsg rreq_rrep_sn @→) onll Γ_{AODV} (λ(_, a, _). anycast rreq_rrep_sn a)"
proof

have sqnf_kno: "paodv i |= onll Γ_{AODV} (λξ. (l ∈ {PRreq-:16} → dip ξ ∈ kD (rt ξ) ∧ sqnf (rt ξ) (dip ξ) = kno))"
  by (inv_cterms)

have rrep_sqn_greater_dsn: "paodv i |= (recvmsg rreq_rrep_sn @→) onll Γ_{AODV} (λξ. (l ∈ {PRrep-:1 .. PRrep-:4} → 1 ≤ sqn (rt ξ) (dip ξ))"
  by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]
  onl_invariant_sterms [OF aodv_wf dsn_rrep]
  onl_invariant_sterms [simplified]
  onl_invariant_sterms_TT [OF aodv_wf sqnf_kno]
  onl_invariant_sterms [OF aodv_wf osn_rreq]
  onl_invariant_sterms [OF aodv_wf dsn_rrep]
  onl_invariant_sterms [OF aodv_wf rrep_sqn_greater_dsn])
  (clarsimp simp: update_kno_dsn_greater_zero)
  show ?thesis
  by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf sequence_number_one_or_bigger [THEN invariant_restrict_inD]
    onl_invariant_sterms [OF aodv_wf kD_unk_or_atleast_one]
    onl_invariant_sterms_TT [OF aodv_wf sqnf_kno]
    onl_invariant_sterms [OF aodv_wf osn_rreq]
    onl_invariant_sterms [OF aodv_wf dsn_rrep]
    onl_invariant_sterms [OF aodv_wf rrep_sqn_greater_dsn])
  (auto simp: proj2_eq_sqn)
  qed

Proposition 7.14

lemma rreq_rrep_fresh_any_step_invariant:
"paodv i |= onll Γ_{AODV} (λξ. (l ∈ {PRreq-:3..PRreq-:22} @→ oip ξ ∈ kD (rt ξ) ∧ (sqn (rt ξ) (oip ξ) > (osn ξ)) \∨ (sqn (rt ξ) (oip ξ) = (osn ξ) \land the (dhops (rt ξ) (oip ξ)) ≤ Suc (hops ξ)) \land the (flag (rt ξ) (oip ξ) = val))))"
proof
fix l ξ l' pp p'
assume "((ξ, pp) ∈ reachable (paodv i)) TT" and 
"{PRreq-:2}ξ. (rt := update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ))\p' \in sterms Γ_{AODV} pp" and 
"l' = PRreq-" show "osn ξ < sqn \land the (dhops (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ)) (oip ξ) \land the (flag (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ) (oip ξ)) (oip ξ) \le Suc (hops ξ) \land the (flag (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ)) (oip ξ) = val)"
  unfolding update_def by (clarsimp split: option.split)
  (metis linorder_neqE_nat not_less)
  qed

have rreq_rprrep: "paodv i |= onll Γ_{AODV} (λξ. (l ∈ {PRreq-:2} @→ (dip ξ ∈ kD (rt ξ) \land the (flag (rt ξ) (dip ξ)) = val))))"
  by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf sip_in_kD])

have rreq_oip_kD: "paodv i |= onll Γ_{AODV} (λξ. (l ∈ {PRreq-:1 .. PRreq-:22} @→ oip ξ ∈ kD (rt ξ))))"
  by (inv_cterms)

have rreq_dip_kD_oip_sqn: "paodv i |= onll Γ_{AODV} (λξ. (l ∈ {PRreq-:16 .. PRreq-:17}) @→ (dip ξ ∈ kD (rt ξ) \land (sqn (rt ξ) (oip ξ)) > (osn ξ))
  proof
  have rreq_oip: "paodv i |=_A onll Γ_{AODV} (λξ. (l ∈ {PRreq-:16} → dip ξ ∈ kD (rt ξ) ∧ sqnf (rt ξ) (dip ξ) = kno))"
    by (inv_cterms)
  have rrep_sqn_greater_dsn: "paodv i |=_A (recvmsg rreq_rrep_sn @→) onll Γ_{AODV} (λξ. (l ∈ {PRrep-:1 .. PRrep-:4} → 1 ≤ sqn (rt ξ) (dip ξ)))"
        (auto simp: proj2_eq_sqn)
      qed
Proposition 7.15

Lemma rerr_invalid_any_step_invariant:

\[ \text{paodv } i \mid= \text{onl } \Gamma_{\text{AODV}} (\lambda(\xi, a, \_). \text{anycast (rerr_invalid (rt } \xi) a)) \]

Proof -

have dests_inv: \[ \text{paodv } i \mid= \text{onl } \Gamma_{\text{AODV}} (\lambda(\xi, 1). (1 \in \{\text{PAodv-:15, PPkt-:7, PRreq-:11, PRreq-:20, PRrep-:7, PRerr-:1}\}
\land (1 \in \{\text{PAodv-:16..PAodv-:17}\}
\cup \{\text{PPkt-:8..PPkt-:9}\}
\cup \{\text{PRreq-:12..PRreq-:13}\}
\cup \{\text{PRreq-:21..PRreq-:22}\}
\cup \{\text{PRrep-:8..PRrep-:9}\}
\cup \{\text{PRerr-:2..PRerr-:4}\}
\rightarrow (\forall i \in \text{dom(dests } \xi). i \in \text{vD(rt } \xi)))
\land (1 = \text{PRerr-:1}
\rightarrow (\forall i \in \text{dom(dests } \xi). i \in \text{vD(rt } \xi)))" 
\]

by inv_cterms (clarsimp split: split_if_asm option.split_asm simp: domIff) +

show \(?thesis\)
by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf dests_inv])

qed

Proposition 7.16

Some well-definedness obligations are irrelevant for the Isabelle development:

1. In each routing table there is at most one entry for each destination: guaranteed by type.
2. In each store of queued data packets there is at most one data queue for each destination: guaranteed by structure.
3. Whenever a set of pairs (rip, rsn) is assigned to the variable dests of type \[ i \rightarrow \text{sqn}, \text{or to the first argument of the function } \text{rerr}, \text{this set is a partial function, i.e., there is at most one entry (rip, rsn) for each destination rip: guaranteed by type.} \]

Lemma dests_vD_inc_sqn:

\[ \text{paodv } i \mid= \text{onl } \Gamma_{\text{AODV}} (\lambda(\xi, 1). (1 \in \{\text{PAodv-:15, PPkt-:7, PRreq-:11, PRreq-:20, PRrep-:7}\}
\land (1 \in \{\text{PAodv-:16..PAodv-:17}\}
\cup \{\text{PPkt-:8..PPkt-:9}\}
\cup \{\text{PRreq-:12..PRreq-:13}\}
\cup \{\text{PRreq-:21..PRreq-:22}\}
\cup \{\text{PRrep-:8..PRrep-:9}\}
\cup \{\text{PRerr-:2..PRerr-:4}\}
\rightarrow (\forall i \in \text{dom(dests } \xi). i \in \text{vD(rt } \xi)))
\land (1 = \text{PRerr-:1}
\rightarrow (\forall i \in \text{dom(dests } \xi). i \in \text{vD(rt } \xi)))" 
\]

by inv_cterms (clarsimp split: split_if_asm option.split_asm simp: domIff) +

Lemma route_tables_fresher:

\[ \text{paodv } i \mid= \text{onl } \Gamma_{\text{AODV}} (\lambda(\xi, \_). \text{anycast (rerr_invalid (rt } \xi) a))" 
\]

by inv_cterms (clarsimp split: split_if_asm option.split_asm simp: domIff) +

qed

Proposition 7.27

Some well-definedness obligations are irrelevant for the Isabelle development:

1. In each routing table there is at most one entry for each destination: guaranteed by type.
2. In each store of queued data packets there is at most one data queue for each destination: guaranteed by structure.
3. Whenever a set of pairs (rip, rsn) is assigned to the variable dests of type \[ i \rightarrow \text{sqn}, \text{or to the first argument of the function } \text{rerr}, \text{this set is a partial function, i.e., there is at most one entry (rip, rsn) for each destination rip: guaranteed by type.} \]
5.7 The quality increases predicate

definition quality_increases :: "state ⇒ bool" where "quality_increases ξ ξ' ≡ (∀ dip∈kD(rt ξ). dip ∈ kD(rt ξ') ∧ rt ξ ⊑ dip rt ξ') ∧ (∀ dip. sqn (rt ξ) dip ≤ sqn (rt ξ') dip)"

lemma quality_increasesI [intro!]:
  assumes "∀ dip. dip ∈ kD(rt ξ) ⇒ dip ∈ kD(rt ξ')" and "∀ dip. dip ∈ kD(rt ξ')" and "∀ dip. sqn (rt ξ) dip ≤ sqn (rt ξ') dip"
  shows "quality_increases ξ ξ'"

unfolding quality_increases_def using assms by clarsimp

lemma quality_increasesE [elim]:
  fixes dip
  assumes "quality_increases ξ ξ'" and "dip∈kD(rt ξ)" and "dip∈kD(rt ξ')" and "dip∈kD(rt ξ')" and "∀ dip. sqn (rt ξ) dip ≤ sqn (rt ξ') dip"
  shows "R dip ξ'"

using assms unfolding quality_increases_def by clarsimp

lemma quality_increases_rt_fresherD [dest]:
fixes ip
assumes "quality_increases ξ ξ'"
shows "rt ξ ⊑ ip rt ξ'"
using assms by auto

lemma quality_increases_sqnE [elim]:
fixes dip
assumes "quality_increases ξ ξ'"
shows "sqn (rt ξ) dip ≤ sqn (rt ξ') dip =⇒ R dip ξ ξ'"
using assms unfolding quality_increases_def by clarsimp

lemma quality_increases_refl [intro, simp]: "quality_increases ξ ξ"
by rule simp_all

lemma strictly_fresher_quality_increases_right [elim]:
fixes σ σ' dip
assumes "rt (σ i) ⊑ dip rt (σ nhip)"
shows "rt (σ i) ⊑ dip (σ nhip)" by auto
proof -
  from qinc have "rt (σ nhip) ⊑ dip (σ nhip)" using ⟨dip ∈ kD(rt (σ nhip))⟩
  by auto
next
  from ⟨质量_increases sqnE ⟩ show ?thesis ..
qed

lemma kD_quality_increases [elim]:
assumes "i ∈ kD(rt ξ)"
shows "i ∈ kD(rt ξ')" using assms by auto

lemma kD_nsqn_quality_increases [elim]:
assumes "i ∈ kD(rt ξ)" and "quality_increases ξ ξ'"
shows "i ∈ kD(rt ξ') ∧ nsqn (rt ξ) i ≤ nsqn (rt ξ') i" by (rule kD_nsqn_quality_increases [THEN conjunct2])

lemma nsqn_quality_increases [elim]:
assumes "i ∈ kD(rt ξ)" and "quality_increases ξ ξ'"
shows "nsqn (rt ξ) i ≤ nsqn (rt ξ') i" using assms by (rule kD_nsqn_quality_increases [THEN conjunct2])

lemma kD_nsqn_quality_increases_trans [elim]:
assumes "i ∈ kD(rt ξ)" and "s ≤ nsqn (rt ξ) i" and "quality_increases ξ ξ'"
shows "i ∈ kD(rt ξ') ∧ s ≤ nsqn (rt ξ') i" by (rule le_trans)
proof
  from ⟨质量_increases sqnE ⟩ and ⟨质量_increases sqnE ⟩ show "i ∈ kD(rt ξ')" ..
  next
  from ⟨质量_increases sqnE ⟩ and ⟨质量_increases sqnE ⟩ have "nsqn (rt ξ) i ≤ nsqn (rt ξ') i" ..
  with ⟨质量_increases sqnE ⟩ show "s ≤ nsqn (rt ξ') i" by (rule le_trans)
qed

461
lemma nsqn_quality_increases_nsqn_lt_lt [elim]:
assumes "i ∈ kD(rt ξ)"
and "quality_increases ξ ξ'"
and "s < nsqn (rt ξ) i"
shows "s < nsqn (rt ξ') i"
proof -
from assms(1-2) have "nsqn (rt ξ) i ≤ nsqn (rt ξ') i" ..
with \( s < nsqn (rt ξ) i \), show "s < nsqn (rt ξ') i" by simp
qed

lemma nsqn_quality_increases_dhops [elim]:
assumes "i ∈ kD(rt ξ)"
and "quality_increases ξ ξ'"
and "nsqn (rt ξ) i = nsqn (rt ξ') i"
shows "the (dhops (rt ξ) i) ≥ the (dhops (rt ξ') i)"
using assms unfolding quality_increases_def
by (clarsimp) (drule(1) bspec, clarsimp simp: rt_fresher_def2)

lemma nsqn_quality_increases_nsqn_eq_le [elim]:
assumes "i ∈ kD(rt ξ)"
and "quality_increases ξ ξ'"
and "s = nsqn (rt ξ) i"
shows "s < nsqn (rt ξ') i ∨ (s = nsqn (rt ξ') i ∧ the (dhops (rt ξ) i) ≥ the (dhops (rt ξ') i))"
using assms by (metis nat_less_le nsqn_quality_increases nsqn_quality_increases_dhops)

lemma quality_increases_rreq_rrep_props [elim]:
fixes sn ip hops sip
assumes qinc: "quality_increases (σ sip) (σ' sip)"
and "1 ≤ sn"
and *: "ip ∈ kD(rt (σ sip)) ∧ sn ≤ nsqn (rt (σ sip)) ip
∧ (nsqn (rt (σ sip)) ip = sn
→ (the (dhops (rt (σ sip)) ip) ≤ hops
∨ the (flag (rt (σ sip)) ip) = inv))"
shows "ip ∈ kD(rt (σ' sip)) ∧ sn ≤ nsqn (rt (σ' sip)) ip
∧ (nsqn (rt (σ' sip)) ip = sn
→ (the (dhops (rt (σ' sip)) ip) ≤ hops
∨ the (flag (rt (σ' sip)) ip) = inv))"
(is "_ ∧ ?nsqnafter")
proof -
from * obtain "ip ∈ kD(rt (σ sip))" and "sn ≤ nsqn (rt (σ sip)) ip" by auto
from quality_increases (σ sip) (σ' sip)
have "sqn (rt (σ sip)) ip ≤ sqn (rt (σ' sip)) ip" ..
from quality_increases (σ sip) (σ' sip) and ip ∈ kD (rt (σ sip))
have "ip ∈ kD (rt (σ' sip))" ..
from sn ≤ nsqn (rt (σ sip)) ip have ?nsqnafter
proof
assume "sn < nsqn (rt (σ sip)) ip"
also from ip ∈ kD(rt (σ sip)) and quality_increases (σ sip) (σ' sip)
have "... ≤ nsqn (rt (σ' sip)) ip" ..
finally have "sn < nsqn (rt (σ' sip)) ip" .
thus ?thesis by simp
next
assume "sn = nsqn (rt (σ sip)) ip"
with ip ∈ kD(rt (σ sip)) and quality_increases (σ sip) (σ' sip)
have "sn < nsqn (rt (σ' sip)) ip
∨ (sn = nsqn (rt (σ' sip)) ip
∧ the (dhops (rt (σ' sip)) ip) ≤ the (dhops (rt (σ sip)) ip))" ..
hence "sn < nsqn (rt (σ' sip)) ip
∨ (nsqn (rt (σ' sip)) ip = sn ∧ the (dhops (rt (σ' sip)) ip) ≤ hops
∨ the (flag (rt (σ' sip)) ip) = inv))"
proof
assume "sn < nsqn (rt (σ' sip)) ip" thus ?thesis ..
next

assume "sn = nsqn (rt (σ' sip)) ip
∧ the (dhops (rt (σ sip)) ip) ≥ the (dhops (rt (σ' sip)) ip)"
hence "sn = nsqn (rt (σ sip)) ip"
and "the (dhops (rt (σ' sip)) ip) ≤ the (dhops (rt (σ sip)) ip)" by auto

from * and ⟨sn = nsqn (rt (σ sip)) ip⟩ have "the (dhops (rt (σ sip)) ip) ≤ hops
∨ the (flag (rt (σ sip)) ip) = inv"

by simp
thus ?thesis
proof
assume "the (dhops (rt (σ sip)) ip) ≤ hops"
with ⟨the (dhops (rt (σ' sip)) ip) ≤ the (dhops (rt (σ sip)) ip)⟩
have "the (dhops (rt (σ' sip)) ip) ≤ hops" by simp
with ⟨sn = nsqn (rt (σ' sip)) ip⟩ show ?thesis by simp

next

assume "the (flag (rt (σ sip)) ip) = inv"
with ⟨ip ∈ kD(rt (σ sip))⟩ have "nsqn (rt (σ' sip)) ip = sqn (rt (σ sip)) ip - 1" ..

with ⟨sn ≥ 1⟩ and ⟨sn = nsqn (rt (σ sip)) ip⟩
have "sqn (rt (σ sip)) ip > 1" by simp

from ⟨ip ∈ kD(rt (σ' sip))⟩ show ?thesis
proof (rule vD_or_iD)
  assume "ip ∈ vD(rt (σ' sip))"
  hence "nsqn (rt (σ' sip)) ip = sqn (rt (σ' sip)) ip" ..
  with ⟨sqn (rt (σ sip)) ip ≤ sqn (rt (σ' sip)) ip⟩
  have "nsqn (rt (σ' sip)) ip ≥ sqn (rt (σ sip)) ip" by simp

with ⟨sqn (rt (σ sip)) ip > 1⟩
  have "nsqn (rt (σ' sip)) ip > sqn (rt (σ sip)) ip - 1" by simp
with ⟨nsqn (rt (σ' sip)) ip = sqn (rt (σ sip)) ip - 1⟩
  have "nsqn (rt (σ' sip)) ip > nsqn (rt (σ sip)) ip" by simp
with ⟨sn = nsqn (rt (σ sip)) ip⟩ have "nsqn (rt (σ' sip)) ip > sn"
by simp
thus ?thesis ..
qed

lemma quality_increases_rreq_rrep_props':
  fixes sn ip hops sip
  assumes "∀ j. quality_increases (σ j) (σ' j)"
  and "1 ≤ sn"
  and *: "ip ∈ kD (rt (σ' sip)) ∧ sn ≤ nsqn (rt (σ sip)) ip
  ∧ (nsqn (rt (σ sip)) ip = sn
      → (the (dhops (rt (σ sip)) ip) ≤ hops
          ∨ the (flag (rt (σ sip)) ip) = inv))"
  shows "ip ∈ kD (rt (σ' sip)) ∧ sn ≤ nsqn (rt (σ' sip)) ip
  ∧ (nsqn (rt (σ' sip)) ip = sn
      → (the (dhops (rt (σ' sip)) ip) ≤ hops
          ∨ the (flag (rt (σ' sip)) ip) = inv))"
proof -
  from assms(1) have "quality_increases (σ sip) (σ' sip)" ..
thus \( ?\text{thesis} \) using \( \text{assms}(2-3) \) by (rule \( \text{quality}\_\text{increases}\_\text{rreq}\_\text{rrep}\_\text{props} \))

\[ \text{qed} \]

**Lemma rreq\_quality\_increases:**

assumes \( \forall j. \ j \neq i \rightarrow \text{quality}\_\text{increases} \ (\sigma j) \ (\sigma' j) \)

and \( \text{rt} \ (\sigma' i) = \text{rt} \ (\sigma i) \)

shows \( \forall j. \text{quality}\_\text{increases} \ (\sigma j) \ (\sigma' j) \)

using \( \text{assms} \) by \( \text{clarsimp} \) (metis \( \text{order}\_\text{refl} \) \( \text{quality}\_\text{increases}\_I \) \( \text{rt}\_\text{fresher}\_\text{refl} \))

**Definition msg\_fresh ::** "(ip \Rightarrow \text{state}) \Rightarrow \text{msg} \Rightarrow \text{bool}"

where \( \text{msg}\_\text{fresh} \ (\sigma) m \equiv \)

\[ \text{case } m \text{ of } \text{Rreq} \ h o \ oip c \ osnc \ sipc \ \Rightarrow \osnc \geq 1 \land (sipc \neq oipc \rightarrow \]
\[ \begin{align*}
& \text{oipc} \in \text{kD} \ (\text{rt} \ (\sigma \ sipc)) \land \text{nsqn} \ (\text{rt} \ (\sigma \ sipc)) \ oipc \geq \osnc \\
& \rightarrow (\text{hopsc} \geq \text{the} \ (\text{dhops} \ (\text{rt} \ (\sigma \ sipc)) \ oipc) \\
& \lor \text{the} \ (\text{flag} \ (\text{rt} \ (\sigma \ sipc)) \ oipc) = \text{inv}) \))
\end{align*} \]

| Rrep \ h o \ oip c \ osnc \ sipc \ \Rightarrow \dsnc \geq 1 \land (sipc \neq dipc \rightarrow |)
\[ \begin{align*}
& \text{dipc} \in \text{kD} \ (\text{rt} \ (\sigma \ sipc)) \land \text{nsqn} \ (\text{rt} \ (\sigma \ sipc)) \ dipc \geq \dsnc \\
& \rightarrow (\text{hopsc} \geq \text{the} \ (\text{dhops} \ (\text{rt} \ (\sigma \ sipc)) \ dipc) \\
& \lor \text{the} \ (\text{flag} \ (\text{rt} \ (\sigma \ sipc)) \ dipc) = \text{inv}) \))
\end{align*} \]

| Rerr \ destsc \ sipc \ \Rightarrow \forall ripc \in \text{dom} (\text{destsc}). \ (\text{ripc} \in \text{kD} \ (\text{rt} \ (\sigma \ sipc)) |)
\[ \begin{align*}
& \text{the} \ (\text{destsc} \ ripc) - 1 \leq \text{nsqn} \ (\text{rt} \ (\sigma \ sipc)) \ ripc)
\end{align*} \]

| _ \ \Rightarrow \text{True} |

**Lemma msg\_fresh [simp]:**

"\( \land \text{hopsc} \ dsnc \ \text{dsn} \ \text{dsk} \ \text{oip} \ \text{osn} \ \text{sip} \ \text{handled} \). \ 
\text{msg}\_\text{fresh} \ (\sigma) \ (\text{Rreq} \ \text{hopsc} \ \text{dsn} \ \text{dsk} \ \text{oip} \ \text{osn} \ \text{sip} \ \text{handled}) = \)

\[ \begin{align*}
& (\text{osnc} \geq 1 \land (\text{sip} \neq \text{oip} \rightarrow \text{oipc} \in \text{kD} \ (\text{rt} \ (\sigma \ sipc)) \\
& \land \text{nsqn} \ (\text{rt} \ (\sigma \ sipc)) \ oipc \geq \osnc \\
& \land (\text{nsqn} \ (\text{rt} \ (\sigma \ sipc)) \ oipc = \osnc \\
& \rightarrow (\text{hopsc} \geq \text{the} \ (\text{dhops} \ (\text{rt} \ (\sigma \ sipc)) \ oipc) \\
& \lor \text{the} \ (\text{flag} \ (\text{rt} \ (\sigma \ sipc)) \ oipc) = \text{inv})) \))
\end{align*} \]

"\( \land \text{hopsc} \ dsnc \ \text{dsn} \ \text{oip} \ \text{sip} \). \ 
\text{msg}\_\text{fresh} \ (\sigma) \ (\text{Rreq} \ \text{hopsc} \ \text{dsn} \ \text{oip} \ \text{sip}) = \)

\[ \begin{align*}
& (\text{dsn} \geq 1 \land (\text{sip} \neq \text{dip} \rightarrow \text{dipc} \in \text{kD} \ (\text{rt} \ (\sigma \ sipc)) \\
& \land \text{nsqn} \ (\text{rt} \ (\sigma \ sipc)) \ dipc \geq \dsnc \\
& \land (\text{nsqn} \ (\text{rt} \ (\sigma \ sipc)) \ dipc = \dsnc \\
& \rightarrow (\text{hopsc} \geq \text{the} \ (\text{dhops} \ (\text{rt} \ (\sigma \ sipc)) \ dipc) \\
& \lor \text{the} \ (\text{flag} \ (\text{rt} \ (\sigma \ sipc)) \ dipc) = \text{inv})) \))
\end{align*} \]

"\( \land destsc \ \text{sip} \). \ 
\text{msg}\_\text{fresh} \ (\sigma) \ (\text{Rerr} \ \text{destsc} \ \text{sip}) = \)

\[ \begin{align*}
& (\forall \text{ripc} \in \text{dom} (\text{destsc}). \ (\text{ripc} \in \text{kD} \ (\text{rt} \ (\sigma \ sipc)) \\
& \land \text{the} \ (\text{destsc} \ ripc) - 1 \leq \text{nsqn} \ (\text{rt} \ (\sigma \ sipc)) \ ripc)
\end{align*} \]

"\( \land \text{d} \ \text{dip} \). \ 
\text{msg}\_\text{fresh} \ (\sigma) \ (\text{Newpkt} \ \text{d} \ \text{dip}) = \text{True}" \n
"\( \land \text{d} \ \text{dip} \). \ 
\text{msg}\_\text{fresh} \ (\sigma) \ (\text{Pkt} \ \text{d} \ \text{dip} \ \text{sip}) = \text{True}"

unfolding \( \text{msg}\_\text{fresh}\_\text{def} \) by \( \text{simp} \_\text{all} \)

**Lemma msg\_fresh\_inc\_sn [simp, elim]:**

"\( \text{msg}\_\text{fresh} \ (\sigma) m \Rightarrow \text{rreq}\_\text{rrep}\_\text{sn} \ m \)"

by (cases \( m \)) \( \text{simp} \_\text{all} \)

**Lemma recv\_msg\_fresh\_inc\_sn [simp, elim]:**

"\( \text{orecvmsg} \ (\text{msg}\_\text{fresh}) \ (\sigma) m \Rightarrow \text{recvmsg} \ \text{rreq}\_\text{rrep}\_\text{sn} \ m \)"

by (cases \( m \)) \( \text{simp} \_\text{all} \)

**Lemma rreq\_nsqn\_is\_fresh [simp]:**

fixes \( \sigma \) msg hops dip dsnc dsnc oip osn sip handled

assumes "\( \text{rreq}\_\text{rrep}\_\text{fresh} \ (\text{rt} \ (\sigma \ sipc)) \ (\text{Rreq} \ \text{hops} \ \text{dsn} \ \text{dsk} \ \text{oip} \ \text{osn} \ \text{sip} \ \text{handled}) \)"

and "\( \text{rreq}\_\text{rrep}\_\text{sn} \ (\text{Rreq} \ \text{hops} \ \text{dsn} \ \text{dsk} \ \text{oip} \ \text{osn} \ \text{sip} \ \text{handled}) \)"

shows "\( \text{msg}\_\text{fresh} \ (\text{Rreq} \ \text{hops} \ \text{dsn} \ \text{dsk} \ \text{oip} \ \text{osn} \ \text{sip} \ \text{handled}) \)"

(is "\( \text{msg}\_\text{fresh} \ (\sigma) \ ?\text{msg}\)"

proof -

let \( \text{rt} = \text{rt} \ (\sigma \ sipc) \)

from \( \text{assms}(2) \) have "\( 1 \leq \text{osn} \)" by \( \text{simp} \)

thus \(?\text{thesis} \)
unfolding msg_fresh_def
proof (simp only: msg.case, intro conjI impI)
  assume "sip ≠ oip"
  with assms(1) show "oip ∈ kD(?rt)" by simp
next
  assume "sip ≠ oip"
  and "nsqn ?rt oip = osn"
  show "the (dhops ?rt oip) ≤ hops ∨ the (flag ?rt oip) = inv"
  proof (cases "oip ∈ vD(?rt)"
    assume "oip ∈ vD(?rt)"
    hence "nsqn ?rt oip = sqn ?rt oip" ..
    with ⟨nsqn ?rt oip = osn⟩ have "sqn ?rt oip = osn" by simp
    with assms(1) and ⟨sip ≠ oip⟩ have "the (dhops ?rt oip) ≤ hops"
    by simp
    thus ?thesis ..
  next
    assume "oip /∈ vD(?rt)"
    moreover from assms(1) have "oip ∈ kD(?rt)" by simp
    ultimately have "oip ∈ D(?rt)" by auto
    hence "the (flag ?rt oip) = inv" ..
    thus ?thesis ..
  qed
next
  assume "sip ≠ oip"
  with assms(1) have "osn ≤ sqn ?rt oip" by auto
  thus "osn ≤ nsqn (rt (σ sip)) oip" (is "msg_fresh σ ?msg")
  proof
    let ?rt = "rt (σ sip)"
    from assms have "sip ≠ dip → dip ∈ D(?rt) ∧ sqn ?rt dip = dsn ∧ the (flag ?rt dip) = val"
    by simp
    hence "sip ≠ dip → dip ∈ kD(?rt) ∧ nsqn ?rt dip ≥ dsn"
    by clarsimp
    with assms show "msg_fresh σ ?msg"
    by clarsimp
  qed

lemma rrep_nsqn_is_fresh [simp]:
fixes σ msg hops dip dsn oip sip
assumes "rreq_rrep_fresh (rt (σ sip)) (Rrep hops dip dsn oip sip)"
and "rreq_rrep_sn (Rrep hops dip dsn oip sip)"
shows "msg_fresh σ (Rrep hops dip dsn oip sip)"
(is "msg_fresh σ ?msg")
proof -
  let ?rt = "rt (σ sip)"
  from assms have "sip ≠ dip → dip ∈ D(?rt) ∧ sqn ?rt dip = dsn ∧ the (flag ?rt dip) = val"
  by simp
  hence "sip ≠ dip → dip ∈ kD(?rt) ∧ nsqn ?rt dip ≥ dsn"
  by clarsimp
  with assms show "msg_fresh σ ?msg"
  by clarsimp
qed

lemma rerr_nsqn_is_fresh [simp]:
fixes σ msg dests sip
assumes "rerr_invalid (rt (σ sip)) (Rerr dests sip)"
shows "msg_fresh σ (Rerr dests sip)"
(is "msg_fresh σ ?msg")
proof

let \(?rt = \"rt (\sigma \ sip)\"
from assms have \(*: \("(\forall \text{rip} \in \text{dom(dests)). (\text{rip} \in \text{D}(\text{rt (\sigma \ sip)))}
\land \text{the (dests rip) = sqn (rt (\sigma \ sip)) rip})\)"
by clarsimp
have \("(\forall \text{rip} \in \text{dom(dests)). (\text{rip} \in \text{kD}(\text{rt (\sigma \ sip)))}
\land \text{the (dests rip) - 1 \leq nsqn (rt (\sigma \ sip)) rip})\)"
proof
fix rip
assume "rip \in \text{dom dests"
with \(* have \"(\text{rip} \in \text{D}(\text{rt (\sigma \ sip)))\)" and \"(\text{the (dests rip) = sqn (rt (\sigma \ sip)) rip})\"
by auto
from this(2) have \"(\text{the (dests rip) - 1 \leq nsqn (rt (\sigma \ sip)) rip})\"
qed
thus "msg_fresh \sigma \ ?msg"
by simp
qed

lemma quality_increases_msg_fresh [elim]:
assumes q-inc: "\(\forall \ j. \text{quality_increases (\sigma \ j) (\sigma' \ j)}\)"
and "msg_fresh \sigma \ m"
shows "msg_fresh \sigma' \ m"
using assms(2)
proof (cases m)
fix hops dip dsn dsk dip oip osn sip handled
assume [simp]: "m = \text{Rreq hops dip dsn dsk oip osn sip handled"
and "msg_fresh \sigma \ m"
then have "osn \geq 1" and "sip = oip \lor (\text{osn} \leq \text{nsqn (rt (\sigma \ sip)) oip}
\land (\text{nsqn (rt (\sigma \ sip)) oip = osn}
\lor (\text{the (dhops (rt (\sigma \ sip)) oip) \leq hops}
\lor \text{the (flag (rt (\sigma \ sip)) oip) = inv}))\)"
by auto
from this(2) show \(?thesis
proof
assume "sip = oip" with \(?osn \geq 1\) show \(?thesis by simp
next
assume "oip \in \text{kD}(\text{rt (\sigma \ sip)))" and "osn \leq \text{nsqn (rt (\sigma \ sip)) oip}
\land (\text{nsqn (rt (\sigma \ sip)) oip = osn}
\lor (\text{the (dhops (rt (\sigma \ sip)) oip) \leq hops}
\lor \text{the (flag (rt (\sigma \ sip)) oip) = inv}))"
moreover from q-inc have "quality_increases (\sigma \ sip) (\sigma' \ sip)"
ultimately have "oip \in \text{kD}(\text{rt (\sigma' \ sip)))" and "osn \leq \text{nsqn (rt (\sigma' \ sip)) oip}
\land (\text{nsqn (rt (\sigma' \ sip)) oip = osn}
\lor (\text{the (dhops (rt (\sigma' \ sip)) oip) \leq hops}
\lor \text{the (flag (rt (\sigma' \ sip)) oip) = inv}))"
using \(?osn \geq 1\) by (rule quality_increases_rreq_rrep_props [rotated 2])
with \(?osn \geq 1\) show "msg_fresh \sigma' \ m"
by (clarsimp)
qed
next
fix hops dip dsn oip sip
assume [simp]: "m = \text{Rrep hops dip dsn oip sip"
and "msg_fresh \sigma \ m"
then have "dsn \geq 1" and "sip = dip \lor (\text{dsn} \leq \text{nsqn (rt (\sigma \ sip)) dip}
\land (\text{nsqn (rt (\sigma \ sip)) dip = dsn}
\lor (\text{the (dhops (rt (\sigma \ sip)) dip) \leq hops}
\lor \text{the (flag (rt (\sigma \ sip)) dip) = inv}))"

466
by auto
from this(2) show "?thesis"
proof
  assume "sip = dip" with \( \langle dsn \geq 1 \rangle \) show ?thesis by simp
next
  assume "\( \text{dip} \in \text{kD}(\text{rt}(\sigma \text{sip})) \) \( \land \) \( dsn \leq \text{nsqn}(\text{rt}(\sigma \text{sip})) \) \( \text{dip} \) \\
  \land (\text{nsqn}(\text{rt}(\sigma \text{sip})) \text{dip} = \text{dsn} \\
  \rightarrow (\text{the}(\text{dhops}(\text{rt}(\sigma \text{sip})) \text{dip}) \leq \text{hops} \\
  \lor \text{the}(\text{flag}(\text{rt}(\sigma \text{sip})) \text{dip}) = \text{inv})")

  moreover from qinc have "quality_increases(\sigma \text{sip})(\sigma' \text{sip})"
ultimately have "\( \text{dip} \in \text{kD}(\text{rt}(\sigma' \text{sip})) \) \( \land \) \( dsn \leq \text{nsqn}(\text{rt}(\sigma' \text{sip})) \) \( \text{dip} \) \\
  \land (\text{nsqn}(\text{rt}(\sigma' \text{sip})) \text{dip} = \text{dsn} \\
  \rightarrow (\text{the}(\text{dhops}(\text{rt}(\sigma' \text{sip})) \text{dip}) \leq \text{hops} \\
  \lor \text{the}(\text{flag}(\text{rt}(\sigma' \text{sip})) \text{dip}) = \text{inv})")

  using \( \langle dsn \geq 1 \rangle \) by (rule quality_increases_rreq_rrep_props [rotated 2])
with \( \langle dsn \geq 1 \rangle \) show "msg_fresh \sigma' m"
by clarsimp
qed
next
fix dests sip
  assume \[\text{simp}\]: "m = \text{Rerr} dests sip"
  and "msg_fresh \sigma m"
then have *: "\( \forall \text{rip} \in \text{dom(dests)}. \text{rip} \in \text{kD}(\text{rt}(\sigma \text{sip})) \)
  \land (\text{the}(\text{dests}\text{rip}) - 1 \leq \text{nsqn}(\text{rt}(\sigma \text{sip})) \text{rip})"
by simp
have "\( \forall \text{rip} \in \text{dom(dests)}. \text{rip} \in \text{kD}(\text{rt}(\sigma' \text{sip})) \)
  \land (\text{the}(\text{dests}\text{rip}) - 1 \leq \text{nsqn}(\text{rt}(\sigma' \text{sip})) \text{rip})"
proof
  fix rip
  assume "\text{rip} \in \text{dom(dests)}"
  with * have "\( \text{rip} \in \text{kD}(\text{rt}(\sigma \text{sip})) \)"
  and "the(\text{dests}\text{rip}) - 1 \leq \text{nsqn}(\text{rt}(\sigma \text{sip})) \text{rip}" \\
  by \(-\) (drule(1) bspec, clarsimp)+
moreover from qinc have "quality_increases(\sigma \text{sip})(\sigma' \text{sip})" by simp
ultimately show "\( \text{rip} \in \text{kD}(\text{rt}(\sigma' \text{sip})) \) \( \land \) (\text{the}(\text{dests}\text{rip}) - 1 \leq \text{nsqn}(\text{rt}(\sigma' \text{sip})) \text{rip})" ..
qed
thus ?thesis by simp
qed simp_all
end

5.8 The 'open' AODV model

theory E_OAodv
imports E_Aodv ../../../AWN/OAWN_SOS_Labels ../../../AWN/OAWN_Convert
begin

Definitions for stating and proving global network properties over individual processes.

definition \( \sigma_{AODV}' \) :: "((ip \Rightarrow state) \times ((state, msg, pseqp, pseqp label) seqp)) set"
where "\( \sigma_{AODV}' \equiv \{\lambda i. \text{aodv}_\text{init} i, \text{\Gamma}_{AODV} \text{PAodv}\}\)"

abbreviation opaodv :: "ip \Rightarrow ((ip \Rightarrow state) \times (state, msg, pseqp, pseqp label) seqp, msg seq_action) automaton"
where "\( \text{opaodv}\ i \equiv \{ \text{init} = \sigma_{AODV}', \text{trans} = \text{oseqp}\_sos \text{\Gamma}_{AODV} i \}\)"

lemma initiali_aodv [intro!, simp]: "\text{initiali} i (\text{init}(\text{opaodv} i)) (\text{init}(\text{paodv} i))"
unfolding \( \sigma_{AODV}\_\text{def} \) \( \sigma_{AODV}'\_\text{def} \) by rule simp_all

lemma oaodv_control_within [simp]: "\text{control_within} \text{\Gamma}_{AODV} (\text{init}(\text{opaodv} i))"
unfolding \( \sigma_{AODV}'\_\text{def} \) by (rule control_withinI) (auto simp del: \( \text{\Gamma}_{AODV}\_\text{sims} \))

lemma \( \sigma_{AODV}'\_\text{labels} \) [simp]: "(\sigma, p) \in \sigma_{AODV}' \Rightarrow \text{labels} \text{\Gamma}_{AODV} p = \{\text{PAodv}:-0\}"
unfolding \( \sigma_{AODV}'\_\text{def} \) by simp

467
lemma oaodv_init_kD_empty [simp]:
"(σ, p) ∈ σ AODV’ ⇒ kD (rt (σ i)) = {}"
unfolding σ AODV’_def kD_def by simp

lemma oaodv_init_vD_empty [simp]:
"(σ, p) ∈ σ AODV’ ⇒ vD (rt (σ i)) = {}"
unfolding σ AODV’_def vD_def by simp

lemma oaodv_trans: "trans (opaodv i) = oseqp_sos Γ AODV i"
by simp

declare oseq_invariant_ctermsI [OF aodv_wf oaodv_control_within aodv_simple_labels oaodv_trans, cterms_intros]
oseq_step_invariant_ctermsI [OF aodv_wf oaodv_control_within aodv_simple_labels oaodv_trans, cterms_intros]
end

5.9 Global invariant proofs over sequential processes

theory E_Global_Invariants
imports E_Seq_Invariants
E_Aodv_Predicates
E_Fresher
E_Quality_Increases
"../../../AWN/OAWN_Convert"
E_OAodv
begin

lemma other_quality_increases [elim]:
assumes other quality_increases I σ σ’
shows ∀ j. quality_increases (σ j) (σ’ j)
using assms by (rule, clarsimp) (metis quality_increases_refl)

lemma weaken_otherwith [elim]:
fixes m
assumes *: "otherwith P I (orecvmsg Q) σ σ’ a"
and weakenP: "∀ σ m. P σ m ⇒ P’ σ m"
and weakenQ: "∀ σ m. Q σ m ⇒ Q’ σ m"
shows "otherwith P’ I (orecvmsg Q’ ) σ σ’ a"
proof
fix j
assume j /∈ I
with * have "P (σ j) (σ’ j)" by auto
thus "P’ (σ j) (σ’ j)" by (rule weakenP)
next
from * have "orecvmsg Q σ a" by auto
thus "orecvmsg Q’ σ a" by rule (erule weakenQ)
qed

lemma oreceived_msg_inv:
assumes other: "∀ σ σ’ m. [ P σ m; other Q {i} σ σ’ ] ⇒ P σ’ m"
and local: "∀ σ m. P σ m ⇒ P (σ (i := σ i (msg := m))) m"
shows "opaodv i |= (otherwith Q {i} (orecvmsg P), other Q {i} →)
    onl Γ AODV (λ(σ, l). l ∈ {PAodv:-1} → P σ (msg (σ i))))"
proof (inv_cterms, intro impl)
fix σ σ’ l
assume "l = PAodv:-1 → P σ (msg (σ i))"
and "l = PAodv:-1" and "other Q {i} σ σ’"
from this(1-2) have "P σ (msg (σ i))" .
hence "P σ (msg (σ i))" using ⟨other Q {i} σ σ’⟩
    by (rule other)
moreover from ⟨other Q {i} σ σ’⟩ have "σ’ i = σ i" ..
ultimately show "P σ' (msg (σ' i))" by simp
next
fix σ σ' msg
assume "otherwith Q {i} (orecvmsg P) σ σ' (receive msg)"
and "σ' i = σ i[msg := msg]"
from this(1) have "P σ msg"
and "∀ j. j ⩾ i σ σ' (msg (σ' j))" by auto
from this(1) have "P (σ i := σ i[msg := msg]) msg" by (rule local)
thus "P σ' msg"
proof (rule other)
from ⟨σ' i = σ i[msg := msg]⟩ and ⟨∀ j. j ⩾ i σ σ' (msg (σ' j))⟩
show "other Q {i} (σ i := σ i[msg := msg]) σ'" by (rule otherI, auto)
qed

(Equivalent to) Proposition 7.27

lemma local_quality_increases:
"paodv i |=A (orecvmsg rreq_rrep_sn → onll Γ AODV (λ((ξ, _), _, (ξ', _)), quality_increases ξ ξ'))"
proof (rule step_invariantI)
fix s a s'
assume sr: "s ∈ reachable (paodv i) (orecvmsg rreq_rrep_sn)"
and tr: "(s, a, s') ∈ trans (paodv i)"
and rm: "orecvmsg rreq_rrep_sn a"
from sr have srTT: "s ∈ reachable (paodv i) TT" ..
from route_tables_fresher sr tr rm
have "onll Γ AODV (λ((ξ, _), _, (ξ', _)). ∀ dip ∈ kD (rt ξ). rt ξ ⊑ dip rt ξ') (s, a, s')"
by (rule step_invariantD)
moreover from known_destinations_increase srTT tr TT_True
have "onll Γ AODV (λ((ξ, _), _, (ξ', _)). kD (rt ξ) ⊆ kD (rt ξ')) (s, a, s')"
by (rule step_invariantD)
moreover from sqns_increase srTT tr TT_True
have "onll Γ AODV (λ((ξ, _), _, (ξ', _)). ∀ ip. sqn (rt ξ) ≤ sqn (rt ξ')) (s, a, s')"
by (rule step_invariantD)
ultimately show "onll Γ AODV (λ((ξ, _), _, (ξ', _)). quality_increases ξ ξ') (s, a, s')"
unfolding onll_def by auto
qed

lemmas olocal_quality_increases =
open_seq_step_invariant [OF local_quality_increases initiali_aodv oaodv_trans aodv_trans, simplified seqll_onll_swap]

lemma quality_increases:
"opaodv i |=A (otherwith quality_increases {i} (orecvmsg (λ_. rreq_rrep_sn)),
other_quality_increases {i} →) onll Γ AODV (λ((σ, _), _, (σ', _)). ∀ j. quality_increases (σ j) (σ' j))"
(is "_ |=A (?S, _ → _) _")
proof (rule onll_ostep_invariantI, simp)
fix σ p 1 a σ' p' 1'
assume or: "("σ, p) ∈ reachable (opaodv i) ?S (other quality_increases {i})"
and 11: "1 ∈ labels Γ AODV p"
and "?S σ σ' a"
and tr: "("σ, p), a, (σ', p') ∈ oseqp_sos Γ AODV i"
and 11': "1' ∈ labels Γ AODV p'"
from this(1-3) have "orecvmsg (λ_. rreq_rrep_sn) σ a" by (auto dest!: reachable_weakenE [where QS="act (orecvmsg rreq_rrep_sn)"
and QU="other quality_increases {i}"], otherwith_actionD)
with or have orw: "("σ, p) ∈ reachable (opaodv i) (act (orecvmsg rreq_rrep_sn))
(other quality_increases {i})"
lemma rreq_rrep_snq_fresh_any_step_invariant:
"opaodv i \models A (act (recvmsg rreq_rrep_sn), other A {i} →)
onll Γ_AODV (λ((ξ, _), a, _). anycast (rreq_rrep_fresh (rt ξ)) a)"
proof (rule ostep_invariantI, simp del: act_simp)
  fix σ p a σ' p'
  assume or: "((σ, p), a, (σ', p')) ∈ trans (opaodv i)"
  have tr: "((σ, p), a, (σ', p')) ∈ trans (opaodv i)"
  thus ?thesis
qed

moreover have "anycast (rreq_rrep_fresh (rt (σ i))) a"
proof -
  have "opaodv i \models A (act (recvmsg rreq_rrep_sn), other A {i} →)
onll Γ_AODV (seqll i (λ((ξ, _), a, _). anycast rreq_rrep_sn a))"
  by (rule ostep_invariantI, simp del: act_simp)
  have tr: "((σ, p), a, (σ', p')) ∈ trans (opaodv i)"
  thus ?thesis
qed

moreover have "anycast (rerr_invalid (rt (σ i))) a"
proof -
  have "opaodv i \models A (act (recvmsg rreq_rrep_sn), other A {i} →)
onll Γ_AODV (seqll i (λ((ξ, _), a, _). anycast rerr_invalid (rt ξ)) a)"
  by (rule ostep_invariantI, simp del: act_simp)
  have tr: "((σ, p), a, (σ', p')) ∈ trans (opaodv i)"
  thus ?thesis
qed

moreover have "anycast rreq_rrep_sn a"
proof -
  from tr recv
  have "anycast rreq_rrep_sn a"
  thus ?thesis
qed

moreover have "anycast (λm. not_Pkt m → msg_sender m = i) a"
proof -
  have "opaodv i \models A (act (recvmsg rreq_rrep_sn), other A {i} →)
onll Γ_AODV (seqll i (λ((ξ, _), a, _). anycast rreq_rrep_sn a))"
  by (rule ostep_invariantI, simp del: act_simp)
  have tr: "((σ, p), a, (σ', p')) ∈ trans (opaodv i)"
  thus ?thesis
qed
by (rule ostep_invariant_weakenE [OF
  open_seq_step_invariant [OF sender_ip_valid initiali_aodv,
simplified seqll_onll_swap]]) auto
thus ?thesis using or tr recv (1∈labels Γ\textsubscript{\textsc{aodv}} p) and (1'∈labels Γ\textsubscript{\textsc{aodv}} p')
by - (drule(3) onll_ostep_invariantD, auto)

qed

ultimately have "anycast (msg\_fresh σ) a"
by (simp_all add: anycast_def
del: msg\_fresh
  split: seq\_action.split\_asm msg\_split\_asm) simp_all
thus "onll Γ\textsubscript{\textsc{aodv}} (λ((σ, _), a, _). anycast (msg\_fresh σ) a) (((σ, p), a, (σ', p')))"
by auto

qed

lemma oreceived_rreq_rrep_nsqn_fresh_inv:
"opaodv i \models (otherwith quality\_increases {i} (orecmsg msg\_fresh),
other quality\_increases {i} →)
onl Γ\textsubscript{\textsc{aodv}} (λ(σ, l). l ∈ {PAodv-:1} → msg\_fresh σ (msg (σ i)))"
proof (rule oreceived_msg_inv)
fix σ σ'
assume *: "msg\_fresh σ m" and "other quality\_increases {i} σ σ'"
from this(2) have "∀ j. quality\_increases (σ j) (σ' j)"..
thus "msg\_fresh σ' m" using * ..

next
fix σ m
assume "msg\_fresh σ m"
thus "msg\_fresh (σ(i := σ i∥msg := m)) m"
proof (cases m)
fix dests sip
assume "m = Rerr dests sip"
with ⟨msg\_fresh σ m⟩ show ?thesis by auto

qed auto

lemma oquality\_increases_nsqn\_fresh:
"opaodv i \models (otherwith quality\_increases {i} (orecmsg msg\_fresh),
other quality\_increases {i} →)
onl Γ\textsubscript{\textsc{aodv}} (λ((σ, l), _, (σ', _)). ∀ j. quality\_increases (σ j) (σ' j))"
by (rule ostep\_invariant\_weakenE [OF oquality\_increases]) auto

lemma oosn\_rreq:
"opaodv i \models (otherwith quality\_increases {i} (orecmsg msg\_fresh),
other quality\_increases {i} →)
onl Γ\textsubscript{\textsc{aodv}} (seql i (λ(ξ, l). l ∈ {PAodv-:4, PAodv-:5} ∪ {PRreq-:n |n. True} → 1 ≤ osn ξ))"
by (rule oinvariant\_weakenE [OF open_seq\_invariant [OF osn\_rreq initiali_aodv]])
(auto simp: seql\_onl\_swap)

lemma rreq\_sip:
"opaodv i \models (otherwith quality\_increases {i} (orecmsg msg\_fresh),
other quality\_increases {i} →)
onl Γ\textsubscript{\textsc{aodv}} (λ(σ, l). l ∈ {PAodv-:4, PAodv-:5, PRreq-:0, PRreq-:2} ∧ sip (σ i) ≠ oip (σ i)) →
oip (σ i) ∈ kD(rt (σ (sip (σ i)))) ∧ nsqn (rt (σ (sip (σ i)))) (oip (σ i)) ≥ osn (σ i)
∧ (nsqn (rt (σ (sip (σ i)))) (oip (σ i)) = osn (σ i) 
  → (hops (σ i) ≥ the (dhops (rt (σ (sip (σ i)))) (oip (σ i))) 
       ∨ the (flag (rt (σ (sip (σ i)))) (oip (σ i))) = inv)))"
(is "_/ = (?S, ?U → ) _")
proof (inv\_cterms inv add: oseq\_step\_invariant\_sterms [OF oquality\_increases\_nsqn\_fresh
  aodv\_wf aodv\_trans]
onl\_oinvariant\_sterms [OF aodv\_wf oreceived\_rreq\_rrep\_nsqn\_fresh\_inv]
onl\_oinvariant\_sterms [OF aodv\_wf oosn\_rreq]

471
simp add: seqlsimp
simp del: One_nat_def, rule impI)

fix σ σ' p l
assume "(σ, p) ∈ ooreachable (opaodv i) ?S ?U"
and "l ∈ labels Γ_AODV p"
and pre:

"(l = PAodv-:4 ∨ l = PAodv-:5 ∨ l = PRreq-:0 ∨ l = PRreq-:2) ∧ sip (σ i) ≠ oip (σ i)
  → oip (σ i) ∈ kD (rt (σ (sip (σ i)))) "
∧ osn (σ i) ≤ nsqn (rt (σ (sip (σ i)))) (oip (σ i))
∧ (nsqn (rt (σ (sip (σ i)))) (oip (σ i))) = osn (σ i)
  → the (dhops (rt (σ (sip (σ i)))) (oip (σ i))) ≤ hops (σ i)
  ∧ the (flag (rt (σ (sip (σ i)))) (oip (σ i))) = inv)"

and "other quality_increases {i} σ σ'"
and hyp: "(l=PAodv-:4 ∨ l=PAodv-:5 ∨ l=PRreq-:0 ∨ l=PRreq-:2) ∧ sip (σ i) ≠ oip (σ i)"

from this(4) have "σ' i = σ i" ..
with hyp have hyp': "?labels ∧ sip (σ i) ≠ oip (σ i)" by simp

show "oip (σ i) ∈ kD (rt (σ' (sip (σ' i)))) "
∧ osn (σ' i) ≤ nsqn (rt (σ' (sip (σ' i)))) (oip (σ' i))
∧ (nsqn (rt (σ' (sip (σ' i)))) (oip (σ' i))) = osn (σ' i)
  → the (dhops (rt (σ' (sip (σ' i)))) (oip (σ' i))) ≤ hops (σ' i)
  ∧ the (flag (rt (σ' (sip (σ' i)))) (oip (σ' i))) = inv)"

proof (cases "sip (σ i) = i")
assume "sip (σ i) ≠ i"

from "other quality_increases {i} σ σ'"
  have "quality_increases (σ (sip (σ i))) (σ' (sip (σ' i)))"
    by (rule otherE) (clarsimp simp: seqlsimp)

moreover from "(σ, p) ∈ ooreachable (opaodv i) ?S ?U" l ∈ labels Γ_AODV p and hyp
have "1 ≤ osn (σ' i)"

by (auto dest!: onl_oinvvariant_weakenD [OF oosn_rreq]
  simp add: seqlsimp σ' i = σ i)

moreover from sip (σ i) ≠ i hyp' and pre
have "oip (σ i) ∈ kD (rt (σ (sip (σ i)))) "
∧ osn (σ i) ≤ nsqn (rt (σ (sip (σ i)))) (oip (σ i))
∧ (nsqn (rt (σ (sip (σ i)))) (oip (σ i))) = osn (σ i)
  → the (dhops (rt (σ (sip (σ i)))) (oip (σ i))) ≤ hops (σ i)
  ∧ the (flag (rt (σ (sip (σ i)))) (oip (σ i))) = inv)

by (auto simp: σ' i = σ i)

ultimately show ?thesis

by (rule quality_increases_rreq_rrep_props)

next
assume "sip (σ i) = i" thus ?thesis

using σ' i = σ i hyp and pre by auto

qed

(auto elim!: quality_increases_rreq_rrep_props')

lemma odsn_rrep:

"opaodv i |=(otherwith quality_increases {i} (orecvmsg msg_fresh),
 other_quality_increases {i}→)

onl Γ_AODV (seql i (λ(ξ, 1). l ∈ {PAodv-:6, PAodv-:7} ∪ {PRrep-:n|n. True} → 1 ≤ dsn ξ))"

by (rule oinvvariant_weakenE [OF open_seq_invariant [OF dsn_rrep initiali_aodv]])
(auto simp: seql_onl_swap)

lemma rrep_sip:

"opaodv i |=(otherwith quality_increases {i} (orecvmsg msg_fresh),
 other_quality_increases {i}→)

onl Γ_AODV (λ(σ, i).

(l ∈ {PAodv-:6, PAodv-:7, PRrep-:0, PRrep-:1} ∧ sip (σ i) ≠ dip (σ i))
  → dip (σ i) ∈ kD(rt (σ (sip (σ i))))
∧ nsqn (rt (σ (sip (σ i)))) (dip (σ i)) ≥ dsn (σ i)
∧ (nsqn (rt (σ (sip (σ i)))) (dip (σ i))) = dsn (σ i)
  → (hops (σ i) ≥ the (dhops (rt (σ (sip (σ i)))) (dip (σ i)))
  → the (flag (rt (σ (sip (σ i)))) (dip (σ i))) = inv))"

(is "|=(?S, ?U →) _")

472
proof (inv_terms inv add: oseq_step_invariant_terms [OF equality_increases_nsqn_fresh aodv_wf]
oadv_trans)
onl_onoinvariant_terms [OF aodv_wf oeredd1 rreq_rrep_nsqn_fresh_inv]
onl_onoinvariant_terms [OF aodv_wf odsn_rrep]
simp del: One_nat_def, rule mpi)

fix σ σ' p l
assume "("σ, p) ∈ reachable (opaodv i) ?S ?U"
and "1 ∈ labels Γ_{AODV} p"
and pre:
((1 = PAodv-:6 ∨ 1 = PAodv-:7 ∨ 1 = PRrep-:0 ∨ 1 = PRrep-:1) ∧ σ (i) ≠ dip (σ i))
→ dip (σ i) ∈ kD (rt (σ (sip σ i))))
∧ dsn (σ i) ≤ nsqn (rt (σ (sip σ i)))) (dip (σ i))
∧ (nsqn (rt (σ (sip σ i)))) (dip (σ i)) = dsn (σ i)
→ the (dhops (rt (σ (sip σ i)))) (dip (σ i)) ≤ hops (σ i)
∧ the (flag (rt (σ (sip σ i)))) (dip (σ i)) = inv"

and "other_quality_increases {i} σ σ'"
and hyp: "(1=PAodv-:6 ∨ 1=PAodv-:7 ∨ 1=PRrep-:0 ∨ 1=PRrep-:1) ∧ σ (i) ≠ dip (σ i)"
(is "?labels ∧ σ (sip σ i) ≠ dip (σ i)"
from this(4) have "σ' i = σ i" ..
with hyp have hyp': "?labels ∧ σ (sip σ i) ≠ dip (σ i)" by simp
show "dip (σ i) ∈ kD (rt (σ (sip σ i))))
∧ dsn (σ i) ≤ nsqn (rt (σ (sip σ i)))) (dip (σ i))
∧ (nsqn (rt (σ (sip σ i)))) (dip (σ i)) = dsn (σ i)
→ the (dhops (rt (σ (sip σ i)))) (dip (σ i)) ≤ hops (σ i)
∧ the (flag (rt (σ (sip σ i)))) (dip (σ i)) = inv"

proof (cases "σ (i) = i")
assume "σ (i) ≠ i"
from "other_quality_increases {i} σ σ'"
have "quality_increases (σ (sip σ i)) (σ' (sip σ i))"
by (rule otherE) (clarsimp simp: (σ (sip σ i)) ≠ i)
moreover from ⟨(σ, p) ∈ reachable (opaodv i) ?S ?U | l ∈ labels Γ_{AODV} p⟩ and hyp
have "1 ≤ dsn (σ' i)"
by (auto dest!: onl_onoinvariant_weakenD [OF odsn_rrep]
simp add: seqlsimp (σ' i = σ i))
moreover from "σ (i) ≠ i" hyp' and pre
have "dip (σ i) ∈ kD (rt (σ (sip σ i)))
∧ dsn (σ i) ≤ nsqn (rt (σ (sip σ i))) (dip (σ i))
∧ (nsqn (rt (σ (sip σ i))) (dip (σ i)) = dsn (σ i)
→ the (dhops (rt (σ (sip σ i))) (dip (σ i)) ≤ hops (σ i)
∧ the (flag (rt (σ (sip σ i)))) (dip (σ i)) = inv"
by (auto simp: (σ' i = σ i))
ultimately show ?thesis
by (rule quality_increases_rreq_rrep_props)
next
assume "σ (i) = i" thus ?thesis
using (σ' i = σ i) hyp and pre by auto
qed (auto simp add: seqlsimp elim!: quality_increases_rreq_rrep_props')

lemma rerr_rsip:
"opaodv i |= (otherquality_increases {i} (orecvmsg msg_fresh),
other_quality_increases {i} →)
onl Γ_{AODV} (λ(σ, i).
1 ∈ {PAodv-:8, PAodv-:9, PRerr-:0, PRerr-:1}
→ (∀ ripc∈dom(dests (σ i)). ripc∈kD (rt (σ (sip σ i)))) ∧
the (dests (σ i) ripc) - 1 ≤ nsqn (rt (σ (sip σ i))) ripc))"
(is _ |= (?S, ?U → _)"
proof -
{ have "dests rip sip ran and σ σ' :: "ip ⇒ state"
assume qinc: "∀ j. quality_increases (σ j) (σ' j)"
and *: "∀ rip∈dom dests. rip ∈ kD (rt (σ sip))
∧ the (dests rip) - 1 ≤ nsqn (rt (σ sip)) rip"
and "dests rip = Some rsn"
from this(3) have "rip∈dom dests" by auto

with \* and \langle \text{dests rip} = \text{Some rsn} \rangle \text{ have } "\text{rip} \in kD(\text{rt (sip)})"
and "\text{rsn} - 1 \leq \text{nsqn (rt (sip)) rip}"

by (auto dest!: bspec)
from qinc have "\text{quality_increases (sip) (s' sip)}" ..
have "\text{rip} \in kD(\text{rt (s' sip)}) \land \text{rsn} - 1 \leq \text{nsqn (rt (s' sip)) rip}"

proof
from \langle \text{rip} \in kD(\text{rt (s' sip)}) \rangle \text{ and } \langle \text{quality_increases (sip) (s' sip)} \rangle
show "\text{rip} \in kD(\text{rt (s' sip)})" ..
next
from \langle \text{rip} \in kD(\text{rt (s' sip)}) \rangle \text{ and } \langle \text{quality_increases (sip) (s' sip)} \rangle
have "\text{nsqn (rt (s' sip)) rip} \leq \text{nsqn (rt (s' sip)) rip}" ..
with \langle \text{rsn} - 1 \leq \text{nsqn (rt (s' sip)) rip} \rangle \text{ show } "\text{rsn} - 1 \leq \text{nsqn (rt (s' sip)) rip}"
by (rule le_trans)

qed

\} note partial = \text{this}

show \text {?thesis}
by (inv_cterms inv add: oseq_step_invariant_sterms [OF \text{quality_increases nsqn fresh aodv wf oadv_trans}]
onl.invariant_sterms [OF \text{oadv wf oreceived_rreq_rrep nsqn fresh_inv}]
\text{other_quality_increases other_localD simp del: One_nat_def, intro conjI})
(clarsimp simp del: One_nat_def split: split_if_asm option.split_asm, erule(2) partial)+

qed

lemma prerr.guard: "\text{paodv i \|= onl }\Gamma_{\text{AODV}} (\lambda (\xi, l). (l = \text{PRerr-:1} \rightarrow (\forall \text{ip} \in \text{dom(dests }\xi). \text{ip} \in \text{vD(\text{rt }\sigma) )})
\land \text{the (nhop (rt }\xi) \text{ ip) = sip } \xi \land \text{sqn (rt }\sigma \text{ ip) < the (dests }\xi \text{ ip})))"
by (inv_cterms) (clarsimp split: option.split_asm split_if_asm)

lemmas odests_vD_inc_sqn =
open_seq_invariant [OF \text{dests_vD_inc_sqn initiali_aodv oadv_trans aodv_trans},
simplified seql_onl_swap, THEN oinvariant_anyact]

lemmas oprerr.guard =
open_seq_invariant [OF \text{prerr_guard initiali_aodv oadv_trans aodv_trans},
simplified seql_onl_swap, THEN oinvariant_anyact]

Proposition 7.28

lemma seq_compare_next_hop':
"\text{opaodv i \|= (otherwith quality_increases {i} (orecvmsg msg_fresh)}, other_quality_increases {i} \rightarrow onl \Gamma_{\text{AODV}} (\lambda (\sigma, _). \forall \text{dip}. \text{let nhip = the (nhop (rt }\sigma \text{ dip})})
in dip \in kD(\text{rt (sip)}) \land nhip \neq dip \rightarrow
dip \in kD(\text{rt (s' nhip)}) \land \text{nsqn (rt (s ip)) dip} \leq \text{nsqn (rt (s' nhip)) dip}"
(is "\text{\_ \|= (?S, ?U \rightarrow \_)}")
proof -

{ fix nhip and \sigma \sigma' :: "ip \Rightarrow \text{state}"
assume \text{pre}:
\forall dip \in kD(\text{rt (s' i)}). \text{nhop dip} \neq dip \rightarrow dip \in kD(\text{rt (s (nhop dip))}) \land \text{nsqn (rt (s i)) dip} \leq \text{nsqn (rt (s (nhop dip)) dip}"
and qinc: \forall j. \text{quality_increases (s j) (s' j)}"
have \forall dip \in kD(\text{rt (s' i)}). \text{nhop dip} \neq dip \rightarrow dip \in kD(\text{rt (s' (nhop dip))}) \land \text{nsqn (rt (s i)) dip} \leq \text{nsqn (rt (s' (nhop dip)) dip}"
proof (intro ballI impI)
fix dip
assume "\text{dip} \in kD(\text{rt (s i)})"
and "\text{nhop dip} \neq dip"
with \text{pre} have "\text{dip} \in kD(\text{rt (s (nhop dip))})"
and "\text{nsqn (rt (s i)) dip} \leq \text{nsqn (rt (s (nhop dip)) dip}"
by auto
from qinc have qinc_nhop: "quality_increases (σ ⟨nhop dip⟩) (σ' ⟨nhop dip⟩)" ..
with ⟨dip∈kD(rt (σ ⟨nhop dip⟩))⟩ have "dip∈kD (rt (σ' ⟨nhop dip⟩))" ..

moreover have "nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' ⟨nhop dip⟩)) dip"
proof -
from ⟨dip∈kD(rt (σ ⟨nhop dip⟩))⟩ qinc_nhop
have "nsqn (rt (σ ⟨nhop dip⟩)) dip ≤ nsqn (rt (σ' ⟨nhop dip⟩)) dip" ..
with nsqn (rt (σ i)) dip ≤ nsqn (rt (σ ⟨nhop dip⟩)) dip show ?thesis
by simp
qed

ultimately show "dip∈kD(rt (σ' ⟨nhop dip⟩))
∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' ⟨nhop dip⟩)) dip"..

qed

} note basic = this

{ fix nhop and σ σ' :: "ip ⇒ state"
assume pre: "∀ dip∈kD(rt (σ i)). nhop dip ≠ dip → dip∈kD(rt (σ (nhop dip)))
∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (nhop dip))) dip"
and ndest: "∀ ripc∈dom (dests (σ i)). ripc ∈ kD (rt (σ (sip (σ i))))
∧ the (dests (σ i) ripc) - 1 ≤ nsqn (rt (σ (sip (σ i)))) ripc"
and issip: "∀ ip∈dom (dests (σ i)). nhop ip = sip (σ i)"
and qinc: "∀ j. quality_increases (σ j) (σ' j)"
have "∀ dip∈kD(rt (σ i)). nhop dip ≠ dip → dip ∈ kD (rt (σ' ⟨nhop dip⟩))
∧ nsqn (invalidate (rt (σ i)) (dests (σ i))) dip ≤ nsqn (rt (σ' ⟨nhop dip⟩)) dip"
proof (intro ballI impI)
fix dip
assume "dip∈kD(rt (σ i))"
and "nhop dip ≠ dip"
with pre and qinc have "dip∈kD(rt (σ' ⟨nhop dip⟩))"
and "nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' ⟨nhop dip⟩)) dip"
by (auto dest!: basic)

have "nsqn (invalidate (rt (σ i)) (dests (σ i))) dip ≤ nsqn (rt (σ' ⟨nhop dip⟩)) dip"
proof (cases "dip∈dom (dests (σ i))")
assume "dip∈dom (dests (σ i))"
with ⟨dip∈kD(rt (σ i))⟩ obtain dsn where "dests (σ i) dip = Some dsn"
by auto
with ⟨dip∈kD(rt (σ i))⟩ have "nsqn (invalidate (rt (σ i)) (dests (σ i))) dip = dsn - 1"
by (rule nsqn_invalidate_eq)
moreover have "dsn - 1 ≤ nsqn (rt (σ' ⟨nhop dip⟩)) dip"
proof -
from ⟨dests (σ i) dip = Some dsn⟩ have "the (dests (σ i) dip) = dsn" by simp
with ndest and ⟨dip∈dom (dests (σ i))⟩ have "dip ∈ kD (rt (σ (sip (σ i))))"
"dsn - 1 ≤ nsqn (rt (σ (sip (σ i)))) dip"
by auto
moreover from issip and ⟨dip∈dom (dests (σ i))⟩ have "nhop dip = sip (σ i)" ..
ultimately have "dip ∈ kD (rt (σ (nhop dip)))"
and "dsn - 1 ≤ nsqn (rt (σ (nhop dip))) dip" by auto
with qinc show "dsn - 1 ≤ nsqn (rt (σ' ⟨nhop dip⟩)) dip"
by simp (metis kD_nsqn_quality_increases_trans)
qed
ultimately show ?thesis by simp
next
assume "dip ∈ dom (dests (σ i))"
with ⟨dip∈kD(rt (σ i))⟩ have "nsqn (invalidate (rt (σ i)) (dests (σ i))) dip = nsqn (rt (σ i)) dip"
by (rule nsqn_invalidate_other)
with nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' ⟨nhop dip⟩)) dip show ?thesis by simp
qed
with ⟨dip∈kD(rt (σ' ⟨nhop dip⟩))⟩
show "dip ∈ kD (rt (σ' ⟨nhop dip⟩))
∧ nsqn (invalidate (rt (σ i)) (dests (σ i))) dip ≤ nsqn (rt (σ' ⟨nhop dip⟩)) dip" ..
qed
}

} note basic_perr = this

{ fix σ σ' :: "ip ⇒ state"
  assume a1: "∀ dip ∈ kD (rt (σ i)). the (nhop (rt (σ i)) dip) ≠ dip
      → dip ∈ kD (rt (σ (the (nhop (rt (σ i)) dip))))
      ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (the (nhop (rt (σ i)) dip)))) dip"
  and a2: "∀ j. quality_increases (σ j) (σ' j)"
  have "∀ dip ∈ kD (rt (σ i)).
      the (nhop (update (rt (σ i)) (sip (σ i)) (0, unk, val, Suc 0, sip (σ i))) dip) ≠ dip →
      dip ∈ kD (rt (σ' (the (nhop (update (rt (σ i)) (sip (σ i))
      (0, unk, val, Suc 0, sip (σ i)))) dip)))
      ∧ nsqn (update (rt (σ i)) (sip (σ i)) (0, unk, val, Suc 0, sip (σ i))) dip
      ≤ nsqn (rt (σ' (the (nhop (update (rt (σ i)) (sip (σ i))
      (0, unk, val, Suc 0, sip (σ i)))) dip))) (is "∀ dip ∈ kD (rt (σ i)). ?P dip")

proof
  fix dip
  assume "dip ∈ kD (rt (σ i))"
  with a1 and a2
  have "the (nhop (rt (σ i)) dip) ≠ dip → dip ∈ kD (rt (σ' (the (nhop (rt (σ i)) dip))))
      ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (the (nhop (rt (σ i)) dip)))) dip" (is "∀ dip ∈ kD (rt (σ i)). ?P dip")
    by (drule(1) basic, auto)
  thus "?P dip" by (cases "dip = sip (σ i)") auto
qed

} note nhop_update_sip = this

{ fix σ σ' :: "ip ⇒ state"
  assume a1: "∀ dip ∈ kD (rt (σ i)). the (nhop (rt (σ i)) dip) ≠ dip
      → dip ∈ kD (rt (σ (the (nhop (rt (σ i)) dip))))
      ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (the (nhop (rt (σ i)) dip)))) dip"
  and qinc: "∀ j. quality_increases (σ j) (σ' j)"
  and a2: "∀ dip ∈ kD (rt (σ sip))
      ∧ osn ≤ nsqn (rt (σ sip)) oip
      ∧ (nsqn (rt (σ sip)) oip = osn
      → the (dhops (rt (σ sip)) oip) ≤ hops
      ∨ the (flag (rt (σ sip)) oip) = inv)"

from pre and qinc

have pre': "∀ dip ∈ kD (rt (σ i)). the (nhop (rt (σ i)) dip) ≠ dip
      → dip ∈ kD (rt (σ' (the (nhop (rt (σ i)) dip))))
      ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (the (nhop (rt (σ i)) dip)))) dip" (is "∀ dip ∈ kD (rt (σ i)). ?P dip")
      by (rule basic)
  have "(the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip)) oip) ≠ oip
      → oip ∈ kD (rt (σ' (the (nhop (update (rt (σ i)) oip
      (osn, kno, val, Suc hops, sip)) oip))))
      ∧ nsqn (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip)) oip
      ≤ nsqn (rt (σ' (the (nhop (update (rt (σ i)) oip
      (osn, kno, val, Suc hops, sip)) oip)))) oip)"
      (is "?nhop_not_oip → ?oip_in_kD ∧ ?nsqn_le_nsqn")
    proof (rule, split update Rt, split_asm)
      assume "rt (σ i) = update (rt (σ i)) oip (osn, kno, val, Suc hops, sip)"
      and "the (nhop (rt (σ i)) oip) ≠ oip"
      with pre' show "?oip_in_kD ∧ ?nsqn_le_nsqn" by auto

next
  assume rtnot: "rt (σ i) ≠ update (rt (σ i)) oip (osn, kno, val, Suc hops, sip)"
  and notoip: "?nhop_not_oip
  with * qinc have ?oip_in_kD
    by (clarsimp elim!: kD_quality_increases)
  moreover with * pre qinc rtnot notoip have ?nsqn_le_nsqn
    by simp (metis kD nsqn_quality_increases_trans)
  ultimately show "?oip_in_kD ∧ ?nsqn_le_nsqn" ..
qed

476
\[
\begin{align*}
\text{fix } & \sigma, \sigma', \text{ oip, sip, osn, hops} \\
\text{assume pre: } & "\forall \text{ dip } \in kD(\text{ rt (}{\sigma}_i\text{))}. \text{ the (nhop (rt (}{\sigma}_i\text{)) dip) } \neq \text{ dip} \\
& \quad \rightarrow \text{ dip } \in kD(\text{ rt (}{\sigma}_i\text{)) dip) \bigwedge \text{ nsqn (rt (}{\sigma}_i\text{)) dip } \leq \text{ nsqn (rt (}{\sigma}_i\text{)) dip}) \text{ dip}" \\
\text{and qinc: } & "\forall j. \text{ quality_increases (}{\sigma}_j\text{)} (}{\sigma}_j'\text{)" \\
\text{and } & "\text{ sip } \neq \text{ oip} \rightarrow \text{ oip } \in kD(\text{ rt (}{\sigma}_i\text{)) dip) \bigwedge \text{ osn } \leq \text{ nsqn (rt (}{\sigma}_i\text{)) dip} \bigvee \text{ (flag (rt (}{\sigma}_i\text{)) dip) } = \text{ inv}" \\
\text{from pre and qinc have pre': } & "\forall \text{ dip } \in kD(\text{ rt (}{\sigma}_i\text{))}. \text{ the (nhop (rt (}{\sigma}_i\text{)) dip) } \neq \text{ dip} \\
& \quad \rightarrow \text{ dip } \in kD(\text{ rt (}{\sigma}_i\text{' (the (nhop (rt (}{\sigma}_i\text{)) dip))}) \bigwedge \text{ nsqn (rt (}{\sigma}_i\text{)) dip } \leq \text{ nsqn (rt (}{\sigma}_i\text{' (the (nhop (rt (}{\sigma}_i\text{)) dip))}) dip}" \\
\text{by (rule basic) have } & "\forall \text{ dip } \in kD(\text{ rt (}{\sigma}_i\text{))}. \text{ the (nhop (update (rt (}{\sigma}_i\text{)) oip (osn, kno, val, Suc hops, sip)) dip) } \neq \text{ dip} \\
& \quad \rightarrow \text{ dip } \in kD(\text{ rt (}{\sigma}_i\text{' (the (nhop (rt (}{\sigma}_i\text{)) dip))}) \bigwedge \text{ nsqn (update (rt (}{\sigma}_i\text{)) oip (osn, kno, val, Suc hops, sip)) dip } \leq \text{ nsqn (rt (}{\sigma}_i\text{' (the (nhop (rt (}{\sigma}_i\text{)) dip))}) dip}" \\
\text{is } & "\forall \text{ dip } \in kD(\text{ rt (}{\sigma}_i\text{) i}). _ \rightarrow \text{ dip } \in kD(\text{ rt (}{\sigma}_i\text{)) dip} \land \text{ ?nsqn_le_nsqn dip}\)" \\
\text{proof (intro ballI impI, split update_rt_split_asm) fix dip} \\
\text{assume "dip } \in kD(\text{ rt (}{\sigma}_i\text{))"} \\
\text{and "the (nhop (rt (}{\sigma}_i\text{)) dip) } \neq \text{ dip} \\
\text{and "rt (}{\sigma}_i\text{) } = \text{ update (rt (}{\sigma}_i\text{)) oip (osn, kno, val, Suc hops, sip)"} \\
\text{with pre' show "?dip } \in kD(\text{ rt (}{\sigma}_i\text{)) dip} \land \text{ ?nsqn_le_nsqn dip}" \text{ by simp next fix dip} \\
\text{assume "dip } \in kD(\text{ rt (}{\sigma}_i\text{))"} \\
\text{and notdip: "the (nhop (update (rt (}{\sigma}_i\text{)) oip (osn, kno, val, Suc hops, sip)) dip) } \neq \text{ dip} \\
\text{and "rt (}{\sigma}_i\text{) } \neq \text{ update (rt (}{\sigma}_i\text{)) oip (osn, kno, val, Suc hops, sip)"} \\
\text{show "?dip } \in kD(\text{ rt (}{\sigma}_i\text{)) dip} \land \text{ ?nsqn_le_nsqn dip}" \\
\text{proof (cases "dip } = \text{ oip")} \\
\text{assume "dip } \neq \text{ oip} \\
\text{with pre' (dip } \in kD(\text{ rt (}{\sigma}_i\text{)) notdip} \\
\text{show ?thesis by clarsimp next} \\
\text{assume "dip } = \text{ oip} \\
\text{with notrtnot qinc (dip } \in kD(\text{ rt (}{\sigma}_i\text{)) notdip *} \\
\text{have "?dip } \in kD(\text{ rt (}{\sigma}_i\text{)) dip"} \\
\text{by simp (metis kD_quality_increases) moreover from (dip } = \text{ oip) rtnot qinc (dip } \in kD(\text{ rt (}{\sigma}_i\text{)) notdip *} \\
\text{have "?nsqn_le_nsqn dip" \text{ by simp (metis kD_nsqn_quality_increases_trans) ultimately show ?thesis .. qed qed} \\
\text{ have "opaodv i } \mid (}{?S}, } \cup \rightarrow \text{ onl } \Gamma_{OADV} \lambda(\sigma, _). \forall \text{ dip } \in kD(\text{ rt (}{\sigma}_i\text{)}). \text{ the (nhop (rt (}{\sigma}_i\text{)) dip) } \neq \text{ dip} \\
\text{\quad \rightarrow \text{ dip } \in kD(\text{ rt (}{\sigma}_i\text{)) dip) \bigwedge \text{ nsqn (rt (}{\sigma}_i\text{)) dip } \leq \text{ nsqn (rt (}{\sigma}_i\text{)) dip}) \text{ dip}"
\end{align*}
\]
thus \( \text{thesis} \) unfolding \( \text{Let} \text{\_def} \) by auto

qed

Proposition 7.30

lemmas \( \text{okD\_unk\_or\_atleast\_one} \) =
  \( \text{open\_seq\_invariant} [\text{OF} kD\_unk\_or\_atleast\_one \text{initiali\_aodv}, \)
  \( \text{simplified seql\_onl\_swap}] \)

lemmas \( \text{ozero\_seq\_unk\_hops\_one} \) =
  \( \text{open\_seq\_invariant} [\text{OF} zero\_seq\_unk\_hops\_one \text{initiali\_aodv}, \)
  \( \text{simplified seql\_onl\_swap}] \)

lemma \( \text{oreachable\_fresh\_okD\_unk\_or\_atleast\_one}: \)
  fixes dip
  assumes "\((\sigma, p) \in \text{oreachable} (\text{opaodv} i) \)
  \((\text{otherwith (op=} \{i\} (\text{orecvmsg (} \lambda\sigma m. \text{msg\_fresh} \sigma m \)
  \land \text{msg\_zhops m}))\)
  \((\text{other quality\_increases \{i\})})" \(\land \) "\( \text{dip}\in kD(\text{rt (} \sigma i))\)"
  shows "\(\pi_3(\text{the (rt (} \sigma i) \text{ dip)}) = \text{unk} \lor 1 \leq \pi_2(\text{the (rt (} \sigma i) \text{ dip)})\)"
  (is "?P dip")
  proof -
  have "\(\exists l. l \in \text{labels } \Gamma_{AODV} p\)" by (metis aodv_ex_label)
  with assms(1) have "\(\forall \text{dip} \in kD(\text{rt (} \sigma i)). ?P dip\)"
  by - (drule oinvariant_weakenD [OF okD_unk_or_atleast_one [OF oaodv_trans aodv_trans]], \)
  auto dest!: otherwith_actionD onlD simp: seqlsimp)
  with \(\langle \text{dip}\in kD(\text{rt (} \sigma i)) \rangle\) show ?thesis by simp
  qed

lemma \( \text{oreachable\_fresh\_ozero\_seq\_unk\_hops\_one}: \)
  fixes dip
  assumes "\((\sigma, p) \in \text{oreachable} (\text{opaodv} i) \)
  \((\text{otherwith (op=} \{i\} (\text{orecvmsg (} \lambda\sigma m. \text{msg\_fresh} \sigma m \)
  \land \text{msg\_zhops m}))\)
  \((\text{other quality\_increases \{i\})})" \(\land \) "\( \text{dip}\in kD(\text{rt (} \sigma i))\)"
  shows "\(\text{sqn (rt (} \sigma i) \text{ dip}) = 0 \rightarrow \)
  \(\text{sqnf (rt (} \sigma i) \text{ dip}) = \text{unk} \land \text{the (dhops (rt (} \sigma i) \text{ dip)}) = 1} \)
  \(\land \text{the (nhop (rt (} \sigma i) \text{ dip)}) = \text{dip}\)"
  (is "?P dip")
  proof -
  have "\(\exists l. l \in \text{labels } \Gamma_{AODV} p\)" by (metis aodv_ex_label)
  with assms(1) have "\(\forall \text{dip} \in kD(\text{rt (} \sigma i)). ?P dip\)"
  by - (drule oinvariant_weakenD [OF ozero_seq_unk_hops_one [OF oaodv_trans aodv_trans]], \)
  auto dest!: onlD otherwith_actionD onlD simp: seqlsimp)
  with \(\langle \text{dip}\in kD(\text{rt (} \sigma i)) \rangle\) show ?thesis by simp
  qed

lemma \( \text{seq\_nhop\_quality\_increases}: \)
  shows "\(\text{opaodv} i \models (\text{otherwith (op=} \{i\} \)
  \((\text{orecvmsg (} \lambda\sigma m. \text{msg\_fresh} \sigma m \land \text{msg\_zhops m}))\), \)
  \((\text{other quality\_increases (i} \rightarrow) \)
  onl \(\Gamma_{AODV} (\lambda(\sigma, _). \forall \text{dip. let nhip = the (nhop (rt (} \sigma i) \text{ dip)}) \)
  \text{in dip} \in \text{vD (rt (} \sigma i)) \cap \text{vD (rt (} \sigma \text{nhip)}) \land \text{nhip} \neq \text{dip}\)"
(is "\( - | (?S i, _ \rightarrow ) _ - \)"

proof

- have weaken:
  "\( \land I R P. P | (\text{otherwith} \text{quality\_increases} I \text{orecmsg} Q, \text{other quality\_increases} I \rightarrow) P \)
  \( \Rightarrow p | (\text{otherwith (op=)} I \text{orecmsg} (\lambda \sigma m. Q \sigma m \land R \sigma m)), \text{other quality\_increases} I \rightarrow) P \)
  by auto

\{ 
  fix i a and \( \sigma \sigma' :: \"ip \Rightarrow \) state"
  assume a1: "\( \forall dip. dip \in VD(rt (\sigma i)) \)
  \( \land dip \in VD(rt (\sigma (the (nhop (rt (\sigma i)) dip)))) \)
  \( \land (the (nhop (rt (\sigma i)) dip)) \neq dip \)
  \( \Rightarrow rt (\sigma i) \sqsubseteq dip rt (\sigma (the (nhop (rt (\sigma i)) dip))))" 
  and ov: "\( ?S i \sigma \sigma' a \)"
  have "\( \forall dip. dip \in VD(rt (\sigma i)) \)
  \( \land dip \in VD(rt (\sigma' (the (nhop (rt (\sigma i)) dip)))) \)
  \( \land (the (nhop (rt (\sigma i)) dip)) \neq dip \)
  \( \Rightarrow rt (\sigma i) \sqsubseteq dip rt (\sigma' (the (nhop (rt (\sigma i)) dip))))" 

proof clarify

fix dip

assume a2: "\( dip \in VD(rt (\sigma i)) \)"

and a3: "\( dip \in VD(rt (\sigma' (the (nhop (rt (\sigma i)) dip)))) \)"

and a4: "\( (the (nhop (rt (\sigma i)) dip)) \neq dip \)"

from ov have "\( \forall j. j \neq i \rightarrow j = \sigma j \)" by auto

show "\( rt (\sigma i) \sqsubseteq dip rt (\sigma' (the (nhop (rt (\sigma i)) dip)))) \)"

proof (cases "\( (\text{the (nhop (rt (\sigma i)) dip)) = i) \)"

assume "\( (\text{the (nhop (rt (\sigma i)) dip)) = i) \)"

with \( dip \in VD(rt (\sigma i)) \) have "\( dip \in VD(rt (\sigma (the (nhop (rt (\sigma i)) dip)))) \)" by simp

with a1 a2 a4 have "\( rt (\sigma i) \sqsubseteq dip rt (\sigma' (the (nhop (rt (\sigma i)) dip)))) \)" by simp

with "\( (\text{the (nhop (rt (\sigma i)) dip)) = i) \) have "\( rt (\sigma i) \sqsubseteq dip rt (\sigma (the (nhop (rt (\sigma i)) dip)))) \)" by simp

hence False by simp

thus ?thesis ..

next

assume "\( (\text{the (nhop (rt (\sigma i)) dip)) \neq i) \)"

with "\( \forall j. j \neq i \rightarrow j = \sigma j \)"

have "\( \sigma (\text{the (nhop (rt (\sigma i)) dip)) = \sigma' (the (nhop (rt (\sigma i)) dip))) \)" by simp

with "\( dip \in VD(rt (\sigma' (the (nhop (rt (\sigma i)) dip)))) \)" by simp

with a1 a2 a4 have "\( rt (\sigma i) \sqsubseteq dip rt (\sigma (the (nhop (rt (\sigma i)) dip)))) \)" by simp

with * show ?thesis by simp

qed

qed

\} note basic = this

\{ 
  fix \( \sigma \sigma' \) a dip sip i
  assume a1: "\( \forall dip. dip \in VD(rt (\sigma i)) \)
  \( \land dip \in VD(rt (\sigma (the (nhop (rt (\sigma i)) dip)))) \)
  \( \land (the (nhop (rt (\sigma i)) dip)) \neq dip \)
  \( \Rightarrow rt (\sigma i) \sqsubseteq dip rt (\sigma (the (nhop (rt (\sigma i)) dip))))" 
  and ov: "\( ?S i \sigma \sigma' a \)"
  have "\( \forall dip. dip \in VD(update (rt (\sigma i)) sip (0, unk, val, Suc 0, sip)) \)
  \( \land dip \in VD(rt (\sigma' (the (nhop (update (rt (\sigma i)) sip (0, unk, val, Suc 0, sip)) dip)))) \)
  \( \land the (nhop (update (rt (\sigma i)) sip (0, unk, val, Suc 0, sip)) dip)) \neq dip \)
  \( \Rightarrow update (rt (\sigma i)) sip (0, unk, val, Suc 0, sip) \)
  \( \sqsubseteq dip rt (\sigma' (the (nhop (update (rt (\sigma i)) sip (0, unk, val, Suc 0, sip)) dip))))" 

proof clarify

fix dip

assume a2: "\( dip \in VD(update (rt (\sigma i)) sip (0, unk, val, Suc 0, sip)) \)"

and a3: "\( dip \in VD(rt (\sigma' (the (nhop (update (rt (\sigma i)) sip (0, unk, val, Suc 0, sip)) dip)))) \)"

and a4: "\( the (nhop (update (rt (\sigma i)) sip (0, unk, val, Suc 0, sip)) dip) \neq dip \)"

show "update (rt (\sigma i)) sip (0, unk, val, Suc 0, sip)
  \( \sqsubseteq dip rt (\sigma' (the (nhop (update (rt (\sigma i)) sip (0, unk, val, Suc 0, sip)) dip))))" 

proof (cases "\( dip = sip \)"

479
assume "dip = sip"
with ⟨the (nhop (update (rt (σ i)) sip (0, unk, val, Suc 0, sip)) dip) ≠ dip⟩
have False by simp
thus ?thesis ..
next
assume [simp]: "dip ≠ sip"
from a2 have "dip∈vD(rt (σ i)) ∨ dip = sip"
  by (rule vD_update_val)
with ⟨dip ≠ sip⟩ have "dip∈vD(rt (σ i))" by simp
moreover from a3 have "dip∈vD(rt (σ' (the (nhop (rt (σ i)) dip))))" by simp
moreover from a4 have "the (nhop (rt (σ i)) dip) ≠ dip" by simp
ultimately have "rt (σ i) ⊏ dip rt (σ' (the (nhop (rt (σ i)) dip)))"
  using a1 ow by - (drule(1) basic, simp)
with ⟨dip ≠ sip⟩ show ?thesis
  by - (erule rt_strictly_fresher_update_other, simp)
qed
qed
}

note update_0_unk = this

{
  fix σ a σ' nhop
  assume pre: "∀dip. dip∈vD(rt (σ i)) ∧ dip∈vD(rt (σ (nhop dip))) ∧ nhop dip ≠ dip
                        → rt (σ i) ⊏ dip rt (σ (nhop dip))"
  and ow: "?S i σ σ' a"
  have "∀dip. dip ∈ vD (invalidate (rt (σ i)) (dests (σ i)))
         ∧ dip ∈ vD (rt (σ' (nhop dip))) ∧ nhop dip ≠ dip
         → rt (σ i) ⊏ dip rt (σ' (nhop dip))"
proof clarify
  fix dip
  assume "dip∈vD(invalidate (rt (σ i)) (dests (σ i)))"
  and "dip∈vD(rt (σ' (nhop dip)))"
  and "nhop dest ≠ dip"
  from this(1) have "dip∈vD (rt (σ i))"
    by (clarsimp dest!: vD_not_dests)
  moreover from ow have "∀j. j ≠ i → σ j = σ' j" by auto
  ultimately have "rt (σ i) ⊏ dip rt (σ' (nhop dip))"
    using pre ⟨dip ∈ vD (rt (σ' (nhop dip))) ⟩ nhop dip ≠ dip
    by metis
  with ∀j. j ≠ i → σ j = σ' j show "rt (σ i) ⊏ dip rt (σ' (nhop dip))"
    by (metis rt_strictly_fresher_irefl)
qed
}

note invalidate = this

{
  fix σ a σ' dip oip osn sip hops i
  assume pre: "∀dip. dip ∈ vD (rt (σ i))
            ∧ dip ∈ vD (rt (σ (the (nhop (rt (σ i)) dip))))
            ∧ the (nhop (rt (σ i)) dip) ≠ dip
            → rt (σ i) ⊏ dip rt (σ (the (nhop (rt (σ i)) dip)))"
  and ow: "?S i σ σ' a"
  and "Suc 0 ≤ osn"
  and a6: "sip ≠ oip → oip ∈ kD (rt (σ sip))
            ∧ osn ≤ nsqn (rt (σ sip)) oip
            ∧ (nsqn (rt (σ sip)) oip = osn
                → the (dhops (rt (σ sip)) oip) ≤ hops
                → the (flag (rt (σ sip)) oip) = inv)"
  and after: "σ' i = σ i |rt := update (rt (σ i)) oip (osn, kno, val, Suc hops, sip)|"
  have "∀dip. dip ∈ vD (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip))
            ∧ dip ∈ vD (rt (σ' (the (nhop (update (rt (σ i)) oip
                                    (osn, kno, val, Suc hops, sip)) dip))))
            ∧ the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip)) dip) ≠ dip
            → update (rt (σ i)) oip (osn, kno, val, Suc hops, sip)
            ⊏ dip
            rt (σ' (the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip)) dip)))"
proof clarify
  fix dip
assume a2: "dip ∈ vD(update (rt (σ i)) oip (osn, kno, val, Suc (hops), sip))"
and a3: "dip ∈ vD(update (rt (σ i)) oip (osn, kno, val, Suc hops, sip)))"
and a4: "the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip)) dip) ≠ dip"
from ow have a5: "∀ j. j ≠ i → σ j = σ' j" by auto
show "update (rt (σ i)) oip (osn, kno, val, Suc hops, sip) ⊏ dip (is "?rt1 ⊏ dip ?rt2 dip")
proof (cases "?rt1 = rt (σ i)"
assume nochange [simp]: "update (rt (σ i)) oip (osn, kno, val, Suc hops, sip) = rt (σ i)"
from after have "σ i = σ i" by simp
with a5 have "∀ j. σ j = σ' j" by metis
from a2 have "dip ∈ vD (rt (σ i))" by simp
moreover from a3 have "dip ∈ vD (σ (the (nhop (rt (σ i)) dip)))"
  using nochange and "∀ j. σ j = σ' j" by clarsimp
moreover from a4 have "the (nhop (rt (σ i)) dip) ≠ dip" by simp
ultimately have "rt (σ i) ⊏ dip rt (σ (the (nhop (rt (σ i)) dip)))"
  using pre by simp
hence "rt (σ i) ⊏ dip rt (σ' (the (nhop (rt (σ i)) dip)))"
  using ⟨dip ≠ oip⟩ by simp
thus "?thesis" by simp
next
assume change: "?rt1 ≠ rt (σ i)"
from after a2 have "dip ∈ kD (rt (σ i))" by auto
show ?thesis
proof (cases "dip = oip"
assumedip ≠ oip"
with a2 have "dip ∈ vD (rt (σ i))" by simp
moreover with a3 a5 after and (dip ≠ oip)
  have "dip ∈ vD (σ (the (nhop (rt (σ i)) dip)))"
    by simp metis
moreover from a4 and (dip ≠ oip) have "the (nhop (rt (σ i)) dip) ≠ dip" by simp
ultimately have "rt (σ i) ⊏ dip rt (σ (the (nhop (rt (σ i)) dip)))"
  using pre by simp
with after and a5 and (dip ≠ oip) show ?thesis
  by simp (metis rt_strictly_fresher_update_other
    rt_strictly_fresher_irefl)
next
assume "dip = oip"
with a4 and change have "sip ≠ oip" by simp
with a6 have "oip ∈ kD (σ sip))"
  and "osn ≤ nsqn (rt (σ sip)) oip" by auto
from a3 change 'dip = oip' have "oip ∈ vD (σ' sip))" by simp
hence "the (flag (rt (σ' sip)) oip) = val" by simp
from oip ∈ kD (rt (σ sip))
have "osn ≤ nsqn (rt (σ' sip)) oip ∨ (osn = nsqn (rt (σ' sip)) oip
    ∧ the (dhops (rt (σ' sip)) oip) ≤ hops)"
proof
  assume "oip ∈ vD (rt (σ sip))"
  hence "the (flag (rt (σ sip)) oip) = val" by simp
with a6 (sip ≠ oip) have "nsqn (rt (σ sip)) oip = osn →
  the (dhops (rt (σ sip)) oip) ≤ hops"
    by simp
show ?thesis
proof (cases "sip = i")
assume "sip ≠ i"
with a5 have "σ sip = σ' sip" by simp
with Ωosn ≤ nsqn (rt (σ sip)) oip
and nsqn (rt (σ sip)) oip = osn → the (dhops (rt (σ sip)) oip) ≤ hops
show ?thesis by auto
next
— alternative to using sip_not_ip
assume [simp]: "sip = i"
have "?rt1 = rt (σ i)"
proof (rule update_cases_kD, simp_all)
from ⟨Suc 0 ≤ osn⟩ show "0 < osn" by simp
next
from ⟨oip ∈ kD(rt (σ sip))⟩ and ⟨sip = i⟩ show "oip ∈ kD(rt (σ i))" by simp
next
assume "sqn (rt (σ i)) oip < osn"
also from ⟨osn ≤ nsqn (rt (σ sip)) oip⟩ have "... ≤ nsqn (rt (σ i)) oip" by simp
also have "... ≤ sqn (rt (σ i)) oip" by (rule nsqn_sqn)
finally have "sqn (rt (σ i)) oip < sqn (rt (σ i)) oip".
hence False by simp
thus "(λa. if a = oip then Some (osn, kno, val, Suc hops, i) else rt (σ i) a) = rt (σ i)" ..
next
assume "the (flag (rt (σ i)) oip) = inv"
with ⟨the (flag (rt (σ sip)) oip) = val⟩ have False by simp
thus "(λa. if a = oip then Some (osn, kno, val, Suc hops, i) else rt (σ i) a) = rt (σ i)" ..
next
from ⟨oip ∈ kD(rt (σ sip))⟩
show "(λa. if a = oip then Some (the (rt (σ i) oip)) else rt (σ i) a) = rt (σ i)" by (auto dest!: kD_Some)
qed
with change have False ..
thus ?thesis ..
qed
next
assume "oip ∈ iD(rt (σ sip))"
with ⟨the (flag (rt (σ sip)) oip) = val⟩ and a5 have "sip = i"
by (metis f.distinct(1) iD_flag_is_inv)
from ⟨oip ∈ iD(rt (σ sip))⟩ have "the (flag (rt (σ sip)) oip) = inv" by auto
with ⟨sip = i⟩ ⟨Suc 0 ≤ osn⟩ change after ⟨oip ∈ kD(rt (σ sip))⟩
have "nsqn (rt (σ sip)) oip < nsqn (rt (σ' sip)) oip"
unfolding update_def
by (clarsimp split: option.split_asm split_if_asm)
(auto simp: sqn_def)
with ⟨osn ≤ nsqn (rt (σ sip)) oip⟩ have "osn < nsqn (rt (σ' sip)) oip" by simp
thus ?thesis ..
qed  thus ?thesis
proof
  assume osnlt: "osn < nsqn (rt (σ' i)) oip"
  from ⟨dip∈kD(rt (σ' i))⟩ and ⟨dip = oip⟩ have "dip ∈ kD (?rt1)" by simp
moreover from a3 have "dip ∈ kD(?rt2 dip)" by simp
moreover have "nsqn ?rt1 dip < nsqn (?rt2 dip) dip"
    proof
      have "nsqn ?rt1 oip = osn"
        by (simp add: ⟨dip = oip⟩ nsqn_update_changed_kno_val [OF change [THEN not_sym]])
    also have "... < nsqn (rt (σ' i)) oip" using osnlt.
    also have "... = nsqn (?rt2 oip) oip" by (simp add: change)
    finally show ?thesis
      using ⟨dip = oip⟩ by simp
    qed
ultimately show ?thesis
by (rule rt_strictly_fresher_ltI)
next
  assume osneq: "osn = nsqn (rt (σ' i)) oip ∧ the (dhops (rt (σ' i)) oip) ≤ hops"
  have "oip∈kD(?rt1)" by simp
moreover have "nsqn ?rt1 dip = nsqn (?rt2 oip) dip"
    proof
      from osneq have "osn = nsqn (rt (σ' i)) oip" ..
    also have "osn = nsqn ?rt1 oip"
      by (simp add: ⟨dip = oip⟩ nsqn_update_changed_kno_val [OF change [THEN not_sym]])
    also have "nsqn (rt (σ' i)) oip = nsqn (?rt2 oip) oip"
      by (simp add: change)
    finally show ?thesis
      .
  qed
moreover have "π5(the (?rt2 oip oip)) < π5(the (?rt1 oip))"
    proof
      from osneq have "the (dhops (rt (σ' i)) oip) ≤ hops" ..
      moreover from ⟨oip ∈ vD (rt (σ' i))⟩ have "oip∈kD(rt (σ' i))" by auto
    ultimately have "π5(the (rt (σ' i) oip)) ≤ hops"
      by (auto simp add: proj5_eq_dhops)
    also from change after have "hops < π5(the (rt (σ' i) oip))"
      by (simp add: proj5_eq_dhops) (metis dhops_update_changed lessI)
    finally have "π5(the (rt (σ' i) oip)) < π5(the (rt (σ' i) oip))" .
with change after show ?thesis by simp
qed
ultimately have "?rt1 ⊑oip ?rt2 oip"
  by (rule rt_strictly_fresher_eqI)
with ⟨dip = oip⟩ show ?thesis by simp
qed
qed

note rreq_rrep_update = this

have "opaodv i |-(otherwith (op=) {i} (orecvmsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
  other quality_increases {i} →)
  onl ΓAODV
    (λ(σ, _). ∀dip. dip ∈ vD (rt (σ i)) ∩ vD (rt (σ (the (nhop (rt (σ i)) dip))))
      ∧ the (nhop (rt (σ i)) dip) ≠ dip
      → rt (σ i) ⊑oip rt (σ (the (nhop (rt (σ i)) dip))))"
proof (inv_cterms inv add: onl_oinvariant_sterms [OF aodv_wf rreq_sip [THEN weaken]]
  onl_oinvariant_sterms [OF aodv_wf rrep_sip [THEN weaken]]
  onl_oinvariant_sterms [OF aodv_wf rerr_sip [THEN weaken]]
  483
\[
\text{onl\_oinvariant\_sterms \ [OF \ aodv\_wf \ oosn\_rreq \ [THEN \ weaken]\]}
\]
\[
\text{onl\_oinvariant\_sterms \ [OF \ aodv\_wf \ odsn\_rrep \ [THEN \ weaken]\]}
\]
solve: basic update\_0\_unk invalidate rreq\_rrep\_update
simp add: seqlsimp)

fix \(\sigma, \sigma'\) l

assume or: "\((\sigma, p) \in \text{oreachable \ (opaodv \ i) \ (?S \ i)}\) \ (other quality\_increases \ \{i\})"
and "other quality\_increases \ \{i\} \ \sigma, \sigma'" and ll: "\(l \in \text{labels} \ \Gamma_{AODV} \ p\)"
and pre: "\(\forall \text{dip. dip} \in vD (rt (\sigma \ i)) \land \text{dip} \in vD (rt (\sigma (\text{nhop (rt (\sigma \ i)) dip})))\)
\land \ (\text{the (nhop (rt (\sigma \ i)) dip}) \neq \text{dip} \narrow rt (\sigma \ i) \sqsubseteq dip \ rt (\sigma (\text{the (nhop (rt (\sigma \ i)) dip})))\)"

from this(1-2)
have or': "\((\sigma', p) \in \text{oreachable \ (opaodv \ i) \ (?S \ i)}\) \ (other quality\_increases \ \{i\})"
by - (rule oreachable_other')

from or and ll have next_hop: "\(\forall \text{dip. let nhip = the (nhop (rt (\sigma \ i)) dip) in dip} \in kD (rt (\sigma \ i)) \land \text{nhip} \neq \text{dip} \narrow dip \in kD (rt (\sigma \ nhip))\)
\land \ (\text{sqn (rt (\sigma \ i)) dip} \leq \text{sqn (rt (\sigma \ nhip)) dip} )\)"
by (auto dest!: onl\_oinvariant\_weakenD [OF seq\_compare\_next\_hop']

from or and ll have unk\_hops\_one: "\(\forall \text{dip} \in kD (rt (\sigma \ i)). \text{sqn (rt (\sigma \ i)) dip} = 0 \narrow \text{sqnf (rt (\sigma \ i)) dip} = \text{unk} \land \text{the (dhops (rt (\sigma \ i)) dip}) = 1 \land \text{the (nhop (rt (\sigma \ i)) dip}) = \text{dip} )\)"
by (auto dest!: onl\_oinvariant\_weakenD [OF ozero_seq\_unk\_hops\_one [OF aodv\_trans aodv\_trans]

otherwith\_actionD
simp: seqlsimp)

from \(\text{other quality\_increases \ \{i\} \ \sigma, \sigma'}\) have "\(\sigma' \ i = \sigma \ i\)" by auto

hence "other quality\_increases \ \{i\} \ (\sigma' \ i)" by auto

with "other quality\_increases \ \{i\} \ (\sigma' \ i)" have "\(\forall \text{j. quality\_increases \ (\sigma \ j) \ (\sigma' \ j)}\)"
by - (erule otherE, metis singleton\_iff)

show "\(\forall \text{dip. dip} \in vD (rt (\sigma' \ i)) \land \text{dip} \in vD (rt (\sigma' (\text{the (nhop (rt (\sigma' \ i)) dip})))))\)
\land \ (\text{the (nhop (rt (\sigma' \ i)) dip}) \neq \text{dip} \narrow rt (\sigma' \ i) \sqsubseteq dip \ rt (\sigma' (\text{the (nhop (rt (\sigma' \ i)) dip})))\)"

proof clarify

fix dip

assume "\(\text{dip} \in vD (rt (\sigma' \ i))\)"
and "\(\text{dip} \in vD (rt (\sigma' (\text{the (nhop (rt (\sigma' \ i)) dip}))))\)"
and "\(\text{the (nhop (rt (\sigma' \ i)) dip}) \neq \text{dip} )\)"

from this(1) and \(\sigma' \ i = \sigma \ i\) have "\(\text{dip} \in vD (rt (\sigma \ i))\)"
and "\(\text{dip} \in kD (rt (\sigma \ i))\)"
by auto

from \(\text{the (nhop (rt (\sigma' \ i)) dip}) \neq \text{dip} )\) and \(\sigma' \ i = \sigma \ i\)
have "\(\text{the (nhop (rt (\sigma \ i)) dip}) \neq \text{dip} )\) (is "\(?\text{nhip} \neq \text{ _}\)"
by simp

with \(\text{dip} \in kD (rt (\sigma \ i))\) and next\_hop
have "\(\text{dip} \in kD (rt (\sigma (?\text{nhip})))\)"
and nsqns: "\(\text{sqn (rt (\sigma \ i)) dip} \leq \text{sqn (rt (\sigma (?\text{nhip}))) dip} )\)"
by (auto simp: Let_def)

have "\(0 < \text{sqn (rt (\sigma \ i)) dip} )"
proof (rule neq\_0\_conv [THEN iffD1, OF notI])
assume "\(\text{sqn (rt (\sigma \ i)) dip} = 0\)"
with \(\text{dip} \in kD (rt (\sigma \ i))\) and unk\_hops\_one
have "\(?\text{nhip} = \text{ dip} )\) by simp
with (?nhip \neq dip) show False ..

qed

also have "\(\ldots = \text{sqn (rt (\sigma \ i)) dip} )"

484
by \(\text{rule vD\_nsqn\_sqn \ [OF } \langle \text{dip} \in \text{vD}(\sigma i) \rangle, \ \text{THEN sym}\rangle\)
also have "\(\ldots \leq \text{nsqn} (\rt (\sigma \ ?nhip)) \ \text{dip}\)"
\
by \(\text{rule nsqns}\)
also have "\(\ldots \leq \text{sqn} (\rt (\sigma \ ?nhip)) \ \text{dip}\)"
\
by \(\text{rule nsqn\_sqn}\)
finally have "\(0 < \text{sqn} (\rt (\sigma \ ?nhip)) \ \text{dip}\)".

have "rt (\sigma i) \sqsubseteq dip rt (\sigma' \ ?nhip)"
proof \(\langle \text{cases } \langle \text{dip} \in \text{vD}(\sigma i) \rangle \rangle\) have "\(\text{dip} \sqsubseteq \text{vD}(\sigma i)\)" as well

by \(\text{rule nsqns}\)
also have "\(\ldots \leq \text{nsqn} (\rt (\sigma \ ?nhip)) \ \text{dip}\)"
\
by \(\text{rule nsqns}\)
finally have "\(0 < \text{nsqn} (\rt (\sigma \ ?nhip)) \ \text{dip}\)".

proof from \(\langle \\forall j. \ \text{quality\_increases} (\sigma j) (\sigma' j) \rangle\) have "\(\text{quality\_increases} (\sigma \ ?nhip) (\sigma' \ ?nhip)\)" ..

ultimately show ?thesis
using \(\langle \text{dip} \in \text{kD}(\sigma i) \rangle\) by \(\text{rule strictly\_fresher\_quality\_increases\_right}\)

next
assume "\(\text{dip} \in \text{kD}(\sigma i)\)"

with \(\langle \text{dip} \in \text{kD}(\sigma i) \rangle\) have "\(\text{dip} \in \text{nhop}(\sigma i)\)" ..

hence "\(\text{dip} \in \text{nhop}(\sigma i)\)" as well

by \(\text{auto}\)

moreover from \(\langle \text{dip} \in \text{iD}(\sigma i) \rangle\) have "\(\ldots = \text{sqn} (\rt (\sigma \ ?nhip)) \ \text{dip} - 1\)"..

also have "\(\ldots < \text{sqn} (\rt (\sigma' \ ?nhip)) \ \text{dip}\)"

proof -
from \(\langle \\forall j. \ \text{quality\_increases} (\sigma j) (\sigma' j) \rangle\) have "\(\text{quality\_increases} (\sigma \ ?nhip) (\sigma' \ ?nhip)\)" ..

hence "\(\forall ip. \ \text{sqn} (\rt (\sigma \ ?nhip)) ip \leq \text{sqn} (\rt (\sigma' \ ?nhip)) ip\)" by \(\text{auto}\)

hence "\(\text{sqn} (\rt (\sigma \ ?nhip)) dip \leq \text{sqn} (\rt (\sigma' \ ?nhip)) dip\)"..

with \(\langle 0 < \text{sqn} (\rt (\sigma \ ?nhip)) dip \rangle\) show ?thesis by \(\text{auto}\)

qed

also have "\(\ldots = \text{nsqn} (\rt (\sigma' \ ?nhip)) \ \text{dip}\)"

proof \(\langle \text{rule vD\_nsqn\_sqn \ [THEN sym]\rangle} \) from \(\langle \text{dip} \in \text{vD}(\sigma i) \rangle\) and \(\langle \sigma' i = \sigma i \rangle\)

show "\(\text{dip} \in \text{vD}(\sigma' \ ?nhip)\)" by \(\text{simp}\)

qed

finally have "\(\text{nsqn} (\rt (\sigma i)) dip < \text{nsqn} (\rt (\sigma' \ ?nhip)) dip\)".

moreover from \(\langle \text{dip} \in \text{vD}(\sigma \ ?nhip) \rangle\) and \(\langle \sigma' i = \sigma i \rangle\)

have "\(\text{dip} \in \text{kD}(\sigma \ ?nhip)\)" by \(\text{auto}\)

ultimately show "rt (\sigma i) \sqsubseteq dip rt (\sigma' \ ?nhip)"

using \(\langle \text{dip} \in \text{kD}(\sigma i) \rangle\) by \(\text{rule rt\_strictly\_fresher\_ltI}\)

qed

with \(\langle \sigma' i = \sigma i \rangle\) show "rt (\sigma i) \sqsubseteq dip rt (\sigma' \ ?nhip)"

by \(\text{simp}\)

qed

thus ?thesis unfolding Let\_def .

qed

lemma seq\_compare\_next\_hop: 
fixes \(w\)
shows "\(\text{opaodv } i \parallel (\text{otherwith } \text{op=} \{i\} \ (\text{orecvmsg } \text{msg\_fresh}), \\)
\(\text{other quality\_increases } \{i\} \rightarrow)\)
\(\text{global } (\lambda \sigma. \ \forall dip. \ \text{let nhip } = \text{the } \text{nhop}(\rt (\sigma i)) dip)
\ \in dip \in \text{kD}(\sigma i) \ \wedge \ \text{nhip } \neq \ \text{dip} \rightarrow \)
\(\text{dip} \in \text{kD}(\sigma \ \text{nhip})\)

\(\wedge \ \text{nsqn} (\rt (\sigma i)) dip \leq \text{nsqn} (\rt (\sigma \ \text{nhip}) dip)\)"

by \(\text{rule oinvariant\_weakenE \ [OF seq\_compare\_next\_hop'\]}\) (auto dest!: onlD)

lemma seq\_nhop\_quality\_increases:
5.10 Routing graphs and loop freedom

theory E_Loop_Freedom
imports E_Aodv_Predicates E_Fresher
begin

Define the central theorem that relates an invariant over network states to the absence of loops in the associate routing graph.

definition
  rt_graph :: "(ip ⇒ state) ⇒ ip ⇒ ip rel"
where
  "rt_graph σ = (λdip. {(ip, ip') | ip ip' dsn dsk hops.
  ip ≠ dip ∧ rt (σ ip) dip = Some (dsn, dsk, val, hops, ip')})"

Given the state of a network σ, a routing graph for a given destination ip address dip abstracts the details of routing tables into nodes (ip addresses) and vertices (valid routes between ip addresses).

lemma rt_graphE [elim]:
  fixes n dip ip ip'
  assumes "(ip, ip') ∈ rt_graph σ dip"
  shows "ip ≠ dip ∧ (∃r. rt (σ ip) = r ∧ (∃dsn dsk hops. r dip = Some (dsn, dsk, val, hops, ip')))"
  using assms unfolding rt_graph_def by auto

lemma rt_graph_vD [dest]:
  "∀ip ip' σ dip. (ip, ip') ∈ rt_graph σ dip ⇒ dip ∈ vD(rt (σ ip))"
  unfolding rt_graph_def vD_def by auto

lemma rt_graph_vD_trans [dest]:
  "∀ip ip' σ dip. (ip, ip') ∈ rt_graph σ dip ⊢ dip ∈ vD(rt (σ ip))"
  by (erule converse_tranclE) auto

lemma rt_graph_not_dip [dest]:
  "∀ip ip' σ dip. (ip, ip') ∈ rt_graph σ dip ⇒ ip ≠ dip"
  unfolding rt_graph_def by auto

lemma rt_graph_not_dip_trans [dest]:
  "∀ip ip' σ dip. (ip, ip') ∈ rt_graph σ dip ⊢ ip ≠ dip"
  by (erule converse_tranclE) auto

NB: the property below cannot be lifted to the transitive closure

lemma rt_graph_nhip_is_nhop [dest]:
  "∀ip ip' σ dip. (ip, ip') ∈ rt_graph σ dip ⇒ ip' = the (nhop (rt (σ ip)) dip)"
  unfolding rt_graph_def by auto

theorem inv_to_loop_freedom:
  assumes "∀i dip. let nhip = the (nhop (rt (σ i)) dip)
  in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
  → (rt (σ i)) ⊏ dip (rt (σ nhip))"
  shows "∀dip. irrefl ((rt_graph σ dip)⁺)"
  using assms proof (intro allI)
  fix σ :: "ip ⇒ state" and dip
  assume inv: "∀ip dip.

end
let nhip = the (nhop (rt (σ ip)) dip) in dip ∈ vD(rt (σ ip)) ∩ vD(rt (σ nhip)) ∧ nhip ≠ dip → rt (σ ip) ⊏ dip rt (σ nhip)"

{ fix ip ip' assume "(ip, ip') ∈ (rt_graph σ dip)" and "dip ∈ vD(rt (σ ip))" and "ip' ≠ dip" hence "rt (σ ip) ⊑ dip rt (σ ip')" proof induction fix nhip assume "(ip, nhip) ∈ rt_graph σ dip" and "dip ∈ vD(rt (σ nhip))" and "nhip ≠ dip" from ⟨(ip, nhip) ∈ rt_graph σ dip⟩ have "dip ∈ vD(rt (σ ip))" and "nhip = the (nhop (rt (σ ip)) dip)" by auto from ⟨dip ∈ vD(rt (σ ip))⟩ and ⟨dip ∈ vD(rt (σ nhip))⟩ have "dip ∈ vD(rt (σ ip)) ∩ vD(rt (σ nhip))" .. with "nhip = the (nhop (rt (σ ip)) dip)" and inv show "rt (σ ip) ⊏ dip rt (σ nhip)" by (clarsimp simp: Let_def) next fix nhip nhip' assume "(ip, nhip) ∈ (rt_graph σ dip)" and "(nhip, nhip') ∈ rt_graph σ dip" and IH: 
[ 
[ [ dip ∈ vD(rt (σ nhip)); nhip ≠ dip ] → rt (σ ip) ⊏ dip rt (σ nhip) ]
and "dip ∈ vD(rt (σ nhip'))" and "nhip' ≠ dip"
] have "dip ∈ vD(rt (σ nhip)) ∩ vD(rt (σ nhip'))" with inv show "rt (σ nhip) ⊏ dip rt (σ nhip')" by (clarsimp simp: Let_def) qed finally show "rt (σ ip) ⊏ dip rt (σ nhip')" . qed } note fresher = this

show "irrefl ((rt_graph σ dip)"
unfolding irrefl_def proof (intro allI notI)
fix ip assume "(ip, ip) ∈ (rt_graph σ dip)" moreover then have "dip ∈ vD(rt (σ ip))" and "ip ≠ dip"
by auto ultimately have "rt (σ ip) ⊑ dip rt (σ ip)" by (rule fresher) thus False by simp qed

end
5.11 Lift and transfer invariants to show loop freedom

theory E_Aodv_Loop_Freedom
imports "../../../AWN/OClosed_Transfer" "../../../AWN/Qmsg_Lifting" E_Global_Invariants E_Loop_Freedom
begin

5.11.1 Lift to parallel processes with queues

lemma par_step_no_change_on_send_or_receive:
  fixes σ s a σ' s'
  assumes "((σ, s), a, (σ', s')) ∈ oparp_sos i (oseqp_sos ΓAODV i) (seqp_sos ΓQM SG)
  and "a ≠ τ"
  shows "σ' i = σ i"
  using assms by (rule qmsg_no_change_on_send_or_receive)

lemma par_nhop_quality_increases:
  shows "opaodv i ⟨⟨ i qmsg |– (otherwith (op=) {i} (orecvmsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
  other_quality_increases {i} →)
global (λσ. ∀ dip. let nhip = the (nhop (rt σ i)) dip)
in dip ∈ VD (rt (σ i)) ∩ VD (rt (σ nhip)) ∧ nhip ≠ dip
→ (rt (σ i)) ⊏ dip (rt (σ nhip)))"
proof (rule lift_into_qmsg [OF seq_nhop_quality_increases])
  show "opaodv i ⟨⟨ i qmsg |– A (otherwith (op=) {i} (orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) →))
globala (λ(σ, a, σ'). quality_increases (σ i) (σ' i))"
  proof (rule ostep_invariant_weakenE [OF olocal_quality_increases], simp_all)
    fix t :: "(((nat ⇒ state) × (state, msg, pseqp, pseqp label) seqp), msg seq_action) transition"
    assume "onll ΓAODV (λ((σ, _, _), _, (σ', _)). quality_increases (σ j) (σ' j)) t"
    thus "quality_increases (fst (fst t) i) (fst (snd (snd t)) i)"
    by (cases t) (clarsimp dest!: onllD, metis aodv_ex_label)
  next
    fix σ σ' a
    assume "otherwith quality_increases (i) (orecvmsg (λ_. rreq_rrep_sn)) σ σ' a"
    thus "otherwith quality_increases (i) (orecvmsg (λ_. rreq_rrep_sn)) σ σ' a"
    by (erule weaken_otherwith, auto)
  qed
  qed auto

lemma par_rreq_rrep_sn_quality_increases:
  "opaodv i ⟨⟨ i qmsg |– A (λ_ _. orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) →))
globala (λ(σ, a, σ'). quality_increases (σ i) (σ' i))"
proof -
  have "opaodv i ⟨⟨ i qmsg |– (otherwith (op=) {i} (orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) →))
globala (λ(σ, a, σ'). quality_increases (σ i) (σ' i))"
  proof (rule ostep_invariant_weakenE [OF olocal_quality_increases], simp_all)
    auto dest!: onllD seqllD elim!: aodv_ex_labelE
  hence "opaodv i ⟨⟨ i qmsg |– A (λ_ _. orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) →))
globala (λ(σ, a, σ'). quality_increases (σ i) (σ' i))"
  by (rule lift_step_into_qmsg_statelessassm) simp_all
  thus ?thesis by rule auto
  qed

lemma par_rreq_rrep_qmsg_statelessssasm:
  "opaodv i ⟨⟨ i qmsg |– A (λσ __. orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_. True) {i} →))
globala (λ(σ, _, σ'). quality_increases (σ i) (σ' i))"
proof -
  have "opaodv i ⟨⟨ i qmsg |– A (λσ __. orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_. True) {i} →))
globala (λ(σ, a, σ'). anycast (msg_fresh σ) a)"
  proof (rule ostep_invariant_weakenE [OF rreq_rrep_qmsg_statelessssasm], simp_all)
    fix t
assume  
\(\text{onll} \, \Gamma_{AODV} (\lambda (\sigma, a, \sigma'). \text{anycast } (\text{msg\_fresh} \sigma) \, a) \, t\)

thus \("\text{globala} (\lambda (\sigma, a, \sigma'). \text{anycast } (\text{msg\_fresh} \sigma) \, a) \, t\"

by (cases \(t\)) (clarsimp dest!: onllD, metis aodv_ex_label)

qed

hence \("\text{opaodv } i \, \llangle \llangle \, \text{qmsg} \, = \, \Lambda (\lambda \sigma_. \, \text{orecvmsg} (\lambda_. \, \text{rreq\_rrep\_sn}) \, \sigma)\, \text{other } (\lambda_. \, \text{True}) \, \{i\} \rightarrow \text{globala} (\lambda (\_, a, \_). \text{anycast } (\text{msg\_fresh} \sigma) \, a)"

by (rule lift_step_into_qmsg_statelessassm) simp_all

thus \(?\text{thesis}\) by rule auto

qed

lemma \(\text{par\_anycast\_msg\_zhops}\):
shows \("\text{opaodv } i \, \llangle \llangle \, \text{qmsg} \, = \, \Lambda (\lambda \sigma_. \, \text{orecvmsg} (\lambda_. \, \text{rreq\_rrep\_sn}) \, \sigma)\, \text{other } (\lambda_. \, \text{True}) \, \{i\} \rightarrow \text{globala} (\lambda (\_, a, \_). \text{anycast } \text{msg\_zhops} \, a)"

proof -

from \(\text{anycast\_msg\_zhops\ initiali\_aodv \oaodv\_trans \aodv\_trans}\)

have \("\text{opaodv } i \, \llangle \llangle \, \text{act} \, TT\, \text{other } (\lambda_. \, \text{True}) \, \{i\} \rightarrow \text{seqll } i \, (\text{onll} \, \Gamma_{AODV} (\lambda (\_, a, \_). \text{anycast } \text{msg\_zhops} \, a))\)

by (rule open_seq_step_invariant)

hence \("\text{opaodv } i \, \llangle \llangle \, \text{globala} (\lambda (\_, a, \_). \text{anycast } \text{msg\_zhops} \, a)\)

proof (rule ostep_invariant_weakenE)

fix \(t\) :: "\(((\text{nat} \Rightarrow \text{state}) \times (\text{state}, \text{msg}, \text{pseqp}, \text{pseqp label}) \, \text{seqp}), \text{msg \, seq\_action})\, \text{transition}\"

assume \("\text{seqll } i \, (\text{onll} \, \Gamma_{AODV} (\lambda (\_, a, \_). \text{anycast } \text{msg\_zhops} \, a)) \, t\"

thus \("\text{globala} (\lambda (\_, a, \_). \text{anycast } \text{msg\_zhops} \, a) \, t\"

by (cases \(t\)) (clarsimp dest!: seqllD onllD, metis aodv_ex_label)

qed simp_all

hence \("\text{opaodv } i \, \llangle \llangle \, \text{globala} (\lambda (\_, a, \_). \text{anycast } \text{msg\_zhops} \, a)\)

by (rule lift_step_into_qmsg_statelessassm) simp_all

thus \(?\text{thesis}\) by rule auto

qed

5.11.2 Lift to nodes

lemma \(\text{node\_step\_no\_change\_on\_send\_or\_receive}\):
assumes \("((\sigma, \text{NodeS} \, i \, P \, R), \, a, \, (\sigma', \text{NodeS} \, i' \, P' \, R')) \in \text{onode\_sos} (\text{oparp\_sos} \, i \, (\text{oseqp\_sos} \, \Gamma_{AODV} \, i) \, (\text{seqp\_sos} \, \Gamma_{QM\, SG})))"

and \("a \neq \tau\"

shows \("\sigma' \, i = \sigma \, i""

using \(\text{assms}\)

by (cases \(a\)) (auto elim!: \text{par\_step\_no\_change\_on\_send\_or\_receive})

lemma \(\text{node\_nhop\_quality\_increases}\):
shows \("(\text{node\_nhop\_quality\_increases})\"

by (rule \text{node\_lift} [OF \text{par\_nhop\_quality\_increases}]) auto

lemma \(\text{node\_quality\_increases}\):
shows \("(\text{node\_quality\_increases})\"

by (rule \text{node\_lift\_step\_statelessassm} [OF \text{par\_rreq\_rrep\_sn\_quality\_increases}]) simp

lemma \(\text{node\_rreq\_rrep\_nsqn\_fresh\_any\_step}\):
shows \("(\text{node\_rreq\_rrep\_nsqn\_fresh\_any\_step})\"

by (rule \text{node\_lift\_anycast\_statelessassm} [OF \text{par\_rreq\_rrep\_nsqn\_fresh\_any\_step}])
lemma node_anycast_msg_zhops:
shows "((i : opaodv i ((i qmsg : R) o) =_A (σ _, oarrivemsg (λσ_. rreq_rrep_sn) σ, other (λ _ _. True) {i} →)
globa (λ(σ_, a, a'). castmsg msg_zhops a)"
by (rule node_lift_anycast_statelessassm [OF par_anycast_msg_zhops])

lemma node_silent_change_only:
shows "((i : opaodv i (((i qmsg : R) o) =_A (λσ _, oarrivemsg (λσ_. True) σ, other (λ _ _. True) {i} →)
globa (λ(σ, a, a'). σ_ → σ' i = σ i)"
proof (rule ostep_invariantI, simp (no_asm), rule impl)
fix σ ζ a σ' ζ'
assume or: "((σ, ζ) ∈ orreachable ((i : opaodv i (((i qmsg : R) o) =_A (λσ _, oarrivemsg (λσ_. True) σ, other (λ _ _. True) {i} →)
and tr: "((σ, ζ), a, (σ', ζ')) ∈ trans ((i : opaodv i (((i qmsg : R) o) =_A
and "a ≠ τ_n"
from or obtain p R where "ζ = NodeS i p R"
by (drule node_net_state, metis)
with tr have "((σ, i : NodeS i p R), a, (σ', ζ')) ∈ onode_sos (oparp_sos i (trans (opaodv i)) (trans qmsg))"
by simp
thus "σ' i = σ i" using (a ≠ τ_n)
by (cases rule: onode_sos.cases)
(auto elim: qmsg_no_change_on_send_or_receive)
qed

5.11.3 Lift to partial networks
lemma arrive_repreq_rep_mseq_fresh_inc_sn [simp]:
assumes "oarrivemsg (λσ m. msg_fresh σ m ∧ σ m) σ m"
shows "oarrivemsg (λσ. rreq_rrep_sn) σ m"
using assms by (cases m) auto

lemma opnet_nhopt_quality_increases:
shows "opnet (λi. opaodv i (((i qmsg : R) p =
(otherwith (op=) (net_tree_ips p) =
(oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
other_quality_increases (net_tree_ips p) →)
globa (λσ. ∀i∈net_tree_ips p. ∀dip.
let nhimp = the (nhop (rt (σ i)) dip)
in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhimp)) ∧ nhimp ≠ dip
→ (rt (σ i)) ⊑ dip (rt (σ nhimp)))"
proof (rule pnet_lift [OF node_nhopt_quality_increases])
fix i R
have "((i : opaodv i (((i qmsg : R) o) =_A (λσ _, oarrivemsg (λσ_. rreq_rrep_sn) σ, other (λ _ _. True) {i} →)
globa (λ(σ, a, a'). castmsg (λσ m. msg_fresh σ m ∧ msg_zhops m) a)"
proof (rule ostep_invariantI, simp (no_asm))
fix σ a a' s' s''
assume or: "((σ, s) ∈ orreachable ((i : opaodv i (((i qmsg : R) o) =_A (λσ _, oarrivemsg (λσ_. rreq_rrep_sn) σ, other (λ _ _. True) {i} →)
and tr: "((σ, s), a, (σ', s')) ∈ trans ((i : opaodv i (((i qmsg : R) o) =_A
and am: "oarrivemsg (λσ. rreq_rrep_sn) σ a"
from or tr am have "castmsg (msg_fresh σ) a"
by (auto dest!: ostep_invariantD [OF node_req_rep_msg_seq_fresh_any_step])
moreover from or am have "castmsg (msg_zhops m) a"
by (auto dest!: ostep_invariantD [OF node_anycast_msg_zhops])
ultimately show "castmsg (λσ m. msg_fresh σ m ∧ msg_zhops m) a"
by (case_tac a) auto
qed
thus "((i : opaodv i (((i qmsg : R) o) =_A (λσ _, oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m) σ,
other quality_increases {i} \rightarrow \text{globa} (\lambda (\sigma, a, \_).
\text{castmsg} (\lambda m. \text{msg\_fresh} \sigma m \land \text{msg\_zhops} m) a)\)

by rule auto

next

fix i R

show "i : opaodv i \langle i \text{ qmsg} : R \rangle \models_A
(\lambda \_ _. \text{oarrivemsg} (\lambda m. \text{msg\_fresh} \sigma m \land \text{msg\_zhops} m) \sigma,
other quality_increases {i} \rightarrow \text{globa} (\lambda (\sigma, a, \sigma').
a \neq \tau \land (\forall i d. a \neq i: \text{deliver}(d)) \rightarrow \sigma \ i = \sigma' \ i)"

by (rule ostep_invariant_weakenE [OF node_silent_change_only]) auto

next

fix i R

show "i : opaodv i \langle i \text{ qmsg} : R \rangle \models_A
(\lambda \_ _. \text{oarrivemsg} (\lambda m. \text{msg\_fresh} \sigma m \land \text{msg\_zhops} m) \sigma,
other quality_increases {i} \rightarrow \text{globa} (\lambda (\sigma, a, \sigma').
a = \tau \lor (3 d . a = i: \text{deliver}(d)) \rightarrow \text{quality\_increases} (\sigma \ i) (\sigma' \ i))"

by (rule ostep_invariant_weakenE [OF node_quality_increases]) auto

qed simp_all

5.11.4 Lift to closed networks

lemma onet_nhop_quality_increases:
shows "oclosed (opnet (\lambda i. \text{opaodv} i \langle i \text{ qmsg} \rangle p)
| (\lambda \_ _. \text{True}, other quality_increases (\text{net\_tree\_ips} p) \rightarrow)
\text{global} (\lambda \sigma. \forall i \in \text{net\_tree\_ips} p. \forall dip.
let nhip = \text{the} (\text{nhop} (\text{rt} (\sigma i)) dip)
in dip \in vD (\text{rt} (\sigma i)) \cap vD (\text{rt} (\sigma \ nhip)) \land nhip \neq dip
\rightarrow (\text{rt} (\sigma i)) \sqsubseteq dip (\text{rt} (\sigma \ nhip)))"

(is "_ | (\_, U \rightarrow) ?inv")

proof (rule inclosed_closed)

from opnet_nhop_quality_increases
show "opnet (\lambda i. \text{opaodv} i \langle i \text{ qmsg} \rangle p)
| (otherwith (op=) (\text{net\_tree\_ips} p) inoclosed, ?U \rightarrow) ?inv"

proof (rule oinvariant_weakenE)

fix \sigma, \sigma' :: \"ip \Rightarrow \text{state}\" and a :: \"msg node\_action\"
assume "otherwith (op=) (\text{net\_tree\_ips} p) inoclosed \ \sigma \ \sigma' \ a"
thus "otherwith (op=) (\text{net\_tree\_ips} p)
(\text{oarrivemsg} (\lambda m. \text{msg\_fresh} \sigma m \land \text{msg\_zhops} m) \ \sigma \ \sigma' \ a)"

proof (rule otherwithEI)

fix \sigma :: \"ip \Rightarrow \text{state}\" and a :: \"msg node\_action\"
assume "inoclosed \ \sigma \ a"
thus "oarrivemsg (\lambda m. \text{msg\_fresh} \sigma m \land \text{msg\_zhops} m) \ a"
proof (cases a)

fix ii ni ms
assume "a = ii \_ ni: \text{arrive}(ms)"
moreover with \langle \text{inoclosed} \ \sigma \ a \rangle obtain d di where "ms = \text{newpkt}(d, di)"
by (cases ms) auto
ultimately show "\text{thesis}\" by simp

qed simp_all

qed

qd

5.11.5 Transfer into the standard model

interpretation aodv_openproc: openproc paodv opaodv id
where "aodv_openproc.initmissing = initmissing"

proof -

show "openproc paodv opaodv id"

proof unfold_locales

fix i :: ip

have "\{(\sigma, \_ .). (\sigma i, \_ .) \in \sigma_{AODV} i \land (\forall j. j \neq i \rightarrow \sigma j \in \text{fst}' \ \sigma_{AODV} j)\} \subseteq \sigma_{AODV}'\"
unfolding \sigma_{AODV}'_def \sigma_{AODV}'_def

proof (rule equalityD1)
show "\( \forall p. \{(\sigma, \zeta). (\sigma i, \zeta) \in \{(f i, p)\} \land (\forall j. j \neq i \rightarrow \sigma j \in \text{fst} ' \{(f j, p)\})\} = \{(f, p)\} \)"
by (rule set_eqI) auto
qed
thus "\{ (\sigma, \zeta). |\sigma \zeta s. s \in \text{init} (paodv i) \land (\sigma i, \zeta) = \text{id} s \land (\forall j. j \neq i \rightarrow \sigma j \in (\text{fst o id} ' \text{init} (paodv j)) \} \subseteq \text{init} (opaodv i)" by simp
next
show "\( \forall j. \text{init} (paodv j) \neq \{\} \)"
unfolding \( \sigma_{AODV\_def} \) by simp
next
fix i s a s' \( \sigma, \sigma' \)
assume "\( \sigma i = \text{fst} (\text{id} s) \)"
and "\( \sigma' i = \text{fst} (\text{id} s') \)"
and "\((\sigma, a, \sigma') \in \text{trans} (paodv i)\)"
then obtain q q'
where "\( s = (\sigma i, q) \)"
and "\( s' = (\sigma' i, q') \)"
and "\(((\sigma i, q), a, (\sigma' i, q')) \in \text{trans} (paodv i)\)"
by (cases s, cases s') auto
from this(3)
have "\(((\sigma, q), a, (\sigma', q')) \in \text{trans} (opaodv i)\)" by simp
with (cases s, cases s') auto
show "\(((\sigma, \text{snd} (\text{id} s)), a, (\sigma', \text{snd} (\text{id} s'))) \in \text{trans} (opaodv i)\)" by simp
qed
then interpret op: openproc paodv opaodv id .
have [simp]: "\( \\forall i. (\text{SOME} x. x \in (\text{fst o id} ' \text{init} (paodv i))) = \text{aodv_init} i \)"
unfolding \( \sigma_{AODV\_def} \) by simp
unfolding op.initmissing_def op.someinit_def initmissing_def
by (clarsimp cong: option.case_cong)
thus "openproc.initmissing (\lambda i. \text{paodv} i \langle\langle \text{qmsg} \rangle (\lambda (p, q). (\text{fst} (\text{id} p), \text{snd} (\text{id} p), q)) = \text{initmissing} \)" by (rule ext)
have "\( \\forall P. \sigma. \text{openproc.netglobal} (\lambda i. \text{paodv} i \langle\langle \text{qmsg} \rangle (\lambda (p, q). (\text{fst} (\text{id} p), \text{snd} (\text{id} p), q)) \) \) P \sigma = \text{netglobal} P \sigma" by (clarsimp cong: option.case_cong)
unfolding opq.netglobal_def netglobal_def opq.initmissing_def initmissing_def
by (clarsimp cong: option.case_cong)
thus "openproc.netglobal (\lambda i. \text{paodv} i \langle\langle \text{qmsg} \rangle (\lambda (p, q). (\text{fst} (\text{id} p), \text{snd} (\text{id} p), q)) \) = \text{netglobal}" by auto
qed

interpretation aodv_openproc_par_qmsg: openproc_parq paodv opaodv id qmsg
where "\( \text{aodv_openproc_par_qmsg.netglobal} = \text{netglobal} \)"
and "\( \text{aodv_openproc_par_qmsg.initmissing} = \text{initmissing} \)"
proof -
show "openproc_parq paodv opaodv id qmsg"
by (unfold_locales) simp
then interpret opq: openproc_parq paodv opaodv id qmsg .

have im: "\( \\\forall \sigma. \text{openproc.initmissing} (\lambda i. \text{paodv} i \langle\langle \text{qmsg} \rangle (\lambda (p, q). (\text{fst} (\text{id} p), \text{snd} (\text{id} p), q))) \sigma = \text{initmissing} \)"
unfolding opq.initmissing_def opq.someinit_def initmissing_def
unfolding \( \sigma_{AODV\_def} \) \( \sigma_{QMSG\_def} \) by (clarsimp cong: option.case_cong)
thus "\( \\\forall \sigma. \text{openproc.netglobal} (\lambda i. \text{paodv} i \langle\langle \text{qmsg} \rangle (\lambda (p, q). (\text{fst} (\text{id} p), \text{snd} (\text{id} p), q)) \) \) P \sigma = \text{netglobal} P \sigma" by (clarsimp cong: option.case_cong)
unfolding opq.netglobal_def netglobal_def opq.initmissing_def initmissing_def
unfolding \( \sigma_{AODV\_def} \) \( \sigma_{QMSG\_def} \)
by (clarsimp cong: option.case_cong)
thus "\( \\\forall \sigma. \text{openproc.netglobal} (\lambda i. \text{paodv} i \langle\langle \text{qmsg} \rangle (\lambda (p, q). (\text{fst} (\text{id} p), \text{snd} (\text{id} p), q)) \) = \text{netglobal}" by auto
qed

lemma net_nhopt_quality_increases:
assumes "\( \text{wf_net_tree} n \)"

492
shows "closed (pnet (\lambda i. paodv i (\langle qmsg \rangle n)) \models \text{netglobal} \\
(\lambda \sigma. \forall i dip. \text{let nhip} = \text{the (nhop (rt (\sigma i)) dip)} \\
in dip \in V D (rt (\sigma i)) \cap V D (rt (\sigma nhip)) \land nhip \neq dip \\
\rightarrow (rt (\sigma i)) \Box dip (rt (\sigma nhip)))"
(is "\_ \models \text{netglobal} (\lambda \sigma. \forall i. \text{?inv} \sigma i")")

proof -
from \langle \text{wf_net_tree n} \rangle
have proto: "closed (pnet (\lambda i. paodv i (\langle qmsg \rangle n)) \models \text{netglobal} (\lambda \sigma. \forall i \in \text{net_tree_ips n}. \forall dip. \\
\text{let nhip} = \text{the (nhop (rt (\sigma i)) dip)} \\
in dip \in V D (rt (\sigma i)) \cap V D (rt (\sigma nhip)) \land nhip \neq dip \\
\rightarrow (rt (\sigma i)) \Box dip (rt (\sigma nhip)))"

by (rule aodv_openproc_par_qmsg.close_opnet [OF \_ onet_nhop_quality_increases])

show ?thesis
unfolding invariant_def opnet_sos.opnet_tau1
proof (rule, simp only: aodv_openproc_par_qmsg.netglobalsimp
\text{fst_initmissing_netgmap_pair_fst}, rule allI)

fix \sigma i
assume sr: "\sigma \in \text{reachable (closed (pnet (\lambda i. paodv i (\langle qmsg \rangle n))) TT}\n
hence "\forall i \in \text{net_tree_ips n}. \text{?inv} (\text{fst (initmissing (netgmap fst \sigma))) i}"

by - (drule invariantD [OF proto],
\text{simp only: aodv_openproc_par_qmsg.netglobalsimp
\text{fst_initmissing_netgmap_pair_fst})

thus "?inv (\text{fst (initmissing (netgmap fst \sigma))) i}"

proof (cases "i \in \text{net_tree_ips n}"

assume "i \notin \text{net_tree_ips n}"

from sr have "\sigma \in \text{reachable (pnet (\lambda i. paodv i (\langle qmsg \rangle n)) TT}\n
hence "\text{net_ips} \sigma = \text{net_tree_ips n}"

with \langle i \notin \text{net_tree_ips n} \rangle have "i \notin \text{net_ips} \sigma" by simp

hence "(\text{fst (initmissing (netgmap fst \sigma))) i} = \text{aodv_init i}"

by simp

thus ?thesis by simp
qed metis
qed

5.11.6 Loop freedom of AODV

theorem aodv_loop_freedom:
assumes "\text{wf_net_tree n}"
shows "closed (pnet (\lambda i. paodv i (\langle qmsg \rangle n)) \models \text{netglobal} (\lambda \sigma. \forall \text{dip}. \text{irrefl ((rt_graph \sigma dip)})"
using assms by (rule aodv_openproc_par_qmsg.netglobal_weakenE
\text{[OF net_nhop_quality_increases inv_to_loop_freedom]})
end
Bibliography

http://afp.sf.net/entries/AWN.shtml.


