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} \hspace{1cm} 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show that Isabelle/HOL can re-establish most proof obligations automatically and identify exactly the steps that are no longer valid.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].
Contents

0.1 Basic data types and constants .......................................................... 5
0.2 Predicates and functions used in the AODV model ............................... 5
  0.2.1 Sequence Numbers ........................................................................ 5
  0.2.2 Modelling Routes ......................................................................... 6
  0.2.3 Routing Tables .............................................................................. 6
  0.2.4 Updating Routing Tables ................................................................. 9
  0.2.5 Route Requests .............................................................................. 19
  0.2.6 Queued Packets ............................................................................ 19
  0.2.7 Comparison with the original technical report .................................. 20
0.3 AODV protocol messages .................................................................... 20
0.4 The AODV protocol ............................................................................ 21
  0.4.1 Data state ..................................................................................... 22
  0.4.2 Auxilliary message handling definitions ....................................... 23
  0.4.3 The protocol process .................................................................... 24
0.5 Invariant assumptions and properties .................................................. 30
0.6 Quality relations between routes .......................................................... 32
  0.6.1 Net sequence numbers .................................................................. 32
  0.6.2 Comparing routes ........................................................................ 34
  0.6.3 Comparing routing tables ............................................................... 35
  0.6.4 Strictly comparing routing tables .................................................... 40
0.7 Invariant proofs on individual processes ................................................. 44
0.8 The quality increases predicate ............................................................. 54
0.9 The ‘open’ AODV model .................................................................... 61
0.10 Global invariant proofs over sequential processes ................................. 61
0.11 Routing graphs and loop freedom ....................................................... 79
0.12 Lift and transfer invariants to show loop freedom ............................... 81
  0.12.1 Lift to parallel processes with queues .......................................... 81
  0.12.2 Lift to nodes ............................................................................... 83
  0.12.3 Lift to partial networks ................................................................. 84
  0.12.4 Lift to closed networks ................................................................. 84
  0.12.5 Transfer into the standard model .................................................. 85
  0.12.6 Loop freedom of AODV ............................................................... 87

1 Variant A: Skipping the RREQ ID ............................................................. 88
  1.1 Predicates and functions used in the AODV model ............................ 88
    1.1.1 Sequence Numbers .................................................................... 88
    1.1.2 Modelling Routes ...................................................................... 88
    1.1.3 Routing Tables .......................................................................... 89
    1.1.4 Updating Routing Tables ......................................................... 92
    1.1.5 Route Requests ........................................................................ 101
    1.1.6 Queued Packets ...................................................................... 102
    1.1.7 Comparison with the original technical report ............................ 103
  1.2 AODV protocol messages ................................................................. 103
  1.3 The AODV protocol ....................................................................... 104
    1.3.1 Data state ................................................................................. 104
    1.3.2 Auxilliary message handling definitions ................................... 105
    1.3.3 The protocol process ................................................................. 107
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.11 Lift and transfer invariants to show loop freedom</td>
<td></td>
</tr>
<tr>
<td>1.11.1 Lift to nodes</td>
<td>165</td>
</tr>
<tr>
<td>1.11.2 Lift to partial networks</td>
<td>166</td>
</tr>
<tr>
<td>1.11.3 Lift to closed networks</td>
<td>167</td>
</tr>
<tr>
<td>1.11.4 Transfer into the standard model</td>
<td>168</td>
</tr>
<tr>
<td>1.11.5 Loop freedom of AODV</td>
<td>169</td>
</tr>
<tr>
<td>2.1 Predicates and functions used in the AODV model</td>
<td>170</td>
</tr>
<tr>
<td>2.1.1 Sequence Numbers</td>
<td>170</td>
</tr>
<tr>
<td>2.1.2 Modelling Routes</td>
<td>170</td>
</tr>
<tr>
<td>2.1.3 Routing Tables</td>
<td>171</td>
</tr>
<tr>
<td>2.1.4 Updating Routing Tables</td>
<td>174</td>
</tr>
<tr>
<td>2.1.5 Route Requests</td>
<td>184</td>
</tr>
<tr>
<td>2.1.6 Queued Packets</td>
<td>184</td>
</tr>
<tr>
<td>2.1.7 Comparison with the original technical report</td>
<td>185</td>
</tr>
<tr>
<td>2.2 AODV protocol messages</td>
<td>185</td>
</tr>
<tr>
<td>2.3 The AODV protocol</td>
<td>186</td>
</tr>
<tr>
<td>2.3.1 Data state</td>
<td>186</td>
</tr>
<tr>
<td>2.3.2 Auxiliary message handling definitions</td>
<td>188</td>
</tr>
<tr>
<td>2.3.3 The protocol process</td>
<td>189</td>
</tr>
<tr>
<td>2.4 Invariant assumptions and properties</td>
<td>195</td>
</tr>
<tr>
<td>2.5 Quality relations between routes</td>
<td>197</td>
</tr>
<tr>
<td>2.5.1 Net sequence numbers</td>
<td>197</td>
</tr>
<tr>
<td>2.5.2 Comparing routes</td>
<td>199</td>
</tr>
<tr>
<td>2.5.3 Comparing routing tables</td>
<td>200</td>
</tr>
<tr>
<td>2.5.4 Strictly comparing routing tables</td>
<td>205</td>
</tr>
<tr>
<td>2.6 Invariant proofs on individual processes</td>
<td>209</td>
</tr>
<tr>
<td>2.7 The quality increases predicate</td>
<td>219</td>
</tr>
<tr>
<td>2.8 The ‘open’ AODV model</td>
<td>226</td>
</tr>
<tr>
<td>2.9 Global invariant proofs over sequential processes</td>
<td>227</td>
</tr>
<tr>
<td>2.10 Routing graphs and loop freedom</td>
<td>245</td>
</tr>
<tr>
<td>2.11 Lift and transfer invariants to show loop freedom</td>
<td>247</td>
</tr>
<tr>
<td>2.11.1 Lift to parallel processes with queues</td>
<td>247</td>
</tr>
<tr>
<td>2.11.2 Lift to nodes</td>
<td>248</td>
</tr>
<tr>
<td>2.11.3 Lift to partial networks</td>
<td>249</td>
</tr>
<tr>
<td>2.11.4 Lift to closed networks</td>
<td>250</td>
</tr>
<tr>
<td>2.11.5 Transfer into the standard model</td>
<td>250</td>
</tr>
<tr>
<td>2.11.6 Loop freedom of AODV</td>
<td>252</td>
</tr>
<tr>
<td>3.1 Predicates and functions used in the AODV model</td>
<td>253</td>
</tr>
<tr>
<td>3.1.1 Sequence Numbers</td>
<td>253</td>
</tr>
<tr>
<td>3.1.2 Modelling Routes</td>
<td>253</td>
</tr>
<tr>
<td>3.1.3 Routing Tables</td>
<td>254</td>
</tr>
<tr>
<td>3.1.4 Updating Routing Tables</td>
<td>257</td>
</tr>
</tbody>
</table>

2 Variant B: Forwarding the Route Reply

3 Variant C: From Groupcast to Broadcast
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.

type_synonym \(r = "sqn \times k \times f \times nat \times ip \times ip set"\)

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"

definition proj7 :: "r \Rightarrow ip set" ("\pi_7")
  where "\pi_7 \equiv \lambda(_, _, _, _, _, 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 ] \Rightarrow P (\pi_3 x)"
by (rule k.induct)

lemma proj4_pred [intro]: "[ P val; P inv ] \Rightarrow 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

0.2.3 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)" => "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) | 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) | None \Rightarrow unk"
abbreviation flag :: "\rt \Rightarrow \ip \Rightarrow f\n\nwhere \ "flag rt dip \equiv map_option \pi_4 (\sigma_{route}(rt, dip))"

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

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

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

definition vD :: "\rt \Rightarrow \ip set\n\nwhere \ "vD rt \equiv \{dip. flag rt dip = Some val\}"

definition iD :: "\rt \Rightarrow \ip set\n\nwhere \ "iD rt \equiv \{dip. flag rt dip = Some inv\}"

definition kD :: "\rt \Rightarrow \ip set\n\nwhere \ "kD rt \equiv \{dip. rt dip \neq None\}"

lemma kD_is_vD_and_iD: \ "kD rt = vD rt \cup iD rt"\nunfolding kD_def vD_def iD_def by auto

lemma vD_iD_gives_kD: \ "\forall ip rt. ip \in vD rt \Rightarrow ip \in kD rt"\n\n\"\forall ip rt. ip \in iD rt \Rightarrow ip \in kD rt"\nunfolding kD_is_vD_and_iD by simp_all

lemma kD_Some [dest]: \ fixes dip rt\n\nassumes \ "dip \in kD rt"\n\nshows \ "\exists dsn dsk flag hops nhip pre. \n\\sigma_{route}(rt, dip) = Some (dsn, dsk, flag, hops, nhip, pre)"\nusing assms unfolding kD_def by simp

lemma kD_None [dest]: \ fixes dip rt\n\nassumes \ "dip \notin kD rt"\n\nshows \ "\sigma_{route}(rt, dip) = None"\nusing assms unfolding kD_def by (metis (mono_tags) mem_Collect_eq)

lemma vD_Some [dest]: \ fixes dip rt\n\nassumes \ "dip \in vD rt"\n\nshows \ "\exists dsn dsk hops nhip pre. \n\\sigma_{route}(rt, dip) = Some (dsn, dsk, flag, hops, nhip, pre)"\nusing assms unfolding vD_def by simp

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

lemma iD_Some [dest]: \ fixes dip rt\n\nassumes \ "dip \in iD rt"\n\nshows \ "\exists dsn dsk hops nhip pre. \n\\sigma_{route}(rt, dip) = Some (dsn, dsk, inv, hops, nhip, pre)"\nusing assms unfolding iD_def by simp

lemma val_is_vD [elim]: \ fixes ip rt\n\nassumes \ "ip \in 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 pre
      where rtip: "rt ip = Some (dsn, dsk, f, hops, nhop, pre)"
        by (metis kD_Some)
    from ⟨ip /∈ vD(rt)⟩ have "f ≠ val"
      proof (rule contrapos_nn)
        assume "f = val"
        with rtip have "the (flag rt ip) = val" by simp
        with ⟨ip ∈ kD(rt)⟩ show "ip ∈ vD(rt)" ..
      qed
    with rtip have "the (flag rt ip) = inv" by simp
    with ⟨ip ∈ kD(rt)⟩ show "ip ∈ iD(rt)" ..
  qed

lemma vD_or_iD [elim]
  fixes ip rt
  assumes "ip ∈ kD(rt)"
    and "ip ∈ vD(rt) =⇒ P rt ip"
    and "ip ∈ iD(rt) =⇒ P rt ip"
  shows "P rt ip"
  proof -
    from ⟨ip ∈ kD(rt)⟩
    have "ip ∈ vD(rt) ∪ iD(rt)"
      by (simp add: kD_is_vD_and_iD)
    thus ?thesis by (auto elim: assms(2-3))
  qed

lemma proj5_eq_dhops: "∀ dip rt. dip ∈ kD(rt) =⇒ π_5 (the (rt dip)) = the (dhops rt dip)"
  unfolding sqn_def by (drule kD_Some) clarsimp

lemma proj4_eq_flag: "∀ dip rt. dip ∈ kD(rt) =⇒ π_4 (the (rt dip)) = the (flag rt dip)"
  unfolding sqn_def by (drule kD_Some) clarsimp

lemma proj2_eq_sqn: "∀ dip rt. dip ∈ kD(rt) =⇒ π_2 (the (rt dip)) = sqn rt dip"
  unfolding sqn_def by (drule kD_Some) clarsimp

lemma kD_sqnf_is_proj3 [simp]: "∀ ip rt. ip ∈ kD(rt) =⇒ sqnf rt ip = π_3 (the (rt ip))"
  unfolding sqnf_def by auto
lemma vD_flag_val [simp]:
"\forall dip rt. dip \in vD (rt) \implies the (flag rt dip) = val"
unfolding vD_def by clarsimp

lemma kD_update [simp]:
"\forall nip v. kD (rt(nip \mapsto v)) = 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 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 "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' npre
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]:
  "\(\forall \text{ip dip. } \text{ip} \in kD(\text{rt} x) \Rightarrow \text{sqnf (the (addpreRT (\text{rt} x) \text{ip} npre)) dip = sqnf (\text{rt} x) dip}\)"
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 = \pi_4(r) = \text{val} ∧
\pi_2(r) = 0 = (\pi_3(r) = \text{unk}) ∧
(\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]:
  "\(\forall nhip pre. \text{update_arg_wf } (0, \text{unk}, \text{val}, \text{Suc } 0, nhip, pre)\)"
  "\(\forall n hops nhip pre. \text{update_arg_wf } (\text{Suc } n, \text{kno}, \text{val}, \text{hops, nhip, pre})\)"
unfolding update_arg_wf_def by auto

lemma update_arg_wf_tuples' [elim]:

11
lemma wf_r_cases [intro]:
  fixes P r
  assumes "update_arg_wf r"
  and c1: "n hops nhip pre. Suc 0 G n =⇒ update_arg_wf (n, kno, val, hops, 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 = 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 ?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 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_2(s), snd (addpre r (π_7(s)))))
    else rt (ip =⇒ addpre s (π_7(r)))"

lemma update_simps [simp]:
  fixes r s nrt nr nr' ns rt ip
  defines "s ≡ the 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 "route (rt, ip) = None ..
  thus "update rt ip r = rt (ip =⇒ r)"
  unfolding update_def by simp

12
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)) 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 "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) = (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 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 "(π₃(r) = 0) = (π₃(r) = unk)"
  and c1: "[\[ ip \notin kD(rt) \] \Longrightarrow P (rt (ip ↦→ r))]"
  and c2: "[\[ ip \in kD(rt); sqn rt ip < π₂(r) \] \Longrightarrow P (rt (ip ↦→ addpre r (π₇(the σroute(rt, ip))))))"
  and c3: "[\[ ip \in kD(rt); sqn rt ip = π₂(r); the (dhops rt ip) > π₅(r) \] \Longrightarrow P (rt (ip ↦→ addpre r (π₇(the σroute(rt, ip))))))"
  and c4: "[\[ ip \in kD(rt); sqn rt ip = π₂(r); the (flag rt ip) = inv \] \Longrightarrow P (rt (ip ↦→ addpre r (π₇(the σroute(rt, ip))))))"
  and c5: "[\[ ip \in kD(rt); π₃(r) = unk \] \Longrightarrow 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: "$[\| \| \in kD(rt); sqn \ rt \ ip \ \geq \ \pi_2(r); \ \pi_3(r) = kno; $
\sqn \ rt \ ip = \pi_3(r) \implies \ (\dhops \ rt \ ip) \leq \pi_5(r) \ \land \ \text{the (flag rt ip) = val} \]
\implies P (rt (ip \mapsto \text{addpre (the } \sigma_{\text{route}(rt, ip)} (\pi_7(r)))) )$"
shows "(P \ (update \ rt \ ip \ r))"

proof (cases "ip \in kD(rt)"

assume "ip \notin kD(rt)"
with c1 show \textbf{?thesis}
by simp

next
assume "ip \in kD(rt)"
moreover then obtain dsn dsk fl hops nhip pre
where rteq: "rt \ ip = \text{Some (dsn, dsk, fl, hops, nhip, pre)"
by (metis kD_Some)
m魅over obtain dsn' dsk' fl' hops' nhip' pre'
where req: "$r = (dsn', dsk', fl', hops', nhip', pre)'"
by (cases r) metis
ultimately show \textbf{?thesis}
using \langle (\pi_2(r) = 0) \ = \ (\pi_3(r) = \text{unk}) \rangle
\text{c2 [OF \langle \text{ip} \in kD(rt) \rangle]}
\text{c3 [OF \langle \text{ip} \in kD(rt) \rangle]}
\text{c4 [OF \langle \text{ip} \in kD(rt) \rangle]}
\text{c5 [OF \langle \text{ip} \in kD(rt) \rangle]}
\text{c6 [OF \langle \text{ip} \in kD(rt) \rangle]}

unfolding \text{update_def sqn_def} \ by \ auto

qed

lemma update_cases_kD:
assumes "(\pi_2(r) = 0) \ = \ (\pi_3(r) = \text{unk})"
and \text{c2: sqn \ rt \ ip < \pi_2(r) \implies P (rt (ip \mapsto \text{addpre r (pi7 (the } \sigma_{\text{route}(rt, ip)}))))")
and \text{c3: [sqn \ rt \ ip = \pi_2(r); the (dhops \ rt \ ip) > \pi_5(r)]
\implies P (rt (ip \mapsto \text{addpre r (pi7 (the } \sigma_{\text{route}(rt, ip)}))))")
and \text{c4: [sqn \ rt \ ip = \pi_2(r); the (flag \ rt \ ip) = inv]
\implies P (rt (ip \mapsto \text{addpre r (pi7 (the } \sigma_{\text{route}(rt, ip)}))))")
and \text{c5: [sqn \ rt \ ip = \pi_2(r); \pi_3(r) = unk \implies P (rt (ip \mapsto \text{addpre r (pi7 (the } \sigma_{\text{route}(rt, ip)}))))")}
and \text{c6: [sqn \ rt \ ip \ \geq \ \pi_2(r); \pi_3(r) = kno; $\sqn \ rt \ ip = \pi_3(r) \implies \ (\dhops \ rt \ ip) \leq \pi_5(r) \ \land \ \text{the (flag rt ip) = val} \]
\implies P (rt (ip \mapsto \text{addpre (the } \sigma_{\text{route}(rt, ip)} (\pi_7(r)))) )$"
shows "(P \ (update \ rt \ ip \ r))"
using \text{assms(1) proof (rule update_cases)
assume "sqn \ rt \ ip < \pi_2(r)"
thus "P (rt(ip \mapsto \text{addpre r (pi7 (the (rt ip)))))" by (rule c2)
next
assume "sqn \ rt \ ip = \pi_2(r)"
and "the (dhops rt ip) > \pi_5(r)"
thus "P (rt(ip \mapsto \text{addpre r (pi7 (the (rt ip)))))" by (rule c3)
next
assume "sqn \ rt \ ip = \pi_3(r)"
and "the (flag rt ip) = inv"
thus "P (rt(ip \mapsto \text{addpre r (pi7 (the (rt ip)))))" by (rule c4)
next
assume "\pi_3(r) = unk"
thus "P (rt (ip \mapsto \text{addpre r (pi7 (rt ip)))})")" by (rule c5)
next
assume "sqn \ rt \ ip \ \geq \ \pi_2(r)"
and "\pi_3(r) = kno"
and "sqn \ rt \ ip = \pi_3(r) \implies \ (dhops \ rt \ ip) \leq \pi_5(r) \ \land \ \text{the (flag rt ip) = val}"
thus "P (rt (ip \mapsto \text{addpre (the (rt ip)) (pi7(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]
  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 "1 ≤ hops"
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 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. 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, {})"
⇒ 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 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 dsk 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 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 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]:
"\A 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) \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 "invalidate rt dests \equiv
\l dip. case (rt ip, dests ip) of
  (None, _) \Rightarrow None
  | (Some s, None) \Rightarrow Some s
  | (Some (_, ds, _, hops, nhip, pre), Some rsn) \Rightarrow
    Some (rsn, ds, inv, hops, nhip, pre)"

lemma proj3_invalidate [simp]:
"\A dip. \pi_3 (the ((invalidate rt dests) dip)) = \pi_3 (the (rt dip))"
unfolding invalidate_def by (clarsimp split: option.split)

lemma proj5_invalidate [simp]:
"\A dip. \pi_5 (the ((invalidate rt dests) dip)) = \pi_5 (the (rt dip))"
unfolding invalidate_def by (clarsimp split: option.split)

lemma proj6_invalidate [simp]:
"\A dip. \pi_6 (the ((invalidate rt dests) dip)) = \pi_6 (the (rt dip))"
unfolding invalidate_def by (clarsimp split: option.split)

lemma proj7_invalidate [simp]:
"\A dip. \pi_7 (the ((invalidate rt dests) dip)) = \pi_7 (the (rt dip))"
unfolding invalidate_def by (clarsimp split: option.split)

lemma invalidate_kD_inv [simp]:
"\A 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 "\A rsn. dests dip = Some rsn \implies sqn rt dip \leq rsn"
shows "sqn rt dip \leq 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 = Some (dsn, dsk, flag, hops, nhip, pre)"
  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 dests) ipa = rsn"
unfolding invalidate_def sqn_def
using assms(1) assms(2) [THEN kD_Some]
by clarsimp

lemma dhops_invalidate [simp]:
"\A dip. the (dhops (invalidate rt dests) dip) = the (dhops rt dip)"
unfolding invalidate_def by (clarsimp split: option.split)

lemma sqnf_invalidate [simp]:
"\A 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]:
"\( \forall \text{dip}. \text{the} (\text{nhop} (\text{invalidate} (\text{rt} \, \xi) (\text{dests} \, \xi)) \, \text{dip}) = \text{the} (\text{nhop} (\text{rt} \, \xi) \, \text{dip}) \)"
unfolding invalidate_def by (clarsimp split: option.split)

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

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

lemma vD_invalidate_vD_not_dests:
"\( \forall \text{dip rt dests}. \text{dip} \in \text{vD} (\text{invalidate rt dests}) \Rightarrow \text{dip} \in \text{vD} (\text{rt}) \land \text{dests dip} = \text{None} \)"
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 \text{dom} (\text{dests})"
  shows "\text{sqn} (\text{invalidate rt dests dip}) = \text{sqn rt dip}"
  using assms unfolding sqn_def by simp

lemma invalidate_changes:
  fixes rt dests dip dsn dsk flag hops nhip pre
  assumes "\text{invalidate rt dests dip} = \text{Some} (\text{dsn, dsk, flag, hops, nhip, pre})"
  shows "dsn = (\text{case dests dip of None \Rightarrow } \pi_2 (\text{the} (\text{rt dip})) \land \text{Some} \, \text{rsn} \Rightarrow \text{rsn})
  \land dsk = \pi_3 (\text{the} (\text{rt dip}))
  \land flag = (\text{if dests dip = None then } \pi_4 (\text{the} (\text{rt dip})) \text{ else inv})
  \land hops = \pi_5 (\text{the} (\text{rt dip}))
  \land nhip = \pi_6 (\text{the} (\text{rt dip}))
  \land pre = \pi_7 (\text{the} (\text{rt dip}))"
  using assms unfolding invalidate_def
  by (cases "\text{rt dip}", clarsimp, cases "\text{dests dip}"
  auto

lemma proj3_inv: "\( \forall \text{dip rt dests}. \text{dip} \in \text{kD (rt)} \Rightarrow \pi_3 (\text{the} (\text{invalidate rt dests dip})) = \pi_3 (\text{the} (\text{rt dip})) \)"
by (clarsimp simp: invalidate_def kD_def split: option.split)

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

0.2.5 Route Requests

Generate a fresh route request identifier.

definition nrreqid :: "(ip \times rreqid) set \Rightarrow \text{ip} \Rightarrow rreqid"
  where "nrreqid \text{reqs} \, \text{ip} \equiv \text{Max} \{(n. \, \text{ip}, n) \in \text{reqs} \cup \{0\}) + 1\"

0.2.6 Queued Packets

Functions for sending data packets.

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

definition sigma_p_flag :: "store \Rightarrow ip \Rightarrow p" ("\sigma_p-flag'(\_, \_')")
where "\sigma_p-flag(store, dip) \equiv map_option fst (store dip)"

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

definition setRRF :: "store \Rightarrow (ip \Rightarrow sqn) \Rightarrow store"
where "setRRF store dests \equiv \lambda dip. if dests dip = None then store dip
        else map_option (\lambda (_, 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. nhop is partial, thus a ‘the’ is needed, similarly for dhops and addpreRT.
2. precs is partial.
3. \sigma_p-flag'(\_, \_') 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

0.3 AODV protocol messages

theory Aodv_Message
imports Aodv_Basic
begin

datatype msg =
    Rreq nat rreqid ip sqn k ip sqn ip
    | Rrep nat ip sqn ip ip

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**

\[ \text{rreq} :: \text{n} \times \text{rreqid} \times \text{i} \times \text{dsn} \times \text{dsk} \times \text{oip} \times \text{osn} \times \text{sip} \Rightarrow \text{msg} \]

where \[ \text{rreq} \equiv \lambda(hops, rreqid, dip, dsn, dsk, oip, osn, sip). Rreq \text{ hops} \text{ rreqid} \text{ dip} \text{ dsn} \text{ dsk} \text{ oip} \text{ osn} \text{ sip} \]

**Lemma**

\[ \text{rreqsimp} [\text{simp}]: \]

\[ \text{rreq(hops, rreqid, dip, dsn, dsk, oip, osn, sip) = Rreq \text{ hops} \text{ rreqid} \text{ dip} \text{ dsn} \text{ dsk} \text{ oip} \text{ osn} \text{ sip} } \]

unfolding rreq_def by simp

**Definition**

\[ \text{rrep} :: \text{n} \times \text{i} \times \text{dsn} \times \text{oip} \times \text{sip} \Rightarrow \text{msg} \]

where \[ \text{rrep} \equiv \lambda(hops, dip, dsn, oip, sip). Rrep \text{ hops} \text{ dip} \text{ dsn} \text{ oip} \text{ sip} \]

**Lemma**

\[ \text{rrepsimp} [\text{simp}]: \]

\[ \text{rrep(hops, dip, dsn, oip, sip) = Rrep \text{ hops} \text{ dip} \text{ dsn} \text{ oip} \text{ sip} } \]

unfolding rrep_def by simp

**Definition**

\[ \text{rerr} :: \text{i} \times \text{i} \times \text{dsn} \times \text{osn} \times \text{sip} \Rightarrow \text{msg} \]

where \[ \text{rerr} \equiv \lambda(dests, sip). Rerr \text{ dests} \text{ sip} \]

**Lemma**

\[ \text{rrerrsimp} [\text{simp}]: \]

\[ \text{rerr(dests, sip) = Rerr \text{ dests} \text{ sip} } \]

unfolding rerr_def by simp

**Lemma**

\[ \text{not_eq_newpkt_rreq} [\text{simp}]: \]

\[ \lnot \text{eq_newpkt} (Rreq \text{ hops} \text{ rreqid} \text{ dip} \text{ dsn} \text{ dsk} \text{ oip} \text{ osn} \text{ sip}) \]

unfolding eq_newpkt_def by simp

**Lemma**

\[ \text{not_eq_newpkt_rrep} [\text{simp}]: \]

\[ \lnot \text{eq_newpkt} (Rrep \text{ hops} \text{ dip} \text{ dsn} \text{ oip} \text{ sip}) \]

unfolding eq_newpkt_def by simp

**Lemma**

\[ \text{not_eq_newpkt_rerr} [\text{simp}]: \]

\[ \lnot \text{eq_newpkt} (Rerr \text{ dests} \text{ sip}) \]

unfolding eq_newpkt_def by simp

**Lemma**

\[ \text{not_eq_newpkt_pkt} [\text{simp}]: \]

\[ \lnot \text{eq_newpkt} (Pkt \text{ d} \text{ dip} \text{ sip}) \]

unfolding eq_newpkt_def by simp

**Definition**

\[ \text{pkt} :: \text{data} \times \text{i} \times \text{i} \Rightarrow \text{msg} \]

where \[ \text{pkt} \equiv \lambda(d, dip, sip). Pkt \text{ d} \text{ dip} \text{ sip} \]

**Lemma**

\[ \text{pktsimp} [\text{simp}]: \]

\[ \text{pkt(d, dip, sip) = Pkt \text{ d} \text{ dip} \text{ sip} } \]

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"
begin

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]: "¬(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 (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

0.4.2 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' ∧
  ξ' = ξ(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 ξ") simp_all

definition is_rrep where "is_rrep ξ ≡ case msg ξ of
  Rrep hops' dip' dsn' oip' sip' ⇒
  { ξ(hops := hops', dip := dip', dsn := dsn', oip := oip', sip := 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 ⟹ 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(λmsg' ξ, η | msg := msg')

(is_newpkt) NEWPKT(λξ. (data ξ, ip η))

(is_pkt) PKT(λξ. (data ξ, dip η, oip η))

(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, 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))

AODV()

λξ. (dests ξ, sip ξ))

AODV(

λξ. pre η, λξ. rerr(dests ξ, ip ξ)). AODV()

AODV()

λξ. (dests ξ, sip ξ))

AODV()

λξ. (dests ξ, sip ξ))

AODV()

λξ. (dests ξ, sip ξ))

AODV()

λξ. (dests ξ, sip ξ))

AODV()
AODV()
)
")

"Γ_AODV PRreq = labelled PRreq (∠. (cip ξ, rreqid ξ) ∈ rreqs ∠)
AODV()
alia
(ξ. (cip ξ, rreqid ξ) ∈ rreqs ∠

[ξ. ξ | rt := update (rt ξ) (cip ξ) (osn ξ, kno, val, hops ξ + 1, sip ξ, {}) []
[ξ. ξ | rreqs := rreqs ξ ∪ {(cip ξ, rreqid ξ)} []

(ξ. dip ξ = ip ξ)
[ξ. ξ | sn := max (sn ξ) (dsn ξ) []
unicast(λξ. the (nhop (rt ξ) (cip ξ)), λξ. rrep(0, dip ξ, sn ξ, cip ξ, ip ξ)).AODV()
> [ξ. ξ | dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (cip ξ)) then Some (inc (sqn (rt ξ) rip)) else None) []
[ξ. ξ | rt := invalidate (rt ξ) (dests ξ) []
[ξ. ξ | store := setRFF (store ξ) (dests ξ) []
[ξ. ξ | pre := \{ the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) \} []
[ξ. ξ | dests := (λrip. if ((dests ξ rip) = None ∧ the (precs (rt ξ) rip) ≠ {}) then (dests ξ rip else None) []
groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)).AODV()
alia
(ξ. dip ξ ≠ ip ξ)
[ξ. dip ξ ∈ vD (rt ξ) ∧ dsn ξ ≤ sqn (rt ξ) (dip ξ) ∧ sqnf (rt ξ) (dip ξ) = kno
[ξ. ξ | rt := the (addpreRT (rt ξ) (dip ξ) {cip ξ}) []
[ξ. ξ | store := setRFF (store ξ) (dests ξ) []
[ξ. ξ | pre := \{ the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) \} []
[ξ. ξ | dests := (λrip. if ((dests ξ rip) = None ∧ the (precs (rt ξ) rip) ≠ {}) then (dests ξ rip else None) []
groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)).AODV()

alia
(ξ. dip ξ ∈ vD (rt ξ))
[ξ. ξ | rt := the (addpreRT (rt ξ) (dip ξ) {the (nhop (rt ξ) (cip ξ))}) []
[ξ. ξ | rt := the (addpreRT (rt ξ) (the (nhop (rt ξ) (dip ξ))))
{the (nhop (rt ξ) (cip ξ))} []
unicast(λξ. the (nhop (rt ξ) (cip ξ)), λξ. rrep(hops ξ + 1, rreqid ξ, dip ξ, max (sqn (rt ξ) (dip ξ)) (dsn ξ), dsk ξ, cip ξ, osn ξ, ip ξ)).AODV()

alia
(ξ. dip ξ ≠ ip ξ)
[ξ. dip ξ ∈ vD (rt ξ)
[ξ. ξ | rt := update (rt ξ) (dip ξ) (dsn ξ, kno, val, hops ξ + 1, sip ξ, {}) []
[ξ. oip ξ = ip ξ
AODV()

alia
(ξ. oip ξ ≠ ip ξ)
[ξ. oip ξ ∈ vD (rt ξ)
[ξ. ξ | rt := the (addpreRT (rt ξ) (dip ξ) {the (nhop (rt ξ) (oip ξ))}) []
[ξ. 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)), λξ. rrep(hops ξ + 1, dip ξ, dsn ξ, oip ξ, ip ξ)).AODV()
groupcast(\lambda \xi. pre \xi, \lambda \xi. rerr(dests \xi, ip \xi)). AODV()

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

fun \Gamma_{AODV}_skeleton
where
  "\Gamma_{AODV}_skeleton PAodv = seqp_skeleton (\Gamma_{AODV} PAodv)"
/ "\Gamma_{AODV}_skeleton PNewPkt = seqp_skeleton (\Gamma_{AODV} PNewPkt)"
/ "\Gamma_{AODV}_skeleton PPkt = seqp_skeleton (\Gamma_{AODV} PPkt)"
/ "\Gamma_{AODV}_skeleton PRreq = seqp_skeleton (\Gamma_{AODV} PRreq)"
/ "\Gamma_{AODV}_skeleton PRrep = seqp_skeleton (\Gamma_{AODV} PRrep)"
/ "\Gamma_{AODV}_skeleton PRerr = seqp_skeleton (\Gamma_{AODV} PRerr)"

lemma \Gamma_{AODV}_skeleton_wf [simp]:
  "wellformed \Gamma_{AODV}_skeleton"
proof (rule, intro allI)
  fix pn pn'
  show "call(pn') \notin 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 p pn
  shows "p \in ctermsl (\Gamma_{AODV} pn) \implies
    (p \in ctermsl (\Gamma_{AODV} PAodv) \lor
    p \in ctermsl (\Gamma_{AODV} PNewPkt) \lor
    p \in ctermsl (\Gamma_{AODV} PPkt) \lor
    p \in ctermsl (\Gamma_{AODV} PRreq) \lor
    p \in ctermsl (\Gamma_{AODV} PRrep) \lor
    p \in ctermsl (\Gamma_{AODV} PRerr))"
  using assms
  by (cases pn) simp_all

definition \sigma_{AODV} :: "ip \Rightarrow (state \times (state, msg, pseq, pseq label) seqp) set"
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 
\[
\begin{align*}
paodv i &\equiv (\text{init} = \sigma_{AODV} i, \text{trans} = \text{seqp_sos} \Gamma_{AODV})
\end{align*}
\]

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: \( \Gamma_{AODV}_{\text{simps}} \))

lemma aodv_wf [simp]: 
\[
\text{wellformed} \Gamma_{AODV}
\]

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]: 
\[
\text{assumes } \forall l \in \text{labels} \Gamma_{AODV} p. P \ l \ p \\
\text{and } \exists p . \ l \ P \ l \ p \ \Rightarrow \ Q
\]

shows "Q"

using assms by (metis aodv_ex_label)

lemma aodv_simple_labels [simp]: 
\[
\text{simple_labels} \Gamma_{AODV}
\]

proof

fix \( p \)

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]: 
\[
(\xi, p) \in \sigma_{AODV} i \Longrightarrow \text{labels} \Gamma_{AODV} p = \{PAodv \ - 0\}
\]

unfolding \( \sigma_{AODV}_{\text{def}} \) by simp

lemma aodv_init_kD_empty [simp]: 
\[
(\xi, p) \in \sigma_{AODV} i \Longrightarrow \text{kD (rt } \xi \text{)} = \{\}
\]

unfolding \( \sigma_{AODV}_{\text{def}} \) \( \text{kD}_{\text{def}} \) by simp

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

by simp

lemma aodv_init_sip_not_ip' [simp]: 
\[
\text{assumes } (\xi, p) \in \sigma_{AODV} i \\
\text{shows } \neg \text{sip } \xi \neq \text{ip } \xi
\]

using assms unfolding \( \sigma_{AODV}_{\text{def}} \) by simp

lemma aodv_init_sip_not_i [simp]: 
\[
\text{assumes } (\xi, p) \in \sigma_{AODV} i \\
\text{shows } \neg \text{sip } \xi \neq i
\]

using assms unfolding \( \sigma_{AODV}_{\text{def}} \) by simp

lemma clear_locals_sip_not_ip':
\[
\text{assumes } \text{ip } \xi = i \\
\text{shows } \neg (\text{sip (clear_locals } \xi) = i)
\]

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 [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

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 sip. msg_zhops (Newpkt 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 \textbf{rreq\_rrep\_fresh} :: "rt ⇒ msg ⇒ bool"
where "\textbf{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 \textbf{rreq\_rrep\_fresh} [simp]:
  "\(\forall\) hops rreqid dip dsn dsnc oip osnc sip.
   \textbf{rreq\_rrep\_fresh} crt (Rreq hops rreqid dip dsn dsnc oip osnc sip) =
   (sip ≠ oip → oip∈kD(crt)
    ∧ (sqn crt oip > osnc
    ∨ (sqn crt oip = osnc
      ∧ the (dhops crt oip) ≤ hopsc
      ∧ the (flag crt oip) = val)))" 

  "\(\forall\) hops dip dsn oip sip. \textbf{rreq\_rrep\_fresh} crt (Rrep hops dip dsn oip sip) =
   (sip ≠ dip → dip∈kD(crt)
    ∧ sqn crt dip = dsn
    ∧ the (dhops crt dip) = hopsc
    ∧ the (flag crt dip) = val)"

  "\(\forall\) dests sip. \textbf{rreq\_rrep\_fresh} crt (Rerr dests sip) = True"

  "\(\forall\) dip. \textbf{rreq\_rrep\_fresh} crt (Newpkt dip) = True"

  "\(\forall\) dip sip. \textbf{rreq\_rrep\_fresh} crt (Pkt dip sip) = True"

unfolding \textbf{rreq\_rrep\_fresh\_def} by simp_all

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

lemma \textbf{rerr\_invalid} [simp]:
  "\(\forall\) hops rreqid dip dsn dsnc oip osnc sip.
   \textbf{rerr\_invalid} crt (Rreq hops rreqid dip dsn dsnc oip osnc sip) = True"

  "\(\forall\) hops dip dsn oip sip. \textbf{rerr\_invalid} crt (Rrep hops dip dsn oip sip) = True"

  "\(\forall\) dests sip. \textbf{rerr\_invalid} crt (Rerr dests sip) = (∀ ripc∈dom(destsc).
    ripc∈iD(crt) ∧ the (destsc ripc) = sqn crt ripc)"

  "\(\forall\) dip. \textbf{rerr\_invalid} crt (Newpkt dip) = True"

  "\(\forall\) dip sip. \textbf{rerr\_invalid} crt (Pkt dip sip) = True"

unfolding \textbf{rerr\_invalid\_def} by simp_all

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

lemma \textbf{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 \textbf{initmissing\_def} by simp

lemma \textbf{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 \textbf{initmissing\_def} by auto
We introduce a streamlined alternative to \textit{initmissing} with \texttt{netgmap} to simplify invariant statements and thus facilitate their comprehension and presentation.

\begin{verbatim}
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)
end
\end{verbatim}

definition
netglobal :: "((nat ⇒ state) ⇒ bool) ⇒ ((state × 'b) × 'c) net\_state ⇒ bool"
where
  "netglobal P ≡ (λs. P (default aodv\_init (netlift fst s)))"

\section{Quality relations between routes}

\begin{verbatim}
theory Fresher
imports Aodv\_Data
begin

\subsection{Net sequence numbers}

On individual routes

definition
nsqn\_r :: "r ⇒ sqn"
where
  "nsqn\_r r ≡ if π₄(r) = val ∨ π₂(r) = 0 then π₂(r) else (π₂(r) - 1)"

lemma nsqn\_r\_def':
  "nsqn\_r 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:
  "∀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"
\end{verbatim}
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 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 - 1"
  using assms unfolding nsqn_sqn_def by auto

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 simp: option.split_asm option.split split_if_asm)
  (metis fun_upd_triv)
lemma nsqn_addpreRT_inv [simp]:
"\forall rt dip npre dip'. dip ∈ kD(rt) \implies
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 \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 ?thesis
    using \(dests dip = Some rsn\) by simp
  qed

lemma nsqn_invalidate_other [simp]:
assumes "dip \in kD(rt)"
and "dip /\ \notin \ \text{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 \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 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.

lemma fresherE [elim]:
assumes "r \sqsubseteq r'"
and "nsqn, r < nsqn, r' \implies P r r'"
and "nsqn, r = nsqn, r' \land \pi_5(r) \geq \pi_5(r') \implies P r r'"
shows "P r r'"
using assms unfolding fresher_def by auto

lemma fresher_refl [simp]: "r \sqsubseteq 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

0.6.3 Comparing routing tables

definition
rt_fresher :: "ip ⇒ rt ⇒ rt ⇒ bool"
where
"rt_fresher ≡ λ dip rt rt'. (the (σ_route(rt, dip))) ⊑ (the (σ_route(rt', dip)))"

abbreviation
rt_fresher_syn :: "rt ⇒ ip ⇒ rt ⇒ bool" ("(_/ ⊑ _/ _)"
where
"rt1 ⊑ ip rt2 ≡ rt_fresher i rt1 rt2"

lemma rt_fresher_def':
"(rt1 ⊑ rt2) = (nsqn rt1 dip < nsqn rt2 dip ∨ 
nsqn rt1 dip = nsqn rt2 dip ∧ π₅ (the (rt 1 i)) ≤ π₅ (the (rt 1 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_def fresher_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 ≈1 rt2 ≡ rt_fresh_as i rt1 rt2"

lemma rt_fresh_as_refl [simp]: "∀rt dip. rt ⇒ 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_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 \( \text{nsqn \ rt \ dip = nsqn \ rt' \ dip} \)
and \( \pi_5(\text{the \ (rt \ dip)}) = \pi_5(\text{the \ (rt' \ dip)}) \)
shows \( rt \approx_{\text{dip}} rt' \)

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_{\text{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_{\text{dip}} rt"
  by (rule rt_fresherI2)
qed

lemma rt_fresh_asE [elim]:
  assumes "rt1 \approx_{\text{dip}} rt2"
  and "; rt1 \sqsubseteq_{\text{dip}} rt2; rt2 \sqsubseteq_{\text{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_{\text{dip}} rt2"
  shows "rt1 \sqsubseteq_{\text{dip}} rt2"
  using assms unfolding rt_fresh_as_def by simp

lemma rt_fresh_asD2 [dest]:
  assumes "rt1 \approx_{\text{dip}} rt2"
  shows "rt2 \sqsubseteq_{\text{dip}} rt1"
  using assms unfolding rt_fresh_as_def by simp

lemma rt_fresh_as_sym:
  assumes "rt1 \approx_{\text{dip}} rt2"
  shows "rt2 \approx_{\text{dip}} rt1"
  using assms unfolding rt_fresh_as_def by simp

lemma not_rt_fresh_asI1 [intro]:
  assumes "\neg (rt1 \sqsubseteq_{\text{dip}} rt2)"
  shows "\neg (rt1 \approx_{\text{dip}} rt2)"
  proof
  assume "rt1 \approx_{\text{dip}} rt2"
  hence "rt1 \sqsubseteq_{\text{dip}} rt2" ..
  with \neg (rt1 \sqsubseteq_{\text{dip}} rt2) show False ..
  qed

lemma not_rt_fresh_asI2 [intro]:
  assumes "\neg (rt2 \sqsubseteq_{\text{dip}} rt1)"
  shows "\neg (rt1 \approx_{\text{dip}} rt2)"
  proof
  assume "rt1 \approx_{\text{dip}} rt2"
  hence "rt2 \sqsubseteq_{\text{dip}} rt1" ..
  with \neg (rt2 \sqsubseteq_{\text{dip}} rt1) show False ..
  qed

lemma not_single_rt_fresher [elim]:
  assumes "\neg (\text{the \ (rt1 \ ip) \sqsubseteq \ the \ (rt2 \ ip)})"
  shows "\neg (rt1 \sqsubseteq_{\text{ip}} rt2)"
  proof
  assume "rt1 \sqsubseteq_{\text{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 (rt1 \sqsubseteq ip rt2)"
shows "\neg (the (rt1 ip) \sqsubseteq the (rt2 ip))"
proof
assume "the (rt1 ip) \sqsubseteq the (rt2 ip)"
  hence "rt1 \sqsubseteq ip rt2" ..
  with \neg (rt1 \sqsubseteq ip rt2) 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 \langle dip \in kD(rt1) \rangle \langle dip \in kD(rt2) \rangle]
kD_nsqn [OF \langle dip \in kD(rt2) \rangle \langle dip \in kD(rt1) \rangle])

lemma rt_fresher_mapupd [intro!]:
assumes "dip \in kD(rt)"
and "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 inkD: "dip \in kD(rt)"
and "dip \neq ip"
shows "rt \subseteq dip 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 "the (dhops rt dip) \geq 1"
and "update_arg_wf r"
shows "rt \subseteq dip update rt ip r"
proof (cases "dip = ip")
  assume "dip \neq 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, 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) \geq 1) and (dip\in kD(rt)) have "hops \geq 1"
by (metis proj5_eq_dhops projs(4))
from (dip\in 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)"

38
proof \(\text{rule \textit{wf\_r\_cases}}\)
fix \(n\) nip pre
from \(\langle \text{hops}_n \geq 1 \rangle\) have "\(\forall \text{pre}'. \ (\text{dsn}_n, \text{dsk}_n, f_n, \text{hops}_n, \text{nhip}_n, \text{pre}_n) \subseteq (\text{dsn}_n, \text{unk}, \text{val}, \text{Suc 0}, \text{nhip}, \text{pre}')""
unfolding \textit{fresher\_def} \textit{sqn\_def} by (cases \(f_n\)) auto
thus "(\text{dsn}_n, \text{dsk}_n, f_n, \text{hops}_n, \text{nhip}_n, \text{pre}_n) \subseteq (\text{dsn}_n, \text{kno}, \text{val}, \text{hops}, \text{nhip}, \text{pre} \cup \text{pre}_n)"
unfolding \textit{fresher\_def} \textit{sqn\_def} by \(\textit{simp\_all}\)
next
fix \(dsn \cdot \text{sqn} \) and \(\text{hops} \) nip pre
assume "\(0 < dsn\)"
show "(\text{dsn}_n, \text{dsk}_n, f_n, \text{hops}_n, \text{nhip}_n, \text{pre}_n) \subseteq (\text{dsn}_n, \text{kno}, \text{val}, \text{hops}, \text{nhip}, \text{pre} \cup \text{pre}_n)"
proof \(\text{rule \textit{update\_cases\_kD \[OF \_ (dip\in\text{kD\(\text{rt}\))], \text{simp\_all add: \(0 < dsn\)]}}\)
assume "\(dsm_n < dsm\)"
thus "(\text{dsn}_n, \text{dsk}_n, f_n, \text{hops}_n, \text{nhip}_n, \text{pre}_n) \subseteq (\text{dsn}_n, \text{kno}, \text{val}, \text{hops}, \text{nhip}, \text{pre} \cup \text{pre}_n)"
unfolding \textit{fresher\_def} \textit{nsqn\_def} by \(\textit{simp}\)
next
assume "\(dsm_n = dsm\)"
with \(\langle 0 < dsm \rangle\) show "(\text{dsn}_n, \text{dsk}_n, \text{inv}, \text{hops}_n, \text{nhip}_n, \text{pre}_n) \subseteq (\text{dsn}_n, \text{kno}, \text{val}, \text{hops}, \text{nhip}, \text{pre} \cup \text{pre}_n)"
unfolding \textit{fresher\_def} by \(\textit{simp}\)
qed

hence "\(rt \subseteq dip\) \text{update \(rt\) dip \(r\)}" by - (rule \textit{single\_rt\_fresher}, \textit{simp})
with \(\langle \text{dip} = \text{ip} \rangle\) show \?thesis by \textit{simp}
qed

\textbf{theorem} \textit{rt\_fresher\_invalidate} [simp]:
assumes "\(\text{dip} \in \text{kD\(\text{rt}\))}\)"
and \textit{indests}: "\(\forall \text{rip} \in \text{dom\(\text{dests}\))}. \text{rip} \in \text{vD\(\text{rt}\) \land \text{sqn \(\text{rt}\) rip < the \(\text{dests rip}\)}\)"
shows "\(rt \subseteq dip\) \text{invalidate \(rt\) \text{dests}\)"
proof \(\text{cases "\(\text{dip} \in \text{dom\(\text{dests}\))\)\}"}\)
assume "\(\text{dip} \in \text{dom\(\text{dests}\))}\)"
thus \?thesis using \(\langle \text{dip} \in \text{kD\(\text{rt}\))\rangle\)
by - (rule \textit{single\_rt\_fresher}, \textit{simp})
next
assume "\(\text{dip} \in \text{dom\(\text{dests}\))}\)"
moreover with \textit{indests} have "\(\text{dip} \in \text{vD\(\text{rt}\)) \land \text{sqn \(\text{rt}\) dip < the \(\text{dests dip}\)}\)"
by auto
ultimately show \?thesis
unfolding invalidate_def \textit{sqn\_def} by - (rule \textit{single\_rt\_fresher}, auto \textit{simp: fresher\_def})
qed

\textbf{lemma} \textit{nsqn\_r\_invalidate} [simp]:
assumes "\(\text{dip} \in \text{kD\(\text{rt}\))}\)
and "\(\text{dip} \in \text{dom\(\text{dests}\))}\)
shows "\(\text{nsqn \(\text{r}\) (the \(\text{invalidate \(rt\) \text{dests dip}\)) = the \(\text{dests dip}\) - 1}\)"
using \textit{assms} unfolding invalidate_def by auto

\textbf{lemma} \textit{rt\_fresh\_as\_inc\_invalidate} [simp]:
assumes "\(\text{dip} \in \text{kD\(\text{rt}\))}\)
and "∀ rip∈dom(dests). rip∈vD(rt) ∧ the (dests rip) = inc (sqn rt rip)"
shows "rt≈ dip invalidate rt dests"

proof (cases "rip∈dom(dests)")
assume "rip∈dom(dests)"
with (rip∈kD(rt)) have "rip∈kD(invalidate rt dests)"
by simp
with (rip∈kD(rt)) show ?thesis
by rule (simp_all add: rip∈dom(dests))

next
assume "rip∈dom(dests)"
with asms (2) have "rip∈vD(rt)" and "the (dests rip) = inc (sqn rt rip)"

by auto
from ⟨rip∈kD(rt)⟩ have "rip∈kD(invalidate rt dests)"
by simp
moreover then have "rip∈kD(invalidate rt dests)" by simp
ultimately show ?thesis
proof (rule nsqn_rt_fresh_asI)
from ⟨rip∈vD(rt)⟩ have "nsqn rt rip = sqn rt rip" by simp
also have "sqn rt rip = nsqr (the (invalidate rt dests rip))" proof
from ⟨rip∈kD(rt)⟩ have "nsqr (the (invalidate rt dests rip)) = the (dests rip) - 1"
using ⟨rip∈dom(dests)⟩ by (rule nsqr_invalidate)
with ⟨the (dests rip) = inc (sqn rt rip)⟩
show "sqn rt rip = nsqr (the (invalidate rt dests rip))" by simp
qed
also from ⟨rip∈kD(invalidate rt dests)⟩ have "nsqr (the (invalidate rt dests rip)) = nsqr (invalidate rt dests) rip"
by (simp add: kD_nsqn)
finally show "nsqr rt rip = nsqr (invalidate rt dests) rip".
qed simp
qed

lemmas rt_fresher_inc_invalidate [simp] = rt_fresh_as_inc_invalidate [THEN rt_fresh_asD1]

lemma rt_fresh_as_addpreRT [simp]:
amss "ip∈kD(rt)"
shows "rt≈ dip the (addpreRT rt ip npre)"
using asms [THEN kD_Some] by (auto simp: addpreRT_def)

lemmas rt_fresher_addpreRT [simp] = rt_fresh_as_addpreRT [THEN rt_fresh_asD1]

0.6.4 Strictly comparing routing tables
definition rt_strictly_fresher :: "ip ⇒ rt ⇒ rt ⇒ bool"
where
"rt_strictly_fresher ≡ λdip rt1 rt2. (rt1 ⊑ dip rt2) ∧ ¬(rt1 ≈ dip rt2)"

abbreviation rt_strictly_fresher_syn :: "rt ⇒ ip ⇒ rt ⇒ bool" ("_/ ⊑_/ _") [51, 999, 51] 50
where
"rt1 ⊑_ i rt2 = rt_strictly_fresher i rt1 rt2"

lemma rt_strictly_fresher_def':
"rt1 ⊑_ i rt2 = ((rt1 ⊑_ i rt2) ∧ ¬(rt2 ⊑_ i rt1))"
unfolding rt_strictly_fresher_def rt_fresh_as_def by auto

lemma rt_strictly_fresherI' [intro]:
amss "rt1 ⊑_ i rt2"
and "¬(rt2 ⊑_ i rt1)"
shows "rt1 ⊑_ i rt2"
using asms unfolding rt_strictly_fresher_def' by simp

lemma rt_strictly_fresherE' [elim]:
amss "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 \sqsubseteq_i rt2"
and "¬(rt1 \approx_i rt2)"
shows "rt1 \sqsubset_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 \sqsubset_i rt2"
and "[ rt1 \sqsubseteq_i rt2; ¬(rt1 \approx_i rt2) ] \implies 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 = (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 (rt1 i)) > \pi_5 (the (rt2 i))))"
unfolding rt_strictly_fresher_def'' rt_fresher_def fresher_def by auto

lemma rt_strictly_fresher_fresherD [dest]:
assumes "rt1 \sqsubseteq_i dip rt2"
shows "the (rt1 dip) \sqsubseteq 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 \sqsubseteq_i dip rt2"
shows "¬ rt1 \approx dip rt2"
using assms unfolding rt_strictly_fresher_def by auto

lemma rt_strictly_fresher_trans [elim, trans]:
assumes "rt1 \sqsubseteq dip rt2"
and "rt2 \sqsubseteq dip rt3"
shows "rt1 \sqsubseteq dip rt3"
using assms proof -
from rt1 \sqsubseteq dip rt2: obtain "the (rt1 dip) \sqsubseteq the (rt2 dip)" by auto
also from rt2 \sqsubseteq dip rt3: obtain "the (rt2 dip) \sqsubseteq the (rt3 dip)" by auto
finally have "the (rt1 dip) \sqsubseteq the (rt3 dip)" .

moreover have "¬ (rt1 \approx dip rt3)"
proof -
from rt1 \sqsubseteq dip rt2: obtain "¬(the (rt2 dip) \sqsubseteq the (rt1 dip))" by auto
also from rt2 \sqsubseteq dip rt3: obtain "¬(the (rt3 dip) \sqsubseteq the (rt2 dip))" by auto
finally have "¬(the (rt3 dip) \sqsubseteq the (rt1 dip))" .
thus ?thesis ..
qed
ultimately show "rt1 \sqsubseteq dip rt3" ..
qed

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

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 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 ⊑ 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"
  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 ⊑ dip rt2"
and "i ∈ kD rt1"
and "i ∈ kD rt2"
shows "nsqn rt1 i ≤ nsqn rt2 i"
using assms(1)
by (auto simp add: rt_fresher_def2 [OF assms(2-3)])

lemma rt_strictly_fresher_ltI [intro]:
assumes "i ∈ kD(rt1)"
and "i ∈ kD(rt2)"
and "nsqn rt1 i < nsqn rt2 i"
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 i ≤ nsqn rt1 i"
    using ⟨i ∈ kD(rt2)⟩ ⟨i ∈ kD(rt1)⟩
    by (rule rt_fresher_imp_nsqn_le)
  with ⟨nsqn rt1 i < nsqn rt2 i⟩ show "False"
    by simp
qed

lemma rt_strictly_fresher_eqI [intro]:
assumes "i ∈ kD(rt1)"
and "i ∈ kD(rt2)"

and "nsqn rt1 i = nsqn rt2 i"
and \("\pi_5(\text{the } (rt2 i)) < \pi_5(\text{the } (rt1 i))\)"
s
shows "rt1 ⪯ rt2"
using assms unfolding rt_strictly_fresher_def' by (auto simp add: kD_nsqn)

lemma invalidate_unfolding rt_strictly_fresher_def':
"\(\forall \text{dests } dip \ rt \ '. \ \text{dests } dip = \text{None} \implies \text{invalidate rt dests } \sqsubseteq dip \ rt \') = (rt \ sqsubseteq dip \ rt)"
unfolding invalidate_def rt_strictly_fresher_def'
by (rule iffI) (auto split: option.split_asm)

lemma vD_invalidate_rt_strictly_fresher: assumes "dip \in vD(invalidate rt1 dests)"
shows "(invalidate rt1 dests \sqsubseteq dip \ rt2) = (rt1 \ sqsubseteq dip \ rt2)"
proof (cases "dip \in dom(dests)")
  assume "dip \notin dom(dests)"
hence "dip \notin vD(invalidate rt1 dests)"
  unfolding invalidate_def vD_def
  by (clarsimp (metis (hide_lams, no_types) assms vD_Some vD_invalidate_vD_not_dests))
  with ⟨dip \in vD(invalidate rt1 dests)⟩
  have "dip \in vD(rt1)"
  unfolding invalidate_def vD_def
  by (clarsimp (metis (hide_lams) assms vD_iD_gives_kD(1)))
  ultimately show ?thesis
  unfolding invalidate_def rt_strictly_fresher_def'
  by (auto)
next
  assume "dip \notin dom(dests)"
hence "dests dip = None" by auto
moreover with ⟨dip \notin vD(invalidate rt1 dests)⟩ have "dip \notin 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: assumes "\(dip \in kD(rt)\)"
shows "(the (update rt dip (osn, kno, val, Suc hops, nhip, {})) \sqsubseteq dip \ rt2) = (rt \ sqsubseteq dip \ rt2)"
using assms unfolding rt_strictly_fresher_def' by clarsimp

lemma lt_sqn_imp_update_strictly_fresher: assumes "dip \in vD(rt2 nhip)"
and *: "osn < sqn (rt2 nhip dip)"
and **: "rt \notin update rt dip (osn, kno, val, hops, nhip, {})"
shows "update rt dip (osn, kno, val, hops, nhip, {}) \sqsubseteq dip rt2 nhip"
proof (rule dirjI1) from ** have "nsqn (update rt dip (osn, kno, val, hops, nhip, {})) dip = osn"
  by (rule nsqn_update_changed_kno_val)
with ⟨dip \in 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 \in 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 \in vD(rt2 nhip)"
and sqn: "sqn (rt2 nhip dip) = osn"
and hop: "\(\text{the } (\text{dhops } (rt2 nhip dip)) \leq hops\)"
and **: "rt \notin update rt dip (osn, kno, val, Suc hops, nhip, {})"
s
shows "update rt dip (osn, kno, val, Suc hops, nhip, {}) \sqsubseteq dip rt2 nhip"
unfolding \textit{rt\_strictly\_fresher\_def}.  

proof (rule disjI2, rule conjI)
  from \(*\) have \("\mathtt{nsqn} (\text{update} \ \mathtt{rt} \ \text{dip} (\mathtt{osn}, \ \text{kno}, \ \text{val}, \ \text{Suc} \ \text{hops}, \ \text{nhip}, \ \{\}) \ \text{dip} = \osn)"
  by (rule \textit{nsqn\_update\_changed\_kno\_val})
  with \(\langle \text{dip} \in \text{vD} (\text{rt2} \ \text{nhip}) \rangle\)
    have \("\mathtt{nsqn}, \ (\text{the} \ (\text{update} \ \mathtt{rt} \ \text{dip} (\mathtt{osn}, \ \text{kno}, \ \text{val}, \ \text{Suc} \ \text{hops}, \ \text{nhip}, \ \{\}) \ \text{dip})) = \osn"
  by (simp add: \textit{kD\_nsqn})
    also have \("\mathtt{osn} = \mathtt{sqn} (\text{rt2} \ \text{nhip}) \ \text{dip}\)" by (rule \textit{sqn \ [symmetric]})
    also have \("\mathtt{sqn} (\text{rt2} \ \text{nhip}) \ \text{dip} = \mathtt{nsqn}, \ (\text{the} \ (\text{rt2} \ \text{nhip} \ \text{dip}))"
    unfolding \textit{nsqn\_def} using \(\langle \text{dip} \in \text{vD} (\text{rt2} \ \text{nhip}) \rangle\)
    by - (metis \textit{vD\_flag\_val} \textit{proj2\_eq\_sqn} \textit{proj4\_eq\_flag} \textit{vD\_iD\_gives\_kD(1)})
    finally have \("\mathtt{nsqn}, \ (\text{the} \ (\text{update} \ \mathtt{rt} \ \text{dip} (\mathtt{osn}, \ \text{kno}, \ \text{val}, \ \text{Suc} \ \text{hops}, \ \text{nhip}, \ \{\}) \ \text{dip})) = \mathtt{nsqn}, \ (\text{the} \ (\text{rt2} \ \text{nhip} \ \text{dip}))" .
  next
    have \("\text{the} \ (\text{dhops} (\text{rt2} \ \text{nhip}) \ \text{dip}) \leq \text{hops}"\) by (rule \textit{hop})
    also have \("\text{hops} < \text{hops} + 1"\) by simp
    finally have \("\text{the} \ (\text{dhops} (\text{update} \ \mathtt{rt} \ \text{dip} (\mathtt{osn}, \ \text{kno}, \ \text{val}, \ \text{Suc} \ \text{hops}, \ \text{nhip}, \ \{\}) \ \text{dip})) < \text{the} \ (\text{dhops} (\text{rt2} \ \text{nhip}) \ \text{dip})" .
    thus \("\pi_5 (\text{the} \ (\text{rt2} \ \text{nhip} \ \text{dip})) < \pi_5 (\text{the} \ (\text{update} \ \mathtt{rt} \ \text{dip} (\mathtt{osn}, \ \text{kno}, \ \text{val}, \ \text{Suc} \ \text{hops}, \ \text{nhip}, \ \{\}) \ \text{dip}))"
    using \(\langle \text{dip} \in \text{vD} (\text{rt2} \ \text{nhip}) \rangle\) by (simp add: \textit{proj5\_eq\_dhops})
qed


lemma \textit{nsqn\_invalidate}:
  assumes \("\text{dip} \in \text{kD} (\text{rt})"\)
  and \("\forall \text{ip} \in \text{dom} (\text{dests}). \ \text{ip} \in \text{vD} (\text{rt}) \land \text{the} \ (\text{dests} \ \text{ip}) = \text{inc} (\text{sqn} \ \text{rt} \ \text{ip})"\)
  shows \("\mathtt{nsqn} (\text{invalidate} \ \text{rt} \ \text{dests}) \ \text{dip} = \mathtt{nsqn} \ \text{rt} \ \text{dip}"\)
proof -
  from \(\langle \text{dip} \in \text{kD} (\text{rt}) \rangle\) have \("\text{dip} \in \text{kD} (\text{invalidate} \ \text{rt} \ \text{dests})"\) by simp
  from \textit{assms} have \("\text{rt} \approx_{\text{dip}} \text{invalidate} \ \text{rt} \ \text{dests}"\)
    by (rule \textit{rt\_fresh\_as\_inc\_invalidate})
  with \(\langle \text{dip} \in \text{kD} (\text{rt}) \ \langle \text{dip} \in \text{kD} (\text{invalidate} \ \text{rt} \ \text{dests}) \rangle \ \text{show \ \text{?thesis}}\)
    by (simp add: \textit{kD\_nsqn} del: \textit{invalidate\_kD\_inv})
    (erule (2) \textit{rt\_fresh\_as\_nsqn})
qed


0.7 Invariant proofs on individual processes

theory \textit{Seq\_Invariants}
imports "./AWN/Invariants" \textit{Aodv} \textit{Aodv\_Data} \textit{Aodv\_Predicates} \textit{Fresher}

begin

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

Proposition 7.2

lemma \textit{sequence\_number\_increases}:
  \"\text{paodv} i \triangleright_{\text{A}} \text{onl1} \ \Gamma_{\text{AODV}} (\lambda((\xi, \_), \_ , (\xi', \_)). \ \text{sn} \ \xi \leq \text{sn} \ \xi')"\)
  by \textit{inv\_cterms}

lemma \textit{sequence\_number\_one\_or\_bigger}:
  \"\text{paodv} i \triangleright_{\text{A}} \Gamma_{\text{AODV}} (\lambda(\xi, \_). \ 1 \leq \text{sn} \ \xi)"\)
  by (rule \textit{onl\_step\_to\_invariantI} \textit{[OF \textit{sequence\_number\_increases}])}
    (auto \simp: \textit{\sigma_{AODV\_def}})

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

lemma \textit{sequence\_number\_increases'}:
  \"\text{paodv} i \triangleright_{\text{A}} (\lambda((\xi, \_), \_ , (\xi', \_)). \ \text{sn} \ \xi \leq \text{sn} \ \xi')"\)
  by (rule \textit{step\_invariant\_weakenE} \textit{[OF \textit{sequence\_number\_increases}])} (auto \dest\!: \textit{onllD})
lemma sequence_number_one_or_bigger':
"paodv i \models (\lambda(\xi, _). 1 \leq sn \xi)"
by (rule invariant_weakenE [OF sequence_number_one_or_bigger]) auto

lemma sip_in_kD:
"paodv i \models onl \Gamma^\text{AODV} (\lambda(\xi, l). l \in \{PAodv-:7 \cup \{PAodv-:5\} \cup \{PRrep-:0..PRrep-:1\}
\cup \{PRreq-:0..PRreq-:3\} \to sip \xi \in kD (rt \xi))"
by inv_cterms

lemma rrep_1_update_changes:
"paodv i \models onl \Gamma^\text{AODV} (\lambda(\xi, l). (l = PRrep-:1 \to rt \xi \neq update (rt \xi) (\text{dip} \xi) (dsn \xi, kno, val, hops \xi + 1, sip \xi, {})))"
by inv_cterms

lemma addpreRT_partly_welldefined:
"paodv i \models onl \Gamma^\text{AODV} (\lambda(\xi, l). (l \in \{PRreq-:16..PRreq-:18\} \cup \{PRreq-:2..PRreq-:6\} \to dip \xi \in kD (rt \xi))
\wedge (l \in \{PRreq-:3..PRreq-:17\} \to oip \xi \in kD (rt \xi)))"
by inv_cterms

Proposition 7.38

lemma includes_nhip:
"paodv i \models onl \Gamma^\text{AODV} (\lambda(\xi, l). \forall \text{dip} \in kD(rt \xi). \text{nhop} (rt \xi) dip \in kD(rt \xi))"

proof -
{ fix ip and \xi \xi' :: state
  assume "\forall dip \in kD (rt \xi). \text{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).
  \text{nhop} (\text{update} (rt \xi) ip (0, unk, val, Suc 0, ip, {})) dip = ip
  \vee \text{nhop} (\text{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 one_hop = this

{ fix sip sn hops and \xi \xi' :: state
  assume "\forall dip \in kD (rt \xi). \text{nhop} (rt \xi) dip \in kD (rt \xi)"
  and "\xi' = \xi([rt := update (rt \xi) sip (sn, kno, val, Suc hops, sip, {})])"
  and "sip \in kD (rt \xi)"
  hence "(\text{nhop} (\text{update} (rt \xi) sip (sn, kno, val, Suc hops, sip, {})) dip = ip
  \vee \text{nhop} (\text{update} (rt \xi) sip (sn, kno, val, Suc hops, sip, {})) dip \in kD (rt \xi))
  \wedge \text{nhop} (\text{update} (rt \xi) ip (sn, kno, val, Suc hops, sip, {})) 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:
"paodv i \models onl \Gamma^\text{AODV} (\lambda(\xi, l). (l \in \{PRreq-:16..PRreq-:18\} \cup \{PRreq-:2..PRreq-:6\} \to dip \xi \in kD (rt \xi))
\wedge (l \in \{PRreq-:3..PRreq-:17\} \to oip \xi \in kD (rt \xi)))"

(is "_. \models onl \Gamma^\text{AODV} ?P")
unfolding invariant_def

proof
  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)

  fix s

have "onl \( \Gamma_{AODV} \) ?P (\( \xi, p \))"
proof (rule onlI)
fix \( l \)
assume "\( l \in \text{labels} \ \Gamma_{AODV} \) p"
with \( (\xi, p) \in \text{reachable (paodv i) TT} \)
have I1: "\( l \in \{\text{PRreq-:16..PRreq-:18}\} \rightarrow \text{dip } \xi \in \text{kD(}\text{rt } \xi)\)"
and I2: "\( l = \text{PRreq-:17} \rightarrow \text{oip } \xi \in \text{kD(}\text{rt } \xi)\)"
and I3: "\( l = \text{PRreq-:6} \rightarrow (\text{the (nhop (rt } \xi) \text{ ( dip } \xi))) \in \text{kD(}\text{rt } \xi)\)"
by (auto dest!: invariantD [OF addpreRT_partly_welldefined])
morerfrom \( (\xi, p) \in \text{reachable (paodv i) TT} \) \( l \in \text{labels} \ \Gamma_{AODV} \) p and I3
have "\( l = \text{PRreq-:6} \rightarrow (\text{the (nhop (rt } \xi) \text{ ( dip } \xi))) \in \text{kD(}\text{rt } \xi)\)"
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:
"paodv i \|= \ A 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:
"paodv i \|= \ A 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:
"paodv i \|= onl \( \Gamma_{AODV} \) (\( \lambda((\xi, l)). l \in \{\text{PAodv-:15..PAodv-:19}\}
\cup \{\text{PPkt-:7..PPkt-:11}\}
\cup \{\text{PRreq-:9..PRreq-:13}\}
\cup \{\text{PRreq-:21..PRreq-:25}\}
\cup \{\text{PRreq-:10..PRreq-:14}\}
\cup \{\text{PRerr-:1..PRerr-:5}\}
\rightarrow (\forall \text{ip} \in \text{dom(dests } \xi). \text{ip} \in \text{kD(rt } \xi) \wedge \text{sqn (rt } \xi) \text{ ip} \leq \text{the (dests } \xi \text{ ip)))}\)"
proof -
have sqninv:
"\( \forall \text{dests rt rsn ip}. \left[ \exists \text{ip} \in \text{dom(dests)}. \text{ip} \in \text{kD(rt)} \wedge \text{sqn rt ip} \leq \text{the (dests ip)}; \text{dests ip} = \text{Some rsn} \right] \Rightarrow \text{sqn (invalidate rt dests) ip} \leq \text{rsn}\)"
by (rule sqn_invalidate_in_dests [THEN eq_imp_le], assumption) auto
have indests:
"\( \forall \text{dests rt rsn ip}. \left[ \exists \text{ip} \in \text{dom(dests)}. \text{ip} \in \text{kD(rt)} \wedge \text{sqn rt ip} \leq \text{the (dests ip)}; \text{dests ip} = \text{Some rsn} \right] \Rightarrow \text{ip} \in \text{kD(rt)} \wedge \text{sqn rt ip} \leq \text{rsn}\)"
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:
"paodv i \|= \ A onll \( \Gamma_{AODV} \) (\( \lambda((\xi, _), _, (\xi', _)). \forall \text{ip}. \text{sqn (rt } \xi) \text{ ip} \leq \text{sqn (rt } \xi') \text{ ip})\)"
proof -
{ fix \( \xi :: \text{state} \)
assume *: "\( \forall \text{ip} \in \text{dom(dests } \xi). \text{ip} \in \text{kD (rt } \xi) \wedge \text{sqn (rt } \xi) \text{ ip} \leq \text{the (dests } \xi \text{ ip})\)"
have "\( \forall \text{ip}. \text{sqn (rt } \xi) \text{ ip} \leq \text{sqn (invalidate (rt } \xi) \text{ (dests } \xi)) \text{ ip})\)"
proof
fix ip
from * have "ip \notin \text{dom}(\text{dests } \xi) \lor \text{sqn}(\text{rt } \xi) \text{ ip} \leq \text{the}(\text{dests } \xi \text{ ip})" by simp
thus "\text{sqn}(\text{rt } \xi) \text{ ip} \leq \text{sqn}(\text{invalidate}(\text{rt } \xi)(\text{dests } \xi)) \text{ 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 \models \Gamma_{\text{AODV}} (\lambda(\xi, _). \text{ip } \xi = i)"
bysimp

Proposition 7.8

lemma sender_ip_valid':
"paodv i \models \text{onll } \Gamma_{\text{AODV}} (\lambda((\xi,_), a, _). \text{anycast}(\lambda m. \text{not}_\text{Pkt } m \rightarrow \text{msg}_\text{sender } m = \text{ip } \xi) a)"
bysimp

lemma sender_ip_valid:
"paodv i \models \text{onll } \Gamma_{\text{AODV}} (\lambda((\xi,)_). \text{msg}_\text{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 \models \text{onl} \Gamma_{\text{AODV}} (\lambda(\xi, l). l \in \{\text{PAodv-:1}\} \rightarrow P (\text{msg } \xi))"
bysimp

Proposition 7.9

lemma sip_not_ip':
"paodv i \models \text{onl} \Gamma_{\text{AODV}} (\lambda(l). \text{sip } r \neq \text{ip } r)")"
bysimp

lemma sip_not_ip:
"paodv i \models \text{onl} \Gamma_{\text{AODV}} (\lambda(l). \text{sip } r \neq \text{ip } r)")"
bysimp

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

Proposition 7.10

lemma hop_count_positive:
"paodv i \models \text{onl} \Gamma_{\text{AODV}} (\lambda(\xi, _). \forall ip \in \text{kd}(\text{rt } \xi). \text{the}(\text{dhops}(\text{rt } \xi) \text{ ip}) \geq 1)"
bysimp

lemma rreq_dip_in_vD_dip_eq_ip:
"paodv i \models \text{onl} \Gamma_{\text{AODV}} (\lambda(l, 1). l \in \{\text{PReq-:16..PReq-:18}\} \rightarrow P (\text{msg } \xi))"
bysimp

proof (inv_cterms, elim conjE)

fix l \xi pp p'
assume "((\xi, pp) \in \text{reachable}(\text{paodv } i) \text{ TT}"
and "(\text{PReq-:17}[\lambda rt \cdot \text{the}(\text{addpreRT}(\text{rt } \xi)(\text{oip } \xi)(\text{the}(\text{nhop}(\text{rt } \xi)(\text{dip } \xi))))]) p' \in \text{sterms} \Gamma_{\text{AODV}} pp"
and "1 = \text{PReq-:17}"
and "\text{dip } \xi \in \text{vd}(\text{rt } \xi)"
from this(1-3) have "\text{oip } \xi \in \text{kd}(\text{rt } \xi)"
by (auto dest: onl_invariant_sterms [OF aodv_wf addpreRT_welldefined, where l="\text{PReq-:17}""])
with \( \text{dip } \xi \in vD \ (rt \ \xi) \)
show "dip \( \xi \in vD \ (\text{the (addpreRT } (rt \ \xi) \ (oip \ \xi) \ \{\text{the (nhop } (rt \ \xi) \ (dip \ \xi))\}))" by simp
qed

Proposition 7.11

lemma anycast_msg_zhops:
"∀ \text{rreqid dip dsn dsk oip osn sip. paodv } i \models \text{Gamma}_\text{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 } (\text{paodv } i) \ TT"
  and "{PRreq-:18}unicast(λξ. \text{the (nhop } (rt \ \xi) \ (oip \ \xi)),
  λξ. \text{Rrep (the (dhops } (rt \ \xi) \ (dip \ \xi))) \ (dip \ \xi) \ (sqn } (rt \ \xi) \ (dip \ \xi)) \ (oip \ \xi) \ (ip \ \xi))
  p' \triangleright pp' \in \text{sterms } \text{Gamma}_\text{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 "∀ ip \in kD } (rt \ \xi). Suc 0 ≤ the (dhops } (rt \ \xi) ip"
  and "dip \ \xi \in vD } (rt \ \xi)"
from "dip \ \xi \in vD } (rt \ \xi)" have "dip \ \xi \in kD } (rt \ \xi)"
  by (rule \ vD\_iD\_gives\_kD(1))
  with "Suc 0 ≤ the (dhops } (rt \ \xi) ip"
thus "0 < the (dhops } (rt \ \xi) (dip \ \xi))" by simp
qed

lemma hop_count_zero_oip_dip_sip:
"paodv } i \models (\text{recvmsg } msg\_zhops } \rightarrow \text{onl } \text{Gamma}_\text{AODV} (\lambda(\_ , l). }\text{hop\_count\_zero\_oip\_dip\_sip } a)"
proof (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv])
  fix \ a
  assume "recvmsg (λm. \text{rreq\_rrep\_sn } m \ \wedge \ \text{msg\_zhops } m) \ a" 
  thus "recvmsg \text{rreq\_rrep\_sn } a" by (cases \ a) simp_all
qed

lemma osn\_rreq:
"paodv } i \models (\text{recvmsg } \text{rreq\_rrep\_sn } \rightarrow \text{onl } \text{Gamma}_\text{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':
"paodv } i \models (\text{recvmsg } (\lambda m. \text{rreq\_rrep\_sn } m \ \wedge \ \text{msg\_zhops } m) \rightarrow \text{onl } \text{Gamma}_\text{AODV} (\lambda(\_ , l). }\text{osn\_rreq' } a)"
proof (rule invariant\_weakenE \ OF \ osn\_rreq)
fix \ a
  assume "recvmsg (λm. \text{rreq\_rrep\_sn } m \ \wedge \ \text{msg\_zhops } m) \ a"
thus "recvmsg \text{rreq\_rrep\_sn } a" by (cases \ a) simp_all
qed

lemma dsn\_rrep:
"paodv } i \models (\text{recvmsg } \text{rreq\_rrep\_sn } \rightarrow \text{onl } \text{Gamma}_\text{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 \models (\text{recvmsg } (\lambda m. \text{rreq\_rrep\_sn } m \ \wedge \ \text{msg\_zhops } m) \rightarrow \text{onl } \text{Gamma}_\text{AODV} (\lambda(\_ , l). }\text{dsn\_rrep' } a)"
proof (rule invariant\_weakenE \ OF \ dsn\_rrep)
fix \ a
  assume "recvmsg (λm. \text{rreq\_rrep\_sn } m \ \wedge \ \text{msg\_zhops } m) \ a"
thus "recvmsg \text{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 \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 \in kD(rt \xi). (sqn (rt \xi) dip = 0 \rightarrow sqnf (rt \xi) dip = unk)
    \land (sqnf (rt \xi) dip = unk \rightarrow \text{the} (dhops (rt \xi) dip) = 1)
    \land \text{the} (dhops (rt \xi) dip) = 1 \rightarrow \text{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 \rightarrow 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 dip :: ip
    assume "\forall dip \in kD(rt \xi).
      (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) = Suc 0)
      \land
      (\text{the} (dhops rt dip) = Suc 0 \rightarrow \text{the} (nhop rt dip) = dip)"
    and "hops = 0 \rightarrow sip = dip"
    and "Suc 0 \leq dsn"
    and "ip \neq dip \rightarrow dip \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!: bspec)
    note prreq_ok1 [simp] = this
  }
  { fix ip dsn hops sip oip rt dip
    assume "\forall dip \in kD(rt \xi).
      (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) = Suc 0)
      \land
      (\text{the} (dhops rt dip) = 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
      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 \xi).
      (sqn rt dip = 0 \rightarrow \pi_3 (\text{the} (rt dip)) = unk) \land
      \ldots"
  }

49
\[
\pi_3(\text{the (rt dip)}) = \text{unk} \rightarrow (\text{dhops rt dip}) = \text{Suc 0} \land \\
(\text{the (dhops rt dip)}) = \text{Suc 0} \rightarrow (\text{nhop rt dip}) = \text{dip})
\]

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

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

by \text{- (rule update_cases, auto simp add: sqn_def dest!: bspec)}

\}}

\text{note prreq_ok3 [simp] = this}

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

by \text{- (rule update_cases, simp_all add: sqnf_def sqn_def)}

\} \text{note prreq_ok4 [simp] = this}

\text{have prreq_ok5 [simp]: "}\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} \land \\
(\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, {})) 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}) = \text{dip})"

by \text{- (rule update_cases, simp_all add: sqn_def sqn_def)}

\} \text{note prreq_ok6 [simp]: "}\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} \land \\
(\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, {})) 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}) = \text{dip})"

by \text{- (rule update_cases, simp_all add: sqn_def sqn_def)}

\text{show ?thesis by (inv_cterms inv add: onl_invariant_sterms_TT [OF aodv_wf addpreRT_welldefined]}
\text{onl_invariant_sterms [OF aodv_wf hop_count_zero_oip_dip_sip']} 
\text{seq_step_invariant_sterms_TT [OF sqns_increase aodv_wf aodv_trans]}
\text{onl_invariant_sterms [OF aodv_wf osn_rreq']}
\text{onl_invariant_sterms [OF aodv_wf dsn_rrep']} \text{clarsimp+}
\text{qed}

\text{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 \in \text{kD rt}. (\text{sqn (rt dip)} = 0 \rightarrow (\text{sqn (rt dip)} = \text{unk}
\land \\
(\text{the (dhops (rt dip)}) = 1
\land \\
(\text{the (nhop (rt dip)}) = \text{dip}))")"

by \text{(rule invariant_weakenE [OF zero_seq_unk_hops_one']) auto}

\text{lemma kD_unk_or_atleast_one:
"paodv i |= (recvmsg rreq_rrep_sn \rightarrow ) onl \Gamma_{AODV} (\lambda(\xi, 1).}
\forall dip \in \text{kD rt}. \pi_3(\text{(the (rt dip)}) = \text{unk} \land 1 \leq \pi_2(\text{(the (rt dip)})")

proof -
\{ \text{fix sip rt dsn1 dsn2 dsk1 dsk2 flag1 flag2 hops1 hops2 nhip1 nhip2 pre1 pre2}
\text{assume "dsk1 = unk \lor Suc 0 \leq dsn2"}
\text{hence "\pi_3(\text{(the (update rt sip (dsn1, dsk1, flag1, hops1, nhip1, pre1) sip)}) = unk}
\land \\
(\text{Suc 0} \leq \text{sqn (update rt sip (dsn2, dsk2, flag2, hops2, nhip2, pre2)) sip}"

\text{unfolding update_def by (cases "dsk1 =unk") (clarsimp split: option.split)+
}\text{note fromsip [simp] = this}
\} \text{fix dip sip rt dsn1 dsn2 dsk1 dsk2 flag1 flag2 hops1 hops2 nhip1 nhip2 pre1 pre2}
\text{assume allkd: "\forall dip \in \text{kD (rt).} \pi_3(\text{(the (rt dip)}) = unk \lor Suc 0 \leq sqn rt dip"}
\text{and **: "dsk1 = unk \lor Suc 0 \leq dsn2"}
\text{have "\forall dip \in \text{kD (rt).} \pi_3(\text{(the (update rt sip (dsn1, dsk1, flag1, hops1, nhip1, pre1) dip)}) = unk}
Proposition 7.13

lemma rreq_rrep_sn_any_step_invariant:
  "paodv \i\ \|=\ A (recvmsg rreq_rrep_sn \rightarrow onll Γ\AODV (λ(_, a, _). ancast rreq_rrep_sn a))"
proof -
  have sqnf_kno: "paodv \i\ \|= onl Γ\AODV (λ(ξ, 1). ancast rreq_rrep_sn (λξ. sqn (rt ξ) (dip ξ) = kno))"
    by (inv_cterms inv add: onl_invariant_sterms_TT [OF aodv_wf addpreRT_welldefined]
      onl_invariant_sterms [OF aodv_wf osn_rreq]
      onl_invariant_sterms [OF aodv_wf dsn_rrep]
      simp add: proj3_inv proj2_eq_sqn)
qed

51
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 [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 |= onl Γ_{AODV} (λ(ξ, 1).
                   (1 ∈ {PRreq:-3, PRreq:-4, PRreq:-15, PRreq:-27}
                   ⇒ oip ξ ∈ kD(rt ξ)
                   ∧ (sqn (rt ξ) (oip ξ) > (osn 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 "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 |= onl Γ_{AODV} (λ(ξ, 1).
                   (1 ∈ {PAodv:-15, PPkt:-7, PRreq:-9, PRreq:-21, PRreq:-10, PRreq:-1} →
                   (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_prrep]
      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 |= onl Γ_{AODV} (λ(ξ, 1).
                   (1 ∈ {PAodv:-15, PPkt:-7, PRreq:-9, PRreq:-21, PRreq:-10, PRreq:-1} →
                   (∀ip ∈ dom(dests ξ). ip ∈ vD(rt ξ)))
                   ∧ (1 ∈ {PAodv:-16..PAodv:-19}
                       ∪ {PPkt:-8..PPkt:-11}
                       ∪ {PRreq:-10..PRreq:-13})
                   ∧ (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))))"
lemma dests_vD_inc_sqn:
\[
\text{"paodv i \vdash } \Gamma_{\text{AODV}} (\lambda(\xi, 1). \langle \forall \text{ip} \in \text{dom(dests } \xi)\rangle. \text{ip} \in V\text{rt(rt } \xi) \wedge \text{(dests } \xi \text{ip) = inc (sqn (rt } \xi) \text{ip)}) \\
\wedge (l = \text{PPkt-:14} \rightarrow \text{dip } \xi \in D\text{rt(rt } \xi)))"
\]
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 \((\text{rip}, \text{rsn})\) is assigned to the variable \text{dests} of type \text{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 \((\text{rip}, \text{rsn})\) for each destination \text{rip}: guaranteed by type.

lemma dests_vD_inc_sqn:
\[
\text{"paodv i \vdash } \Gamma_{\text{AODV}} (\lambda(\xi, 1). \langle \forall \text{ip} \in \text{dom(dests } \xi)\rangle. \text{ip} \in V\text{rt(rt } \xi) \wedge \text{(dests } \xi \text{ip) = inc (sqn (rt } \xi) \text{ip)}) \\
\wedge (l = \text{PPkt-:14} \rightarrow \text{dip } \xi \in D\text{rt(rt } \xi)))"
\]
by inv_cterms (clarsimp split: split_if_asm option.split_asm)+

Proposition 7.27

lemma route_tables_fresher:
\[
\text{"paodv i \vdash } \Gamma_{\text{AODV}} (\lambda((\xi, _), _, (\xi', _)). \forall dip \in kD(\text{rt } \xi). \text{rt } \xi \subseteq dip \text{ rt } \xi')"
\]

proof (inv_cterms inv add:
\[
\begin{align*}
onl\text{invariant_sterms} \{OF aodv_wf dests_vD_inc_sqn \{THEN invariant_restrict_inD\}\} \\
onl\text{invariant_sterms} \{OF aodv_wf hop_count_positive \{THEN invariant_restrict_inD\}\} \\
onl\text{invariant_sterms} \{OF aodv_wf osn_rreq\} \\
onl\text{invariant_sterms} \{OF aodv_wf dsn_rrep\} \\
onl\text{invariant_sterms} \{OF aodv_wf addpreRT_welldefined \{THEN invariant_restrict_inD\}\}
\end{align*}
\]
fix \(\xi\) pp \(p'\)
assume "\((\xi, pp) \in \text{reachable (paodv i) (recvmsg rreq_rrep_sn)}\)"
and "\(\{\text{PPreq-:2}\}[(\xi, rt := \text{update (rt } \xi) \text{ (oip } \xi) \text{ (osn } \xi, \text{kno, val, Suc (hops } \xi), \text{sip } \xi, \{\})])p' \in \text{sterms } \Gamma_{\text{AODV}} pp"
and "Suc 0 \leq \text{osn } \xi"
and *: "\(\forall dip \in kD (rt } \xi). \text{Suc 0} \leq \text{the (dhops (rt } \xi) dip)\)"

show "\(\forall dip \in kD (rt } \xi). \text{rt } \xi \subseteq dip \text{ update (rt } \xi) \text{ (oip } \xi) \text{ (osn } \xi, \text{kno, val, Suc (hops } \xi), \text{sip } \xi, \{\})\)"
proof
fix ip
assume "ip \in kD (rt } \xi)"
moreover with * have "1 \leq \text{the (dhops (rt } \xi) ip)" by simp
moreover from Suc 0 \leq \text{osn } \xi:
have "\text{update_arg wf (osn } \xi, \text{kno, val, Suc (hops } \xi), \text{sip } \xi, \{\})" ..
ultimately show "\(rt } \xi \subseteq ip \text{ update (rt } \xi) \text{ (oip } \xi) \text{ (osn } \xi, \text{kno, val, Suc (hops } \xi), \text{sip } \xi, \{\})\)"
by (rule rt_fresher_update)

qed

next

fix \(\xi\) pp \(p'\)
assume "\((\xi, pp) \in \text{reachable (paodv i) (recvmsg rreq_rrep_sn)}\)"
and "\(\{\text{PPreq-:1}\}[(\xi, rt := \text{update (rt } \xi) \text{ (dip } \xi) \text{ (dsn } \xi, \text{kno, val, Suc (hops } \xi), \text{sip } \xi, \{\})])p' \in \text{sterms } \Gamma_{\text{AODV}} pp"
and "Suc 0 \leq \text{dsn } \xi"
and *: "\(\forall dip \in kD (rt } \xi). \text{Suc 0} \leq \text{the (dhops (rt } \xi) dip)\)"
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

theory Quality_Increases
imports Aodv_Predicates 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)"
      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 \langle rt (\sigma \ i) \sqsubseteq dip \ rt (\sigma \ nhip) \rangle \ show ?thesis ..
qed

lemma kD_quality_increases [elim]:
  assumes "i\in kD(rt (\xi))"
  and "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 "quality_increases \xi \xi'"
  shows "i\in kD(rt (\xi')) \land nsqn (rt (\xi)) \ i \leq 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 "nsqn (rt (\xi)) \ i \leq 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 "quality_increases \xi \xi'"
  shows "nsqn (rt (\xi)) \ i \leq 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 nsqn (rt (\xi)) \ i"
  and "quality_increases \xi \xi'"
  shows "i\in kD(rt (\xi')) \land s \leq nsqn (rt (\xi')) \ i"
proof
  from i\in kD(rt (\xi)) and (quality_increases \xi \xi') show "i\in kD(rt (\xi'))" ..
  next
  from i\in kD(rt (\xi)) and (quality_increases \xi \xi') have "nsqn (rt (\xi)) \ i \leq nsqn (rt (\xi')) \ i" ..
  with s \leq nsqn (rt (\xi)) \ i show "s \leq nsqn (rt (\xi')) \ i" by (rule le_trans)
qed

lemma nsqn_quality_increases_nsqn_lt_lt [elim]:
  assumes "i\in kD(rt (\xi))"
  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 nsqn_quality_increases_dhops [elim]:
  assumes "i\in kD(rt (\xi))"
  and "quality_increases \xi \xi'"
  and "nsqn (rt (\xi)) \ i = nsqn (rt (\xi')) \ i"
  shows "the (dhops (rt (\xi)) \ i) \geq the (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 "quality_increases \xi \xi'"
  and "s = nsqn (rt (\xi)) \ i"
  shows "s \leq nsqn (rt (\xi')) \ i" by (simp, clarsimp)
qed

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))" 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 "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) 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 ∈ D(rt (σ' sip))"
hence "the (flag (rt (σ' sip)) ip) = inv" ..
with :sn = nsqn (rt (σ' sip)) ip 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)) ip show thesis
  by simp

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 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]:
"orevmsg (msg_fresh) σ m =⇒ recvmsg rreq_rrep_sn m" 
by (cases m) simp_all 

lemma rreq_misq_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_misq (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 ⟨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 \texttt{assms(1)} have "osn \leq sqn ?rt oip" by auto 
thus "osn \leq nsqn (rt (\sigma sip)) oip"

proof (rule nat_le_eq_or_lt)
  assume "osn < sqn ?rt oip"
  hence "osn \leq sqn ?rt oip - 1" by simp
  also have "... \leq nsqn ?rt oip" by (rule sqn_nsqn)
  finally show "osn \leq nsqn ?rt oip".

next
  assume "osn = sqn ?rt oip"
  with \texttt{assms(1)} and \langle \sigma sip \neq oip \rangle have "oip \in kD(?rt)"
  and "the (flag ?rt oip) = val" by auto
  hence "osn \leq nsqn ?rt oip".

qed 
qed simp

lemma \texttt{rrep_nsqn_is_fresh [simp]}:
 fixes \sigma msg hops dip dsn oip sip
 assumes "rreq_rrep_fresh (rt (\sigma sip)) (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 \texttt{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 \texttt{assms} show "msg_fresh \sigma ?msg"
  by clarsimp

qed

lemma \texttt{rerr_nsqn_is_fresh [simp]}:
 fixes \sigma msg dests sip
 assumes "rerr_invalid (rt (\sigma sip)) (Rerr dests sip)"
 shows "msg_fresh \sigma (Rerr dests sip)"
  (is "msg_fresh \sigma ?msg")

proof -
  let ?rt = "rt (\sigma sip)"
  from \texttt{assms} have *: "(\forall rip \in \text{dom(dests)}. (rip \in iD(rt (\sigma sip))
  \land the (dests rip) = sqn (rt (\sigma sip)) rip))"
  by clarsimp
  have "(\forall rip \in \text{dom(dests)}. (rip \in kD(rt (\sigma sip))
  \land the (dests rip) - 1 \leq nsqn (rt (\sigma sip)) rip))"
  proof
    fix rip
    assume "rip \in \text{dom dests}"
    with * have "rip \in iD(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 "... \leq nsqn (rt (\sigma sip)) rip" by (rule sqn_nsqn)
    finally have "the (dests rip) - 1 \leq nsqn (rt (\sigma sip)) rip".
    with \langle rip \in iD(rt (\sigma sip)) \rangle
    show "rip \in kD(rt (\sigma sip)) \land the (dests rip) - 1 \leq nsqn (rt (\sigma sip)) rip"
    by clarsimp
    qed
    thus "msg_fresh \sigma ?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 osn sip
  assume [simp]: "m = Rreq hops rreqid dip dsn 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

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 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 by rule simp_all
lemma oaodv_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' ⇒ labels Γ_AODV p = {PAodv-0}"
unfolding σ_AODV'_def by simp
lemma oaodv_init_kD_empty [simp]: "(σ, p) ∈ σ_AODV' ⇒ kD (rt (σ i)) = {}"
unfolding σ_AODV'_def by simp
lemma oaodv_init_vD_empty [simp]: "(σ, p) ∈ σ_AODV' ⇒ vD (rt (σ i)) = {}"
unfolding σ_AODV'_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

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 otherI)
    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 otherI)
      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 |\|= (recvmsg 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) (recvmsg rreq_rrep_sn)"
      and tr: "(s, a, s') ∈ trans (paodv i)"

70
and rm: \texttt{"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) ⪯ 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', _)). ∀ dip∈kD (rt (ξ) ⊆ kD (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 aodv\_trans aodv\_trans, simplified seqll\_onll\_swap]

lemma quality\_increases:
"opaodv i \models (other\_with quality\_increases {i} (orecvmsg (λ_. rreq\_rrep\_sn)),
other quality\_increases {i} \rightarrow)

"onll Γ\_AODV (λ((σ, _), _, (σ', _)). ∀ j. quality\_increases (σ j) (σ' j))"
(is "\_ \models (??S, _ → ) _")

proof (rule onll\_ostep\_invariantI, simp)

fix σ p a σ' p'

assumption or: "((σ, p) ∈ reachable (opaodv i) ??S (other quality\_increases {i})))"

and ll: "1 ∈ labels Γ\_AODV p"

and "??S σ σ' a"

and tr: "((σ, p), a, (σ', p')) ∈ oseqp\_sos Γ\_AODV i"

and ll': "1' ∈ 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) ∈ 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 (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!: other\_with\_syncD)

qed

lemma rreq\_rrep\_nsqn\_fresh\_any\_step\_invariant:
"opaodv i \models (act (recvmsg rreq\_rrep\_sn), other A {i} \rightarrow)

"onll Γ\_AODV (λ((σ, _), a, _). anycast (msg\_fresh σ) a)"

proof (rule ostep\_invariantI, simp del: act\_simp)

fix σ p a σ' p'

assumption or: "((σ, p) ∈ reachable (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 "1∈labels Γ\_AODV p" and "1'∈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 \models (act (recvmsg rreq\_rrep\_sn), other A {i} \rightarrow)"
by \(\text{rule ostep_invariant_weakenE} \ [\text{OF open_seq_step_invariant} \ [\text{OF rreq_rrep_fresh_any_step_invariant initiali_aodv}, \\\text{simplified seqll_onll_swap}]])\) auto

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

by \(\text{rule ostep_invariant_weakenE} \ [\text{OF open_seq_step_invariant} \ [\text{OF rreq_rrep_fresh_any_step_invariant initiali_aodv}, \\\text{simplified seqll_onll_swap}]])\) auto

thus \(?thesis\)

using \(\| \in \text{labels} \Gamma_{AODV} p\) and \(\|' \in \text{labels} \Gamma_{AODV} p'\) by auto

qed

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

proof -

have "\(\text{opaodv} i \models A \ (\text{act} \ (\text{recvmsg} \ rreq_rrep_sn), \ \text{other} A \ {i} \rightarrow)\)

\(\text{onll} \Gamma_{AODV} \ (\text{seqll} \ i \ (\lambda((\xi, _), a, _). \ \text{anycast} \ (\text{rreq_rrep_sn} a)))\)

by \(\text{rule ostep_invariant_weakenE} \ [\text{OF open_seq_step_invariant} \ [\text{OF rreq_rrep_sn_any_step_invariant initiali_aodv}, \\\text{oaodv_trans aodv_trans}, \\\\text{simplified seqll_onll_swap}]])\) auto

thus \(?thesis\)

using \(\| \in \text{labels} \Gamma_{AODV} p\) and \(\|' \in \text{labels} \Gamma_{AODV} p'\) by auto

qed

moreover have "anycast \((\lambda \sigma. \text{msg}_\text{fresh} \sigma)\) \ a"

by \(\text{simp_all add: anycast_def} \\\\\\text{del: msg_fresh} \\\\\text{split: seq.action.split_asm msg.split_asm})\) simp_all

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

by auto

qed

lemma oreceived_rreq_rrep_msg-fresh_inv:

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

\(\text{onll} \Gamma_{AODV} \ (\lambda(\sigma, l). \ l \in \{PAodv\rightarrow\} \rightarrow \text{msg}_\text{fresh} \sigma \ (\text{msg} \ (\sigma \ i)))\)"

proof \(\text{rule oreceived_msg_inv})

fix \(\sigma, \sigma'\)

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

and "other quality_increases \ {i} \ \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 \((\sigma \ i := \sigma \ i(\text{msg} := \text{m})) \ m\)"

proof (cases \(m\))
fix dests sip
assume "m = Rerr dests sip"
with msg_fresh \(\sigma \ m\) show ?thesis
  by auto
qed auto

qed

lemma quality_increases_nsqn_fresh:
"opaodv \(\vdash_{A}\) \((\text{otherwith quality_increases} \ \{i\} \ (\text{orecvmsg msg_fresh}),\)
other_quality_increases \(\{i\} \rightarrow\)
onl \(\Gamma_{AODV} \ (\lambda((\sigma, \_), \_, (\sigma', \_)). \ \forall j. \text{quality_increases} \ (\sigma \ j) \ (\sigma' \ j))\)"
by (rule ostep_invariant_weakenE [OF quality_increases]) auto

lemma osn_rreq:
"opaodv \(\vdash_{A}\) \((\text{otherwith quality_increases} \ \{i\} \ (\text{orecvmsg msg_fresh}),\)
other_quality_increases \(\{i\} \rightarrow\)
onl \(\Gamma_{AODV} \ (\text{seql} \ (\lambda(\xi, \ l). \ 1 \in \{\text{PAodv}:-4, \text{PAodv}:-5\} \cup \{\text{PRreq}:-n \ \text{ln} \ \text{True}\} \rightarrow \ 1 \leq \text{osn} \ (\xi))\)"
by (rule oinvariant_weakenE [OF open_seq_invariant [OF osn_rreq initiali_aodv]])
(auto simp: seql_onl_swap)

lemma rreq_sip:
"opaodv \(\vdash_{A}\) \((\text{otherwith quality_increases} \ \{i\} \ (\text{orecvmsg msg_fresh}),\)
other_quality_increases \(\{i\} \rightarrow\)
onl \(\Gamma_{AODV} \ (\lambda(\sigma, \ l). \ (1 \in \{\text{PAodv}:-4, \text{PAodv}:-5, \text{PRreq}:-0, \text{PRreq}:-2\} \ \& \ \text{sig} \ (\sigma \ i) \neq \text{oip} \ (\sigma \ i))\)
\(\rightarrow\) oip \(\ (\sigma \ i) \in \text{kD}(\text{rt} \ (\sigma \ (\text{sip} \ (\sigma \ i))))\)
\(\&\) nsqn \(\ (\text{rt} \ (\sigma \ (\text{sip} \ (\sigma \ i))))\) \ (oip \(\ (\sigma \ i)) \geq \text{osn} \ (\sigma \ i))\)
\(\&\) \(\ (\text{nsqn} \ (\text{rt} \ (\sigma \ (\text{sip} \ (\sigma \ i))))\) \ (\text{oip} \ (\sigma \ i)) = \text{osn} \ (\sigma \ i))\)
\(\rightarrow\) \(\ (\text{hops} \ (\sigma \ i) \ \geq \ (\text{dhops} \ (\text{rt} \ (\sigma \ (\text{sip} \ (\sigma \ i))))\) \ (\text{oip} \ (\sigma \ i)))\)
\(\vee\) the \(\ (\text{flag} \ (\text{rt} \ (\sigma \ (\text{sip} \ (\sigma \ i))))\) \ (\text{oip} \ (\sigma \ i))) = \text{inv}\))"
(is "_ \(\vdash_{(\Sigma, \ ?U \rightarrow \ _})\)"

proof (inv_cterms inv add: oseq_step_invariant_sterms [OF quality_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 osn_rreq]
simp add: seqlsimp
simp del: One_nat_def, rule impI)
fix \(\sigma \ \sigma' \ p \ l\)
assume ""(\sigma, \ p) \in \text{oreachable} \ (opaodv \ i) \ ?S \ ?U"
and "\ l \in \text{labels} \ \Gamma_{AODV} \ p"\nand pre:
"(1 = \text{PAodv}:-4 \ \lor \ 1 = \text{PAodv}:-5 \ \lor \ 1 = \text{PRreq}:-0 \ \lor \ 1 = \text{PRreq}:-2) \ \& \ \text{sig} \ (\sigma \ i) \neq \text{oip} \ (\sigma \ i))\)
\(\rightarrow\) oip \(\ (\sigma \ i) \in \text{kD}(\text{rt} \ (\sigma \ (\text{sip} \ (\sigma \ i))))\)
\(\&\) osn \(\ (\sigma \ i) \leq \text{nsqn} \ (\text{rt} \ (\sigma \ (\text{sip} \ (\sigma \ i))))\) \ (oip \ (\sigma \ i))\)
\(\&\) \(\ (\text{nsqn} \ (\text{rt} \ (\sigma \ (\text{sip} \ (\sigma \ i))))\) \ (\text{oip} \ (\sigma \ i)) = \text{osn} \ (\sigma \ i))\)
\(\rightarrow\) \(\ (\text{dhops} \ (\text{rt} \ (\sigma \ (\text{sip} \ (\sigma \ i))))\) \ (\text{oip} \ (\sigma \ i))) \ \leq \text{hops} \ (\sigma \ i))\)
\(\vee\) the \(\ (\text{flag} \ (\text{rt} \ (\sigma \ (\text{sip} \ (\sigma \ i))))\) \ (\text{oip} \ (\sigma \ i))) = \text{inv}\))"
and "other quality_increases \(\{i\} \ \sigma \ \sigma'\)"
and hyp: "(1=PAodv:-4 \ \lor 1=PAodv:-5 \ \lor 1=PRreq:-0 \ \lor 1=PRreq:-2) \ \& \ \text{sig} \ (\sigma' \ i) \neq \text{oip} \ (\sigma' \ i))"
(is "\text{labels} \ \& \ \text{sig} \ (\sigma' \ i) \neq \text{oip} \ (\sigma' \ i))"
from this(4) have "\(\sigma' \ i = \sigma \ i\)" ..
with hyp have hyp': "\text{labels} \ \& \ \text{sig} \ (\sigma \ i) \neq \text{oip} \ (\sigma \ i))" by simp
show oip \(\ (\sigma' \ i) \in \text{kD}(\text{rt} \ (\sigma' \ (\text{sip} \ (\sigma' \ i))))\)
\(\&\) osn \(\ (\sigma' \ i) \leq \text{nsqn} \ (\text{rt} \ (\sigma' \ (\text{sip} \ (\sigma' \ i))))\) \ (oip \ (\sigma' \ i))\)
\(\&\) \(\ (\text{nsqn} \ (\text{rt} \ (\sigma' \ (\text{sip} \ (\sigma' \ i))))\) \ (\text{oip} \ (\sigma' \ i)) = \text{osn} \ (\sigma \ i))\)
\(\rightarrow\) \(\ (\text{dhops} \ (\text{rt} \ (\sigma' \ (\text{sip} \ (\sigma' \ i))))\) \ (\text{oip} \ (\sigma' \ i))) \ \leq \text{hops} \ (\sigma' \ i))\)
\(\vee\) the \(\ (\text{flag} \ (\text{rt} \ (\sigma' \ (\text{sip} \ (\sigma' \ i))))\) \ (\text{oip} \ (\sigma' \ i))) = \text{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: simp (σ i) ≠ i)
moreover from (σ, p) ∈ reachable (opaodv i) ?S ?U l ∈ labels ΓAODV p' and hyp
have "1 ≤ osn (σ i)"
  by (auto elim!: quality increases rreq rrep props)
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 dsn_rrep:
"opaodv i ⊏ (otherwith quality increases {i} (orecvmsg msg_fresh),
other quality increases {i} →)
onsl ΓAODV (seql i (λ(τ, 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 (sip (σ i)))) (dip (σ i))))
  ∨ the (flag (rt (σ (sip (σ i)))) (dip (σ i))) = inv))"
(is "_ ⊏ (?S, ?U → _)")

proof (intro terms inv add: oseq_step_invariant_sterms [OF quality increases nsqn fresh aodv_wf
oaodv_trans]
onl_on_orvariant_sterms [OF aodv_wf oreceived_rreq_rrep nsqn fresh_inv]
onl_on_orvariant_sterms [OF aodv_wf dsn_rrep]
simp del: One_nat_def, rule impl)

fix σ σ' p l
assume "σ σ' p l ∈ labels ΓAODV p" and "l ∈ labels ΓAODV p"
and pre:
"(1 = PAodv-6 ∨ 1 = PAodv-7 ∨ l = PRrep-0 ∨ l = 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)"

and "other quality increases {i} σ σ'"
and hyp: "(1=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))))
  ∧ nsqn (rt (σ' (sip (σ' i)))) (dip (σ' i)) = inv"
lemma rerr_sip:
  "opaodv i \{| (otherwith quality_increases \{i\} (orecvmsg msg_fresh),
    other_quality_increases \{i\} \rightarrow
  \}
  onl \Gamma_{AODV} \{\lambda(\sigma, 1),
    1 \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 "\_ \rightarrow (\sigma, \sigma) \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)"
    and rsn - 1 \leq nsqn (rt \sigma sip) rip"
  by (auto dest!: bspec)
  from qinc have "quality_increases (\sigma sip) (\sigma' sip)" ..
  have "rip \in kD (rt \sigma sip) \land rsn - 1 \leq nsqn (rt \sigma sip) rip"
  proof
    from \{rip\in kD (rt \sigma sip)\} and \{quality_increases \sigma sip \sigma' sip\}
    show "rip \in kD (rt \sigma sip)" ..
  next
    from \{rip\in kD (rt \sigma sip)\} and \{quality_increases \sigma sip \sigma' sip\}
    have "nsqn (rt \sigma sip) rip \leq nsqn (rt \sigma sip) rip" ..
    with rsn - 1 \leq nsqn (rt \sigma sip) rip show "rsn - 1 \leq nsqn (rt \sigma sip) rip"
  by (rule le_trans)
  qed
  
  note partial = this

  show \{thesis inv add: oseq_step_invariant_sterms [OF equality_increases_nsqn_fresh aodv_wf
    oaodv_trans]
    onl_o invariant_sterms [OF aodv_wf oreceived_rreq_rrep_nsqn_fresh_inv]
    other_quality_increases other_localD
    simp del: One_nat_def, intro conjl)"
lemma prerr_guard: "paodv i \(\Rightarrow\) onl \(\Gamma\) \(\Lambda\) (\(\xi\), 1). (1 = P\text{Err}):-1
\[
\rightarrow (\forall ip \in \text{dom}(\text{dests} \ \xi). \ ip \in \text{V}(\text{rt} \ \xi)
\wedge \text{the} (\text{nhop} (\text{rt} \ \xi) \ ip) = \text{sip} \ \xi
\wedge \text{sqn} (\text{rt} \ \xi) \ ip < \text{the} (\text{dests} \ \xi \ ip)))"
by (inv\_ctersms) (clarsimp split: option.split_asm split_if_asm)

lemmas oaddpreRT\_welldefined =
open_seq_invariant [OF addpreRT\_welldefined initiali\_aodv oadv\_trans aadv\_trans,
simplified seql\_onl\_swap,
THEN oinvariant\_anyact]

lemmas odests\_vD\_inc\_sqn =
open_seq_invariant [OF dests\_vD\_inc\_sqn initiali\_aodv oadv\_trans aadv\_trans,
simplified seql\_onl\_swap,
THEN oinvariant\_anyact]

lemmas oprerr\_guard =
open_seq_invariant [OF prerr\_guard initiali\_aodv oadv\_trans aadv\_trans,
simplified seql\_onl\_swap,
THEN oinvariant\_anyact]

Proposition 7.28

lemma seq\_compare\_next\_hop':
"opaodv i \(\Rightarrow\) (otherwith quality\_increases \{i\} (orec\_msg msg\_fresh),
other quality\_increases \{i\} \(\Rightarrow\) onl \(\Gamma\) \(\Lambda\) (\(\sigma\), _).
\[
\forall dip. \ \text{let} \ nhip = \text{the} (\text{nhop} (\text{rt} \ (\sigma \ i)) dip)
\in dip \in \text{kD}(\text{rt} \ (\sigma \ i)) \wedge nhip \neq dip \rightarrow
\]
dip \in \text{kD}(\text{rt} \ (\sigma \ nhip)) \wedge \text{sqn} (\text{rt} \ (\sigma \ i)) dip \leq \text{sqn} (\text{rt} \ (\sigma \ nhip)) dip"
(is "\(\_ \Rightarrow \text{(?S, ?U \rightarrow \_)} \ _\)"
proof -

{ fix nhop and \(\sigma \sigma' \) :: "ip \(\Rightarrow\) state"
assume pre: "\(\forall dip \in \text{kD}(\text{rt} \ (\sigma \ i))\). nhop dip \neq dip \rightarrow
\]
dip \in \text{kD}(\text{rt} \ (\sigma \ (\text{nhop} dip))) \wedge \text{sqn} (\text{rt} \ (\sigma \ i)) dip \leq \text{sqn} (\text{rt} \ (\sigma \ (\text{nhop} dip))) dip"
and qinc: "\(\forall j. \text{quality\_increases} \ (\sigma \ j) \ (\sigma' \ j)\)"
have "\(\forall dip \in \text{kD}(\text{rt} \ (\sigma \ i))\). nhop dip \neq dip \rightarrow
\]
dip \in \text{kD}(\text{rt} \ (\sigma \ (\text{nhop} dip))) \wedge \text{sqn} (\text{rt} \ (\sigma \ i)) dip \leq \text{sqn} (\text{rt} \ (\sigma' \ (\text{nhop} dip))) dip"
proof (intro ballI impI)
fix dip
assume "dip \in \text{kD}(\text{rt} \ (\sigma \ i))"
and "nhop dip \neq dip"
with pre have "dip \in \text{kD}(\text{rt} \ (\sigma \ (\text{nhop} dip)))"
and "\text{sqn} (\text{rt} \ (\sigma \ i)) dip \leq \text{sqn} (\text{rt} \ (\sigma \ (\text{nhop} dip))) dip"
by auto
from qinc have qinc\_nhop: "quality\_increases (\sigma \ (\text{nhop dip})) (\sigma' \ (\text{nhop dip}))" ..
with \(\forall dip \in \text{kD}(\text{rt} \ (\sigma \ i))\) have "dip \in \text{kD}(\text{rt} \ (\sigma' \ (\text{nhop dip})))" ..
moreover have "\text{sqn} (\text{rt} \ (\sigma \ i)) dip \leq \text{sqn} (\text{rt} \ (\sigma' \ (\text{nhop dip}))) dip"
proof -
from \(\forall dip \in \text{kD}(\text{rt} \ (\sigma \ (\text{nhop dip})))\) qinc\_nhop
have "\text{sqn} (\text{rt} \ (\sigma \ (\text{nhop dip}))) dip \leq \text{sqn} (\text{rt} \ (\sigma' \ (\text{nhop dip}))) dip" ..
with \(\forall dip \in \text{kD}(\text{rt} \ (\sigma \ i))\) dip \leq \text{sqn} (\text{rt} \ (\sigma \ (\text{nhop dip}))) dip show ?thesis
by simp
qed
ultimately show "dip \in \text{kD}(\text{rt} \ (\sigma' \ (\text{nhop dip})))
\wedge \text{sqn} (\text{rt} \ (\sigma \ i)) dip \leq \text{sqn} (\text{rt} \ (\sigma' \ (\text{nhop dip}))) dip" ..
qed
}

{ note basic = this

68
{ 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)))" 
  ∧ 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))))" 
  ∧ nisip: "∀i. 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)))" 
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)))" 
by (auto dest!: basic)

have "nsqn (invalidate (rt (σ i)) (dests (σ i))) dip ≤ nsqn (rt (σ′ (nhop 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)))" 
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′ (nhop dip)))" 
by auto
moreover from nisip 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)))" by auto
with qinc show "dsn - 1 ≤ nsqn (rt (σ′ (nhop dip)))" 
by simp (metis kD mandate_quality_changes)
qed ultimate 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)))⟩ 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))))" 
  ∧ 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))
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)))" 
by (auto dest!: basic)

have "nsqn (invalidate (rt (σ i)) (dests (σ i))) dip ≤ nsqn (rt (σ′ (nhop 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)))" 
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′ (nhop dip)))" 
by auto
moreover from nisip 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)))" by auto
with qinc show "dsn - 1 ≤ nsqn (rt (σ′ (nhop dip)))" 
by simp (metis kD mandate_quality_changes)
qed ultimate 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)))⟩ 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
}

proof
fix dip
assume "dip ∈ kD(rt (σ))" with a1 and a2
have "the (nhop (rt (σ)) dip) ≠ dip --> dip ∈ kD(rt (σ' (the (nhop (rt (σ)) dip)))) \nsqn (rt (σ)) dip ≤ \nsqn (rt (σ' (the (nhop (rt (σ)) dip)))) dip" by -(drule(1) basic, auto)
thus "?P dip" by (cases "dip = sip (σ)") auto
qed

} note nhop_update_sip = this

{ fix σ σ' oip sip osn hops
assume pre: "\forall dip ∈ kD(rt (σ)). \(\text{the} (\text{nhop} (rt (σ)) dip) \neq dip \rightarrow \text{dip} ∈ kD(rt (σ' (\text{the} (\text{nhop} (rt (σ)) dip))))\) \nsqn (rt (σ)) dip ≤ \nsqn (rt (σ' (the (nhop (rt (σ)) dip)))) dip" and qinc: "\forall j. quality_increases (σ j) (σ' j)"
and *: "sip ≠ oip \rightarrow oip ∈ kD(rt (σ sip)) \nsqn (rt (σ sip)) oip = osn \rightarrow the (dhops (rt (σ sip)) oip) ≤ hops ∨ the (flag (rt (σ sip)) oip) = inv"

from pre and qinc
have pre': "\forall dip ∈ kD(rt (σ)). \(\text{the} (\text{nhop} (rt (σ)) dip) \neq dip \rightarrow \text{dip} ∈ kD(rt (σ' (\text{the} (\text{nhop} (rt (σ)) dip))))\) \nsqn (rt (σ)) dip ≤ \nsqn (rt (σ' (the (nhop (rt (σ)) dip)))) dip" by (rule basic)

have "(the (update (rt (σ)) oip (osn, kno, val, Suc hops, sip, {})) oip) ≠ oip \rightarrow oip ∈ kD(rt (σ' (the (update (rt (σ)) oip (osn, kno, val, Suc hops, sip, {})) oip))) \nsqn (update (rt (σ)) oip (osn, kno, val, Suc hops, sip, {})) oip ≤ \nsqn (rt (σ' (the (update (rt (σ)) oip (osn, kno, val, Suc hops, sip, {})) oip))) oip" (is "?nhop_not_oip \rightarrow ?oip_in_kD \nsqn_le_nsqn")

proof (rule, split update_rt_split_asm)
assume "rt (σ) = update (rt (σ)) oip (osn, kno, val, Suc hops, sip, {})"
and "the (nhop (rt (σ)) oip) ≠ oip" with pre' show "?oip_in_kD \nsqn_le_nsqn" by auto
next
assume rtnot: "rt (σ) ≠ update (rt (σ)) oip (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 "?oip_in_kD \nsqn_le_nsqn" ..
qed
}

} note update1 = this

{ fix σ σ' oip sip osn hops
assume pre: "\forall dip ∈ kD(rt (σ)). \(\text{the} (\text{nhop} (rt (σ)) dip) \neq dip \rightarrow \text{dip} ∈ kD(rt (σ' (\text{the} (\text{nhop} (rt (σ)) dip))))\) \nsqn (rt (σ)) dip ≤ \nsqn (rt (σ' (the (nhop (rt (σ)) dip)))) dip" and qinc: "\forall j. quality_increases (σ j) (σ' j)"
and *: "sip ≠ oip \rightarrow oip ∈ kD(rt (σ sip)) \nsqn (rt (σ sip)) oip = osn \rightarrow the (dhops (rt (σ sip)) oip) ≤ hops ∨ the (flag (rt (σ sip)) oip) = inv"

from pre and qinc
have pre': "\forall dip ∈ kD(rt (σ)). \(\text{the} (\text{nhop} (rt (σ)) dip) \neq dip \rightarrow \text{dip} ∈ kD(rt (σ' (the (nhop (rt (σ)) dip))))\) \nsqn (rt (σ)) dip ≤ \nsqn (rt (σ' (the (nhop (rt (σ)) dip)))) dip" by (rule basic)
have "\(\forall dip \in kD(rt (\sigma i)). \)
\(\text{the (nhop (update (rt (\sigma i)) oip (osn, kno, val, Suc hops, sip, {})) dip) \neq dip} \)
\(\to dip \in kD(rt (\sigma' (the (nhop (update (rt (\sigma i)) oip (osn, kno, val, Suc hops, sip, {})) dip)))) \)
\(\wedge \text{nsqn (update (rt (\sigma i)) oip (osn, kno, val, Suc hops, sip, {})) dip \leq nsqn (rt (\sigma' (the (nhop (update (rt (\sigma i)) oip (osn, kno, val, Suc hops, sip, {})) dip))) dip}" 
(is "\(\forall dip \in kD(rt (\sigma i)). _ \to ?dip_in_kD dip \wedge ?nsqn_le_nsqn dip")

proof (intro ballI impI, split update_rt_split_asm)
fix dip
assume "dip \in kD(rt (\sigma i))"
and "the (nhop (rt (\sigma i)) dip) \neq dip"
and "rt (\sigma i) = update (rt (\sigma i)) oip (osn, kno, val, Suc hops, sip, {})"
with pre' show "?dip_in_kD dip \wedge ?nsqn_le_nsqn dip" by simp

next
fix dip
assume "dip \in kD(rt (\sigma i))"
and notdip: "the (nhop (update (rt (\sigma i)) oip (osn, kno, val, Suc hops, sip, {})) dip) \neq dip"
and rtnot: "rt (\sigma i) \neq update (rt (\sigma i)) oip (osn, kno, val, Suc hops, sip, {})"
show "?dip_in_kD dip \wedge ?nsqn_le_nsqn dip" proof (cases "dip = oip")
assume "dip \neq oip"
with pre' ⟨dip \in kD(rt (\sigma i))⟩ notdip *
show ?thesis byclarsimp
next
assume "dip = oip"
with rtnot qinc ⟨dip \in kD(rt (\sigma i))⟩ notdip *
have "?dip_in_kD dip" by simp (metis kD_quality_increases)
moreover from ⟨dip = oip⟩ rtnot qinc ⟨dip \in kD(rt (\sigma i))⟩ notdip *
have "?nsqn_le_nsqn dip" by simp (metis kD_nsqn_quality_increases_trans)
ultimately show ?thesis ..
qed
qed

} note update2 = this

have "\(\sigma \models ?S, ?U \to \) onl \(\Gamma_{\text{AODV}} (\lambda(\sigma, _). \)
\(\forall dip \in kD(rt (\sigma i)). \) the (nhop (rt (\sigma i)) dip) \neq dip 
\(\to dip \in kD(rt (\sigma' (the (nhop (rt (\sigma i)) dip)))) \)
\(\wedge \text{nsqn (rt (\sigma i)) dip \leq nsqn (rt (\sigma' (the (nhop (rt (\sigma i)) dip)))) dip})" 
by (inv_cterms inv add: oseq_step_invariant_sterms [OF oquality_increases_nsqn_fresh aodv_wf oaddpreRT_welldefined]
onl_invariant_sterms [OF aodv_wf odests_vD_inc_sqn]
onl_invariant_sterms [OF aodv_wf oprerr_guard]
onl_invariant_sterms [OF aodv_wf rreq_sip]
onl_invariant_sterms [OF aodv_wf rrep_sip]
onl_invariant_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) (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. ?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 ororeachable_fresh_ozero_seq_unk_hops_one:
  fixes dip
  assumes "(σ, p) ∈ ororeachable (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. ?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 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 ∈ D(rt (σ i))
      ∧ dip ∈ D(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 ∈ D(rt (σ i))
      ∧ dip ∈ D (rt (σ' (the (nhop (rt (σ i)) dip))))
      ∧ dip ∈ D (rt (σ' (the (nhop (rt (σ i)) dip))))"
\[
\begin{align*}
\wedge (\text{the (nhop (rt (\sigma i)) dip)}) & \neq \text{dip} \\
\rightarrow \text{rt (\sigma i)} & \sqsubseteq_{\text{dip}} \text{rt (\sigma'} \text{(the (nhop (rt (\sigma i)) dip)))}
\end{align*}
\]

**proof clarify**

**fix dip**

**assume a2:** "\(\text{dip} \in \nuD(\text{rt (\sigma i)})\)"

and **a3:** "\(\text{dip} \in \nuD (\text{rt (\sigma'} \text{(the (nhop (rt (\sigma i)) dip))))}\)"

and **a4:** "\(\text{the (nhop (rt (\sigma i)) dip)}) \neq \text{dip}\)"

**from ow have** "\(\forall j. j \neq i \rightarrow j = \sigma' j\)" by auto

**show** "\(\text{rt (\sigma i)} \sqsubseteq_{\text{dip}} \text{rt (\sigma'} \text{(the (nhop (rt (\sigma i)) dip))))}\)"

**proof (cases \("\text{the (nhop (rt (\sigma i)) dip)}) = i\)"

**assume \("\text{the (nhop (rt (\sigma i)) dip)}) = i\)"

with \(\text{dip} \in \nuD(\text{rt (\sigma i)})\) have "\(\text{dip} \in \nuD (\text{rt (\sigma (the (nhop (rt (\sigma i)) dip))))}\)" by simp

with a1 a2 a4 have "\(\text{dip} \in \nuD (\text{rt (\sigma (the (nhop (rt (\sigma i)) dip))))}\)" by simp

with (\(\text{the (nhop (rt (\sigma i)) dip)}) = i\) have "\(\text{rt (\sigma i)} \sqsubseteq_{\text{dip}} \text{rt (\sigma i)}\)" by simp

**hence False by simp**

**thus \(?\text{thesis} \ldots\)**

**next**

**assume \("\text{the (nhop (rt (\sigma i)) dip)}) \neq i\)"

with \(\forall j. j \neq i \rightarrow j = \sigma' j\)

have \(*:\ "\sigma (\text{the (nhop (rt (\sigma i)) dip)}) = \sigma' (\text{the (nhop (rt (\sigma i)) dip)})\)" by simp

with \(\text{dip} \in \nuD (\text{rt (\sigma (the (nhop (rt (\sigma i)) dip))))}\)

have "\(\text{dip} \in \nuD (\text{rt (\sigma (the (nhop (rt (\sigma i)) dip))))}\)" by simp

with a1 a2 a4 have "\(\text{dip} \in \nuD (\text{rt (\sigma (the (nhop (rt (\sigma i)) dip))))}\)" by simp

with \(* \text{show} \ ?\text{thesis} \ \text{by simp} \)

**qed**

**qed**

\}\ note basic = this

\{ fix \(\sigma \sigma' \ a \ \text{dip} \ \text{sip} \ i\)

**assume a1:** "\(\forall \text{dip. dip} \in \nuD(\text{rt (\sigma i)})\)

\(\wedge \text{dip} \in \nuD (\text{rt (\sigma (the (nhop (rt (\sigma i)) dip))))})\)

\(\wedge \text{the (nhop (rt (\sigma i)) dip)}) \neq \text{dip}\)

\(\rightarrow \text{rt (\sigma i)} \sqsubseteq_{\text{dip}} \text{rt (\sigma (the (nhop (rt (\sigma i)) dip))))}\)

and **ow:** "\(\forall i \sigma \sigma' \ a\)"

**have** "\(\forall \text{dip. dip} \in \nuD (\text{update (rt (\sigma i)) sip (0, unk, val, Suc 0, sip, {}))}\)

\(\wedge \text{dip} \in \nuD (\text{rt (\sigma'} \text{(the (nhop (update (rt (\sigma i)) sip (0, unk, val, Suc 0, sip, {})) dip))))}\)

\(\wedge \text{the (nhop (update (rt (\sigma i)) sip (0, unk, val, Suc 0, sip, {})) dip)}) \neq \text{dip}\)

\(\rightarrow \text{update (rt (\sigma i)) sip (0, unk, val, Suc 0, sip, {})}\)

\(\sqsubseteq_{\text{dip}} \text{rt (\sigma'} \text{(the (nhop (update (rt (\sigma i)) sip (0, unk, val, Suc 0, sip, {})) dip))))}\)

**proof clarify**

**fix dip**

**assume a2:** "\(\text{dip} \in \nuD (\text{update (rt (\sigma i)) sip (0, unk, val, Suc 0, sip, {}}))\)"

and **a3:** "\(\text{dip} \in \nuD (\text{rt (\sigma'} \text{(the (nhop (update (rt (\sigma i)) sip (0, unk, val, Suc 0, sip, {})) dip))))})\)"

and **a4:** "\(\text{the (nhop (update (rt (\sigma i)) sip (0, unk, val, Suc 0, sip, {})) dip)}) \neq \text{dip}\)"

**show** "\(\text{update (rt (\sigma i)) sip (0, unk, val, Suc 0, sip, {})}\)

\(\sqsubseteq_{\text{dip}} \text{rt (\sigma'} \text{(the (nhop (update (rt (\sigma i)) sip (0, unk, val, Suc 0, sip, {})) dip))))}\)"

**proof (cases \"\text{dip = sip}\")

**assume \"\text{dip = sip}\"

**with \(\text{the (nhop (update (rt (\sigma i)) sip (0, unk, val, Suc 0, sip, {})) dip)}) \neq \text{dip}\) have False by simp

**thus \(?\text{thesis} \ldots\)**

**next**

**assume [simp]: \"\text{dip} \neq \text{sip}\"

from a2 have "\(\text{dip} \in \nuD (\text{rt (\sigma i)}) \lor \text{dip} = \text{sip}\)"

by \(\text{rule \nuD_update_val}\)

with \(\text{dip} \neq \text{sip}\) have "\(\text{dip} \in \nuD (\text{rt (\sigma i)})\)" by simp

moreover from a3 have "\(\text{dip} \in \nuD (\text{rt (\sigma'} \text{(the (nhop (rt (\sigma i)) dip))))})\)" by simp

moreover from a4 have "\(\text{the (nhop (rt (\sigma i)) dip)}) \neq \text{dip}\)" by simp

ultimately have "\(\text{rt (\sigma i)} \sqsubseteq_{\text{dip}} \text{rt (\sigma'} \text{(the (nhop (rt (\sigma i)) dip))))}\)"

using a1 ow by - \(\text{(erule(1) basic, simp)}\)

with \(\text{dip} \neq \text{sip}\) show \(?\text{thesis}\)

by - \(\text{(erule rt_strictly_fresher_update_other, simp)}\)

**qed**
{ 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: "∀ 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 "∀ i. 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 "∀ i. j ≠ i → σ j = σ' j" have "rt (σ i) □ dip rt (σ' (nhop 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, {})))))" 
  and a4: "the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {}))) ≠ dip"
from ow have a5: "∀ i. j ≠ i → σ j = σ' j" by auto
show "∀ i. j ≠ i → σ j = σ' j" by auto
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 "∀ i. j ≠ i → σ j = σ' j" by metis
} note update_0_unk = this

have "∀ i. j ≠ i → σ j = σ' j" by auto
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 "∀ i. 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. 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
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 i)) 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)" ..

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)
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’ 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 dip = 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 show ?thesis
  using ⟨dip = oip⟩ 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∈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 (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 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' 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

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 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 "... ≤ sqn (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 ⟨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 \(\text{dip} \in kD(r (\sigma \ ?\text{nhip}))\)
by (rule strictly_fresher_quality_increases_right)
next
assume "\(\text{dip} \notin vD(r (\sigma \ ?\text{nhip}))\)"
with \(\text{dip} \in kD(r (\sigma \ ?\text{nhip}))\) have "\(\text{dip} \in iD(r (\sigma \ ?\text{nhip}))\)" ..
hence "the (flag \(r (\sigma \ ?\text{nhip})\) \(\text{dip}\)) = \text{inv}" by auto
have "\(\text{sqn} (r (\sigma \ ?\text{i})) \text{dip} \leq \text{sqn} (r (\sigma \ ?\text{nhip})) \text{dip}\)"
by (rule \text{nsqns})
also from \(\text{dip} \in vD(r (\sigma \ ?\text{nhip}))\)
have "... = \text{sqn} (r (\sigma \ ?\text{nhip})) \text{dip} - 1" ..
also have "... < \text{sqn} (r (\sigma \ ?\text{nhip})) \text{dip}\)"
proof -
from \(\forall j. \text{quality_increases} (\sigma j) (\sigma' j)\)
have "\(\text{quality_increases} (\sigma \ ?\text{nhip}) (\sigma' \ ?\text{nhip})\)" ..
hence "\(\forall ip. \text{sqn} (r (\sigma \ ?\text{nhip}) ip \leq \text{sqn} (r (\sigma' \ ?\text{nhip}) ip)\)" by auto
hence "\(\text{sqn} (r (\sigma \ ?\text{nhip})) \text{dip} \leq \text{sqn} (r (\sigma' \ ?\text{nhip})) \text{dip}\)" ..
with \(\exists 0 < \text{sqn} (r (\sigma \ ?\text{nhip})) \text{dip} \) show \(\text{thesis}\) by auto
qed
also have "... = \text{nsqn} (r (\sigma' \ ?\text{nhip})) \text{dip}\)"
proof (rule \text{vD_nsqn_sqn [THEN sym]})
from \(\text{dip} \in vD(r (\sigma' (\text{the} (\text{nhop} (r (\sigma' \ ?\text{i})) \text{dip}))))\) and \(\sigma' \ ?\text{i} = \sigma \ ?\text{i}\)
show "\(\text{dip} \in vD(r (\sigma' \ ?\text{nhip}))\)" by simp
qed
finally have "\(\text{nsqn} (r (\sigma \ ?\text{i})) \text{dip} < \text{nsqn} (r (\sigma' \ ?\text{nhip})) \text{dip}\)" .
moreover from \(\text{dip} \in vD(r (\sigma' (\text{the} (\text{nhop} (r (\sigma' \ ?\text{i})) \text{dip}))))\) and \(\sigma' \ ?\text{i} = \sigma \ ?\text{i}\)
have "\(\text{dip} \in kD(r (\sigma' \ ?\text{nhip}))\)" by auto
ultimately show "\(r (\sigma \ ?\text{i}) \sqsubseteq \text{dip} \ r (\sigma' \ ?\text{nhip})\)" using \(\text{dip} \in kD(r (\sigma \ ?\text{i}))\) by - (rule \text{rt_strictly_fresher_ltI})
with \(\sigma' \ ?\text{i} = \sigma \ ?\text{i}\) show "\(r (\sigma \ ?\text{i}) \sqsubseteq \text{dip} \ r (\sigma' \ (\text{the} (\text{nhop} (r (\sigma' \ ?\text{i})))) \text{dip}))\)" by simp
qed
qed
thus \(\text{thesis}\) unfolding \text{Let_def} .
qed

lemma \(\text{seq_nhpop_quality_increases}:\)
shows "\(\text{opaodv} \ ?\text{i} = (\text{otherwith} (\text{op}=) \ ?\text{i})\)
\(\text{(orecvmsg (\lambda \sigma m. \text{msg_fresh} \sigma m \land \text{msg_zhops} m)),}\)
other \(\text{quality_increases} \ ?\text{i} \rightarrow\)
global (\(\lambda \sigma. \forall \text{dip}. \text{let nhip = the (nhop (r (\sigma \ ?\text{i}))) dip})\)
\(\text{in dip} \in vD (r (\sigma \ ?\text{i})) \land vD (r (\sigma \ ?\text{nhip})) \land \text{nhip} \neq \text{dip}\)
\(\rightarrow (r (\sigma \ ?\text{i}) \sqsubseteq \text{dip} \ r (\sigma \ ?\text{nhip}))\)"
by (rule \text{oinvariant_weakenE [OF \text{seq_nhpop_quality_increases']}) (auto dest!: onlD)

end

0.11 Routing graphs and loop freedom

theory \text{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 \(\text{rt_graph} : = (\text{ip} \Rightarrow \text{state}) \Rightarrow \text{ip} \Rightarrow \text{ip rel}\) where
\(\text{rt_graph} \ ?\sigma = (\lambda \text{dip.}\)
\((\text{ip}, \text{ip'}) \mid \text{ip ip'} \text{dsn dsk hops pre.}\)
\(\text{ip} \neq \text{dip} \land r (\sigma \ ?\text{ip}) \text{dip} = \text{Some} (\text{dsn}, \text{dsk}, \text{val}, \text{hops}, \text{ip'}, \text{pre}))\)
Given the state of a network $\sigma$, 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** "$(ip, ip') \in rt_graph \sigma dip$"
- **Shows** "$ip \neq dip \land \exists r. rt (\sigma ip) = r$
  $\land \exists 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]:**

"$\forall ip ip' 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 ip ip' 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 dip ip'. (ip, ip') \in rt_graph \sigma dip \Rightarrow ip \neq dip"
- **Unfolding** rt_graph_def by auto

**Lemma rt_graph_not_dip_trans [dest]:**

"$\forall ip dip ip'. (ip, ip') \in rt_graph (\sigma dip) \Rightarrow 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 dip ip'. (ip, ip') \in rt_graph \sigma dip \Rightarrow ip' = the (nhop (rt (\sigma ip)) dip)"
- **Unfolding** rt_graph_def by auto

**Theorem inv_to_loop_freedom:**

**Assumes** "$\forall i dip. let nhip = the (nhop (rt (\sigma i)) dip) in dip \in vD(rt (\sigma i)) \cap vD(rt (\sigma nhip)) \land nhip \neq dip \Rightarrow (rt (\sigma i)) \sqsubseteq dip (rt (\sigma nhip))"
- **Shows** "$\forall dip. irrefl ((rt_graph \sigma dip))""
- **Using** assms proof (intro allI)
- **Fix** $\sigma :: "ip \Rightarrow state"$ and dip
- **Assume inv:** "$\forall ip dip. let nhip = the (nhop (rt (\sigma ip)) dip) in dip \in vD(rt (\sigma ip)) \cap vD(rt (\sigma nhip)) \land nhip \neq dip \Rightarrow rt (\sigma ip) \sqsubseteq dip rt (\sigma nhip)"
- **{** fix ip ip'
- **Assume** "$(ip, ip') \in rt_graph (\sigma dip)"" and "$\exists dip. rt (\sigma ip')""
  **And** "$\exists \exists dip. rt (\sigma ip')""
  **Hence** "$\forall rt (\sigma ip) \sqsubseteq dip rt (\sigma ip')"
  **Proof induction**
  **Fix nhip**
  **Assume** "$(ip, nhip) \in rt_graph (\sigma dip)"
  **And** "$\exists dip. rt (\sigma nhip))""
  **And** "$\exists dip. rt (\sigma nhip))""
  **From**: $(ip, nhip) \in rt_graph (\sigma dip)$ have "$\exists dip. rt (\sigma ip))" and "$nhip = the (nhop (rt (\sigma ip)) dip)"
- **By** auto
- **From**: $(ip, nhip) \in rt_graph (\sigma dip)$ have "$\exists dip. rt (\sigma ip))" and "$nhip = the (nhop (rt (\sigma ip)) dip)"
  **And** "$nhip \neq dip)"
  **And** inv
  **Show** "$\forall 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 \((nhip, nhip') \in rt_graph \sigma dip\)
and \(IH: \[[ dip \in vD(rt (\sigma nhip)); nhip \neq dip ] \implies rt (\sigma ip) \sqsubseteq dip rt (\sigma nhip)\]
and \("dip \in vD(rt (\sigma nhip'))\"
and \("nhip' \neq dip\"
from \((nhip, nhip') \in rt_graph \sigma dip\) have 1: "dip \in vD(rt (\sigma nhip))"
and 2: "nhip \neq dip"
and "nhip' = the (nhop (rt (\sigma nhip)) dip)"
by auto
from 1 2 have "rt (\sigma ip) \sqsubseteq dip rt (\sigma nhip)"
by (rule IH)
also have "rt (\sigma nhip) \sqsubseteq dip rt (\sigma nhip')" proof -
from \(dip \in vD(rt (\sigma nhip))\) and \(dip \in vD(rt (\sigma nhip'))\)
have "dip \in vD(rt (\sigma nhip)) \cap vD(rt (\sigma nhip'))" ..
with \(nhip' \neq dip\)
and \(nhip' = the (nhop (rt (\sigma nhip)) dip)\)
and inv
show "rt (\sigma nhip) \sqsubseteq dip rt (\sigma nhip)"
by (clarsimp simp: Let_def)
qed
finally show "rt (\sigma ip) \sqsubseteq dip rt (\sigma nhip')".
qed } note fresher = this

show "irrefl ((rt_graph \sigma dip)\+)
unfolding irrefl_def proof (intro allI notI)
fix ip
assume "(ip, ip) \in (rt_graph \sigma dip)\+
moreover then have "dip \in vD(rt (\sigma ip))"
and "ip \neq dip"
by auto
ultimately have "rt (\sigma ip) \sqsubseteq dip rt (\sigma 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 oparp_sos i (oseqp_sos \Gamma_{AODV} i) (seqp_sos \Gamma_{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 "opaodv i \((\ni qmsg \models (\text{otherwith (op=} \{i\}) \text{(orecvmsg (} \lambda \sigma m. \text{msg_fresh } \sigma m \wedge \text{msg_zhops } m)) \text{, ostep_invariant_weakenE \{\sigma i\} \rightarrow}}\)
\text{global} (\lambda(\lambda). \forall dip. let nhip = the (nhop (rt (\sigma i)) dip)
in dip \in vD (rt (\sigma i)) \cap vD (rt (\sigma nhip)) \wedge nhip \neq dip \implies (rt (\sigma i)) \sqsubseteq dip (rt (\sigma nhip)))" proof (rule lift_into_qmsg [OF seq_nhop_quality_increases])
show "opaodv i \models_A \((\text{otherwith (op=} \{i\}) \text{(orecvmsg (} \lambda \sigma m. \text{msg_fresh } \sigma m \wedge \text{msg_zhops } m)) \text{, ostep_invariant_weakenE \{\sigma i\} \rightarrow}}\)
\text{globala} (\lambda(\lambda(\sigma, \sigma')). \text{quality_increases (} \sigma i \text{) (} \sigma' i\))" proof (rule ostep_invariant_weakenE [OF 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', _). ∀ 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!: onll1D, 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} →
globala (λ(σ', _). quality_increases (σ i) (σ' i))))"
proof -
  have "opaodv i |=_A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ), other (λ_ _. True) {i} →
globala (λ(σ, _). quality_increases (σ i) (σ' i)))"
    by (rule ostep_invariant_weakenE[of local_quality_increases])
    (auto dest!: ostep_invariant_weakenE elim!: aodv_ex_labelE)
  hence "opaodv i ((i qmsg =_A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ), other (λ_ _. True) {i} →
globala (λ(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_snq_freshany_step:
shows "opaodv i ((i qmsg =_A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ), other (λ_ _. True) {i} →
globala (λ(σ, a, _). anycast (msg_fresh σ) a)))"
proof -
  have "opaodv i |=_A (λσ _. (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_rrep_snq_freshany_step_invariant])
    (auto dest!: ostep_invariant_weakenE elim!: aodv_ex_labelE)
  hence "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)))"
    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} →
globala (λ(σ, a, _). anycast msg_zhops a)))"
proof -
  from anycast_msg_zhops_initiali_aodv oaoav_trans aodv_trans
  have "opaodv i |=_A (act TT, other (λ_ _. True) {i} →)
    seqll i (onll Γ_AODV (λ(σ, a, _). anycast msg_zhops a))"
    by (rule open_seq_step_invariant)
  hence "opaodv i |=_A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ), other (λ_ _. True) {i} →
globala (λ(σ, a, _). anycast msg_zhops a))"
    by (rule ostep_invariant_weakenE)
  fix t :: "(((nat ⇒ state) × (state, msg, pseqp, pseqp label) seqp), msg seq_action) transition"
  assume "seqll i (onll Γ_AODV (λ(σ, a, _). anycast msg_zhops a)) t"
  thus "globala (λ(σ, a, _). anycast msg_zhops a) t"
    by (cases t) (clarsimp dest!: seqll1D onll1D, metis aodv_ex_label)
  qed simp_all
  hence "opaodv i ((i qmsg =_A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ), other (λ_ _. True) {i} →
globala (λ(σ, a, _). anycast 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 : opraodv i ⟨⟨i qmsg : R⟩o|=
(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:
"⟨i : opraodv i ⟨⟨i qmsg : R⟩o|=
(λσ. oarrivemsg (λσ_. 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 : opraodv i ⟨⟨i qmsg : R⟩o|=
(λσ. oarrivemsg (λσ_. rreq_rrep_sn) σ,
other (λ_. True) {i} →
globala (λ(_, a, _). castmsg (msg_fresh σ) a)" by (rule node_lift_anycast_statelessassm [OF par_rreq_rrep_nsqn_fresh_any_step])

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

lemma node_silent_change_only:
shows "⟨i : opraodv i ⟨⟨i qmsg : R⟩o|=
(λσ. oarrivemsg (λσ_. 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 σ' ζ'
  assume or: "⟨σ, ζ⟩ ∈ oreachable (⟨i : opraodv i ⟨⟨i qmsg : R_i⟩o|=
(λσ. oarrivemsg (λσ_. True) σ)
other (λ_. True) {i}⟩)" and tr: "⟨(σ, ζ), a, (σ', ζ')⟩ ∈ trans (⟨i : opraodv i ⟨⟨i qmsg : R_i⟩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, NodeS i p R), a, (σ', ζ')) ∈ onode_sos (oparp_sos i (trans (opadr 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 [simpl]:
  assumes "oarrivemsg (λσ. m. 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) qs msg
  (otherwith (op=) (net_tree_ips p)) (oarrivemsg (λσ. m. msg_fresh σ m ∧ msg_zhops m))
  global (λσ. ∀i∈net_tree_ips p. ∀dip. let nhip = the (nhop (rt (σ i)) dip)
  in dip ∈ νD (rt (σ i)) ∩ νD (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) qs msg : R)oo = A (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)
  by (auto)

  end

proof (rule ostep_invariantI, simp (no_asm))
  fix σ a σ' s'
  assume or: "((σ, a, σ', s') ∈ oract)
  by (auto)

  from or tr am have "castmsg (msg_fresh σ) a"
  by (auto)

  moreover from or tr am have "castmsg (msg_zhops m) a"
  by (auto)

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

qed

thus "(i : opaodv i) qs msg : R)oo = A (λσ_. oarrivemsg (λ_. rreq_rrep_sn) σ,
  other quality_increases {i} →) globala (λ(σ, a, _.)
  castmsg (λm. msg_fresh σ m ∧ msg_zhops m) a)"
  by rule auto

next

show "((i : opaodv i) qs msg : R)oo = A (λσ_. oarrivemsg (λσ. m. msg_fresh σ m ∧ msg_zhops m) σ,
  other quality_increases {i} →) globala (λ(σ, a, σ').
  a ≠ τ ∧ (∀d. a ≠ i:deliver(d) → σ i = σ' i))"
  by (rule ostep_invariant_weakenE [OF node_silent_change_only]) auto

next

show "((i : opaodv i) qs msg : R)oo = A (λσ_. oarrivemsg (λ_. m. msg_fresh σ m ∧ msg_zhops m) σ,
  other quality_increases {i} →) globala (λ(σ, a, σ').
  a = τ ∨ (∀d. a = i:deliver(d) → quality_increases (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)) qs msg
  (otherwith (op=) (net_tree_ips p)) (oarrivemsg (λσ. m. msg_fresh σ m ∧ msg_zhops m))
  global (λσ. ∀i∈net_tree_ips p. ∀dip. let nhip = the (nhop (rt (σ i)) dip)
  in dip ∈ νD (rt (σ i)) ∩ νD (rt (σ nhip)) ∧ nhip ≠ dip
  → (rt (σ i)) ⊆ dip (rt (σ nhip)))"
proof (rule inclosed_closed)
  from opnet_nhop_quality_increases
  show "opnet (λi. opaodv i \(\langle\langle i qmsg \rangle\ p\ = \langle\langle \text{otherwith (op=)} (\text{net_tree_ips} p) \text{inclosed}, ?U \to \rangle \rangle ?\text{inv}\)"
  proof (rule oinvariant_weakenE)
    fix \(\sigma \), \(\sigma'\) :: "\text{ip} \Rightarrow \text{state}"
    and \(a\) :: "\text{msg node_action}"
    assume "\text{otherwith (op=)} (\text{net_tree_ips} p) \text{inclosed} \sigma \sigma' a"
    thus "\text{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 \(\sigma\) :: "\text{ip} \Rightarrow \text{state}"
      and \(a\) :: "\text{msg node_action}"
      assume "\text{inclosed} \sigma a"
      thus "\text{oarrivemsg} (\lambda \sigma m. \text{msg_fresh} \sigma m \land \text{msg_zhops} m)
        \sigma a"
      proof (cases \(a\))
        fix \(\text{ii}, \text{ni}, \text{ms}\)
        assume "\(a = \text{ii} \land \text{ni:arrivemsg}(\text{ms})\)"
        moreover with \(\\langle\text{inclosed} \sigma a\rangle\) obtain \(d, \text{di}\) where 
          "\text{ms = newpkt}(d, \text{di})"
        by (cases \(\text{ms}\)) auto
        ultimately show \(?\text{thesis}\) by simp
      qed simp_all
      qed
    qed
  qed

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 \(i\) :: \text{ip}
    have "\{(\sigma, \zeta). (\sigma i, \zeta) \in \sigma_{AODV} i \land (\forall j. j \neq i \to \sigma j \in \text{fst} (\sigma_{AODV} j)) \leq \sigma_{AODV}'\}"
      unfolding \text{sigma}_{AODV}_def \text{sigma}_{AODV}'_def
    proof (rule equalityD1)
      show "\(\\\forall f p. \{(\sigma, \zeta). (\sigma i, \zeta) \in \{f i, p\} \land (\forall j. j \neq i \to \sigma j \in \text{fst} (\{f j, p\}))\} = \{f, p\}\)"
        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 \to \sigma j \in \text{fst} (\text{id} s) \to \text{init}(paodv \ j)) \} \leq \text{init}(opaodv \ i)"
      by simp
next
  show "\(\\forall j. \text{init}(paodv \ j) \neq \{\}\)"
    unfolding \text{sigma}_{AODV}_def by simp
next
  fix \(i\) \(s\) \(s'\) \(\sigma\) \(\sigma'\)
  assume "\(\sigma i = \text{fst} (\text{id} s)\)"
  and "\(\sigma' i = \text{fst} (\text{id} s')\)"
  and "\(\langle s, a, s'\rangle \in \text{trans}(paodv \ i)\)"
  then obtain \(q, q'\) where "\(s = (\sigma i, q)\)"
    and "\(s' = (\sigma' i, q')\)"
    and "\(\langle \sigma i, q, a, (\sigma' i, q')\rangle \in \text{trans}(paodv \ i)\)"
    by (cases \(s\), cases \(s'\)) auto
  from this(3) have "\(\langle (\sigma, q), a, (\sigma', q')\rangle \in \text{trans}(opaodv \ i)\)"
    by simp (rule open_seqp_action [OF aodv_wf])
  with \(\langle s = (\sigma i, q)\rangle\) and \(\langle s' = (\sigma' i, q')\rangle\)
  show "\(\langle (\sigma, \text{snd} \langle\text{id} s\rangle), a, (\sigma', \text{snd} \langle\text{id} s'\rangle)\rangle \in \text{trans}(opaodv \ i)\)"
    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. opnproc.initmissing paodv id i = initmissing i"

unfolding opn.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: "∀ P σ. openproc.netglobal (λi. paodv i (⟨⟨ qmsg) (λ(p, q). (fst (id p), snd (id p), q)) 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)

thus "openproc.netglobal (λi. 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 (λi. paodv i (⟨⟨ 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 (⟨⟨ 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

86
hence "(fst (initmissing (netgmap fst σ))) i = aodv_init i"
  by simp
thus ?thesis by simp
qed
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 originators 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" ("\(\pi_4\)

88
\[ \pi_4 \equiv \lambda(\ldots, \text{flag}, \ldots). \text{flag} \]

**Definition proj5**: \( \tau \Rightarrow \mathbb{N} \) ("\( \pi_5 \)"

where \( \pi_5 \equiv \lambda(\ldots, \text{hops}, \ldots). \text{hops} \)

**Definition proj6**: \( \tau \Rightarrow \text{ip} \) ("\( \pi_6 \)"

where \( \pi_6 \equiv \lambda(\ldots, \text{nhip}, \ldots). \text{nhip} \)

**Definition proj7**: \( \tau \Rightarrow \text{ip set} \) ("\( \pi_7 \)"

where \( \pi_7 \equiv \lambda(\ldots, \ldots, \text{pre}, \ldots). \text{pre} \)

**Lemma projs [simp]**:
\[
\begin{align*}
\pi_2(\text{dsn}, \text{dsk}, \text{flag}, \text{hops}, \text{nhip}, \text{pre}) &= \text{dsn} \\
\pi_3(\text{dsn}, \text{dsk}, \text{flag}, \text{hops}, \text{nhip}, \text{pre}) &= \text{dsk} \\
\pi_4(\text{dsn}, \text{dsk}, \text{flag}, \text{hops}, \text{nhip}, \text{pre}) &= \text{flag} \\
\pi_5(\text{dsn}, \text{dsk}, \text{flag}, \text{hops}, \text{nhip}, \text{pre}) &= \text{hops} \\
\pi_6(\text{dsn}, \text{dsk}, \text{flag}, \text{hops}, \text{nhip}, \text{pre}) &= \text{nhip} \\
\pi_7(\text{dsn}, \text{dsk}, \text{flag}, \text{hops}, \text{nhip}, \text{pre}) &= \text{pre}
\end{align*}
\]

by (clarsimp simp: proj2_def proj3_def proj4_def proj5_def proj6_def proj7_def)+

**Lemma proj3_pred [intro]**: 
\[
[ [ \text{P kno}; \text{P unk} ] ] \Rightarrow \text{P (} \pi_3 x \text{)}
\]

by (rule k.induct)

**Lemma proj4_pred [intro]**: 
\[
[ [ \text{P val}; \text{P inv} ] ] \Rightarrow \text{P (} \pi_4 x \text{)}
\]

by (rule f.induct)

**Lemma proj6_pair_snd [simp]**:
fixes \( \text{dsn}' \) \( \tau \)

shows \( \pi_6(\text{dsn}', \text{snd}(\tau)) = \pi_6(\tau) \)

by (cases \( \tau \) simp)

### 1.1.3 Routing Tables

Routing tables map ip addresses to route entries.

**Type Synonym** \( \text{rt} = \"\mathbb{ip} \rightarrow \tau \"

**Syntax**
\[
\ldots \Sigma_{\text{route}} :: \"\tau \Rightarrow \mathbb{ip} \rightarrow \tau\" \ (\"\sigma_{\text{route}}'(\ldots, \ldots)'\")
\]

**Translations**
\[
\sigma_{\text{route}}(\text{rt}, \text{dip}) \Rightarrow \text{rt dip}
\]

**Definition sqn** :: \( \tau \Rightarrow \mathbb{ip} \Rightarrow \text{sqn} \)

where \( \text{sqn } \tau \text{ dip } \equiv \text{case } \sigma_{\text{route}}(\text{rt}, \text{dip}) \text{ of Some } \tau \Rightarrow \pi_2(\tau) \text{ | None } \Rightarrow 0 \)

**Definition sqnf** :: \( \tau \Rightarrow \mathbb{ip} \Rightarrow \text{k} \)

where \( \text{sqnf } \tau \text{ dip } \equiv \text{case } \sigma_{\text{route}}(\text{rt}, \text{dip}) \text{ of Some } \tau \Rightarrow \pi_3(\tau) \text{ | None } \Rightarrow \text{unk} \)

**Abbreviation** flag :: \( \tau \Rightarrow \mathbb{ip} \Rightarrow \text{f} \)

where \( \text{flag } \tau \text{ dip } \equiv \text{map_option } \pi_4(\sigma_{\text{route}}(\text{rt}, \text{dip})) \)

**Abbreviation** dhops :: \( \tau \Rightarrow \mathbb{ip} \Rightarrow \mathbb{nat} \)

where \( \text{dhops } \tau \text{ dip } \equiv \text{map_option } \pi_5(\sigma_{\text{route}}(\text{rt}, \text{dip})) \)

**Abbreviation** nhop :: \( \tau \Rightarrow \mathbb{ip} \Rightarrow \mathbb{ip} \)

where \( \text{nhop } \tau \text{ dip } \equiv \text{map_option } \pi_6(\sigma_{\text{route}}(\text{rt}, \text{dip})) \)

**Abbreviation** precs :: \( \tau \Rightarrow \mathbb{ip} \Rightarrow \mathbb{ip set} \)

where \( \text{precs } \tau \text{ dip } \equiv \text{map_option } \pi_7(\sigma_{\text{route}}(\text{rt}, \text{dip})) \)

**Definition** \( \text{vD} \) :: \( \tau \Rightarrow \mathbb{ip set} \)

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

definition $kD :: "rt \Rightarrow ip set"$
  where "$kD\ rt \equiv \{dip.\ rt\ dip \neq 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]:
  "\(\forall ip\ rt.\ ip\in\ vD\ rt\ \Rightarrow\ ip\in\ kD\ rt\)"
  "\(\forall ip\ rt.\ ip\in\ iD\ rt\ \Rightarrow\ ip\in\ kD\ rt\)"
  unfolding $kD\_is\_vD\_and\_iD$ by simp_all

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

lemma $kD\_None$ [dest]:
  fixes dip rt
  assumes "dip \notin kD rt"
  shows "\(\sigma\ route\ (rt,\ dip) = None\)"
  using assms unfolding $kD\_def$
  by (metis (mono_tags) mem_Collect_eq)

lemma $vD\_Some$ [dest]:
  fixes dip rt
  assumes "dip \in vD rt"
  shows "\(\exists dsn\ dsk\ hops\ nhip\ pre.\ 
    \sigma\ route\ (rt,\ dip) = Some\ (dsn,\ dsk,\ val,\ hops,\ nhip,\ pre)\)"
  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 \in iD rt"
  shows "\(\exists dsn\ dsk\ hops\ nhip\ pre.\ 
    \sigma\ route\ (rt,\ dip) = Some\ (dsn,\ dsk,\ inv,\ hops,\ nhip,\ pre)\)"
  using assms unfolding $iD\_def$ by simp

lemma $val\_is\_vD$ [elim]:
  fixes ip rt
  assumes "ip \in kD rt"
  and "the (flag rt ip) = val"
  shows "ip \in vD rt"
  using assms unfolding $vD\_def$ 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"
from \( ip \in iD(rt) \) 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 \( ip \not\in vD(rt) \)
shows \( ip \in iD(rt) \)
proof -
  from \( ip \in kD(rt) \) obtain dsn dsk f hops nhop pre
  where rtip: \( rt \ ip = Some (dsn, dsk, f, hops, nhop, pre) \)
    by (metis kD_Some)
  from \( ip \not\in vD(rt) \) have \( f \neq val \)
    proof (rule contrapos_nn)
      assume \( f = val \)
      with rtip have \( the (flag rt ip) = val \)
        by simp
      with \( ip \in kD(rt) \)
      show \( ip \in vD(rt) \)
    ..
  qed
  with rtip have \( the (flag rt ip) = inv \)
    by simp
  with \( ip \in kD(rt) \)
  show \( ip \in iD(rt) \)
    ..
  qed

lemma \( vD\_or\_iD \) [elim]:
fixes ip rt
assumes \( ip \in kD(rt) \) and \( ip \in vD(rt) \implies P rt ip \)
and \( ip \in iD(rt) \implies P rt ip \)
shows \( P rt ip \)
proof -
  from \( ip \in kD(rt) \)
  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: \( \forall dip rt. dip \in kD(rt) \implies \pi_5 (the (rt dip)) = the (dhops rt dip) \)'
unfolding sqn_def by (drule kD_Some) clarsimp

lemma proj4_eq_flag: \( \forall dip rt. dip \in kD(rt) \implies \pi_4 (the (rt dip)) = the (flag rt dip) \)'
unfolding sqn_def by (drule kD_Some) clarsimp

lemma proj2_eq_sqn: \( \forall dip rt. dip \in kD(rt) \implies \pi_2 (the (rt dip)) = sqn rt dip \)'
unfolding sqn_def by (drule kD_Some) clarsimp

lemma kD_sqnf_is_proj3 [simp]:
  \( \forall ip rt. ip \in kD(rt) \implies sqnf rt ip = \pi_3 (the (rt ip)) \)'
unfolding sqnf_def by auto

lemma vD_flag_val [simp]:
  \( \forall dip rt. dip \in vD(rt) \implies the (flag rt dip) = val \)'
unfolding vD_def by clarsimp

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

lemma kD_empty [simp]: \( kD \operatorname{Map}.empty = {} \)'
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 \( [ ip \neq ip'; ip \in kD(rt) ] \implies P rt ip ip' \)
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)"

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) ∪ 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)) (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'
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 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)"
"∀ 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: "∀ 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 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) ∨ 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 if π3(r) = unk
        then rt (ip ⇒ addpre s (π7(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 = \pi_2(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 = \pi_2(r)) and (flag rt ip = Some inv)
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 "\pi_3(r) = unk"
and "(\pi_2(r) = 0) = (\pi_3(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 ((\pi_2(r) = 0) = (\pi_3(r) = unk)) and (\pi_3(r) = unk)
show "update rt ip r = rt (ip \mapsto nr')"
  unfolding update_def nr'_def nr_def s_def by (cases r) simp
next
assume "ip \in kD(rt)"
and otherassms: "sqn rt ip \geq \pi_2(r)"
  "\pi_3(r) = kno"
  "sqn rt ip = \pi_2(r) \implies (\text{dhops rt ip} \leq \pi_5(r) \land \text{the (flag rt ip)} = \text{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 \mapsto ns)"
  unfolding update_def ns_def s_def by auto
qed

lemma update_cases [elim]:
  assumes "(\pi_2(r) = 0) = (\pi_3(r) = unk)"
  and cl1: "[\[ ip /\not\in kD(rt) \] \implies P (rt (ip \mapsto r))]"
  and cl2: "[\[ \pi_2(r) < \pi_2(r) \] \implies P (rt (ip \mapsto \text{addpre (rt (ip \mapsto r))))])"
  and cl3: "[\[ \pi_2(r) \geq \pi_2(r) \] \implies P (rt (ip \mapsto \text{addpre (rt (ip \mapsto r))))])"
  and cl4: "[\[ \pi_3(r) = \pi_3(r) \] \implies P (rt (ip \mapsto \text{addpre (rt (ip \mapsto r))))])"
  and cl5: "[\[ \pi_4(r) = \pi_4(r) \] \implies P (rt (ip \mapsto (\text{route}_\pi(r = \pi_4(r), \pi_5(r), \pi_6(r), \pi_7(\text{addpre (rt (ip \mapsto r)) = val})))])"
  and cl6: "[\[ \pi_8(r) = \pi_8(r) \] \implies P (rt (ip \mapsto (\text{route}_\pi(r = \pi_4(r), \pi_5(r), \pi_6(r), \pi_7(\text{addpre (rt (ip \mapsto r)) = val})))])"
shows "(P (update rt ip r))"
proof (cases "ip \in kD(rt)")
  assume "ip \notin kD(rt)"
  with cl1 show ?thesis
  by simp
next
  assume "ip \in kD(rt)"
  moreover then obtain dsn dsk fl hops nhip pre
    where rtreq: "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 \(\langle \pi_2 (r) = 0 \rangle = (\pi_3 (r) = \text{unk})\):

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

unfolding update_def sqn_def by auto

qed

lemma update_cases_kD:
assumes \(\langle \pi_2 (r) = 0 \rangle = (\pi_3 (r) = \text{unk})\) and 
\(\text{ip} \in kD (rt)\)

\(c2: \text{sqn} \, rt \, \text{ip} < \pi_2 (r) \Rightarrow \text{P} (rt (ip \mapsto \text{addpre} \, r (\pi_7 (\sigma_{\text{route} (rt, ip)))))\)

\(c3: \text{sqn} \, rt \, \text{ip} = \pi_2 (r); \text{the} (\text{dhops} \, rt \, \text{ip}) > \pi_5 (r) \Rightarrow \text{P} (rt (ip \mapsto \text{addpre} \, r (\pi_7 (\sigma_{\text{route} (rt, ip))))))\)

\(c4: \text{sqn} \, rt \, \text{ip} = \pi_2 (r); \text{the} (\text{flag} \, rt \, \text{ip}) = \text{inv} \Rightarrow \text{P} (rt (ip \mapsto \text{addpre} \, r (\pi_7 (\sigma_{\text{route} (rt, ip))))))\)

\(c5: \pi_3 (r) = \text{unk} \Rightarrow \text{P} (rt (ip \mapsto (\pi_2 (\sigma_{\text{route} (rt, ip)}), \pi_3 (r), \pi_4 (r), \pi_5 (r), \pi_6 (r), \pi_7 (\text{addpre} \, r (\pi_7 (\sigma_{\text{route} (rt, ip))))))))\)

\(c6: \text{sqn} \, rt \, \text{ip} \geq \pi_2 (r); \pi_3 (r) = \text{kno}; \text{sqn} \, rt \, \text{ip} = \pi_2 (r) \Rightarrow \text{the} (\text{dhops} \, rt \, \text{ip}) \leq \pi_5 (r) \land \text{the} (\text{flag} \, rt \, \text{ip}) = \text{val}\) 
\(\Rightarrow \text{P} (rt (ip \mapsto \text{addpre} \, (\sigma_{\text{route} (rt, ip)}) (\pi_7 (r))))\)

shows "(\(\text{P} (\text{update} \, rt \, \text{ip} \, r)\))"

using assms(1) proof (rule update_cases)
assume "sqn \, rt \, \text{ip} < \pi_2 (r)"
thus "\(\text{P} (rt (ip \mapsto \text{addpre} \, r (\pi_7 (\sigma_{\text{route} (rt, ip)))))))\)" by (rule c2)

next
assume "sqn \, rt \, \text{ip} = \pi_2 (r)"

and "the (\text{dhops} \, rt \, \text{ip}) > \pi_5 (r)"

thus "\(\text{P} (rt (ip \mapsto \text{addpre} \, r (\pi_7 (\sigma_{\text{route} (rt, ip)))))))\)" by (rule c3)

next
assume "sqn \, rt \, \text{ip} = \pi_2 (r)"

and "the (\text{flag} \, rt \, \text{ip}) = \text{inv}"

thus "\(\text{P} (rt (ip \mapsto \text{addpre} \, r (\pi_7 (\sigma_{\text{route} (rt, ip)))))))\)" by (rule c4)

next
assume "\pi_3 (r) = \text{unk}"

thus "\(\text{P} (rt (ip \mapsto (\pi_2 (\sigma_{\text{route} (rt, ip)}), \pi_3 (r), \pi_4 (r), \pi_5 (r), \pi_6 (r), \pi_7 (\text{addpre} \, r (\pi_7 (\sigma_{\text{route} (rt, ip))))))))\)" by (rule c5)

next
assume "sqn \, rt \, \text{ip} \geq \pi_2 (r)"

and "\pi_3 (r) = \text{kno}"

and "sqn \, rt \, \text{ip} = \pi_2 (r) \Rightarrow \text{the} (\text{dhops} \, rt \, \text{ip}) \leq \pi_5 (r) \land \text{the} (\text{flag} \, rt \, \text{ip}) = \text{val}\"

thus "\(\text{P} (rt (ip \mapsto \text{addpre} \, (\sigma_{\text{route} (rt, ip)}) (\pi_7 (r))))\)" by (rule c6)

qed (simp add: \langle \text{ip} \in kD (rt)\rangle)

lemma in_kD_after_update [simp]:
fixes rt nip dsn dsk flag hops nhp pre

shows "kD (update \, rt \, nip \, (dsn, dsk, flag, hops, nhp, 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 nhp

assumes "rt \neq update \, rt \, dip \, (dsn, dsk, flag, hops, nhp, \{}\)"

shows "the \,(\text{nhop} \,(update \, rt \, dip \, (dsn, dsk, flag, hops, nhp, \{}\))) \, dip) = \text{nhp}"

proof -
from assms
have update_neq: "\forall \, v. \, rt \, dip = \text{Some} \, v \implies"
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: "∀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 = 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 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 sqnf_def by (clarsimp split: option.splits split_if_asm) auto

lemma update_kno_dsn_greater_zero:
  "∀ rt 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, {})
  ⇒ π_3(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"
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 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 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)) \implies (dip∈vD(rt) \or 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))"
lemma proj5_invalidate [simp]:
"\(\forall \text{dip. } \pi_5(\text{the } (\text{invalidate } \text{rt dests} \text{ dip})) = \pi_5(\text{the } (\text{rt dip}))\)"
unfolding invalidate_def by (clarsimp split: option.split)

lemma proj6_invalidate [simp]:
"\(\forall \text{dip. } \pi_6(\text{the } (\text{invalidate } \text{rt dests} \text{ dip})) = \pi_6(\text{the } (\text{rt dip}))\)"
unfolding invalidate_def by (clarsimp split: option.split)

lemma proj7_invalidate [simp]:
"\(\forall \text{dip. } \pi_7(\text{the } (\text{invalidate } \text{rt dests} \text{ dip})) = \pi_7(\text{the } (\text{rt dip}))\)"
unfolding invalidate_def by (clarsimp split: option.split)

1.1.5 Route Requests

lemma invalidate_kD_inv [simp]:
"\(\forall \text{rt dests. kD } (\text{invalidate } \text{rt dests}) = \text{kD } \text{rt}\)"
unfolding invalidate_def kD_def by (simp split: option.split)

lemma invalidate_sqn:
fixes \(\text{rt dip dests}\)
assumes "\(\forall \text{rsn. dests dip = Some rsn } \rightarrow sqn \text{ rt dip } \leq \text{rsn}\)"
shows "\(sqn \text{ rt dip } \leq sqn (\text{invalidate } \text{rt dests} \text{ dip})\)"
proof (cases "\(\text{dip} \notin kD(\text{rt})\)"
  assume "\(\neg \text{dip} \notin kD(\text{rt})\)"
  hence "\(\text{dip} \in kD(\text{rt})\)" by simp
  then obtain \(dsn \text{ dsk flag hops nhip pre where } \text{rt dip} = \text{Some } (dsn, dsk, flag, hops, nhip, pre)\)"
  by (metis kD_Some)
  with assms show "\(sqn \text{ rt dip } \leq sqn (\text{invalidate } \text{rt dests} \text{ dip})\)"
  by (cases "\(\text{dests dip}\)"
  (auto simp add: invalidate_def sqn_def)
qed simp

lemma sqn_invalidate_in_dests [simp]:
fixes \(\text{dests ipa rsn rt}\)
assumes "\(\text{dests ipa} = \text{Some rsn}\)"
and "\(\text{ipa} \in kD(\text{rt})\)"
shows "\(sqn (\text{invalidate } \text{rt dests}) \text{ ipa} = \text{rsn}\)"
unfolding invalidate_def sqn_def using assms(1) assms(2) [THEN kD_Some] by clarsimp

lemma dhops_invalidate [simp]:
"\(\forall \text{dip. the } (\text{dhops } (\text{invalidate } \text{rt dests} \text{ dip})) = \text{the } (\text{dhops } \text{rt dip})\)"
unfolding invalidate_def by (clarsimp split: option.split)

lemma sqnf_invalidate [simp]:
"\(\forall \text{dip. sqnf } (\text{invalidate } (\text{rt } \xi) (\text{dests } \xi)) \text{ dip} = \text{sqnf } (\text{rt } \xi) \text{ dip}\)"
unfolding sqnf_def invalidate_def by (clarsimp split: option.split)

lemma nhop_invalidate [simp]:
"\(\forall \text{dip. the } (\text{nhop } (\text{invalidate } (\text{rt } \xi) (\text{dests } \xi))) \text{ dip} = \text{the } (\text{nhop } (\text{rt } \xi) \text{ dip})\)"
unfolding invalidate_def by (clarsimp split: option.split)

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

lemma invalidate_none [simp]:
fixes \(\text{rt dests dip}\)

assumes "dip ∈ kD(rt)"
shows "invalidate rt dests dip = None"
using assms unfolding invalidate_def by clarsimp

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

lemma sqn_invalidate_not_in_dests [simp]:
fixes dests dip rt
assumes "dip ∈ 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 ⇒ \(\pi_2\) (the (rt dip)) | Some rsn ⇒ 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 ∈ kD(rt) \implies \(\pi_3\) (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 ∈ 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)

1.1.6 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 := None)
\lor Some (p, q) ⇒ 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 (\(\lambda\) (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)"
```

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.

1.2 AODV protocol messages

definition rreq :: "nat × ip × sqn × k × ip × sqn × ip ⇒ msg"
  where "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
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) × ip ⇒ 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"
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),
    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),
    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

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 ξ \equiv \text{case } msg ξ \text{ of}
\begin{align*}
  \text{Rreq hops'} dip' dsn' dsk' oip' osn' sip' \Rightarrow \\
  \{ ξ | \text{hops} := \text{hops'}, \text{dip} := \text{dip'}, \text{dsn} := \text{dsn'}, \\
  \text{dsk} := \text{dsk'}, \text{oip} := \text{oip'}, \text{osn} := \text{osn'}, \text{sip} := \text{sip'} | \}
\end{align*}
| _ \Rightarrow \{\}"

lemma is_rreq_asm [dest!]:
assumes "ξ' \in is_rreq ξ"
shows "(∀ hops' rreqid' dip' dsn' oip' osn' sip'.
\begin{align*}
  msg ξ = \text{Rreq hops'} dip' dsn' dsk' oip' osn' sip' ∧ \\
  ξ' = ξ | \text{hops} := \text{hops'}, \text{dip} := \text{dip'}, \text{dsn} := \text{dsn'}, \\
  \text{dsk} := \text{dsk'}, \text{oip} := \text{oip'}, \text{osn} := \text{osn'}, \text{sip} := \text{sip'} |)"
using assms unfolding is_rreq_def
by (cases "msg ξ") simp_all

definition is_rrep
where "is_rrep ξ \equiv \text{case } msg ξ \text{ of}
\begin{align*}
  \text{Rrep hops'} dip' dsn' oip' sip' \Rightarrow \\
  \{ ξ | \text{hops} := \text{hops'}, \text{dip} := \text{dip'}, \text{dsn} := \text{dsn'}, \\
  \text{oip} := \text{oip'}, \text{sip} := \text{sip'} | \}
\end{align*}
| _ \Rightarrow \{\}"

lemma is_rrep_asm [dest!]:
assumes "ξ' \in is_rrep ξ"
shows "(∀ hops' dip' dsn' oip' sip'.
\begin{align*}
  msg ξ = \text{Rrep hops'} dip' dsn' oip' sip' ∧ \\
  ξ' = ξ | \text{hops} := \text{hops'}, \text{dip} := \text{dip'}, \text{dsn} := \text{dsn'}, \text{oip} := \text{oip'}, \text{sip} := \text{sip'} |)"
using assms unfolding is_rrep_def
by (cases "msg ξ") simp_all

definition is_rerr
where "is_rerr ξ \equiv \text{case } msg ξ \text{ of}
\begin{align*}
  \text{Rerr dests'} sip' \Rightarrow \\
  \{ ξ | \text{dests} := \text{dests'}, \text{sip} := \text{sip'} | \}
\end{align*}
| _ \Rightarrow \{\}"

lemma is_rerr_asm [dest!]:
assumes "ξ' \in is_rerr ξ"
shows "(∀ dests' sip'.
\begin{align*}
  msg ξ = \text{Rerr dests'} sip' ∧ \\
  ξ' = ξ | \text{dests} := \text{dests'}, \text{sip} := \text{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]:
\begin{align*}
  "ξ' \in is_rerr ξ \quad \Rightarrow \quad \text{ip } ξ' = \text{ip } ξ" \\
  "ξ' \in is_rrep ξ \quad \Rightarrow \quad \text{ip } ξ' = \text{ip } ξ" \\
  "ξ' \in is_rreq ξ \quad \Rightarrow \quad \text{ip } ξ' = \text{ip } ξ" \\
  "ξ' \in is_pkt ξ \quad \Rightarrow \quad \text{ip } ξ' = \text{ip } ξ" \\
  "ξ' \in is_newpkt ξ \quad \Rightarrow \quad \text{ip } ξ' = \text{ip } ξ"
\end{align*}
unfolding is_msg_defs
by (cases "msg ξ", clarsimp+) +

lemma is_msg_inv_sn [simp]:
\begin{align*}
  "ξ' \in is_rerr ξ \quad \Rightarrow \quad \text{sn } ξ' = \text{sn } ξ" \\
  "ξ' \in is_rrep ξ \quad \Rightarrow \quad \text{sn } ξ' = \text{sn } ξ" \\
  "ξ' \in is_rreq ξ \quad \Rightarrow \quad \text{sn } ξ' = \text{sn } ξ" \\
  "ξ' \in is_pkt ξ \quad \Rightarrow \quad \text{sn } ξ' = \text{sn } ξ" \\
  "ξ' \in is_newpkt ξ \quad \Rightarrow \quad \text{sn } ξ' = \text{sn } ξ"
\end{align*}
unfolding is_msg_defs
by (cases "msg ξ", clarsimp+) +
"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+)+

1.3.3 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
\[ \text{PKT args} \equiv \\
(\ \text{clear_locals} \ \xi. \ \text{let } (\text{data}, \text{dip}, \text{oip}) = \text{args} \ \xi \ \text{in} \\
(\ \text{data} := \text{data}, \ \text{dip} := \text{dip}, \ \text{oip} := \text{oip} \ ] \ ]
\text{call(PPkt)} \]

abbreviation NEWPKT
where
\[ \text{NEWPKT args} \equiv \\
(\ \text{clear_locals} \ \xi. \ \text{let } (\text{data}, \text{dip}) = \text{args} \ \xi \ \text{in} \\
(\ \text{data} := \text{data}, \ \text{dip} := \text{dip} \ ] \ ]
\text{call(PNewPkt)} \]

abbreviation RREQ
where
\[ \text{RREQ args} \equiv \\
(\ \text{clear_locals} \ \xi. \ \text{let } (\text{hops}, \text{dip}, \text{dsn}, \text{dsk}, \text{oip}, \text{osn}, \text{sip}) = \text{args} \ \xi \ \text{in} \\
(\ \text{hops} := \text{hops}, \ \text{dip} := \text{dip}, \ \text{dsn} := \text{dsn}, \ \text{dsk} := \text{dsk}, \ \text{oip} := \text{oip}, \ \text{osn} := \text{osn}, \ \text{sip} := \text{sip} \ ] \ ]
\text{call(PRreq)} \]

abbreviation RREP
where
\[ \text{RREP args} \equiv \\
(\ \text{clear_locals} \ \xi. \ \text{let } (\text{hops}, \text{dip}, \text{dsn}, \text{oip}, \text{sip}) = \text{args} \ \xi \ \text{in} \\
(\ \text{hops} := \text{hops}, \ \text{dip} := \text{dip}, \ \text{dsn} := \text{dsn}, \ \text{oip} := \text{oip}, \ \text{sip} := \text{sip} \ ] \ ]
\text{call(PRrep)} \]

abbreviation RERR
where
\[ \text{RERR args} \equiv \\
(\ \text{clear_locals} \ \xi. \ \text{let } (\text{dests}, \text{sip}) = \text{args} \ \xi \ \text{in} \\
(\ \text{dests} := \text{dests}, \ \text{sip} := \text{sip} \ ] \ ]
\text{call(PRerr)} \]

fun \( \Gamma_{AODV} : (\text{state, msg, pseqp, pseqp label}) \ \text{seqp_env} \)
where
\[ \Gamma_{AODV} \ \text{PAodv} = \text{labelled PAodv }\text{ (receive(}\lambda \text{msg' } \xi. \ \text{if } \text{msg := msg' } \ ] \ ]
(\ \text{is_newpkt) NEWPKT(}\lambda \xi. \ \text{(data } \xi, \ \text{ip } \xi) \]
\text{PKT(}\lambda \xi. \ \text{(data } \xi, \ \text{dip } \xi, \ \text{oip } \xi) \]
\text{RREQ(}\lambda \xi. \ \text{(hops } \xi, \ \text{dip } \xi, \ \text{dsn } \xi, \ \text{dsk } \xi, \ \text{oip } \xi, \ \text{osn } \xi, \ \text{sip } \xi) \]
\text{RREP(}\lambda \xi. \ \text{(hops } \xi, \ \text{dip } \xi, \ \text{dsn } \xi, \ \text{oip } \xi, \ \text{sip } \xi) \]
\text{RERR(}\lambda \xi. \ \text{(dests } \xi, \ \text{sip } \xi) \]
\text{all } \xi }}\]
\[
\text{groupcast}(\lambda_x. \text{pre } \xi, \lambda_x. \text{rerr}(\text{dests } \xi, \text{ip } \xi)). \text{AODV()} \\
\text{\{ } \xi\text{. dip }\xi = \text{ dip }\xi \text{\{ } \xi\text{. dip }\xi = \text{ dip }\xi \text{\{ } \xi\text{. dip }\xi = \text{ dip }\xi \\
\text{\} \}
\]
\[
\text{\{ } \xi\text{. dip }\xi \neq \text{ dip }\xi \text{\{ } \xi\text{. dip }\xi \neq \text{ dip }\xi \text{\{ } \xi\text{. dip }\xi \neq \text{ dip }\xi \\
\text{\} \}
\]
\[
\text{\{ } \xi\text{. dip }\xi \in \text{iD (rt }\xi) \text{\{ } \xi\text{. dip }\xi \in \text{iD (rt }\xi) \text{\{ } \xi\text{. dip }\xi \in \text{iD (rt }\xi) \\
\text{\} \}
\]
\[
\text{\{ } \xi\text{. dip }\xi \notin \text{vD (rt }\xi) \text{\{ } \xi\text{. dip }\xi \notin \text{vD (rt }\xi) \text{\{ } \xi\text{. dip }\xi \notin \text{vD (rt }\xi) \\
\text{\} \}
\]
\[
\text{\{ } \xi\text{. dip }\xi \notin \text{vD (rt }\xi) \text{\{ } \xi\text{. dip }\xi \notin \text{vD (rt }\xi) \text{\{ } \xi\text{. dip }\xi \notin \text{vD (rt }\xi) \\
\text{\} \}
\]
\[
\text{\{ } \xi\text{. dip }\xi \notin \text{vD (rt }\xi) \text{\{ } \xi\text{. dip }\xi \notin \text{vD (rt }\xi) \text{\{ } \xi\text{. dip }\xi \notin \text{vD (rt }\xi) \\
\text{\} \}
\]
\[
\text{\{ } \xi\text{. dip }\xi \notin \text{vD (rt }\xi) \text{\{ } \xi\text{. dip }\xi \notin \text{vD (rt }\xi) \text{\{ } \xi\text{. dip }\xi \notin \text{vD (rt }\xi) \\
\text{\} \}
\]
\[
\text{\{ } \xi\text{. dip }\xi \notin \text{vD (rt }\xi) \text{\{ } \xi\text{. dip }\xi \notin \text{vD (rt }\xi) \text{\{ } \xi\text{. dip }\xi \notin \text{vD (rt }\xi) \\
\text{\} \}
\]
\[rt := \text{the (addpreRT (rt ξ) (dip ξ) \{sip ξ\})} \]
\[rt := \text{the (addpreRT (rt ξ) (dip ξ) \{the (nhop (rt ξ) (dip ξ))\})} \]
\[\text{unicast(λξ. the (nhop (rt ξ) (oip ξ)), λξ. rrep(hops (dhops (rt ξ) (dip ξ)), dip ξ, sqn (rt ξ) (dip ξ), oip ξ, ip ξ)}.\]

\[AODV()\]

\[dests := (λrip. if (rip ∈ vD (rt ξ) \& nhop (rt ξ) rip = nhop (rt ξ) (oip ξ)) then Some (inc (sqn (rt ξ) rip)) else None) \]
\[rt := \text{invalidate (rt ξ) (dests ξ)} \]
\[store := \text{setRRF (store ξ)} \]
\[dests := (λrip. if ((dests ξ) rip ≠ None \& the (precs (rt ξ) rip) ≠ {})) then (dests ξ) rip else None) \]
\[\text{unicast(λξ. the (nhop (rt ξ) (oip ξ)), λξ. rrep(hops (dhops (rt ξ) (dip ξ)), dip ξ, sqn (rt ξ) (dip ξ), oip ξ, ip ξ)).} \]

\[AODV()\]

\[dests := (λrip. if (rip ∈ vD (rt ξ) \& nhop (rt ξ) rip = nhop (rt ξ) (oip ξ)) then Some (inc (sqn (rt ξ) rip)) else None) \]
\[rt := \text{invalidate (rt ξ) (dests ξ)} \]
\[store := \text{setRRF (store ξ)} \]
\[dests := (λrip. if ((dests ξ) rip ≠ None \& the (precs (rt ξ) rip) ≠ {})) then (dests ξ) rip else None) \]
\[\text{unicast(λξ. the (nhop (rt ξ) (oip ξ)), λξ. rrep(hops (dhops (rt ξ) (dip ξ)), dip ξ, sqn (rt ξ) (dip ξ), oip ξ, ip ξ)).} \]

\[AODV()\]

\[rt := \text{update (rt ξ) (dip ξ) (dsn ξ, kno, val, hops ξ + 1, sip ξ, {})} \]
\[\text{unicast(λξ. the (nhop (rt ξ) (oip ξ)), λξ. rrep(hops (dhops (rt ξ) (dip ξ)), dip ξ, sqn (rt ξ) (dip ξ), oip ξ, ip ξ)).} \]

\[AODV()\]

\[rt := \text{update (rt ξ) (dip ξ) (dsn ξ, kno, val, hops ξ + 1, sip ξ, {})} \]
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
  "$\Gamma_{AODV}.\text{skeleton} \ P\text{Aodv} = \text{seqp_skeleton} \ (\Gamma_{AODV} \ P\text{Aodv})""
  | "$\Gamma_{AODV}.\text{skeleton} \ P\text{NewPkt} = \text{seqp_skeleton} \ (\Gamma_{AODV} \ P\text{NewPkt})""
  | "$\Gamma_{AODV}.\text{skeleton} \ P\text{Pkt} = \text{seqp_skeleton} \ (\Gamma_{AODV} \ P\text{Pkt})""
  | "$\Gamma_{AODV}.\text{skeleton} \ P\text{Req} = \text{seqp_skeleton} \ (\Gamma_{AODV} \ P\text{Req})""
  | "$\Gamma_{AODV}.\text{skeleton} \ P\text{Rep} = \text{seqp_skeleton} \ (\Gamma_{AODV} \ P\text{Rep})""
  | "$\Gamma_{AODV}.\text{skeleton} \ P\text{Err} = \text{seqp_skeleton} \ (\Gamma_{AODV} \ P\text{Err})""

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

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

lemma aodv_proc_cases [dest]:
  fixes p pn
  shows "p $\in$ ctermsl (\Gamma_{AODV} pn) $\implies$
  (p $\in$ ctermsl (\Gamma_{AODV} \ P\text{Aodv}) $\lor$
  p $\in$ ctermsl (\Gamma_{AODV} \ P\text{NewPkt}) $\lor$
  p $\in$ ctermsl (\Gamma_{AODV} \ P\text{Pkt}) $\lor$
  p $\in$ ctermsl (\Gamma_{AODV} \ P\text{Req}) $\lor$
  p $\in$ ctermsl (\Gamma_{AODV} \ P\text{Rep}) $\lor$
  p $\in$ ctermsl (\Gamma_{AODV} \ P\text{Err})"
  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 \{(\text{aodv}\_\text{init} \ i, \Gamma_{AODV} \ P\text{Aodv})\}"

abbreviation paodv
  :: "ip $\Rightarrow$ (state $\times$ (state, msg, pseqp, pseqp label) seqp, msg seq_action) automaton"
where
  "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_within} \ \Gamma_{AODV} \ (\text{init} (paodv \ i))"
  unfolding $\sigma_{AODV}.\text{def}$ by (rule control_withinI) (auto simp del: $\Gamma_{AODV}.\text{simps}$)

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

lemmas aodv_labels_not_empty [simp] = labels_not_empty [OF aodv_wf]

lemma aodv_ex_label [intro]: "\exists \ l \in \text{labels} \ \Gamma_{AODV} \ p"
  by (metis aodv_labels_not_empty all_not_in_conv)
lemma aodv_ex_labelE [elim]:
  assumes "∀l∈labels Γ AODV p. P l p"
  and "∃p l. P l p ⇒ Q"
  shows "Q"
  using assms by (metis aodv_ex_label)

lemma aodv_simple_labels [simp]: "simple_labels Γ AODV"
proof
  fix pn p
  assume "p∈subterms(Γ AODV pn)"
  thus "∃!l. labels Γ AODV p = {l}" by (cases pn) (simp_all cong: seqp_congs | elim disjE)+
qed

lemma σ AODV_labels [simp]: "((ξ, p) ∈ σ AODV i)" unfolding σ AODV_def by simp

lemma aodv_init_kD_empty [simp]: "((ξ, p) ∈ σ AODV i)" 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 [cterms1_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

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 not_Pkt :: "msg ⇒ bool"
  where "not_Pkt m ≡ case m of Pkt _ _ _ ⇒ False | _ ⇒ True"

definition msg_sender :: "msg ⇒ ip"

112
where \( \text{"msg_sender } m \equiv \text{case } m \text{ of Rreq } \ldots \text{ ipc } \Rightarrow \text{ipc} \)
\| \text{Rrep } \ldots \text{ ipc } \Rightarrow \text{ipc} \\
\| \text{Rerr } \text{ ipc } \Rightarrow \text{ipc} \\
\| \text{Pkt } \ldots \text{ ipc } \Rightarrow \text{ipc}\)

**lemma msg_sender_simps [simp]:**

\(\forall \text{hops dip dsn oip osn sip.} \text{msg_sender (Rreq hops dip dsn dsk oip osn sip) = sip}\\
\forall \text{hops dip dsn oip sip.} \text{msg_sender (Rrep hops dip dsn oip sip) = sip}\\
\forall \text{dests sip.} \text{msg_sender (Rerr dests sip) = sip}\\
\forall \text{d dip.} \text{msg_sender (Pkt d dip sip) = sip}\)

unfolding msg_sender_def by simp_all

**definition msg_zhops :: "msg \Rightarrow bool"**

where \( \text{msg_zhops } m \equiv \text{case } m \text{ of Rreq hops dip dsn dsk oip osn sip = 0 } \Rightarrow \text{oipc = sipc} \\
\| \text{Rrep hops dip dsn oip sip = 0 } \Rightarrow \text{dipc = sipc} \\
\| \ldots \Rightarrow \text{True}\)

**lemma msg_zhops_simps [simp]:**

\(\forall \text{hops dip dsn oip osn sip.} \text{msg_zhops (Rreq hops dip dsn dsk oip osn sip) = (hops = 0 } \Rightarrow \text{dipc = sipc)\\
\forall \text{hops dip dsn oip sip.} \text{msg_zhops (Rrep hops dip dsn oip sip) = (hops = 0 } \Rightarrow \text{dipc = sipc)\\
\forall \text{dests sip.} \text{msg_zhops (Rerr dests sip) = True}\\
\forall \text{d dip.} \text{msg_zhops (Newpkt d dip) = True}\\
\forall \text{d dip.} \text{msg_zhops (Pkt d dip sip) = True}\)

unfolding msg_zhops_def by simp_all

**definition rreq_rrep_sn :: "msg \Rightarrow bool"**

where \( \text{rreq_rrep_sn } m \equiv \text{case } m \text{ of Rreq hops dip dsn dsk oip osn sip = 0 } \Rightarrow \text{osnc = 1} \\
\| \text{Rrep hops dip dsn oip sip = 0 } \Rightarrow \text{dsnc = 1} \\
\| \ldots \Rightarrow \text{True}\)

**lemma rreq_rrep_sn_simps [simp]:**

\(\forall \text{hops dip dsn oip osn sip.} \text{rreq_rrep_sn (Rreq hops dip dsn dsk oip osn sip) = (osn = 1)\\
\forall \text{hops dip dsn oip sip.} \text{rreq_rrep_sn (Rrep hops dip dsn oip sip) = (dsn = 1)\\
\forall \text{dests sip.} \text{rreq_rrep_sn (Rerr dests sip) = True}\\
\forall \text{d dip.} \text{rreq_rrep_sn (Newpkt d dip) = True}\\
\forall \text{d dip.} \text{rreq_rrep_sn (Pkt d dip sip) = True}\)

unfolding rreq_rrep_sn_def by simp_all

**definition rreq_rrep_fresh :: "rt \Rightarrow msg \Rightarrow bool"**

where \( \text{rreq_rrep_fresh } crt \ m \equiv \text{case } m \text{ of Rreq hopsc dipc _ _ oipc osnc ipcc = 0 } \Rightarrow \text{ipcc \notin kD(crt)} \\
\| \text{Rrep hopsc dipc _ _ oipc osnc ipcc = 0 } \Rightarrow \text{dipc = dipc = oipc = ipcc} \\
\| \ldots \Rightarrow \text{True}\)

**lemma rreq_rrep_fresh [simp]:**

\(\forall \text{hops dip dsn oip osn sip.} \text{rreq_rrep_fresh (Rreq hopsc dipc _ _ oipc osnc ipcc = (ipcc = dipc = oipc = ipcc)\\
\| \text{Rrep hopsc dipc _ _ oipc osnc ipcc = (ipcc = dipc = oipc = ipcc)\\
\| \ldots \Rightarrow \text{True}\)

\(\forall \text{hops dip dsn oip osn sip.} \text{rreq_rrep_fresh (Rreq hopsc dipc _ _ oipc osnc ipcc = (ipcc = dipc = oipc = ipcc)\\
\| \text{Rrep hopsc dipc _ _ oipc osnc ipcc = (ipcc = dipc = oipc = ipcc)\\
\| \ldots \Rightarrow \text{True}\)

\(\forall \text{hops dip dsn oip osn sip.} \text{rreq_rrep_fresh (Rreq hopsc dipc _ _ oipc osnc ipcc = (ipcc = dipc = oipc = ipcc)\\
\| \text{Rrep hopsc dipc _ _ oipc osnc ipcc = (ipcc = dipc = oipc = ipcc)\\
\| \ldots \Rightarrow \text{True}\)
definition rerr_invalid :: "rt ⇒ msg ⇒ bool"
where "rerr_invalid crt m ⇒ (∀ ripc∈dom(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) = (∀ rip ∈ dom(dests).
rip∈iD(crt) ∧ the (dests rip) = sqn crt rip)"
"∀ 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

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 = (λs. P (default aodv_init (netlift fst s)))"

end

1.5 Quality relations between routes

theory A_Fresher
imports A_Aodv_Data
begin

1.5.1 Net sequence numbers

On individual routes

definition
\[ nsqn_r :: "r \Rightarrow sqn" \]

where
\[ "nsqn, r \equiv \text{if } \pi_4(r) = \text{val} \lor \pi_2(r) = 0 \text{ then } \pi_2(r) \text{ else } (\pi_2(r) - 1)" \]

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

unfolding \( \text{nsqn}_{-\text{def}} \) by simp

lemma \( \text{nsqn}_{-\text{zero}} \; [\text{simpl}] \):
\[ "\forall dsn \; dsk \; flag \; hops \; nhip \; pre. \; nsqn_r \; (dsn, \; dsk, \; flag, \; hops, \; nhip, \; pre) = 0" \]

unfolding \( \text{nsqn}_{-\text{def}} \) by clarsimp

lemma \( \text{nsqn}_{-\text{val}} \; [\text{simpl}] \):
\[ "\forall dsn \; dsk \; hops \; nhip \; pre. \; nsqn_r \; (dsn, \; dsk, \; val, \; hops, \; nhip, \; pre) = dsn" \]

unfolding \( \text{nsqn}_{-\text{def}} \) by clarsimp

lemma \( \text{nsqn}_{-\text{inv}} \; [\text{simpl}] \):
\[ "\forall dsn \; dsk \; hops \; nhip \; pre. \; nsqn_r \; (dsn, \; dsk, \; inv, \; hops, \; nhip, \; pre) = dsn - 1" \]

unfolding \( \text{nsqn}_{-\text{def}} \) by clarsimp

lemma \( \text{nsqn}_{-\text{lte_dsn}} \; [\text{simpl}] \):
\[ "\forall dsn \; dsk \; flag \; hops \; nhip \; pre. \; nsqn_r \; (dsn, \; dsk, \; flag, \; hops, \; nhip, \; pre) \leq dsn" \]

unfolding \( \text{nsqn}_{-\text{def}} \) by clarsimp

On routes in routing tables

definition
\[ nsqn :: "rt \Rightarrow \text{ip} \Rightarrow sqn" \]

where
\[ "nsqn \equiv \lambda rt \; dip. \; \text{case } \sigma_{\text{route}}(rt, \; dip) \text{ of } \text{None} \Rightarrow 0 \; | \; \text{Some } r \Rightarrow nsqn_r(r)" \]

lemma \( \text{nsqn}_{-\text{sqn_def}} \):
\[ "\forall rt \; dip. \; nsqn \; rt \; dip = (\text{if } flag \; rt \; dip = \text{Some val} \lor \; sqn \; rt \; dip = 0 \text{ then } sqn \; rt \; dip \text{ else } sqn \; rt \; dip - 1)" \]

unfolding \( \text{nsqn}_{-\text{def}} \) \( \text{sqn}_{-\text{def}} \) by (clarsimp split: option.split) auto

lemma \( \text{not_in_kD_nsqn} \; [\text{simpl}] \):
assumes \( \text{"dip} \notin kD(rt)" \)
shows \( "nsqn \; rt \; dip = 0" \)

using assms unfolding \( \text{nsqn}_{-\text{def}} \) by simp

lemma \( \text{kD_nsqn} \):
assumes \( \text{"dip} \in kD(rt)" \)
shows \( "nsqn \; rt \; dip = nsqn_r((\sigma_{\text{route}}(rt, \; dip)))" \)

using assms \([\text{THEN kD_Some}]\) unfolding \( \text{nsqn}_{-\text{def}} \) by clarsimp

lemma \( \text{nsqnr_r_flag_pred} \; [\text{simpl, intro}] \):
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 \( \text{nsqnr_r_addpreRT_inv} \; [\text{simpl}] \):
\[ "\forall rt \; dip \; npre \; dip'. \; dip \in kD(rt) \Rightarrow nsqn_r ((\text{the } (\text{addpreRT } rt \; dip \; npre) \; dip')) = nsqn_r ((\text{the } (rt \; dip'))" \]

unfolding \( \text{addpreRT}_{-\text{def}} \) \( \text{nsqn}_{-\text{def}} \) by (frule \( \text{kD_Some} \)) (clarsimp split: option.split)

lemma \( \text{sqn_nsqn} \):
\[ "\forall rt \; dip. \; sqn \; rt \; dip - 1 \leq nsqn \; rt \; dip" \]

unfolding \( \text{sqn}_{-\text{def}} \) \( \text{nsqn}_{-\text{def}} \) by (clarsimp split: option.split)

lemma \( \text{nsqn_sqn} \):
\[ "nsqn \; rt \; dip \leq sqn \; rt \; dip" \]
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_r_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_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 ≠ 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

116
using \texttt{dests dip = Some rsn} by simp

qed

lemma \texttt{nsqn_invalidate_other [simp]}:
  assumes "dip \in \texttt{kD(rt)}"
  and "dip \notin \texttt{dom dests}"
  shows "\texttt{nsqn (invalidate rt dests) dip = nsqn rt dip}"
  using \texttt{assms} by (clarsimp simp add: \texttt{kD_nsqn})

1.5.2 Comparing routes

definition
\texttt{fresher :: \textit{r} ⇒ \textit{r} ⇒ bool} ("\textit{r} ⊑ \textit{r}'") [51, 51] 50
where
"\texttt{fresher r r'} ≡ ((\texttt{nsqn \_ r < nsqn \_ r'}) \lor (\texttt{nsqn \_ r = nsqn \_ r'} \land \pi_5(r) \geq \pi_5(r')))"

lemma \texttt{fresherI1 [intro]}:
  assumes "\texttt{nsqn \_ r < nsqn \_ r’}"
  shows "\texttt{r ⊑ r’}"
  unfolding \texttt{fresher_def} using \texttt{assms} by simp

lemma \texttt{fresherI2 [intro]}:
  assumes "\texttt{nsqn \_ r = nsqn \_ r’} \land \pi_5(r) \geq \pi_5(r’)"
  shows "\texttt{r ⊑ r’}"
  unfolding \texttt{fresher_def} using \texttt{assms} by simp

lemma \texttt{fresherI [intro]}:
  assumes "\texttt{(nsqn \_ r < nsqn \_ r’) \lor (nsqn \_ r = nsqn \_ r’} \land \pi_5(r) \geq \pi_5(r’))"
  shows "\texttt{r ⊑ r’}"
  unfolding \texttt{fresher_def} using \texttt{assms}.

lemma \texttt{fresherE [elim]}:
  assumes "\texttt{r ⊑ r’}"
  and "\texttt{nsqn \_ r < nsqn \_ r’} = \Rightarrow \texttt{P r r’}"
  and "\texttt{nsqn \_ r = nsqn \_ r’} \land \pi_5(r) \geq \pi_5(r’) = \Rightarrow \texttt{P r r’}"
  shows "\texttt{P r r’}"
  using \texttt{assms} unfolding \texttt{fresher_def} by auto

lemma \texttt{fresher_refl [simp]}: "\texttt{r ⊑ r}"
  unfolding \texttt{fresher_def} by simp

lemma \texttt{fresher_trans [elim, trans]}:
  "[ x ⊑ y; y ⊑ z ] \Rightarrow x ⊑ z"
  unfolding \texttt{fresher_def} by auto

lemma \texttt{not_fresher_trans [elim, trans]}:
  "[ \neg(x ⊑ y); \neg(z ⊑ x) ] \Rightarrow \neg(z ⊑ y)"
  unfolding \texttt{fresher_def} by auto

lemma \texttt{fresher_dsn_flag_hops_const [simp]}:
  fixes \texttt{dsn dsk dsk’ flag hops nhip nhip’ pre pre’}
  shows "\texttt{(dsn, dsk, flag, hops, nhip, pre pre’} \subseteq \texttt{(dsn, dsk’, flag, hops, nhip’, pre’)}"
  unfolding \texttt{fresher_def} by (cases flag) simp_all

lemma \texttt{addpre_fresher [simp]}: "\texttt{\land r npre. r ⊑ (addpre r npre)}"
  by clarsimp

1.5.3 Comparing routing tables

definition
\texttt{rt_fresher :: \textit{ip ⇒ rt ⇒ rt ⇒ bool}}
where
"\texttt{rt_fresher ≡ λdip rt rt'. (the (\sigma_{route}(rt, dip))) ⊊ (the (\sigma_{route}(rt’, dip)))}"
abbreviation
  rt_fresher_syn :: "rt ⇒ ip ⇒ rt ⇒ bool" ("(_/ _/ _)" [51, 999, 51] 50)
where
  "rt1 ⊑ 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 ≈ dip 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') ≤ 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 "¬ (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 `rt_fresher_mapupd [intro!]
  assumes "dip ∈ kD(rt)"
  shows "dip ∈ kD(rt)"
  proof (auto simp: rt_fresher_def2 [OF `dip ∈ kD(rt)`] kD_nsqn [OF `dip ∈ kD(rt)`])

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")
assumption 
next
assumption "dip = ip"
from ⟨dip ∈ kD(rt)⟩ obtain dsn_n dsk_n f_n hops_n nhip_n pre_n
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)"
⊆ 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, nhip_n, pre_n) ⊑ (dsn, kno, val, hops, nhip, pre ⊔ pre_n)"
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

definition of nsqn_r

theorem rt_fresher_invalidate [simp]:
assumes "dip ∈ kD(rt)"
and "dip ∈ dom(dests)"
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
proof (simp_all add: ⟨dip ∈ dom(dests)⟩)
next
assume "dip ∈ dom(dests)"
moreover with assms have "dip ∈ vD(rt)"
and "sqn rt dip < the (dests rip)"
ultimately show ?thesis
unfolding invalidate_def sqn_def
by (rule single_rt_fresher, simp)

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 sqn_def
by auto

lemma rt_fresh_as_inc_invalidate [simp]:
assumes "dip ∈ kD(rt)"
and "∀ 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)"
with ⟨dip ∈ kD(rt)⟩ have "dip ∈ kD(invalidate rt dests)"
by simp
ultimately show ?thesis
proof (rule nsqn_rt_fresh_asI)
from ⟨dip ∈ vD(rt)⟩ have "nsqn rt dip = sqn rt dip" by auto
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))" by simp

qed
also from \( \text{dip} \in \text{kd}(\text{invalidate rt dests}) \),

have "\( \text{nsqn}_r \ (\text{the} \ (\text{invalidate rt dests} \ \text{dip})) = \text{nsqn} \ (\text{invalidate rt dests} \ \text{dip}) \)"

by \((\text{simp add: kd}\_\text{nsqn})\)

finally show "\( \text{nsqn} \ \text{rt} \ \text{dip} = \text{nsqn} \ (\text{invalidate rt dests} \ \text{dip}) \)".

qed simp

lemmas rt_fresher_invalidate \[\text{simp} = \text{rt\_fresh\_as\_inc\_invalidate} \ [\text{THEN rt\_fresh\_asD}1]\]

lemma rt_fresh_as_addpreRT \[\text{simp}:

assumes \( \text{ip} \in \text{kd}(\text{rt}) \)

shows "\( \text{rt} \approx \text{dip} \ (\text{addpreRT} \ \text{rt ip npre}) \)"

using \( \text{assms} \ [\text{THEN kd}\_\text{Some}] \) by \((\text{auto simp: addpreRT}\_\text{def})\)

lemmas rt_fresher_addpreRT \[\text{simp} = \text{rt\_fresh\_as\_addpreRT} \ [\text{THEN rt\_fresh\_asD}1]\]

1.5.4 Strictly comparing routing tables

definition rt_strictly_fresher :: "\text{ip} \Rightarrow \text{rt} \Rightarrow \text{rt} \Rightarrow \text{bool}"

where

\( \text{rt\_strictly\_fresher} \equiv \lambda \text{dip} \ \text{rt}1 \ \text{rt}2. \ (\text{rt}1 \sqsubseteq_{\text{dip}} \text{rt}2) \land \neg(\text{rt}1 \approx_{\text{dip}} \text{rt}2) \)"

abbreviation

\( \text{rt\_strictly\_fresher\_\text{syn}} :: "\text{rt} \Rightarrow \text{ip} \Rightarrow \text{rt} \Rightarrow \text{bool}" \ ("/_/ \sqsubseteq /\_\)" \ [51, 999, 51] 50)

where

\( \text{rt}1 \sqsubseteq_{\text{i}} \text{rt}2 \equiv \text{rt\_strictly\_fresher} \ \text{i} \ \text{rt}1 \ \text{rt}2 \)"

lemma rt_strictly_fresher_def'':

"\( \text{rt}1 \sqsubseteq \_ \text{rt}2 = ((\text{rt}1 \sqsubseteq_{\text{i}} \text{rt}2) \land \neg(\text{rt}2 \sqsubseteq_{\text{i}} \text{rt}1)) \)"

unfolding rt_strictly_fresher_def by auto

lemma rt_strictly_fresherI' \[\text{intro}:

assumes \( \text{rt}1 \sqsubseteq_{\text{i}} \text{rt}2 \)

and \"\( \neg(\text{rt}2 \sqsubseteq_{\text{i}} \text{rt}1) \)"

shows "\( \text{rt}1 \sqsubseteq_{\text{i}} \text{rt}2 \)"

using \( \text{assms} \ [\text{unfolding rt\_strictly\_fresher\_def''}] \) by simp

lemma rt_strictly_fresherE' \[\text{elim}:

assumes \( \text{rt}1 \sqsubseteq_{\text{i}} \text{rt}2 \)

and \"\( \neg(\text{rt}2 \sqsubseteq_{\text{i}} \text{rt}1) \) \]

shows "\( \text{rt}1 \approx_{\text{i}} \text{rt}2 \)"

using \( \text{assms} \ [\text{unfolding rt\_strictly\_fresher\_def''}] \) by simp

lemma rt_strictly_fresherI [intro]:

assumes "\( \text{rt}1 \sqsubseteq_{\text{i}} \text{rt}2 \)"

and "\( \neg(\text{rt}1 \approx_{\text{i}} \text{rt}2) \)"

shows "\( \text{rt}1 \sqsubseteq_{\text{i}} \text{rt}2 \)"

unfolding rt_strictly_fresher_def using assms ..

lemmas rt_strictly_fresher_singleI \[\text{elim} = \text{rt\_strictly\_fresher\_I} \ [\text{OF single\_rt\_fresher}]\]

lemma rt_strictly_fresherE \[\text{elim}:

assumes "\( \text{rt}1 \sqsubseteq_{\text{i}} \text{rt}2 \)"

and "\( \neg(\text{rt}1 \approx_{\text{i}} \text{rt}2) \) \]

shows "\( \text{rt}1 \approx_{\text{i}} \text{rt}2 \)"

using \( \text{assms}(1) \ [\text{unfolding rt\_strictly\_fresher\_def}] \)

by rule \((\text{erule}(1) \ \text{assms}(2))\)

lemma rt_strictly_fresher_def':

"\( \text{rt}1 \sqsubseteq_{\text{i}} \text{rt}2 = (\text{nsqn}_r \ (\text{the} \ (\text{rt}1 \ \text{i})) < \text{nsqn}_r \ (\text{the} \ (\text{rt}2 \ \text{i})) \)

\lor (\text{nsqn}_r \ (\text{the} \ (\text{rt}1 \ \text{i})) = \text{nsqn}_r \ (\text{the} \ (\text{rt}2 \ \text{i})) \land \pi_5(\text{the} \ (\text{rt}1 \ \text{i})) > \pi_5(\text{the} \ (\text{rt}2 \ \text{i})))""

unfolding rt_strictly_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 rt2)" unfolding rt_strictly_fresher_def'' by auto
    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 rt2)" unfolding rt_strictly_fresher_def'' by auto
    moreover from ⟨¬(rt3 ⪯ dip rt1)⟩ 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 ⊑ dip 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 \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 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
by clarsimp (metis (hide_lams, no_types) assms vD_Some vD_invalidate_vD_not_dests)
ultimately show \( \text{thesis} \)
unfolding invalidate_def rt_strictly_fresher_def' by auto
qed

lemma rt_strictly_fresher_update_other [elim!]:
  "\( \forall dip \ ip \ rt \ rt'. \ [ \ dip \neq ip; \ rt \in\ dip \ rt' \ ] \implies \text{update} \ rt \ ip \ rt \in\ dip \ rt' \)"
unfolding rt_strictly_fresher_def' by clarsimp

lemma addpreRT_strictly_fresher [simp]:
assumes "dip \in kD(rt)"
shows "\( \text{addpreRT} \ rt \ dip \ npre \subseteq \text{ip} \ rt2 \) = \( \rt \subseteq \text{ip} \ rt2 \)"
using assms unfolding rt_strictly_fresher_def' by clarsimp

lemma lt_sqn_imp_update_strictly_fresher:
assumes "dip \in vD (rt2 nhip)"
and \( \osn < \text{sqn} (rt2 nhip) \) dip
and \( \osn \neq \text{update} \ rt \ dip (osn, kno, val, hops, nhip, \{\}) \) dip
shows "\( \text{update} \ rt \ dip (osn, kno, val, hops, nhip, \{\}) \in\ dip \ rt2 \)"
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 \( \text{dip} \subseteq \text{vD} \ (rt2 nhip) \) dip
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 \( \text{dip} \subseteq \text{vD} \ (rt2 nhip) \) dip
by - (metis vD_flag_val proj2_eq_sqn proj4_eq_flag vD_iD_gives_kD(1))
finally show "\( \text{nsqn, (the (update rt dip (osn, kno, val, hops, nhip, \{\})) dip)} < \text{nsqn, (the (rt2 nhip dip))} \)".
qed

lemma dhops_le_hops_imp_update_strictly_fresher:
assumes "dip \in vD(rt2 nhip)"
and sqn: "\( \text{sqn} (rt2 nhip) \) dip = osn"
and hop: "the \( \text{dhops} \ (rt2 nhip) \) dip \leq \text{hops}"
and **: "\( \osn \neq \text{update} \ rt \ dip (osn, kno, val, Suc \text{hops}, nhip, \{\}) \) dip"
shows "\( \text{update} \ rt \ dip (osn, kno, val, Suc \text{hops}, nhip, \{\}) \in\ dip \ rt2 \)"
unfolding rt_strictly_fresher_def' proof (rule disjI2, rule conjI)
from ** have "nsqn (update rt dip (osn, kno, val, Suc \text{hops}, nhip, \{\})) dip = osn"
by (rule nsqn_update_changed_kno_val)
with \( \text{dip} \subseteq \text{vD} \ (rt2 nhip) \) dip
have "nsqn, (the (update rt dip (osn, kno, val, Suc \text{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 \( \text{dip} \subseteq \text{vD} \ (rt2 nhip) \) dip
by - (metis vD_flag_val proj2_eq_sqn proj4_eq_flag vD_iD_gives_kD(1))
finally show "\( \text{nsqn, (the (update rt dip (osn, kno, val, Suc \text{hops}, nhip, \{\})) dip)} = \text{nsqn, (the (rt2 nhip dip))} \)".
next
have "the \( \text{dhops} \ (rt2 nhip) \) dip \leq \text{hops}" by (rule hop)
also have "hops < \text{hops} + 1" by simp
also have "\text{hops} + 1 = the \( \text{dhops} \ (update \ rt \ dip (osn, kno, val, Suc \text{hops}, nhip, \{\})) \) dip"
using ** by simp
finally have "the \( \text{dhops} \ (rt2 nhip) \) dip < the \( \text{dhops} \ (update \ rt \ dip (osn, kno, val, Suc \text{hops}, nhip, \{\})) \) dip".
thus "\( \pi_5 (the (rt2 nhip dip)) < \pi_5 (the (update rt dip (osn, kno, val, Suc \text{hops}, nhip, \{\})) dip) \)".
using \( \text{dip} \subseteq \text{vD} \ (rt2 nhip) \) dip by (simp add: proj5_eq_dhops)
qed

126
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

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 │= onl 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 │= (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 ξ) (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

{ 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 (0, unk, val, Suc 0, ip, {})\})"
  and "sip ∈ kD (rt ξ)"
  hence "(the (nhop (update (rt ξ) ip (0, unk, val, Suc 0, ip, {})) ip) = ip"
  ∨ the (nhop (update (rt ξ) ip (0, unk, val, Suc 0, ip, {})) ip) ∈ kD (rt ξ)"
  by clarsimp (metis nhop_update_unk_val update_another)
} note one_hop = 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 ||| = onll Γ AODV (\(\lambda (ξ, l). (l ∈ \{PRreq-:16..PRreq-:18\} → dip ξ ∈ kD (rt ξ)) ∧ (l = PRreq-:17 → oip ξ ∈ kD (rt ξ)) ∧ (l = PRreq-:5 → dip ξ ∈ kD (rt ξ)) ∧ (l = PRreq-:6 → (the (nhop (rt ξ) (dip ξ))) ∈ kD (rt ξ)))")

(is "_ ||| = onll 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 "onll Γ AODV ?P (ξ, p)"
  proof (rule onllI)
    fix onll
    assume "1 ∈ labels Γ AODV p"
    with (\(\xi, p\) ∈ 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-:2..PRreq-:6\} → dip ξ ∈ kD (rt ξ)"
    by (auto dest!: invariantD [OF addpreRT_partly_welldefined])
    moreover from (\(\xi, p\) ∈ 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
    qed
    with \(s = (ξ, p)\) show "onll Γ AODV ?P s"
    by simp
  qed

Proposition 7.4

lemma known_destinations_increase:
"paodv i |= A onll Γ AODV (\(\lambda ((ξ, _), (_), (ξ', _)). kD (rt ξ) ⊆ kD (rt ξ'))\)"
by (inv_cterms inv add: onll_invariant_sterms [OF aodv_wf addpreRT_welldefined] simp add: subset_insert1)
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, _), _, (ξ', _)). rreqs ξ ⊆ rreqs ξ')" 
proof -
  have sqninv:
    "∀ dests rt rsn ip. sqn (rt ξ) ip ≤ sqn (invalidate (rt ξ) (dests ξ ip))" 
  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)
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 -
  have indests:
    "∀ dests rt rsn ip. sqn (rt ξ) ip ≤ sqn (invalidate (rt ξ) (dests ξ ip))" 
  show ?thesis
  by (inv_cterms split_if_asm option.split_asm elim!: sqninv indests)
qed

Proposition 7.7

lemma ip_constant:
  "paodv i |= onll Γ_AODV (λ((ξ, _), _, (ξ', _)). ip = 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 |= onll Γ_AODV (λ((ξ, _), a, _). anycast (λm. not_Pkt m −→ msg_sender m = ip) a)"
by (rule step_invariant_weaken_with_invariantE [OF ip_constant sender_ip_valid'])
lemma received_msg_inv:
"paodv i |= (recvmsg P ⊃) onl $\Gamma_{AODV}$ ($\lambda(\xi, 1). \ l \in \{PAodv←:1\} \ −→ \ P \ (msg \ \xi)$)"
by inv_cterms

Proposition 7.9

lemma sip_not_ip:
"paodv i |= (recvmsg ($\lambda m. \ not_Pkt \ m \ −→ \ msg_sender \ m \neq i$) \ −→ \ onl $\Gamma_{AODV}$ ($\lambda(\xi, \ _). \ sip \ \xi \neq ip \ \xi$)"
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 $\Gamma_{AODV}$ ($\lambda(\xi, \ _). \ sip \ \xi \neq ip \ \xi$)"
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+

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

Proposition 7.10

lemma hop_count_positive:
"paodv i |= onl $\Gamma_{AODV}$ ($\lambda(\xi, \ _). \ \forall ip\in kD \ (rt \ \xi). \ \the (dhops (rt \ \xi) \ ip) \geq 1$)"
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 $\Gamma_{AODV}$ ($\lambda(\xi, 1). \ (l \in \{PRreq←:16..PRreq←:18\} \ −→ \ dip \ \xi \in vD(rt \ \xi))$
∧ (l \in \{PRreq←:5, PRreq←:6\} \ −→ \ dip \ \xi = ip \ \xi)
∧ (l \in \{PRreq←:15..PRreq←:18\} \ −→ \ dip \ \xi \neq ip \ \xi))"

proof (inv_cterms, elim conjE)
fix $l \in pp \ p'$
assume "$((\xi, pp) \in \ reachable \ (paodv i) \ TT$"
and "$\{PRreq←:17\}[l\xi. \ \xi[rt := \ \the (addpreRT (rt \ \xi) \ (oip \ \xi) \ (the (nhop (rt \ \xi) \ (dip \ \xi))))] \ p' \in \ sterms \ \Gamma_{AODV} \ pp$"
and "$l = PRreq←:17"$
and "$dip \ \xi \in vD \ (rt \ \xi)$"
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 \dip \ \xi \in vD \ (rt \ \xi)
show "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:
"\l\lreqid \ dip \ dsn \ dsk \ oip \ osn \ sip. \ paodv i |=A onll $\Gamma_{AODV}$ ($\lambda(\_, \ a, \ _). \ \anycast \ msg_zhops \ a$)"
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 $l \in pp \ p' \ pp'$
assume "$((\xi, pp) \in \ reachable \ (paodv i) \ TT$"
and "$\{PRreq←:18\}unicast(\lambda. \ \the (nhop (rt \ \xi) \ (oip \ \xi)), \lambda. \ Rrep (the (dhops (rt \ \xi) (dip \ \xi))) \ (dip \ \xi) \ (sqn (rt \ \xi) (dip \ \xi)) \ (a \ (oip \ \xi) \ (ip \ \xi)). \ p' \in \ sterms \ \Gamma_{AODV} \ pp$"
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 "$:\forall \ip\in kD \ (rt \ \xi). \ Suc \ 0 \leq \ (\the (dhops (rt \ \xi) \ ip))$"
and "$dip \ \xi \in vD \ (rt \ \xi)"$
from $\dip \ \xi \in vD \ (rt \ \xi)$ have "$dip \ \xi \in kD \ (rt \ \xi)"
by (rule vD_iD_gives_kD(1))
with $*$ have "Suc $0 \leq \ (\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 ΓADIOV (λ(ξ, 1).
(1 ∈{PAodv:-4..PAodv:-5} ∪ {PRreq-n|n. True} →
(hops ξ = 0 → oip ξ = sip ξ))
∧
((1 ∈{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_rqreq:
"paodv i ||= (recvmsg rreq_rrep_sn →) onl ΓADIOV (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_rqreq':
"paodv i ||= (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) →) onl ΓADIOV (λ(ξ, 1).
1 ∈ {PAodv:-6, PAodv:-7} ∪ {PRrep-n|n. True} → 1 ≤ dsn ξ)"

proof (rule invariant_weakenE [OF osn_rqreq])

qed

lemma dsn_rrep:
"paodv i ||= (recvmsg rreq_rrep_sn →) onl ΓADIOV (λ(ξ, 1).
1 ∈ {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_rqreq':
"paodv i ||= (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) →) onl ΓADIOV (λ(ξ, 1).
1 ∈ {PAodv:-6, PAodv:-7} ∪ {PRrep-n|n. True} → 1 ≤ dsn ξ)"

proof (rule invariant_weakenE [OF dsn_rrep])

qed

lemma hop_count_zero_oip_dip_sip':
"paodv i ||= (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) →) onl ΓADIOV (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])

qed

Proposition 7.12

lemma zero_seq_unk_hops_one':
"paodv i ||= (recvmsg rreq_rrep_sn →) onl ΓADIOV (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

131
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

{ fix dsn hops :: nat and sip oip rt and ip dip :: ip
  assume "∀dip ∈ kD(rt).
    (sqn rt dip = 0 → π_3(the (rt dip)) = unk) ∧
    (π_3(the (rt dip)) = unk → the (dhaps rt dip) = Suc 0) ∧
    (the (dhaps 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 (dhaps (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)
}

{ fix ip dsn hops sip oip rt dip
  assume "∀dip ∈ kD(rt).
    (sqn rt dip = 0 → π_3(the (rt dip)) = unk) ∧
    (π_3(the (rt dip)) = unk → the (dhaps rt dip) = Suc 0) ∧
    (the (dhaps rt dip) = Suc 0 → the (nhop rt dip) = dip)"
  and "Suc 0 ≤ dsn"
  and "ip ≠ dip → ip ∈ kD(rt)"
hence "π_3(the (update rt dip (dsn, kno, val, Suc hops, sip, {})) ip)) = unk → the (dhaps (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)
}

{ fix rt sip
  assume "∀dip ∈ kD rt.
    (sqn rt dip = 0 → π_3(the (rt dip)) = unk) ∧
    (π_3(the (rt dip)) = unk → the (dhaps rt dip) = Suc 0) ∧
    (the (dhaps 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 →
      π_3(the (update rt sip (0, unk, val, Suc 0, sip, {})) dip) = unk)
    ∧ (π_3(the (update rt sip (0, unk, val, Suc 0, sip, {})) dip) = unk →
      the (dhaps (update rt sip (0, unk, val, Suc 0, sip, {})) dip) = Suc 0)
    ∧ (the (dhaps (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 add: sqnf_def sqn_def)
}

have prreq_ok5 [simp]: "∀sip rt.
\[\begin{align*}
\pi_3&(\text{the (update rt sip (0, unk, val, Suc 0, sip, \{\} sip))}) = \text{unk} \implies \\
\text{the (dhops (update rt sip (0, unk, val, Suc 0, sip, \{\} sip)) sip)} &= \text{Suc 0} \\
\text{by (rule update_cases) simp_all}
\end{align*}\]

have prreq_ok6 [simp]: "\(\forall \text{sip rt.}
\text{sqn (update rt sip (0, unk, val, Suc 0, sip, \{\} sip)) sip} = \text{Suc 0} \implies \\
\pi_3 \text{ (the (update rt sip (0, unk, val, Suc 0, sip, \{\} sip)) sip)} = \text{unk}\)" \\
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: "\(\text{paodv i ||} \rightarrow \text{recvmsg (λ m. rreq_rrep_sn m ∧ msg_zhops m)} \rightarrow \text{onl} \) Γ AODV (λ (ξ, _).
\∀ \text{dip} \in \text{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: "\(\text{paodv i ||} \rightarrow \text{recvmsg rreq_rrep_sn} \rightarrow \text{onl} \) Γ AODV (λ (ξ, l).
\∀ \text{dip} \in \text{kD(rt ξ)}. \pi_3 \text{(the (rt ξ dip))} = \text{unk} ∨ \text{Suc 0} \leq \pi_2 \text{(the (rt ξ dip))})"
proof - \\
{ fix sip rt dsn1 dsn2 dsk1 dsk2 flag1 flag2 hops1 hops2 nhip1 nhip2 pre1 pre2 \\
assume "dsk1 = unk ∨ Suc 0 ≤ dsn2" \\
hence "\(\pi_3 \text{(the (update rt sip (dsn1, dsk1, flag1, hops1, nhip1, pre1) sip)) = unk} \vee \text{Suc 0} \leq \pi_3 \text{(the (dhops (rt ξ) dip)} = \text{unk) \vee \text{Suc 0} \leq \pi_2 \text{(the (nhop (rt ξ) dip))})\)" \\
unfolding update_def by (cases "dsk1 = unk") (clarsimp split: option.split)+ \\
} note fromsip [simp] = this

{ fix sip rt dsn1 dsn2 dsk1 dsk2 flag1 flag2 hops1 hops2 nhip1 nhip2 pre1 pre2 \\
assume allkd: "\(\forall \text{ip} \in \text{dom(dests). ip} \in \text{kD(rt)} \land \text{sqn rt ip} \leq \text{the (dests ip)})\) \\
and **: "\(\forall \text{ip} \in \text{kD(rt). \pi_3 (the (rt dip))} = \text{unk} \vee \text{Suc 0} \leq \text{sqn rt dip} \leq \text{the (dests ip)})\)" \\
have "\(\forall \text{ip} \in \text{kD(rt). \pi_3 (the (update rt sip (dsn1, dsk1, flag1, hops1, nhip1, pre1) dip)} = \text{unk} \land \text{Suc 0} \leq \text{sqn (update rt sip (dsn2, dsk2, flag2, hops2, nhip2, pre2)) dip} \leq \text{the (dests ip)})\)" \\
(is "\(\forall \text{ip} \in \text{kD(rt). \pi_3 (the (rt dip)} = \text{unk} \land \text{Suc 0} \leq \text{sqn (invalidate rt dests) dip} \leq \text{the (dests ip)})\)"
proof \\
fix dip \\
assume "\(\text{dip} \in \text{kD(rt)}\)" \\
thus "\(\text{?prop dip}\)" \\
proof (cases "\(\text{dip} = \text{sip}\)"
\assume "\(\text{dip} = \text{sip}\)" \\
with ** show \(\text{?thesis}\) \\
by simp \\
next \\
assume "\(\text{dip} \neq \text{sip}\)" \\
with \(\text{dip} \in \text{kD(rt)}\) allkd show \(\text{?thesis}\) \\
by simp \\
qed \\
qed
} note solve_update [simp] = this

{ fix dip rt dests \\
assume *: "\(\forall \text{ip} \in \text{dom(dests). ip} \in \text{kD(rt)} \land \text{sqn rt ip} \leq \text{the (dests ip)}\)" \\
and **: "\(\forall \text{ip} \in \text{kD(rt). \pi_3 (the (rt ip)} = \text{unk} \land \text{Suc 0} \leq \text{sqn rt ip} \leq \text{the (dests ip)})\)" \\
have "\(\forall \text{ip} \in \text{kD(rt). \pi_3 (the (rt dip)} = \text{unk} \land \text{Suc 0} \leq \text{sqn (invalidate rt dests) dip} \leq \text{the (dests ip)})\)"
proof \\
fix dip \\
assume "\(\text{dip} \in \text{kD(rt)}\)"
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 \( \text{Suc 0} \leq \text{sqn rt dip} \)
  have \( \text{Suc 0} \leq \text{sqn (invalidate rt dests) dip} \)
  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 \( \text{Suc 0} \leq \text{sqn rt dip} \) \( \text{dip} \in \text{kD(rt)} \) [THEN \kD_Some] show \( \text{?thesis} \)
  unfolding \invalidate_def \sqn_def
  by auto
next
  assume \( \text{dip} /\in \text{dom(dests)} \)
  with \( \text{Suc 0} \leq \text{sqn rt dip} \) \( \text{dip} \in \text{kD(rt)} \) [THEN \kD_Some]
  show \( \text{?thesis} \)
  unfolding \invalidate_def \sqn_def
  by auto
qed
thus \( \text{?thesis} \) by (rule disjI2)
qed

proposition 7.13
lemma \rreq_rrep_sn_any_step_invariant:
  "paodv i \|=\_ (recvmsg rreq_rrep_sn \rightarrow) onll \Gamma_{AODV} (\lambda(_, \_, A). \text{anycast rreq_rrep_sn a})"
proof -
  have \sqnf_kno: "paodv i \|= onl \Gamma_{AODV} (\lambda(\xi, 1).
    \text{(l} \in \{\text{PRreq-:16..PRreq-:18}\} \rightarrow \text{sqn (rt } \xi \) (dip \xi) = \text{kno})"
  proof
    inv_cterms
    fix \l \xi \l' pp p'
    assume \( \text{dip} \in \text{reachable (paodv i) TT} \)
and \[ \lambda \xi. \xi'(\text{rt} := \text{update (rt } \xi (\text{oip } \xi (\text{osn } \xi, \text{kno}, \text{val}, \text{Suc (hops } \xi), \text{sip } \xi, \{\})) (\text{oip } \xi)) (\text{osn } \xi, \text{kno}, \text{val}, \text{Suc (hops } \xi), \text{sip } \xi, \{\})) (\text{oip } \xi)) \]

show \[ \text{osn } \xi' (\text{rt} := \text{update (rt } \xi (\text{oip } \xi (\text{osn } \xi, \text{kno}, \text{val}, \text{Suc (hops } \xi), \text{sip } \xi, \{})) (\text{oip } \xi)) \lor (\text{osn } \xi (\text{rt} := \text{update (rt } \xi (\text{oip } \xi (\text{osn } \xi, \text{kno}, \text{val}, \text{Suc (hops } \xi), \text{sip } \xi, \{})) (\text{oip } \xi)) \leq \text{Suc (hops } \xi))) \land \text{the (flag (update (rt } \xi (\text{oip } \xi (\text{osn } \xi, \text{kno}, \text{val}, \text{Suc (hops } \xi), \text{sip } \xi, \{})) (\text{oip } \xi)))}

unfolding update_def by (clarsimp split: option.split)

(metis linorder_neqE_nat not_less)

qed

have rrep_prrep: "paodv i |= \lambda (\xi, l). (l \in \{\text{PAodv-:15, PPkt-:7, PRreq-:9, PRreq-:21, PRrep-:10, PRerr-:1} \land (l \in \text{Suc (hops } \xi) \land \text{the (flag (update (rt } \xi (\text{oip } \xi (\text{osn } \xi, \text{kno}, \text{val}, \text{Suc (hops } \xi), \text{sip } \xi, \{})) (\text{oip } \xi)))}

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 |= \lambda (\xi, l). anycast (rerr_invalid (rt } \xi)) a"

proof -
have dests_inv: "paodv i |= \lambda (\xi, l). (l \in \text{Suc (hops } \xi) \land \text{the (flag (update (rt } \xi (\text{oip } \xi (\text{osn } \xi, \text{kno}, \text{val}, \text{Suc (hops } \xi), \text{sip } \xi, \{})) (\text{oip } \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 \((\text{rip}, \text{rsn})\) is assigned to the variable \text{dests} of type \text{ip} \to \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 \((\text{rip}, \text{rsn})\) for each destination \text{rip}: guaranteed by type.

lemma dests_vD_inc_sqn:
"paodv i |=
onl $\Gamma_{AODV} (\lambda(\xi, 1). \ (1 \in \{PAodv=15, PPkt=-7, PReq=-9, PReq=21, PReq=10\}) \\
\quad \rightarrow (\forall ip \in \text{dom}(dests) \ . \ ip \in vD(rt) \land \text{the}(dests \ ip) = \text{inc}(sqn (rt \ \xi) ip))) \\
\quad \land \ (1 = \text{PRerr}=1 \\
\quad \rightarrow (\forall ip \in \text{dom}(dests) \ . \ ip \in VD(rt) \land \text{the}(dests \ ip) > \text{sqn}(rt \ \xi) ip)))$

by inv_terms (clarsimp split: split_if_asm option.split_asm+)

Proposition 7.27

**Lemma route_tables_fresher:**

"paodv \ i \ =_{A} (recvmsg \ rreq_rrep_sn \ \rightarrow) \ onll \ \Gamma_{AODV} (\lambda((\xi, \ _), \ \_, \ (\ξ', \ _)). \\
\quad \forall dip \in kD(rt \ \xi). \ rt \ \xi \sqsubseteq dip \ rt \ ξ')"

**Proof:** (inv_terms inv_add:

\begin{align*}
onl \text{onl_invariant_sterms}[OF \ \text{aodv_wf \ dests_vD_inc_sqn \ [THEN \ \text{invariant_restrict_inD]}] \\
onl \text{onl_invariant_sterms}[OF \ \text{aodv_wf \ hop_count_positive \ [THEN \ \text{invariant_restrict_inD]}] \\
onl \text{onl_invariant_sterms}[OF \ \text{aodv_wf \ osn_rreq} \\
onl \text{onl_invariant_sterms}[OF \ \text{aodv_wf \ dsn_rrep} \\
onl \text{onl_invariant_sterms}[OF \ \text{aodv_wf \ addpreRT_welldefined \ [THEN \ \text{invariant_restrict_inD}}]
\end{align*}

fix $\xi \ pp \ p'$

assume "$(\xi, \ pp) \in \text{reachable}(paodv \ i) \ (recvmsg \ rreq_rrep_sn)"

and "(PReq=2)$\{\xi(1 = \text{update}(rt \ \xi) (ip \ \xi)) (osn \ \xi, \ kno, \ val, \ Suc \ (hops \ \xi), \ sip \ \xi, \ \{\})\]

$p' \in \text{sterms} \ \Gamma_{AODV} \ pp"

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

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

show "$\forall ip \in kD (rt \ \xi). \ rt \ \xi \sqsubseteq ip \ \text{update} (rt \ \xi) (ip \ \xi)) (osn \ \xi, \ kno, \ val, \ Suc \ (hops \ \xi), \ sip \ \xi, \ \{\})"

proof

\begin{align*}
\text{fix} \ ip \\
\text{assume} "\text{ip} \in kD (rt \ \xi)" \\
\text{moreover with} * \ \text{have} "1 \leq \text{the}(dhops (rt \ \xi) ip)" \ \text{by simp}
\end{align*}

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 ip \ \text{update} (rt \ \xi) (ip \ \xi)) (osn \ \xi, \ kno, \ val, \ Suc \ (hops \ \xi), \ sip \ \xi, \ \{\})"

by (rule rt_fresher_update)

qed

next

\begin{align*}
\text{fix} \ ip \\
\text{assume} "\text{ip} \in kD (rt \ \xi)" \\
\text{moreover with} * \ \text{have} "1 \leq \text{the}(dhops (rt \ \xi) ip)" \ \text{by simp}
\end{align*}

moreover from Suc 0 \leq \text{dsn} \ \xi

have "update_arg_wf (dsn \ \xi, \ kno, \ val, \ Suc \ (hops \ \xi), \ sip \ \xi, \ \{\})" ..

ultimately show "rt \ \xi \sqsubseteq ip \ \text{update} (rt \ \xi) (ip \ \xi)) (dsn \ \xi, \ kno, \ val, \ Suc \ (hops \ \xi), \ sip \ \xi, \ \{\})"

by (rule rt_fresher_update)

qed

end

1.7 The quality increases predicate

theory A_Quality_Increases
imports A_Aodv_Predicates A_Fresher
begin

definition quality_increases :: "\text{state} \Rightarrow \text{state} \Rightarrow \text{bool}"
where "quality_increases \ ξ \ ξ' \equiv (\forall dip \in kD(rt \ \xi). \ dip \in kD(rt \ ξ') \land \rt \ \xi \sqsubseteq dip \ rt \ ξ') \\
\land (\forall dip. \ \text{sqn} (rt \ \xi) dip \leq \text{sqn}(rt \ ξ') dip)"

lemma quality_increasesI [intro!]:
assumes \( \forall \text{ dip. } \text{ dip } \in kD(rt \xi) \implies \text{ dip } \in kD(rt \xi') \) and \( \forall \text{ dip. } [ \text{ dip } \in kD(rt \xi); \text{ dip } \in kD(rt \xi')] \implies rt \xi \sqsubseteq_{\text{ dip}} rt \xi' \) and \( \forall \text{ dip. } \text{ sqn (rt \xi) dip } \leq \text{ sqn (rt \xi') dip} \)

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 "\( \forall \text{ dip. } \text{ dip } \in kD(rt \xi) \)"
  and "\( [ \text{ dip } \in kD(rt \xi); \text{ dip } \in kD(rt \xi')] \implies \text{ R dip } \xi \xi' \)"
  using assms unfolding quality_increases_def by clarsimp

lemma quality_increases_rt_fresherD [dest]:
  fixes ip
  assumes "quality_increases \( \xi \xi' \)"
  and "ip \in kD(rt \xi)"
  shows "rt \xi \sqsubseteq_{ip} rt \xi'"
  using assms by auto

lemma quality_increases_sqnE [elim]:
  fixes dip
  assumes "quality_increases \( \xi \xi' \)"
  and "sqn (rt \xi) dip \leq sqn (rt \xi') dip \implies \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 \( \sigma \sigma' \) dip
  assumes "rt (\sigma i) \sqsubseteq_{\text{ dip}} rt (\sigma' \text{ nhip})" and qinc: "quality_increases (\sigma \text{ nhip}) (\sigma' \text{ nhip})" and "\( \forall \text{ dip. } \text{ dip } \in kD(rt (\sigma \text{ nhip})) \)"
  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 \( \forall \text{ dip. } \text{ dip } \in kD(rt (\sigma \text{ nhip})) \)
  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 "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 "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 "quality_increases \( \xi \xi' \)"

137
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_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 ≤ snq (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 $\langle \text{ip} \in kD(r (\sigma \text{ sip})) \rangle$ and $\langle \text{quality_increases (} \sigma \text{ sip) \ (} \sigma' \text{ sip)} \rangle$

have "... $\leq$ nsqn (rt (} \sigma' \text{ sip)) ip" ..

finally have "sn $<$ nsqn (rt (} \sigma' \text{ sip)) ip" .

thus ?thesis by simp

next

assume "sn = nsqn (rt (} \sigma' \text{ sip)) ip"

with $\langle \text{ip} \in kD(r (\sigma \text{ sip})) \rangle$ and $\langle \text{quality_increases (} \sigma \text{ sip) \ (} \sigma' \text{ sip)} \rangle$

have "sn $<$ nsqn (rt (} \sigma' \text{ sip)) ip

$\lor$ (sn $=$ nsqn (rt (} \sigma' \text{ sip)) ip

$\land$ the (dhops (rt (} \sigma' \text{ sip)) ip) $\leq$ the (dhops (rt (} \sigma \text{ sip)) ip))" ..

hence "sn $<$ nsqn (rt (} \sigma' \text{ sip)) ip

$\lor$ (nsqn (rt (} \sigma' \text{ sip)) ip = sn $\land$ (the (dhops (rt (} \sigma' \text{ sip)) ip) $\leq$ hops

$\lor$ the (flag (rt (} \sigma' \text{ sip)) ip) = inv))"

proof

assume "sn $<$ nsqn (rt (} \sigma' \text{ sip)) ip" thus ?thesis ..

next

assume "sn = nsqn (rt (} \sigma' \text{ sip)) ip

$\land$ the (dhops (rt (} \sigma' \text{ sip)) ip) $\geq$ the (dhops (rt (} \sigma \text{ sip)) ip)

hence "sn = nsqn (rt (} \sigma' \text{ sip)) ip"

and "the (dhops (rt (} \sigma' \text{ sip)) ip) $\leq$ the (dhops (rt (} \sigma \text{ sip)) ip)" by auto

from * and $\langle \text{sn} = \text{nsqn (} \text{rt (} \sigma' \text{ sip)) ip} \rangle$ have "the (dhops (rt (} \sigma \text{ sip)) ip) $\leq$ hops

$\lor$ the (flag (rt (} \sigma \text{ sip)) ip) = inv"

by simp

thus ?thesis

proof

assume "the (dhops (rt (} \sigma \text{ sip)) ip) $\leq$ hops"

with "the (dhops (rt (} \sigma' \text{ sip)) ip) $\leq$ the (dhops (rt (} \sigma \text{ sip)) ip)"

have "the (dhops (rt (} \sigma' \text{ sip)) ip) $\leq$ hops" by simp

with $\langle \text{sn} = \text{nsqn (} \text{rt (} \sigma' \text{ sip)) ip} \rangle$ show ?thesis by simp

next

assume "the (flag (rt (} \sigma \text{ sip)) ip) = inv"

with $\langle \text{ip} \in kD(r (\sigma \text{ sip})) \rangle$ have "nsqn (rt (} \sigma \text{ sip)) ip = sqn (rt (} \sigma \text{ sip)) ip - 1" ..

with $\langle \text{sn} \geq 1 \rangle$ and $\langle \text{sn} = \text{nsqn (} \text{rt (} \sigma \text{ sip)) ip} \rangle$

have "sqn (rt (} \sigma \text{ sip)) ip > 1" by simp

from $\langle \text{ip} \in kD(r (\sigma \text{ sip})) \rangle$ show ?thesis

proof (rule vD_or_iD)

assume "ip$\in$D(rt (} \sigma \text{ sip))"

hence "the (flag (rt (} \sigma \text{ sip)) ip) = inv" ..

with $\langle \text{sn} = \text{nsqn (} \text{rt (} \sigma' \text{ sip)) ip} \rangle$ show ?thesis by simp

next

assume "ip$\in$vD(rt (} \sigma \text{ sip))"

hence "nsqn (rt (} \sigma' \text{ sip)) ip = sqn (rt (} \sigma' \text{ sip)) ip" ..

with $\langle \text{sqn (} \text{rt (} \sigma \text{ sip)) ip \leq sqn (rt (} \sigma \text{ sip)) ip} \rangle$

have "nsqn (rt (} \sigma' \text{ sip)) ip $\geq$ sqn (rt (} \sigma \text{ sip)) ip" by simp

with $\langle \text{sqn (} \text{rt (} \sigma \text{ sip)) ip > 1} \rangle$

have "nsqn (rt (} \sigma' \text{ sip)) ip > sqn (rt (} \sigma \text{ sip)) ip - 1" by simp

with $\langle \text{sqn (} \text{rt (} \sigma \text{ sip)) ip = sqn (rt (} \sigma \text{ sip)) ip - 1} \rangle$

have "nsqn (rt (} \sigma' \text{ sip)) ip > nsqn (rt (} \sigma \text{ sip)) ip" by simp

with $\langle \text{sn} = \text{nsqn (} \text{rt (} \sigma' \text{ sip)) ip} \rangle$ have "nsqn (rt (} \sigma' \text{ sip)) ip > sn" by simp

thus ?thesis ..

qed

qed

thus ?thesis by (metis (mono_tags) le_cases not_le)

qed

with $\langle \text{ip} \in kD (} \text{rt (} \sigma \text{ sip)) $ show "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 hops dip dsn dsk oip osn sip) =
    (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 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. 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 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 "oip ≠ oip"
    with assms(1) show "oip ∈ kD(?rt)" by simp
  next
    assume "oip ≠ 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 ⟨oip ≠ oip⟩ have "the (dhops (?rt oip) ≤ hops"
      by simp
      thus ?thesis ..
    next
      assume "oip∈vD(?rt)"
      moreover from assms(1) and ⟨oip ≠ 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 "oip ≠ 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 ⟨oip ≠ 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
    qed 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 "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 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"
\begin{verbatim}
∧ (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

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)}"

end
\end{verbatim}
abbreviation \texttt{opaodv} :: "\texttt{ip} \Rightarrow ((\texttt{ip} \Rightarrow \texttt{state}) \times (\texttt{state}, \texttt{msg}, \texttt{pseqp}, \texttt{pseqp label}) \texttt{seqp}, \texttt{msg} \texttt{seq\_action}) \texttt{automaton}\"

where
\texttt{"opaodv i \equiv \{ init = \sigma_{AODV} ', trans = oseq\_sos \Gamma_{AODV} i \\}"}

lemma \texttt{initiali\_aodv [intro!, simp]}: "\texttt{initiali i (init (opaodv i)) (init (paodv i))}"
unfolding \texttt{\sigma_{AODV}'\_def} \texttt{\sigma_{AODV}\_def} by \texttt{rule simp\_all}

lemma \texttt{oaodv\_control\_within [simp]}: "\texttt{control\_within \Gamma_{AODV} (init (opaodv i))}"
unfolding \texttt{\sigma_{AODV}'\_def} by (\texttt{rule control\_withinI}) (\texttt{auto simp del: \Gamma_{AODV\_simps}})

lemma \texttt{\sigma_{AODV}'\_labels [simp]}: \texttt{"(\sigma, p) \in \sigma_{AODV}' \Rightarrow labels \Gamma_{AODV} p = \{PAodv\:-0\}"}
unfolding \texttt{\sigma_{AODV}'\_def} by \texttt{simp}

lemma \texttt{oaodv\_init\_kD\_empty [simp]}:
\texttt{"(\sigma, p) \in \sigma_{AODV}' \Rightarrow kD (rt (\sigma i)) = {}"}
unfolding \texttt{\sigma_{AODV}'\_def} \texttt{kD\_def} by \texttt{simp}

lemma \texttt{oaodv\_init\_vD\_empty [simp]}:
\texttt{"(\sigma, p) \in \sigma_{AODV}' \Rightarrow vD (rt (\sigma i)) = {}"}
unfolding \texttt{\sigma_{AODV}'\_def} \texttt{vD\_def} by \texttt{simp}

lemma \texttt{oaodv\_trans}: \texttt{"trans (opaodv i) = oseq\_sos \Gamma_{AODV} i"}
by \texttt{simp}

declare
\texttt{oseq\_invariant\_ctermsI [OF aodv\_wf oaodv\_control\_within aodv\_simple\_labels oaodv\_trans, cterms\_intros]}
\texttt{oseq\_step\_invariant\_ctermsI [OF aodv\_wf oaodv\_control\_within aodv\_simple\_labels oaodv\_trans, cterms\_intros]}

end

1.9 Global invariant proofs over sequential processes

theory \texttt{A\_Global\_Invariants}
imports \texttt{A\_Seq\_Invariants} \texttt{A\_Aodv\_Predicates} \texttt{A\_Fresher} \texttt{A\_Quality\_Increases}
\texttt{"../../../AWN/OAWN\_Convert"} \texttt{A\_OAodv}
begin

lemma \texttt{other\_quality\_increases [elim]}:
assumes \texttt{"other quality\_increases I \sigma \sigma'"}
shows \texttt{"\forall j. quality\_increases (\sigma j) (\sigma' j)"}
using \texttt{assms} \texttt{by (rule, clarsimp)} (\texttt{metis quality\_increases\_refl})

lemma \texttt{weaken\_otherwith [elim]}:
fixes \texttt{z}
assumes \texttt{*: "otherwith P I (orecvmsg Q) \sigma \sigma' a"}
and \texttt{weakenP: "\\land \sigma m. P \sigma m \Rightarrow P' \sigma m"}
and \texttt{weakenQ: "\\land \sigma m. Q \sigma m \Rightarrow Q' \sigma m"}
shows \texttt{"otherwith P' I (orecvmsg Q') \sigma \sigma' a"}
proof
fix \texttt{j}
assume \texttt{"j \notin I"}
with \texttt{*} have \texttt{"P (\sigma j) (\sigma' j)"} by \texttt{auto}
thus \texttt{"P' (\sigma j) (\sigma' j)"} by (\texttt{rule weakenP})
next
from \texttt{*} have \texttt{"orecvmsg Q \sigma a"} by \texttt{auto}
thus \texttt{"orecvmsg Q' \sigma a"}
by \texttt{rule (erule weakenQ)}
qed
lemma orreceived_msg_inv:
assumes other: "⋀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, i). 1 ∈ {PAodv{:i} → 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 "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

(Equivalent to) Proposition 7.27

lemma local_quality_increases:
"paodv i |=_A (recvmsg 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) (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 (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 oquality_increases:
"opaodv i |=_A (otherwith quality_increases {i} (orecvmsg (λ_. rreq_rrep_sn)),
other quality_increases {i} →)"
lemma rreq_rrep_nsqn_fresh_any_step_invariant:

  "opaodv i \models_A (act (recvmsg rreq_rrep_sn), other A {i} \rightarrow)
  onll Γ_AODV (λ(σ, _, a, _). anycast (msg_fresh σ) a)"

proof (rule onll_ostep_invariantI, simp)

  fix σ p a σ' p' l'

  assume or: "((σ, p) ∈ orreachable (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: "l' ∈ labels Γ_AODV p'"

  from this(1-3) have "orecvmsg (Λ.. rreq_rrep_sn) σ a" by (auto dest!: oreachable_weakenE)

  have "anycast (rreq_rrep_fresh (rt (opaodv i))) a" by (metis anycast_trans)

  qed

lemma rreq_rrep_nsqn_fresh_any_step_invariant:

  "opaodv i \models_A (act (recvmsg rreq_rrep_sn), other A {i} \rightarrow)
  onll Γ_AODV (λ(σ, _, a, _). anycast (msg_fresh σ) a)"

proof (rule onll_ostep_invariantI, simp del: act_simps)

  fix σ p a σ' p' l'

  assume or: "((σ, p) ∈ orreachable (opaodv i) (act (recvmsg rreq_rrep_sn))) (other A {i})))"

  and 11: "l ∈ labels Γ_AODV p"

  and tr: "((σ, p), a, (σ', p')) ∈ oseqp_sos Γ_AODV i"

  and recv: "act (recvmsg rreq_rrep_sn) ∈ ocosq_sos Γ_AODV σ σ'

  obtain l l' where "l∈labels Γ_AODV p" and "l'∈labels Γ_AODV p'"

  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 \models_A (act (recvmsg rreq_rrep_sn), other A {i} \rightarrow)
  onll Γ_AODV (sequil i (λ(ξ, _, a, _). anycast (rreq_rrep_fresh (rt ξ)) a))"

  by (rule onll_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 (sequil 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 -

have "opaodv i \models_A (act (recvmsg rreq_rrep_sn), other A {i} \rightarrow)
  onll Γ_AODV (sequil i (λ(ξ, _, a, _). anycast (rerr_invalid (rt ξ)) a))"

  by (rule onll_ostep_invariant_weakenE [OF

  open_seq_step_invariant [OF rerr_invalid_any_step_invariant initiali_aodv, simplified seqll_onll_swap]) auto

  hence "onll Γ_AODV (sequil 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 $\Gamma_{AODV}$ (seqll i ($\lambda(_, a, _). \text{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 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 ($\lambda m. \text{not}_\text{Pkt} m \rightarrow \text{msg_sender} m = \ i$) a"
proof -
have "opaodv i $\models_A$ (act (recvmsg rreq_rrep_sn), other A (i) $\rightarrow$
  onll $\Gamma_{AODV}$ (seqll i ($\lambda((\xi, _), a, _). \text{anycast} \ (\lambda m. \text{not}_\text{Pkt} m \rightarrow \text{msg_sender} m = \ i) a)$))"
  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 $\Gamma_{AODV}$ ($\lambda((\sigma, _), a, _). \text{anycast} \ (\lambda m. \text{not}_\text{Pkt} m \rightarrow \text{msg_sender} m = \ i) a)$ ((\sigma, p), a, (\sigma', p'))"
by auto
qed

lemma oreceived_rreq_rrep_nsqn_fresh_inv:
"opaodv i $\models_A$ (otherwith quality_increases \{i\} (orecvmsg msg_fresh),
  other quality_increases \{i\} $\rightarrow$
  onl $\Gamma_{AODV}$ ($\lambda(\sigma, 1). \ 1 \in \{PAodv:-4\} \rightarrow \text{msg}_\text{fresh} \ \sigma \ (\text{msg} (\sigma \ i))$)"
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 ($\sigma (i := \sigma \ i (msg := m)) \ m$"
proof (cases \ $m$)
  fix dests \ sip
  assume "$m = \text{Rerr} \ \text{dests} \ \text{sip}$"
  with $\langle \text{msg}_\text{fresh} \ \sigma \ m \rangle$ show thesis by auto
qed auto

lemma oquality_increases_nsqn_fresh:
"opaodv i $\models_A$ (otherwith quality_increases \{i\} (orecvmsg msg_fresh),
  other quality_increases \{i\} $\rightarrow$
  onll $\Gamma_{AODV}$ ($\lambda((\sigma, _), _, (\sigma', _)). \ \forall j. \text{quality_increases} \ (\sigma \ j) \ (\sigma' \ j)$)"
by (rule ostep_invariant_weakenE[OF oquality_increases]) auto

lemma oosn_rreq:
"opaodv i $\models_A$ (otherwith quality_increases \{i\} (orecvmsg msg_fresh),
  other quality_increases \{i\} $\rightarrow$
  onll $\Gamma_{AODV}$ (seqll i ($\lambda((\xi, 1), 1 \in \{PAodv:-4, PAodv:-5\} \cup \{PRreq:-n \in \text{In. True}\} \rightarrow \ 1 \leq \ \text{osn} \ \xi)$)"
by (rule oinvariant_weakenE[OF open_seq_invariant [OF osn_rreq initiali_aodv]])
(auto simp: seqll_onll_swap)
lemma rreq_sip:

"opadov i |= (otherwith quality_increases {i} (orecvmsg msg_fresh),
other quality_increases {i} i) →

onl AODV ((σ, i).

(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 equality_increases_nsqn_fresh

adov_wf oadov_trans]

onloinvariant_sterms [OF adov_wf oreceived_rreq_rrep_nsqn_fresh_inv

onl invariant_sterms [OF adov_wf oosn_rreq]

simp add: seqlsimp

simp del: Oue Nat_def, rule impl)

fix σ σ' p l

assume "(σ, p) ∈ oreachable (opadov i) ?S ?U"

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(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 (opadov i) ?S ?U 1 ∈ labels Γ AODV p) and hyp

have "1 ≤ osn (σ' i)"

by (auto dest!: onloinvariant_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

qed (auto elim!: quality_increases_rreq_rrep_props')

lemma odso_rrep:

"opadov i |= (otherwith quality_increases {i} (orecvmsg msg_fresh),
other quality_increases {i} i) →

148
lemma rrep_sip:
"opaodv i \models (otherwith quality_increases {i} (orecvmsg msg_fresh),
other quality_increases {i} \rightarrow)
onl \Gamma_{AODV} (λ(σ, i).
(1 \in \{PAodv=:6, PAodv=:? \cup \{PRrep=:n|n. True \} \rightarrow 1 ≤ dsn (ξ))")

by (rule oinvariant_weakenE [OF open_seq_invariant [OF dsn_rrep_initiali_aodv])
(auto simp: seql_onl_swap)

proof (inv_terms 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]
onl oinvariant_sterms [OF aodv_wf odsn_rrep]
simp del: One_nat_def, rule impl)
fix σ σ' p l
assume "("(σ, p) \in ooreachable (opaodv i) ?S ?U"
and "1 \in labels \Gamma_{AODV} p"
and pre:
auto simp: seql_onl_swap)
(dip (σ' i) \in kD (rt (σ' (sip (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' (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: "(1=PAodv=:6 ∨ l=PAodv=:? ∨ l=PRrep=:0 \rightarrow l=PRrep=:1) \land (σ' i) \neq dip (σ' i)"
(is "?labels ∧ (σ' i) \neq dip (σ' i)"
from this(4) have "σ' i = σ i" ..
with hyp have hyp': "?labels ∧ (σ' i) \neq dip (σ' i)" by simp
show "dip (σ' i) \land kD (rt (σ' (sip (σ' i))))
∧ nsqn (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' (sip (σ' i)))) (dip (σ' i)))
∨ the (flag (rt (σ' (sip (σ' i)))) (dip (σ' i))) = inv)"
proof (cases "dip (σ' i) = i")
assume "dip (σ' i) \neq i"
from "other quality_increases {i} σ σ'"
have "quality_increases (σ (sip (σ i))) (σ' (sip (σ' i)))"
by (rule otherE) (clarsimp simp: drip (σ' i) \neq i).
moreover from "((σ, p) \in ooreachable (opaodv i) ?S ?U \land 1 \in labels \Gamma_{AODV} p)" and hyp
have "1 \leq dsn (σ' i)"
by (auto dest!: oinvariant_weakenD [OF odsn_rrep]
simp add: seqlsimp (σ' i = σ i))
moreover from "dip (σ' i) \neq i" hyp' and pre
have "dip (σ' i) \in kD (rt (σ' (sip (σ' i))))
∧ nsqn (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' (sip (σ' i)))) (dip (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 "dip (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 \parallel (otherwith quality_increases \{i\} (orecvmsg msg_fresh),
other quality_increases \{i\} \rightarrow
\Gamma_{AODV} (\lambda (\sigma, 1).
1 \in \{PAodv-:8, PAodv-:9, PRerr-:0, PRerr-:1\}
\rightarrow (\forall ripc \in \text{dom}(dests (i)). ripc \in kD(rt (\sigma (sip (i)))) \land
the (dests (i) ripc) - 1 \leq nsqn (rt (\sigma (sip (i)))) ripc))"
(is "_ \parallel (?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 rip \in \text{dom} dests. rip \in kD (rt (\sigma sip))
\land the (dests rip) - 1 \leq nsqn (rt (\sigma sip)) rip"
from this(3) have "rip \in \text{dom} dests" by auto
with * and \langle dests rip = Some rsn \rangle have "rip \in kD(rt (\sigma sip))"
and "rsn - 1 \leq nsqn (rt (\sigma sip)) rip"
by (auto dest!: bspec)
from qinc have "quality_increases (\sigma sip) (\sigma' sip)"
.. have "rip \in kD(rt (\sigma sip)) \land rsn - 1 \leq nsqn (rt (\sigma sip)) rip"
proof from \langle rip \in kD(rt (\sigma sip)) \rangle and \langle quality_increases (\sigma sip) (\sigma' sip) \rangle
show "rip \in kD(rt (\sigma sip))" ..
next from \langle rip \in kD(rt (\sigma sip)) \rangle and \langle quality_increases (\sigma sip) (\sigma' sip) \rangle
have "nsqn (rt (\sigma sip)) rip \leq nsqn (rt (\sigma sip)) rip" ..
with \langle rsn - 1 \leq nsqn (rt (\sigma sip)) rip \rangle show "rsn - 1 \leq nsqn (rt (\sigma 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 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 conjI)
- (clarsimp simp del: One_nat_def split: split_if_asm option.split_asm, erule(2) partial)+
qed

lemma prerr_guard: "paodv i \parallel
\Gamma_{AODV} (\lambda (\xi, 1). (1 = PRerr-:1
\rightarrow (\forall ripc \in \text{dom}(dests \xi). ripc \in vD(rt \xi)
\land the (nhop (rt \xi) ripc) = sip \xi
\land sqn (rt \xi) ripc < the (dests \xi ripc))))"
by (inv_cterms) (clarsimp split: option.split_asm split_if_asm)

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':
proof -

{ fix nhop and σ σ' :: "ip ⇒ state"
  assume pre: ∀ dip ∈ kD(rt (σ i)). nhop dip ≠ dip →
    dip ∈ kD(rt (σ (nhop dip))) ∧ nhop dip ≠ dip →
    nhop dip = the (nhop (rt (σ i))) dip
    "quality_increases (rt (σ i)) ∧ nhop dip ≠ dip →
    nhop dip = the (nhop (rt (σ i))) dip" 
  nhop dip ≠ dip →
    dip ∈ kD(rt (σ (nhop dip))) ∧ nhop dip ≠ dip →
    nhop dip = the (nhop (rt (σ i))) dip
  "quality_increases (rt (σ i)) ∧ nhop dip ≠ dip →
    nhop dip = the (nhop (rt (σ i))) dip"
  nhop dip ≠ dip →
    dip ∈ kD(rt (σ (nhop dip))) ∧ nhop dip ≠ dip →
    nhop dip = the (nhop (rt (σ i))) dip
  "quality_increases (rt (σ i)) ∧ nhop dip ≠ dip →
    nhop dip = the (nhop (rt (σ i))) 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)))" 
  ∧ 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: ∀ ripc ∈ 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

151
with ndest and dip\in dom (dests (\sigma i)) have "dip \in kD (rt (\sigma (sip (\sigma i))))"

"dsn - 1 \leq nsqn (rt (\sigma (sip (\sigma i)))) dip"

by auto

moreover from issip and dip\in dom (dests (\sigma i)) have "nhop dip = sip (\sigma i)" ..

ultimately have "dip \in kD (rt (\sigma (nhop dip)))"

and "dsn - 1 \leq nsqn (rt (\sigma (nhop dip))) dip" by auto

with qinc show "dsn - 1 \leq nsqn (rt (\sigma' (nhop dip))) dip"

by simp (metis kD_nsqn_quality_increases_trans)

qed

ultimately show ?thesis by simp

next

assume "dip \notin dom (dests (\sigma i))"

with \in kD(rt (\sigma i))

have "nsqn (invalidate (rt (\sigma i)) (dests (\sigma i))) dip = nsqn (rt (\sigma i)) dip"

by (rule nsqn_invalidate_other)

with nsqn (rt (\sigma i)) dip \leq nsqn (rt (\sigma' (nhop dip))) dip show ?thesis by simp

qed

with \in kD(rt (\sigma' (nhop dip)))

show "dip \in kD (rt (\sigma' (nhop dip)))" 

\land nsqn (invalidate (rt (\sigma i)) (dests (\sigma i))) dip \leq nsqn (rt (\sigma' (nhop dip))) dip" ..

qed

\}

note basic_prerr = this

\{

fix \sigma \sigma' :: "\textit{ip} \Rightarrow \textit{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)) (0, unk, val, Suc 0, sip (\sigma i), {\})) dip) \neq dip 

\rightarrow dip\in kD(rt (\sigma' (the (nhop (rt (\sigma i)) dip))) 

(0, unk, val, Suc 0, sip (\sigma i), {\})) 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)) 

(0, unk, val, Suc 0, sip (\sigma i), {\})) 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 \textit{derule(1) basic, auto}

thus "?P dip" by (cases "dip = sip (\sigma i)"

auto)

qed

\}

note nhop_update_sip = this

\{

fix \sigma \sigma' oip sip osn hops

assume 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"

and qinc: "\forall j. quality_increases (\sigma j) (\sigma' j)"

and \*: "sip \neq oip \rightarrow oip\in kD(rt (\sigma sip)) 

\land osn \leq nsqn (rt (\sigma sip)) oip 

\land (nsqn (rt (\sigma sip)) oip = osn 

\rightarrow the (dhops (rt (\sigma sip)) oip) \leq hops

\lor the (flag (rt (\sigma sip)) oip) = inv)"

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 \( (\text{nhop (update (rt (\sigma i)) oip (osn, kno, val, Suc hops, sip, \{\}) oip) \neq oip} \)
\rightarrow oip \in kD((\text{nhop (update (rt (\sigma i)) oip (osn, kno, val, Suc hops, sip, \{\}) oip)})
\wedge nsqn (update (rt (\sigma i)) oip (osn, kno, val, Suc hops, sip, \{\}) oip)
\leq nsqn (rt (\sigma' (the (nhop (update (rt (\sigma i)) oip (osn, kno, val, Suc hops, sip, \{\}) oip))) oip)\)

(is "\text{?nhop_not_oip \rightarrow ?oip_in_kD \wedge ?nsqn_le_nsqn}\)

proof (rule, split update_rt_split_asm)
assume "rt (\sigma i) = 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 \wedge ?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 \wedge ?nsqn_le_nsqn" ..
qed

} note update1 = this

{ fix \sigma \sigma' oip sip osn hops
assume 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))))
\wedge nsqn (rt (\sigma i)) dip \leq nsqn (rt (\sigma (the (nhop (rt (\sigma i)) dip)))) dip"
and qinc: "\forall j. quality_increases (\sigma j) (\sigma' j)"
and *: "sip \neq oip \rightarrow oip \in kD(rt (\sigma sip))
\wedge (nsqn (rt (\sigma sip)) oip = osn
\rightarrow the (dhops (rt (\sigma sip)) oip) \leq hops
\vee the (flag (rt (\sigma sip)) oip) = inv)"

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))))
\wedge nsqn (rt (\sigma i)) dip \leq nsqn (rt (\sigma' (the (nhop (rt (\sigma i)) dip)))) dip"
by (rule basic)

have "\forall dip \in kD(rt (\sigma i)).
the (nhop (update (rt (\sigma i)) oip (osn, kno, val, Suc hops, sip, \{\})) dip) \neq dip
\rightarrow dip \in kD(rt (\sigma' (the (nhop (update (rt (\sigma i)) oip (osn, kno, val, Suc hops, sip, \{\}) dip))))
\wedge nsqn (update (rt (\sigma i)) oip (osn, kno, val, Suc hops, sip, \{\}) dip)
\leq nsqn (rt (\sigma' (the (nhop (update (rt (\sigma i)) oip (osn, kno, val, Suc hops, sip, \{\}) dip)))) dip)"
(is "\forall dip \in kD(rt (\sigma i)). dip \rightarrow ?dip_in_kD dip \wedge ?nsqn_le_nsqn dip")

proof (intro ballI impI, split update_rt_split_asm)
fix dip
assume "dip \in kD(rt (\sigma i))"
and "the (nhop (rt (\sigma i)) dip) \neq dip"
and "rt (\sigma i) = update (rt (\sigma i)) oip (osn, kno, val, Suc hops, sip, \{\})"
with pre' show "?dip_in_kD dip \wedge ?nsqn_le_nsqn dip" by simp

next
fix dip
assume "dip \in kD(rt (\sigma i))"
and notdip: "the (nhop (update (rt (\sigma i)) oip (osn, kno, val, Suc hops, sip, \{\})) dip) \neq dip"
and rtnot: "rt (\sigma i) \neq update (rt (\sigma i)) oip (osn, kno, val, Suc hops, sip, \{\})"
show "?dip_in_kD dip \wedge ?nsqn_le_nsqn dip"

proof (cases "dip = oip")
assume "dip \neq oip"
with pre' (dip \in kD(rt (\sigma i))) notdip
show ?thesis by clarsimp

next
assume "dip = oip"
with \( \text{rtnot qinc } \langle \text{dip} \in kD(\text{rt } (\sigma i)) \rangle \) \( \text{notdip} * \)

have "?dip_in_kD dip"

by \( \text{simp (metis kD_quality_increases)} \)

moreover from \( \langle \text{dip} = \text{oip} \rangle \) \( \text{rtnot qinc } \langle \text{dip} \in kD(\text{rt } (\sigma i)) \rangle \) \( \text{notdip} * \)

have "?nsqn_le_nsqn dip" by \( \text{simp (metis kD_nsqn_quality_increases_trans)} \)

ultimately show ?thesis ..

qed

qed

}

note update2 = this

have "opaodv i \models (?S, ?U \rightarrow) \text{onl } \Gamma_{AODV} (\lambda(\sigma, _). \forall \text{dip} \in kD(\text{rt } (\sigma i)). \text{the (nhop (\text{rt } (\sigma i)) dip) }\neq \text{dip} \rightarrow \text{dip} \in kD(\text{rt } (\sigma (\text{the (nhop (\text{rt } (\sigma i)) dip)))))) \land \text{nsqn (rt } (\sigma i)) dip \leq \text{nsqn (rt } (\sigma (\text{the (nhop (rt } (\sigma i)) dip)))) dip)"


(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 ooreachable_fresh_okD_unk_or_atleast_one:

fixes dip

assumes "((\sigma, p) \in \text{oreachable (opaodv i)}) ((\text{otherwith (op=} i) (orecvmsg (\lambda m. msg_fresh \sigma m \land msg_zhops m))))

\langle \text{other quality_increases } (i) \rangle"

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 \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 ooreachable_fresh_ozero_seq_unk_hops_one:

fixes dip

assumes "((\sigma, p) \in \text{oreachable (opaodv i)}) ((\text{otherwith (op=} i) (orecvmsg (\lambda m. msg_fresh \sigma m \land msg_zhops m))))

\langle \text{other quality_increases } (i) \rangle"


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∈labels Γ_{AODV} p" by (metis aodv_ex_label)
  with assms(1) have "∀ dip∈kd (rt (σ i)). ?P dip"
    by (drule invariant_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_nhops_quality_increases':
  shows "opaodv i |= (otherwith (op=) {i})
    (orecvmsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
    other quality_increases (i) \rightarrow
    onl Γ_{AODV} (λ(σ, _). ∀ dip. let nhop = the (nhop (rt (σ i)) dip)
      in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhop))
      ∧ nhop ≠ dip
      → (rt (σ i)) △ dip (rt (σ (nhop (rt (σ i)) dip)))"
(is "-_ |= (?S i, _ → ) _")
proof -
  have weaken:
    "∀ P I Q R P. p |= (otherwith quality_increases I (orecvmsg Q),
     other quality_increases I \rightarrow) P
     \rightarrow p |= (otherwith (op=) I (orecvmsg (λσ m. Q σ m ∧ R σ m)),
     other quality_increases I \rightarrow) 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)))"
    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 ov: "∀ dip. dip ∈ vD(update (rt (σ i))) ∪ dip = 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))) ∪ dip = 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)))"
  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 ov 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 σ' a 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 ov: "∀ σ i 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 ov 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: "∃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, {}))
∧ 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 = 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. 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)
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)" ..
next
from \(oip \in kD(rt (\sigma \cdot ipp))\)
  show "\((\lambda a. \text{if } a = oip \text{ then } \text{Some (rt (\sigma \cdot i) oip) else rt (\sigma \cdot i)}) a) = rt (\sigma \cdot i)\)"
  by (auto dest!: kD_Some)
qed

with change have False ..
thus ?thesis ..
qed

next
assume "oip \in iD(rt (\sigma \cdot ipp))"
with \(\text{the (flag (rt (\sigma \cdot ipp) oip) = val})\) and a5 have "\(\text{sip = i}\)"
  by (metis f.distinct(1) iD_flag_is_inv)
from \(oip \in kD(rt (\sigma \cdot ipp))\) have "\(\text{the (flag (rt (\sigma \cdot ipp) oip) = inv})\)"
  unfolding update_def
  by (clarsimp split: option.split_asm split_if_asm)
with \(\text{osn \leq nsqn (rt (\sigma \cdot ipp)) oip})\) have "\(\text{osn < nsqn (rt (\sigma \cdot ipp)) oip})\"
  by (simp)
thus ?thesis ..
qed

thus ?thesis proof
assume osnlt: "\(\text{osn < nsqn (rt (\sigma \cdot ipp)) oip})\"
from \(dip \in kD(rt (\sigma \cdot i))\) and \(\text{dip = oip})\) have "\(\text{dip \in kD (?rt1})\)" by simp
moreover from a3 have "\(\text{dip \in kD(?rt2 dip})\)" by simp
moreover have "\(\text{nsqn ?rt1 dip < nsqn (?rt2 dip) dip})\"
  proof
    have "\(\text{nsqn ?rt1 oip = osn})"
      by (simp add: \(\text{dip = oip})\) nsqn_update_changed_kno_val [OF change [THEN not_sym]]
    also have "\(\ldots < nsqn (rt (\sigma \cdot ipp)) oip})\) using osnlt .
    also have "\(\ldots = nsqn (?rt2 oip) oip})" by (simp add: change)
    finally show ?thesis
      using \(\text{dip = oip})\) by simp
  qed
ultimately show ?thesis
  by (rule rt_strictly_fresher_ltI)
next
assume osneq: "\(\text{osn = nsqn (rt (\sigma \cdot ipp)) oip \& the (dhops (rt (\sigma \cdot ipp)) oip) \leq hops})\"
have "\(\text{oip \in kD(?rt1})\)" by simp
moreover from a3 \(\text{dip = oip})\) have "\(\text{oip \in kD(?rt2 oip})\)" by simp
moreover have "\(\text{nsqn ?rt1 oip = nsqn (?rt2 oip) oip})\"
  proof
    from osneq have "\(\text{osn = nsqn (rt (\sigma \cdot ipp)) oip}) ..
    also have "\(\text{osn = nsqn ?rt1 oip})"
      by (simp add: \(\text{dip = oip})\) nsqn_update_changed_kno_val [OF change [THEN not_sym]]
    also have "\(\text{nsqn (rt (\sigma \cdot ipp)) oip = nsqn (?rt2 oip) oip})"
      by (simp add: change)
    finally show ?thesis
  qed
moreover have "\(\pi_5(the (?rt2 oip oip)) < \pi_5(the (?rt1 oip))\)"
  proof
    from osneq have "\(\text{the (dhops (rt (\sigma \cdot ipp)) oip) \leq hops}) ..
    moreover from \(\text{oip \in vD (rt (\sigma \cdot ipp}))\) have "\(\text{oip \in kD(rt (\sigma \cdot ipp})\)" by auto
    ultimately have "\(\pi_5(the (rt (\sigma \cdot ipp) oip) \leq hops})"
      by (auto simp add: proj5_eq_dhops)
    also from change after have "\(hops < \pi_5(the (rt (\sigma \cdot i) oip))\)"
      by (simp add: proj5_eq_dhops) (metis dhops_update_changed lessI)
    finally have "\(\pi_5(the (rt (\sigma \cdot ipp) oip)) < \pi_5(the (rt (\sigma \cdot i) oip))\)" .
    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

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_nsqns_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 sqn_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 ∀ 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 "... = 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_nsqns_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⟩
ultimately 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} (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. 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

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
  rt_graph :: "(ip ⇒ state) ⇒ ip ⇒ ip rel"
where
  "rt_graph σ = (λdip.
  { (ip, ip') | ip ip' dsn dsk hops pre.
  ip ≠ dip ∧ rt σ 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 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]:
"\(\forall ip \ ip' \ \sigma. (ip, ip') \in rt_graph \ \sigma \ dip \implies ip \neq dip\)"
unfolding rt_graph_def by auto

lemma rt_graph_not_dip_trans [dest]:
"\(\forall ip \ ip' \ \sigma. (ip, ip') \in (rt_graph \ \sigma \ dip)^+ \implies ip \neq dip\)"
by (erule converse_tranclE) auto

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

lemma rt_graph_nhip_is_nhop [dest]:
"\(\forall ip \ ip' \ \sigma. (ip, ip') \in rt_graph \ \sigma \ dip \implies ip' = \text{the (nhop (rt (\sigma \ ip))) dip}\)"
unfolding rt_graph_def by auto

theorem inv_to_loop_freedom:
assumes
"\(\forall i \ dip. \text{let nhip} = \text{the (nhop (rt (\sigma \ i))) dip} \in \text{vD (rt (\sigma \ i)) \cap vD (rt (\sigma \ nhip)) \land nhip \neq dip} \implies rt (\sigma \ i) \sqsubseteq dip \text{ rt (\sigma \ nhip)}\)"
shows "\(\forall dip. \text{irrefl ((rt_graph \ \sigma \ dip)^+)}\)"
using assms proof (intro allI)
fix \sigma :: "ip \Rightarrow state" and dip
assume inv: "\(\forall ip \ dip. \text{let nhip} = \text{the (nhop (rt (\sigma \ ip))) dip} \in \text{vD (rt (\sigma \ ip)) \cap vD (rt (\sigma \ nhip)) \land nhip \neq dip} \implies rt (\sigma \ ip) \sqsubseteq dip \text{ rt (\sigma \ nhip)}\)"
{ 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 \text{ 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 "\(\sigma \ ip) \in \text{vD (rt (\sigma \ nhip))}\)" and "\(\sigma \ nhip \neq dip\)"
by auto
from ⟨dip \in vD(rt (\sigma \ nhip))⟩ and ⟨dip \in vD(rt (\sigma \ nhip'))⟩
have "\(\sigma \ ip) \in \text{vD (rt (\sigma \ nhip))\)" ..
with ⟨nhip \in \text{the (nhop (rt (\sigma \ nhip))) dip⟩}
and ⟨nhip \neq dip⟩
and inv
show "rt (\sigma \ ip) \sqsubseteq dip \text{ rt (\sigma \ 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: \[\[ \text{dip} \in vD(rt (\sigma \ nhip)); \text{nhip} \neq dip \] \implies rt (\sigma \ ip) \sqsubseteq dip \text{ rt (\sigma \ nhip)}\]
and "\(\text{dip} \in vD(rt (\sigma \ nhip'))\)" and "\(\text{nhip'} \neq dip\)"
from ⟨(nhip, nhip') \in rt_graph \ \sigma \ dip⟩ have 1: "\(\text{dip} \in vD(rt (\sigma \ nhip))\)"
and 2: "\(\text{nhip} \neq dip\)" and "\(\text{nhip'} \in \text{the (nhop (rt (\sigma \ nhip))) dip}\)"
by auto
from 1 2 have "rt (\sigma \ ip) \sqsubseteq dip \text{ rt (\sigma \ nhip)}" by (rule IH)
also have "rt (\sigma \ nhip) \sqsubseteq dip \text{ rt (\sigma \ nhip')}"
proof -
from dip \in vD(rt (\sigma \ nhip)) and dip \in vD(rt (\sigma \ nhip'))
have "\(\text{dip} \in vD(rt (\sigma \ nhip)) \cap vD(rt (\sigma \ nhip'))\)" ..
with ⟨nhip' \neq dip⟩
and ⟨nhip' \in \text{the (nhop (rt (\sigma \ 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"
by auto
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 "call 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} (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 "call 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} (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
qed auto

lemma par_rreq_rrep_sn_quality_increases:
"opaodv i ⟨⟨i qmsg | A (λσ_. orerecvmsg (λ_. rreq_rrep_sn) σ), other (λ_. _. True) {i} →) globala (λ(σ, _, σ'). quality_increases (σ i) (σ' i))"
proof -
have "opaodv i |=A (λσ_. orecvmsg (λ_. rreq_rrep_sn) σ), other (λ_. True) {i} \rightarrow
  globala (λ(σ, a, σ'). quality_increases (σ i) (σ' i))"
by (rule ostep_invariant_weakenE [OF olocal_quality_increases])
(auto dest!: onllD seqllD elim!: aodv_ex_label)
hence "opaodv i (i1 qmsg |=A (λσ_. orecvmsg (λ_. rreq_rrep_sn) σ), other (λ_. True) {i} \rightarrow
  globala (λ(σ, a, σ'). quality_increases (σ i) (σ' i))"
by (rule lift_step_into_qmsg_statelessasm) simp_all
thus thesis by rule auto
qed

lemma par_rreq_rrep_nsqn_fresh_any_step:
shows "opaodv i (i1 qmsg |=A (λσ_. orecvmsg (λ_. rreq_rrep_sn) σ),
  other (λ_. True) {i} \rightarrow
  globala (λ(σ, a, σ'). anycast (msg_fresh σ) a))"
proof -
have "opaodv i (i1 qmsg |=A (λσ_. (orecvmsg (λ_. rreq_rrep_sn))) σ),
  other (λ_. True) {i} \rightarrow
  globala (λ(σ, a, σ'). anycast (msg_fresh σ) a))"
proof (rule ostep_invariant_weakenE [OF rreq_rrep_nsqn_fresh_any_step_invariant])
fix t
assume "onll Γ_AODV (λ((σ, _), a, _). anycast (msg_fresh σ) a) t"
thus "globala (λ(σ, a, σ'). anycast (msg_fresh σ) a) t"
by (cases t) (clarsimp dest!: onllD, metis aodv_ex_label)
qed auto

hence "opaodv i (i1 qmsg |=A (λσ_. (orecvmsg (λ_. rreq_rrep_sn))) σ),
  other (λ_. True) {i} \rightarrow
  globala (λ(σ, a, σ'). anycast (msg_fresh σ) a))"
by (rule lift_step_into_qmsg_statelessasm) simp_all
thus thesis by rule auto
qed

lemma par_anycast_msg_zhops:
shows "opaodv i (i1 qmsg |=A (λσ_. orecvmsg (λ_. rreq_rrep_sn) σ),
  other (λ_. True) {i} \rightarrow
  globala (λ(_, a, _). anycast msg_zhops a))"
proof -
from anycast_msg_zhops_initial_aodv oadv_trans aodv_trans
have "opaodv i |=A (act TT, other (λ_. True) {i} \rightarrow
  seqll i (onll Γ_AODV (λ(_, a, _). anycast msg_zhops a)))"
by (rule open_seq_step_invariant)
hence "opaodv i |=A (λσ_. orecvmsg (λ_. rreq_rrep_sn) σ), other (λ_. True) {i} \rightarrow
  globala (λ(_, a, _). anycast msg_zhops a))"
proof (rule ostep_invariant_weakenE)
fix t :: "((nat ⇒ state) × (state, msg, pseqp, pseqp label) seqp), msg seq_action) transition"
assume "seqll i (onll Γ_AODV (λ(_, a, _). anycast msg_zhops a)) t"
thus "globala (λ(_, a, _). anycast msg_zhops a) t"
by (cases t) (clarsimp dest!: seqllD onllD, metis aodv_ex_label)
qed auto

hence "opaodv i (i1 qmsg |=A (λσ_. orecvmsg (λ_. rreq_rrep_sn) σ),
  other (λ_. True) {i} \rightarrow
  globala (λ(_, a, _). anycast msg_zhops a))"
by (rule lift_step_into_qmsg_statelessasm) simp_all
thus thesis by rule auto
qed

1.11.1 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 (i1 qmsg : R) |=
  (otherwith (op=) {i})"
(oarrivemsg ($\lambda \sigma \ m. \ \text{msg\_fresh} \ \sigma \ m \land \text{msg\_zhops} \ m$)), 
other quality\_increases (i) 
$\rightarrow$ global ($\lambda \sigma. \ \forall \ \text{dip. let nhip = the (nhop (rt (\sigma \ i)) dip) \ in \ dip \in VD (rt (\sigma \ i)) \cap VD (rt (\sigma \ nhip)) \land nhip \neq dip$ 
$\rightarrow (rt (\sigma \ i)) \sqsubseteq \text{dip} (rt (\sigma \ nhip))")$
by (rule node\_lift [OF par\_nhop\_quality\_increases]) auto

lemma node\_quality\_increases:
"( i : opaodv i \langle i \ qmsg : R \rangle \o \models A ($\lambda \sigma. \ \text{oarrivemsg} \ (\lambda. \ \text{rreq\_rrep\_sn}) \ \sigma$,
other ($\lambda. \ \text{True}) \ (i) \rightarrow"
globala ($\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 : opaodv i \langle i \ qmsg : R \rangle \o \models A ($\lambda \sigma. \ \text{oarrivemsg} \ (\lambda. \ \text{rreq\_rrep\_sn}) \ \sigma$,
other ($\lambda. \ \text{True}) \ (i) \rightarrow"
globala ($\lambda (\sigma, a, \sigma'). \ \text{castmsg} (\text{msg\_fresh} \ \sigma) a$)"
by (rule node\_lift\_anycast\_statelessassm [OF par\_rreq\_rrep\_nsqn\_fresh\_any\_step])

lemma node\_anycast\_msg\_zhops:
shows "( i : opaodv i \langle i \ qmsg : R \rangle \o \models A ($\lambda \sigma. \ \text{oarrivemsg} \ (\lambda. \ \text{rreq\_rrep\_sn}) \ \sigma$,
other ($\lambda. \ \text{True}) \ (i) \rightarrow"
globala ($\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 : opaodv i \langle i \ qmsg : R \rangle \o \models A ($\lambda \sigma. \ \text{oarrivemsg} \ (\lambda. \ \text{rreq\_rrep\_sn}) \ \sigma$,
other ($\lambda. \ \text{True}) \ (i) \rightarrow"
globala ($\lambda (\sigma, a, \sigma'). \ \sigma \neq \tau \rightarrow \sigma' \ i = \sigma \ i$)"
proof (rule ostep\_invariantI, simp (no_asm), rule impI)
fix $\sigma \ \zeta \ a \ \sigma' \ \zeta'$
assume or: "((\sigma, \ \zeta) \in \text{oreachable} \langle i : \text{opaodv} i \langle i \ qmsg : R_i \rangle, \ (\lambda \sigma. \ \text{oarrivemsg} \ (\lambda. \ \text{rreq\_rrep\_sn}) \ \sigma$,
other ($\lambda. \ \text{True}) \ (i) \rightarrow"
globala ($\lambda (\sigma, a, \sigma'). \ \sigma \neq \tau \rightarrow \sigma' \ i = \sigma \ i$)"
and tr: "((\sigma, \ \zeta), a, (\sigma', \ \zeta')) \in \text{trans} \langle i : \text{opaodv} i \langle i \ qmsg : R_i \rangle, a, (\sigma', \ \zeta') \in \text{onode\_sos} (\text{oparp\_sos} i \ (\text{trans} (\text{opaodv} i)) (\text{trans} qmsg))" 
from or obtain $p \ R \ \text{where} "(\zeta = \text{NodeS} \ i \ p \ R)"$ 
by - (drule node\_net\_state, metis)
with tr have "((\sigma, \ \text{NodeS} i \ p \ R), a, (\sigma', \ \zeta') \in \text{onode\_sos} (\text{oparp\_sos} i (\text{trans} (\text{opaodv} i)) (\text{trans} qmsg))" 
by simp 
thus "$\sigma' \ i = \sigma \ i$" using (a $\neq \tau_n$) 
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 \ m. \ \text{msg\_fresh} \ \sigma \ m \land \text{P} \ \sigma \ m) \ \sigma \ m"
shows "\text{oarrivemsg} \ (\lambda. \ \text{rreq\_rrep\_sn}) \ \sigma \ m"
using assms by (cases m) auto

lemma opnet\_nhop\_quality\_increases:
shows "\text{opnet} (\lambda i. \ \text{opaodv} i \langle i \ qmsg \rangle \ p \models (
\text{otherwith \ (op=) \ (net\_tree\_ips \ p)} \ (\text{oarrivemsg} \ (\lambda \ m. \ \text{msg\_fresh} \ \sigma \ m \land \text{msg\_zhops} \ m)), \ 
other quality\_increases \ (\text{net\_tree\_ips} \ p) \rightarrow) \ 
global \ (\lambda \sigma. \ \forall \ i \in \text{net\_tree\_ips} \ p. \ \forall \ \text{dip. let nhip = the (nhop (rt (\sigma \ i)) dip) \ in \ dip \in VD (rt (\sigma \ i)) \cap VD (rt (\sigma \ nhip)) \land nhip \neq dip$ 
$\rightarrow (rt (\sigma \ i)) \sqsubseteq \text{dip} (rt (\sigma \ nhip))")")
proof (rule pnet\_lift [OF node\_nhop\_quality\_increases])
fix i R
have "\(i : \text{opadv} i \langle \lambda i. \text{qmsg} : R \rangle_o \vdash_A (\lambda_\sigma_\_ \_. \text{oarrivemsg} (\lambda_\_. \text{rreq_rrep_sn}) \sigma, \text{other} (\lambda_\_. \_. \text{True}) \{i\} \rightarrow) \text{globala} (\lambda(\sigma, a, \_). \text{castmsg} (\lambda m. \text{msg_fresh} \sigma m \land \text{msg_zhops} m) a)"

proof (rule ostep_invariantI, simp (no_asm))
fix \(s a \sigma' s'
assume or: "(\sigma, s) \in \text{oreachable} \langle \lambda i. \text{opadv} i \langle \lambda i. \text{qmsg} : R \rangle_o \rangle (\lambda_\sigma_\_ \_. \text{oarrivemsg} (\lambda_\_. \text{rreq_rrep_sn}) \sigma) \text{other} (\lambda_\_. \_. \text{True}) \{i\})"
and tr: "((\sigma, s), a, (\sigma', s')) \in \text{trans} \langle \lambda i. \text{opadv} i \langle \lambda i. \text{qmsg} : R \rangle_o \rangle (\lambda_\sigma_\_ \_. \text{oarrivemsg} (\lambda_\_. \text{rreq_rrep_sn}) \sigma a)"
and am: "\text{oarrivemsg} (\lambda_\_. \text{rreq_rrep_sn}) \sigma a"

from or tr am have "\text{castmsg} (\text{msg_fresh} \sigma) a"
by (auto)
moreover from or tr am have "\text{castmsg} (\text{msg_zhops} m) a"
by (auto)
ultimately show "\text{castmsg} (\lambda m. \text{msg_fresh} \sigma m \land \text{msg_zhops} m) a"
by (case_tac a) auto

qed thus "\(i : \text{opadv} i \langle \lambda i. \text{qmsg} : R \rangle_o \vdash_A (\lambda_\sigma_\_ \_. \text{oarrivemsg} (\lambda_\sigma m. \text{msg_fresh} \sigma m \land \text{msg_zhops} m) \sigma, \text{other} \text{quality_increases} \{i\} \rightarrow) \text{globala} (\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 "\(i : \text{opadv} i \langle \lambda i. \text{qmsg} : R \rangle_o \vdash_A (\lambda_\sigma_\_ \_. \text{oarrivemsg} (\lambda_\sigma m. \text{msg_fresh} \sigma m \land \text{msg_zhops} m) \sigma, \text{other} \text{quality_increases} \{i\} \rightarrow) \text{globala} (\lambda(\sigma, a, \_). \text{castmsg} (\lambda m. \text{msg_fresh} \sigma m \land \text{msg_zhops} m) a)\)
by (rule ostep_invariant_weakenE [OF node_silent_change_only]) auto

next
fix \(i R\)
show "\(i : \text{opadv} i \langle \lambda i. \text{qmsg} : R \rangle_o \vdash_A (\lambda_\sigma_\_ \_. \text{oarrivemsg} (\lambda_\sigma m. \text{msg_fresh} \sigma m \land \text{msg_zhops} m) \sigma, \text{other} \text{quality_increases} \{i\} \rightarrow) \text{globala} (\lambda(\sigma, a, \_). \text{castmsg} (\lambda m. \text{msg_fresh} \sigma m \land \text{msg_zhops} m) a)\)
by (rule ostep_invariant_weakenE [OF node_quality_increases]) auto

qed simp_all

1.11.3 Lift to closed networks

lemma onet_nhop_quality_increases:
shows "\text{closed} (\text{opnet} (\lambda i. \text{opadv} i \langle \lambda i. \text{qmsg} : R \rangle) p) \vdash (\lambda_\_. \_. \text{True}, \text{other} \text{quality_increases} (\text{net_tree_ips} p) \rightarrow) \text{globala} (\lambda \sigma. \forall i \in \text{net_tree_ips} p. \forall \text{dip}. \text{let nhip = the (nhop (rt (\sigma i)) dip)} \text{in dip} \in \text{vd} (\text{rt (\sigma i)} \cap \text{vd} (\text{rt (\sigma nhip)} \land \text{nhip} \neq \text{dip} \rightarrow (\text{rt (\sigma i)} \in \text{dip} (\text{rt (\sigma nhip)}))) (\text{is "\_" \vdash (\_)} \Rightarrow \text{?inv})")
proof (rule inclosed_closed)
from onet_nhop_quality_increases
show "\text{opnet} (\lambda i. \text{opadv} i \langle \lambda i. \text{qmsg} : R \rangle) p \vdash (\text{otherwith (op=)} (\text{net_tree_ips} p) \text{inclosed}, \text{?U} \Rightarrow \text{?inv})"

proof (rule ostep_invariant_weakenE)
fix \(\sigma \sigma' : \text{"ip \Rightarrow state" and } a : \text{"msg node_action"}
assume "\text{otherwith (op=} (\text{net_tree_ips} p) \text{inclosed} \sigma \sigma' a"
thus "\text{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 otherwithE)
fix \(\sigma : \text{"ip \Rightarrow state" and } a : \text{"msg node_action"}
assume "\text{inclosed} \sigma a"
thus "\text{oarrivemsg} (\lambda m. \text{msg_fresh} \sigma m \land \text{msg_zhops} m) \sigma a"

proof (cases a)
fix \(i n i m s\)
assume "a = i ni:arrive(ms)"
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 :: ip
have "\{ (\sigma, \zeta). (\sigma i, \zeta) \in \sigma_{AODV} \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 "\{ (\sigma, \zeta). \sigma \in \text{init}(\text{paodv } i) \land (\sigma i, \zeta) = \text{id } s \land (\forall j. j \neq i \rightarrow \sigma j \in \text{fst } \text{id } \sigma_{AODV} j) \} \subseteq \text{init}(\text{opaodv } i)"
by simp
next
show "\forall j. \text{init}(\text{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 "(s, a, s') \in \text{trans}(\text{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}(\text{paodv } i)"
by (cases s, cases s') auto
from this(3) have "\{ (\sigma, q), a, (\sigma', q') \} \in \text{trans}(\text{opaodv } i)"
by simp (rule open_seqp_action [OF aodv_wf])
with \sigma = (\sigma i, q) and \sigma' = (\sigma' i, q'),
show "\{ (\sigma, \text{snd } \text{id } s'), a, (\sigma', \text{snd } \text{id } s') \} \in \text{trans}(\text{opaodv } i)"
by simp
qed
then interpret op: openproc paodv opaodv id.

have [simp]: "\forall i. (\text{SOME } x. x \in \text{fst } \text{id } i) \neq \{}" unfolding \sigma_{AODV}_def by simp
hence "\forall i. \text{openproc.initmissing paodv id } i = \text{initmissing } i"
unfolding op.initmissing_def op.someinit_def initmissing_def by (auto split: option.split)
thus "openproc.initmissing paodv id = \text{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: "\forall \sigma. \text{openproc.initmissing } (\lambda i. \text{paodv } i \langle \langle \text{qmsg} \rangle \rangle \langle \lambda (p, q). (\text{fst } \text{id } p), \text{snd } \text{id } p, q \rangle \rangle \sigma"
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 ⟨⟨ qmsg⟩⟩ (λ(p, q). (fst (id p), snd (id p), q))) = initmissing"
by (rule ext)
have "∀P σ. openproc. netglobal (λi. 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 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 ⟨⟨ 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 (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)))"
(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))))"
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

1.11.5 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 2

Variant B: Forwarding the Route Reply

Explanation [4, §10.2]: In AODV’s 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
import 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" ("π₂")
where \( \pi_2 \equiv \lambda (dsn, \_ , \_ , \_ , \_ ). dsn \)

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

definition proj4 :: "r ⇒ f" ("\(\pi_4\)"
where \( \pi_4 \equiv \lambda (dsn, dsk, flag, \_ , \_ , \_ ). flag \)

definition proj5 :: "r ⇒ nat" ("\(\pi_5\)"
where \( \pi_5 \equiv \lambda (dsn, dsk, flag, hops, \_ , \_ ). hops \)

definition proj6 :: "r ⇒ ip" ("\(\pi_6\)"
where \( \pi_6 \equiv \lambda (dsn, dsk, flag, hops, nhip, \_ ). nhip \)

definition proj7 :: "r ⇒ ip set" ("\(\pi_7\)"
where \( \pi_7 \equiv \lambda (dsn, dsk, flag, hops, nhip, pre). pre \)

lemma proj2[simp]: 
\( \pi_2 (dsn, dsk, flag, hops, nhip, pre) = dsn \)

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

lemma proj4_pred [intro]: "[ \[ P val; P inv \] \] = \Rightarrow 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

2.1.3 Routing Tables

Routing tables map ip addresses to route entries.

type_synonym rt = "ip ⇒ r"

syntax
"_Sigma_route" :: "rt ⇒ ip ⇒ r" ("σ_route'(_, _, ')")

translations
"σ_route(rt, dip)" ⇒ "rt dip"

definition sqn :: "rt ⇒ ip ⇒ sqn"
where "sqn rt dip ≡ case σ_route(rt, dip) of Some r ⇒ \(\pi_2\) (r) | None ⇒ 0"

definition sqnf :: "rt ⇒ ip ⇒ k"
where "sqnf rt dip ≡ case σ_route(rt, dip) of Some r ⇒ \(\pi_3\) (r) | None ⇒ unk"

abbreviation flag :: "rt ⇒ ip ⇒ f"
where "flag rt dip ≡ map_option \(\pi_4\) (σ_route(rt, dip))"

abbreviation dhops :: "rt ⇒ ip ⇒ nat"
where "dhops rt dip ≡ map_option \(\pi_5\) (σ_route(rt, dip))"

abbreviation nhop :: "rt ⇒ ip ⇒ ip"
where "nhop rt dip ≡ map_option \(\pi_6\) (σ_route(rt, dip))"
abbreviation `precs :: "rt ⇒ ip → ip set"
  where "precs rt dip ≡ map_option π7 (σroute(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 iD_flag_is_inv [elim, simp]:
  fixes ip rt
  assumes "ip \in iD(rt)"
  shows "the (flag rt ip) = inv"
proof -
  from \{ip \in iD(rt)\} 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 \{ip \in kD(rt)\} obtain dsn dsk f hops nhop pre
    where rtip: "rt ip = Some (dsn, dsk, f, hops, nhop, pre)"
    by (metis kD_Some)
  from \{ip / \in vD(rt)\} have "f \neq val"
    proof (rule contrapos_nn)
      assume "f = val"
      with rtip have "the (flag rt ip) = val" by simp
    qed
  with rtip have "the (flag rt ip) = inv" by simp
  with \{ip \in kD(rt)\} show "ip \in vD(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 \{ip \in kD(rt)\} 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: "\(\forall dip rt. dip \in kD(rt) \Rightarrow \pi_5(\text{the (rt dip)}) = \text{the (dhops rt dip)})"
  unfolding sqn_def by (drule kD_Some) clarsimp

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

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

lemma kD_sqnf_is_proj3 [simp]:
  "\(\forall ip rt. ip \in kD(rt) \Rightarrow \text{sqnf rt ip} = \pi_3(\text{the (rt ip)})\)"
  unfolding sqn_def by auto

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

lemma kD_update [simp]:
  "\(\forall rt nip v. kD (rtnip \mapsto v)) = \text{insert nip (kD rt)}\)"
  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' ∨ 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 "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) ∪ npre"
  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

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)) (σ_route(rt, dip))"
lemma snd_addpre [simp]:
"\(dsn \; dsn' \; v \; pre. \; (dsn, \; (addpre \; (dsn', \; v) \; pre)) = addpre \; (dsn, \; v) \; pre\)"
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]:
"\(\forall \; \text{rt} \; dip \; ip \; npre. \; \text{dip} \in \kD(\text{rt}) \implies \pi_5(\text{the} \; (\text{addpreRT} \; \text{rt} \; \text{dip} \; npre) \; \text{ip}) = \pi_5(\text{the} \; (\text{rt} \; \text{ip}))\)"
unfolding addpreRT_def by auto

lemma flag_addpreRT [simp]:
fixes rt pre ip dip
assumes "dip \in \kD rt"
shows "\(\text{flag} \; (\text{the} \; (\text{addpreRT} \; \text{rt} \; \text{dip} \; \text{npre}) \; \text{ip}) = \text{flag} \; \text{rt} \; \text{ip}\)"
unfolding addpreRT_def by auto

lemma kD_addpreRT [simp]:
fixes rt dip npre
assumes "dip \in \kD rt"
shows "\(\kD \; (\text{the} \; (\text{addpreRT} \; \text{rt} \; \text{dip} \; \text{npre}) \; \text{ip}) = \kD \; \text{rt}\)"
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 "\(\vD \; (\text{the} \; (\text{addpreRT} \; \text{rt} \; \text{dip} \; \text{npre}) \; \text{ip}) = \vD \; \text{rt}\)"
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 "\(\iD \; (\text{the} \; (\text{addpreRT} \; \text{rt} \; \text{dip} \; \text{npre}) \; \text{ip}) = \iD \; \text{rt}\)"
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 "\(\text{nhop} \; (\text{the} \; (\text{addpreRT} \; \text{rt} \; \text{dip} \; \text{npre}) \; \text{ip}) = \text{nhop} \; \text{rt} \; \text{ip}\)"
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 "\(\text{sqn} \; (\text{the} \; (\text{addpreRT} \; \text{rt} \; \text{dip} \; \text{npre}) \; \text{ip}) = \text{sqn} \; \text{rt} \; \text{ip}\)"
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)"
  "∀ 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]:
  "∀ n 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 \( \text{update_arg_wf} \) have \( \text{wf1: "flag = val"} \)
and \( \text{wf2: "(dsn = 0) = (dsk = unk)"} \)
and \( \text{wf3: "dsk = unk \( \rightarrow \) (hops = 1)"} \)

unfolding \( \text{update_arg_wf_def} \) by auto
have "\( \text{P (dsn, dsk, flag, hops, nhip, pre)} \)"
proof (cases dsk)
assume "dsk = unk"
moreover with \( \text{wf2} \) \( \text{wf3} \) have "dsn = 0" by auto
ultimately show \( \text{?thesis} \) using \( \langle \text{flag} = \text{val} \rangle \) by simp (rule c1)
next
assume "dsk = kno"
moreover with \( \text{wf2} \) have "dsn > 0" by simp
ultimately show \( \text{?thesis} \) using \( \langle \text{flag} = \text{val} \rangle \) by simp (rule c2)
qed
with \( * \) show "\( \text{P r} \)" by simp
qed

definition update :: "rt \Rightarrow ip \Rightarrow r \Rightarrow rt"
where "update rt ip r \equiv \text{case } \sigma_{\text{route}}(rt, ip) \text{ of }
None \Rightarrow rt (ip \mapsto r)
| \text{Some } s \Rightarrow
  \text{if } \pi_2(s) < \pi_2(r) \text{ then } rt (ip \mapsto \text{addpre } r (\pi_7(s)))
  \text{else if } \pi_2(s) = \pi_2(r) \land (\pi_5(s) > \pi_5(r) \lor \pi_4(s) = \text{inv})
  \text{then } rt (ip \mapsto \text{addpre } r (\pi_7(s)))
  \text{else if } \pi_3(r) = \text{unk}
  \text{then } rt (ip \mapsto (\pi_3(s), \text{snd } (\text{addpre } r (\pi_7(s)))))
  \text{else if } \pi_3(r) = \text{unk}
  \text{then } rt (ip \mapsto \text{addpre } s (\pi_7(r)))"

lemma update_simps [simp]:
fixes r s nrt nr nr' ns rt ip
defines "s \equiv \text{the } \sigma_{\text{route}}(rt, ip)"
and "nr \equiv \text{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 \text{addpre } s (\pi_7(r))"
shows
"[ \[ ip /\in kD(rt) \] ] \Rightarrow \text{update } rt ip r = rt (ip \mapsto r)"
"[ \[ ip \in kD(rt); \sqn rt ip < \pi_2(r) \] ] \Rightarrow \text{update } rt ip r = rt (ip \mapsto nr)"
"[ \[ ip \in kD(rt); \sqn rt ip = \pi_2(r); \text{flag rt ip = Some } inv \] ] \Rightarrow \text{update } rt ip r = rt (ip \mapsto nr)"
"[ \[ ip \in kD(rt); \pi_3(r) = \text{unk}; (\pi_2(r) = 0) = (\pi_3(r) = \text{unk}) \] ] \Rightarrow \text{update } rt ip r = rt (ip \mapsto nr')"
"[ \[ ip \in kD(rt); \sqn rt ip \geq \pi_2(r); \pi_3(r) = \text{kno}; \text{flag rt ip = Some } inv \] ] \Rightarrow \text{update } rt ip r = rt (ip \mapsto ns)"
proof -
assume "ip \notin kD(rt)"

hence "\( \sigma_{\text{route}}(rt, ip) = \text{None} \) ..
thus "update rt ip r = rt (ip \mapsto r)"

unfolding \text{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 \( \langle \sqn rt ip < \pi_2(r) \rangle \) show "update rt ip r = rt (ip \mapsto nr)"

unfolding \text{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 ⟨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 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: "[\[ \{ ip \notin kD(rt) \} =⇒ P (rt (ip ↦ r))\]"
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: "[\[ \{ ip ∈ kD(rt); sqn rt ip = π₃(r); the (flag rt ip) = inv \} =⇒ P (rt (ip ↦ addpre r (π₇(\{ the σroute(rt, ip) \}))))\]"
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 \notin kD(rt)"
with c1 show \?thesis
  by simp
next
assume "ip ∈ kD(rt)"
moreover then obtain \( dsn \ dsk \ fl \ hops \ nhip \ pre \)
where \( \text{rteq} \: : \text{rt ip} = \text{Some} \ (dsn, dsk, fl, hops, nhip, pre) \)
by \( \text{(metis kD_Some)} \)
moreover obtain \( dsn' \ dsk' \ fl' \ hops' \ nhip' \ pre' \)
where \( \text{req} \: : \text{r} = (dsn', dsk', fl', hops', nhip', pre') \)
by \( \text{(cases r)} \) \text{metis}
ultimately show \( \text{thesis} \)
using \( \langle \begin{array}{l}
(\pi_2(r) = 0) = (\pi_3(r) = \text{unk}) \\
\langle \text{ip} \in \text{kD}(rt) \rangle \\
\text{c2} \ [\text{OF } \langle \text{ip} \in \text{kD}(rt) \rangle ] \\
\text{c3} \ [\text{OF } \langle \text{ip} \in \text{kD}(rt) \rangle ] \\
\text{c4} \ [\text{OF } \langle \text{ip} \in \text{kD}(rt) \rangle ] \\
\text{c5} \ [\text{OF } \langle \text{ip} \in \text{kD}(rt) \rangle ] \\
\text{c6} \ [\text{OF } \langle \text{ip} \in \text{kD}(rt) \rangle ] \\
\end{array} \rangle \)
unfolding \text{update_def sqn_def} \) by \( \text{auto} \)
qed

lemma \text{update_cases_kD}: 
assumes \( (\pi_2(r) = 0) = (\pi_3(r) = \text{unk}) \)
and \( \text{ip} \in \text{kD}(rt) \)
and \( \text{c2}: \text{sqn rt ip} < \pi_2(r) \Rightarrow P (\text{rt (ip }\mapsto\text{addpre r (\pi_7(\sigma_{route}(rt, ip))))}) \)
and \( \text{c3}: \text{sqn rt ip} = \pi_2(r); \text{the (dhops rt ip)} > \pi_5(r) \Rightarrow P (\text{rt (ip }\mapsto\text{addpre r (\pi_7(\sigma_{route}(rt, ip))))}) \)
and \( \text{c4}: \text{sqn rt ip} = \pi_2(r); \text{the (flag rt ip)} = \text{inv} \Rightarrow P (\text{rt (ip }\mapsto\text{addpre r (\pi_7(\sigma_{route}(rt, ip))))}) \)
and \( \text{c5}: \pi_3(r) = \text{unk} \Rightarrow P (\text{rt (ip }\mapsto\text{addpre r (\pi_2(\sigma_{route}(rt, ip)), \pi_3(r), \pi_4(r), \pi_5(r), \pi_6(r), \pi_7(\text{addpre r (\pi_7(\sigma_{route}(rt, ip))))})}) \)
and \( \text{c6}: \text{sqn rt ip} \geq \pi_2(r); \pi_3(r) = \text{kno}; \text{sqn rt ip} = \pi_2(r) \Rightarrow (\text{dhops rt ip}) \leq \pi_5(r) \Rightarrow (\text{flag rt ip}) = \text{val} \Rightarrow P (\text{rt (ip }\mapsto\text{addpre (\sigma_{route}(rt, ip)))}) \)
shows \( (P (\text{update rt ip r})) \)
using \text{assms(1)} \) proof \( \text{(rule update_cases)} \)
assume \( \text{sqn rt ip} < \pi_2(r) \)
thus \( P (\text{rt (ip }\mapsto\text{addpre r (\pi_7(\sigma_{route}(rt, ip))))}) \) by \( \text{(rule c2)} \)
next
assume \( \text{sqn rt ip} = \pi_2(r) \)
and \( \text{the (dhops rt ip)} > \pi_5(r) \)
thus \( P (\text{rt (ip }\mapsto\text{addpre r (\pi_7(\sigma_{route}(rt, ip))))}) \) by \( \text{(rule c3)} \)
next
assume \( \text{sqn rt ip} = \pi_2(r) \)
and \( \text{the (flag rt ip)} = \text{inv} \)
thus \( P (\text{rt (ip }\mapsto\text{addpre r (\pi_7(\sigma_{route}(rt, ip))))}) \) by \( \text{(rule c4)} \)
next
assume \( \pi_3(r) = \text{unk} \)
thus \( P (\text{rt (ip }\mapsto\text{addpre r (\pi_7(\sigma_{route}(rt, ip))))}) \) by \( \text{(rule c5)} \)
next
assume \( \text{sqn rt ip} \geq \pi_2(r) \)
and \( \pi_3(r) = \text{kno} \)
and \( \text{sqn rt ip} = \pi_2(r) \Rightarrow (\text{dhops rt ip}) \leq \pi_5(r) \Rightarrow (\text{flag rt ip}) = \text{val} \)
thus \( P (\text{rt (ip }\mapsto\text{addpre r (\pi_7(\sigma_{route}(rt, ip))))}) \) by \( \text{(rule c6)} \)
qed \( \text{(simp add: \langle \text{ip} \in \text{kD}(rt) \rangle )} \)

lemma \text{in_kD_after_update [simp]}: 
fixes \text{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 \text{update_def}
by \( \text{(cases "rt nip") auto} \)

lemma \text{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] 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)) dip = rt dip"
  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 = 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 dan 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, kno, flg, hops, sip, {})
  ⇒ 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 dip ip dan hops npre. 1 ≤ dan ⇒ 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 dsk flg hops sip.
\[
\begin{align*}
rt \neq \text{update rt dip (dsn, dsk, flg, hops, sip, {})} \quad & \implies \pi_3(\text{the (update rt dip (dsn, dsk, flg, hops, sip, {})) dip}) = dsk \\
\text{unfolding update_def sqnf_def} \quad & \text{by (clarsimp split: option.splits split_if_asm) auto}
\end{align*}
\]

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

lemma flag_update [simp]: "\(\forall rt dip dsn flg hops sip. \)
\(rt \neq \text{update rt dip (dsn, kno, flg, hops, sip, {})} \quad \implies \text{the (flag (update rt dip (dsn, kno, flg, hops, sip, {})) dip}) = flg}\)
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 "\text{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]: "\(\forall rt dip dsn dsk flg hops sip. \)
\(rt \neq \text{update rt dip (dsn, dsk, flg, hops, sip, {})} \quad \implies \text{the (nhop (update rt dip (dsn, dsk, flg, hops, sip, {})) dip}) = sip}\)
\text{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]:
"\(\forall dip rt dip' dsn dsk hops nhip pre. \)
dip \in vD(update rt dip' (dsn, dsk, val, hops, nhip, pre)) \implies (dip \in vD(rt) \lor dip=dip')"
\text{unfolding update_def vD_def by (clarsimp split: option.split_asm split_if_asm) auto}

Invalidating route entries

definition invalidate :: "rt \Rightarrow (ip \Rightarrow sqn) \Rightarrow rt"
where "invalidate rt dests \equiv \lambda ip. case (rt ip, dests ip) of
(\text{None, _}) \Rightarrow \text{None
}\}
lemma proj3_invalidate [simp]:
"\(\pi_3((\text{invalidate } rt \text{ dests} \ dip)) = \pi_3(\text{rt dip})\)"
unfolding invalidate_def by (clarsimp split: option.split)

lemma proj5_invalidate [simp]:
"\(\pi_5((\text{invalidate } rt \text{ dests} \ dip)) = \pi_5(\text{rt dip})\)"
unfolding invalidate_def by (clarsimp split: option.split)

lemma proj6_invalidate [simp]:
"\(\pi_6((\text{invalidate } rt \text{ dests} \ dip)) = \pi_6(\text{rt dip})\)"
unfolding invalidate_def by (clarsimp split: option.split)

lemma proj7_invalidate [simp]:
"\(\pi_7((\text{invalidate } rt \text{ dests} \ dip)) = \pi_7(\text{rt dip})\)"
unfolding invalidate_def by (clarsimp split: option.split)

lemma invalidate_kD_inv [simp]:
"\(kD(\text{invalidate } rt \text{ dests}) = kD \text{rt}\)"
unfolding invalidate_def kD_def
by (simp split: option.split)

lemma invalidate_sqn:
fixes rt dip dests
assumes "\(\forall \text{rsn}. \text{dests dip = Some rsn} \rightarrow \text{sqn rt dip} \leq \text{rsn}\)"
shows "\(\text{sqn rt dip} \leq \text{sqn (invalidate rt dests) dip}\)"
proof (cases "\(\text{dip} \notin kD(\text{rt})\)"
    assume "\(\neg \text{dip} \notin kD(\text{rt})\)"
    hence "\(\text{dip} \in kD(\text{rt})\)" by simp
    then obtain dsn dsk flag hops nhip pre where "\(\text{rt dip = Some (dsn, dsk, flag, hops, nhip, pre)}\)"
    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 "\(\text{dests ipa = Some rsn}\)"
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]:
"\(\text{dhops (invalidate rt dests) dip} = \text{dhops rt dip}\)"
unfolding invalidate_def by (clarsimp split: option.split)

lemma sqnf_invalidate [simp]:
"\(\text{sqnf (invalidate (rt } \xi \text{) (dests } \xi \text{)) dip} = \text{sqnf (rt } \xi \text{) dip}\)"
unfolding sqnf_def invalidate_def by (clarsimp split: option.split)

lemma nhop_invalidate [simp]:
"\(\text{nhop (invalidate (rt } \xi \text{) (dests } \xi \text{)) dip} = \text{nhop (rt } \xi \text{) dip}\)"
unfolding invalidate_def by (clarsimp split: option.split)

lemma invalidate_other [simp]:
fixes rt dests dip
assumes "\(\text{dip} \notin \text{dom(dests)}\)"
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 ∈ 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 ∈ vD(invalidate rt dests) =⇒ dip ∈ vD(rt) ∧ dests dip = None"
unfolding invalidate_def vD_def
by (clarsimp split: option.split_asm)

lemma sqn_invalid_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 ⇒ 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 ∈ 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 ∈ 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)

2.1.5 Route Requests

Generate a fresh route request identifier.

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

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

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

definition \(\text{sigma\_p\_flag}\) :: "store \Rightarrow \text{ip} \Rightarrow p" ("\(\sigma\_p\_flag\)(\text{\_, \_'})")
where "\(\sigma\_p\_flag\text{store, dip} \equiv\) map_option fst\ (\text{store dip})"

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

definition \(\text{set\_RRF}\) :: "store \Rightarrow \text{ip} \Rightarrow sqn \Rightarrow store"
where "\(\text{set\_RRF}\ \text{store\ dests} \equiv\) \(\lambda\ dip.\ \text{if}\ \text{dests}\ dip = \text{None}\ \text{then}\ \text{store}\ dip\ \
else\ \text{map}\_\text{option}\ (\lambda(\_, q).\ (\text{req}, q)))\)\ (\text{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. \(\text{nhop}\) is partial, thus a ‘the’ is needed, similarly for \(\text{dhops}\) and \(\text{add\_pre\_RT}\).
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 (\(\text{ip} \Rightarrow \text{r option}\)) 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 \(\text{dests}\) mapping passed to \(\text{invalidate}\), and \text{store}.

end

2.2 AODV protocol messages

theory \text{B\_Aodv\_Message}
imports \text{B\_Fwdrreps}
begin

datatype \text{msg} =
\begin{array}{lll}
\text{Rreq} \text{nat rreqid ip sqn k ip sqn ip} & | \ \text{Rrep} \text{nat ip sqn ip ip} & | \ \text{Rerr} \text{"ip \Rightarrow sqn" ip} \\
| \ \text{Newpkt data ip} & | \ \text{Pkt data ip ip} \\
\end{array}

instantiation \text{msg} :: \text{msg}
begin
\text{definition newpkt_def [simp]: "newpkt \equiv \lambda(d, dip).\ Newpkt\ d\ dip"}
\text{definition eq_newpkt_def: "eq\_newpkt\ m \equiv case\ m\ of\ \text{Newpkt}\ d\ dip \Rightarrow \text{True} \ |\ \_ \Rightarrow \text{False}"

instance by intro_classes (simp add: eq_newpkt_def)
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:**

\[
\text{rreq} :: \text{nat} \times \text{rreqid} \times \text{ip} \times \text{sqn} \times \text{k} \times \text{ip} \times \text{sqn} \times \text{ip} \Rightarrow \text{msg}
\]

where \( \text{rreq} \equiv \lambda (\text{hops}, \text{rreqid}, \text{dip}, \text{dsn}, \text{dsk}, \text{oip}, \text{osn}, \text{sip}). \text{Rreq }\text{hops }\text{rreqid }\text{dip }\text{dsn }\text{dsk }\text{oip }\text{osn }\text{sip} \)

**Lemma:**

rreq simp [simp]:

\[\text{rreq}(\text{hops}, \text{rreqid}, \text{dip}, \text{dsn}, \text{dsk}, \text{oip}, \text{osn}, \text{sip}) = \text{Rreq }\text{hops }\text{rreqid }\text{dip }\text{dsn }\text{dsk }\text{oip }\text{osn }\text{sip} \]

unfolding rreq_def by simp

**Definition:**

\[
\text{rrep} :: \text{nat} \times \text{ip} \times \text{sqn} \times \text{ip} \times \text{ip} \Rightarrow \text{msg}
\]

where \( \text{rrep} \equiv \lambda (\text{hops}, \text{dip}, \text{dsn}, \text{oip}, \text{sip}). \text{Rrep }\text{hops }\text{dip }\text{dsn }\text{oip }\text{sip} \)

**Lemma:**

rrep simp [simp]:

\[\text{rrep}(\text{hops}, \text{dip}, \text{dsn}, \text{oip}, \text{sip}) = \text{Rrep }\text{hops }\text{dip }\text{dsn }\text{oip }\text{sip} \]

unfolding rrep_def by simp

**Definition:**

\[
\text{rerr} :: (\text{ip} \Rightarrow \text{sqn}) \times \text{ip} \Rightarrow \text{msg}
\]

where \( \text{rerr} \equiv \lambda (\text{dests}, \text{sip}). \text{Rerr }\text{dests }\text{sip} \)

**Lemma:**

rerr simp [simp]:

\[\text{rerr}(\text{dests}, \text{sip}) = \text{Rerr }\text{dests }\text{sip} \]

unfolding rerr_def by simp

**Lemma:**

not_eq_newpkt_rreq [simp]: \( \neg \text{eq_newpkt }\text{(Rreq }\text{hops }\text{rreqid }\text{dip }\text{dsn }\text{dsk }\text{oip }\text{osn }\text{sip}) \)

unfolding eq_newpkt_def by simp

**Lemma:**

not_eq_newpkt_rrep [simp]: \( \neg \text{eq_newpkt }\text{(Rrep }\text{hops }\text{dip }\text{dsn }\text{oip }\text{sip}) \)

unfolding eq_newpkt_def by simp

**Lemma:**

not_eq_newpkt_rerr [simp]: \( \neg \text{eq_newpkt }\text{(Rerr }\text{dests }\text{sip}) \)

unfolding eq_newpkt_def by simp

**Lemma:**

not_eq_newpkt_pkt [simp]: \( \neg \text{eq_newpkt }\text{(Pkt }\text{d }\text{dip }\text{sip}) \)

unfolding eq_newpkt_def by simp

**Definition:**

\[
\text{pkt} :: \text{data} \times \text{ip} \times \text{ip} \Rightarrow \text{msg}
\]

where \( \text{pkt} \equiv \lambda (\text{d}, \text{dip}, \text{sip}). \text{Pkt }\text{d }\text{dip }\text{sip} \)

**Lemma:**

pkt simp [simp]:

\[\text{pkt}(\text{d}, \text{dip}, \text{sip}) = \text{Pkt }\text{d }\text{dip }\text{sip} \]

unfolding pkt_def by simp

end

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 =

\text{ip} :: "ip"
\text{sn} :: "sqn"
\text{rt} :: "rt"
\text{rreqs} :: "(ip \times \text{rreqid}) set"
\text{store} :: "store"

186
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 = (|
  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]: "¬(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"
2.3.2 Auxilliary message handling definitions

definition is_newpkt
where "is_newpkt ξ ≡ \( \text{case msg } ξ \) of
\begin{align*}
\text{Newpkt data’ dip’ } & \Rightarrow \{ ξ[| data := data’, dip := dip’] \} \\
\_ & \Rightarrow \{\}
\end{align*}


definition is_pkt
where "is_pkt ξ ≡ \( \text{case msg } ξ \) of
\begin{align*}
\text{Pkt data’ dip’ oip’ } & \Rightarrow \{ ξ[| data := data’, dip := dip’, oip := oip’ |] \} \\
\_ & \Rightarrow \{\}
\end{align*}


definition is_rreq
where "is_rreq ξ ≡ \( \text{case msg } ξ \) of
\begin{align*}
\text{Rreq hops’ rreqid’ dip’ dsn’ dsk’ oip’ osn’ sip’ } & \Rightarrow \\
\_ & \Rightarrow \{\}
\end{align*}


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’ ∧
using assms unfolding is_rreq_def
by (cases "msg ξ") simp_all


definition is_rrep
where "is_rrep ξ ≡ \( \text{case msg } ξ \) of
\begin{align*}
\_ & \Rightarrow \{\}
\end{align*}


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 ≡ \( \text{case msg } ξ \) of
\begin{align*}
\text{Rerr dests’ sip’ } & \Rightarrow \{ ξ[| dests := dests’, sip := sip’ |] \} \\
\_ & \Rightarrow \{\}
\end{align*}


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 \Rightarrow \text{ip } ξ’ = \text{ip } ξ"
"ξ’ ∈ is_rrep ξ \Rightarrow \text{ip } ξ’ = \text{ip } ξ"
"ξ’ ∈ is_rreq ξ \Rightarrow \text{ip } ξ’ = \text{ip } ξ"
\[ \xi' \in \text{is_pkt} \xi \implies \text{ip} \xi' = \text{ip} \xi \]
\[ \xi' \in \text{is_newpkt} \xi \implies \text{ip} \xi' = \text{ip} \xi \]
unfolding \text{is_msg_defs}
by (cases "msg \xi", clarsimp+)+

lemma \text{is_msg_inv_sn} [simp]:
\[ \xi' \in \text{is_rerr} \xi \implies \text{sn} \xi' = \text{sn} \xi \]
\[ \xi' \in \text{is_rrep} \xi \implies \text{sn} \xi' = \text{sn} \xi \]
\[ \xi' \in \text{is_rreq} \xi \implies \text{sn} \xi' = \text{sn} \xi \]
\[ \xi' \in \text{is_pkt} \xi \implies \text{sn} \xi' = \text{sn} \xi \]
\[ \xi' \in \text{is_newpkt} \xi \implies \text{sn} \xi' = \text{sn} \xi \]
unfolding \text{is_msg_defs}
by (cases "msg \xi", clarsimp+)+

lemma \text{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 \text{is_msg_defs}
by (cases "msg \xi", clarsimp+)+

lemma \text{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 \text{is_msg_defs}
by (cases "msg \xi", clarsimp+)+

lemma \text{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 \text{is_msg_defs}
by (cases "msg \xi", clarsimp+)+

lemma \text{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 \text{is_msg_defs}
by (cases "msg \xi", clarsimp+)+

2.3.3 The protocol process

datatype \text{pseqp} =
\begin{align*}
\text{PAodv} & \\
\text{PNewPkt} & \\
\text{PPkt} & \\
\text{PRreq} & \\
\text{PRrep} & \\
\text{PRerr} & 
\end{align*}

fun \text{nat_of_seqp} :: "\text{pseqp} \Rightarrow \text{nat}"
where
\begin{align*}
\text{nat_of_seqp PAodv} & = 1 \\
\text{nat_of_seqp PNewPkt} & = 2 \\
\text{nat_of_seqp PPkt} & = 3 \\
\text{nat_of_seqp PRreq} & = 4 \\
\text{nat_of_seqp PRrep} & = 5
\end{align*}
instantiation "pseqp" :: ord
begin

definition less_eq_seqp [iff]: "l1 \leq l2 = (nat_of_seqp l1 \leq nat_of_seqp l2)"
definition less_seqp [iff]: "l1 < l2 = (nat_of_seqp l1 < nat_of_seqp l2)"

instance .. end

abbreviation AODV where
"AODV \equiv \lambda_. [\text{clear_locals}] \text{call(PAodv)}"

abbreviation PKT where
"PKT args \equiv
\begin{array}{l}
[\xi. \text{let } (\text{data}, \text{dip}, \text{oip}) = \text{args } \xi \text{ in}
\begin{array}{l}
(\text{clear_locals } \xi) (| \text{data} := \text{data}, \text{dip} := \text{dip}, \text{oip} := \text{oip})
\end{array}
\text{call(PPkt)}]
\end{array}"

abbreviation NEWPKT where
"NEWPKT args \equiv
\begin{array}{l}
[\xi. \text{let } (\text{data}, \text{dip}) = \text{args } \xi \text{ in}
\begin{array}{l}
(\text{clear_locals } \xi) (| \text{data} := \text{data}, \text{dip} := \text{dip})
\end{array}
\text{call(PNewPkt)}]
\end{array}"

abbreviation RREQ where
"RREQ args \equiv
\begin{array}{l}
[\xi. \text{let } (\text{hops}, \text{rreqid}, \text{dip}, \text{dsn}, \text{dsk}, \text{oip}, \text{osn}, \text{sip}) = \text{args } \xi \text{ in}
\begin{array}{l}
(\text{clear_locals } \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})
\end{array}
\text{call(PRreq)}]
\end{array}"

abbreviation RREP where
"RREP args \equiv
\begin{array}{l}
[\xi. \text{let } (\text{hops}, \text{dip}, \text{dsn}, \text{oip}, \text{sip}) = \text{args } \xi \text{ in}
\begin{array}{l}
(\text{clear_locals } \xi) (| \text{hops} := \text{hops}, \text{dip} := \text{dip}, \text{dsn} := \text{dsn},
\text{oip} := \text{oip}, \text{sip} := \text{sip})
\end{array}
\text{call(PRrep)}]
\end{array}"

abbreviation RERR where
"RERR args \equiv
\begin{array}{l}
[\xi. \text{let } (\text{dests}, \text{sip}) = \text{args } \xi \text{ in}
\begin{array}{l}
(\text{clear_locals } \xi) (| \text{dests} := \text{dests}, \text{sip} := \text{sip})
\end{array}
\text{call(PRerr)}]
\end{array}"

fun \Gamma_{AODV} :: "(state, msg, pseqp, pseqp label) seqp_env"
where
"\Gamma_{AODV} PAdv = labelled PAdv (receive(\lambda msg'. \xi. (| \text{msg} := \text{msg'}))).
\oplus (\text{is_newpkt}) NEWPKT(\lambda\xi. (\text{data } \xi, \text{ip } \xi))
\oplus (\text{is_pkt}) PKT(\lambda\xi. (\text{data } \xi, \text{dip } \xi, \text{oip } \xi))
\oplus (\text{is_rreq})
\begin{array}{l}
[\xi. \text{let } (\text{rt} := \text{update } (\text{rt } \xi) (\text{sip } \xi) (0, \text{unk}, \text{val}, 1, \text{sip } \xi, \{\}))) RREQ(\lambda\xi. (\text{hops } \xi, \text{rreqid } \xi, \text{dip } \xi, \text{dsn } \xi, \text{dsk } \xi, \text{oip } \xi, \text{osn } \xi, \text{sip } \xi))
\end{array}
\oplus (\text{is_rrep})
\begin{array}{l}
[\xi. \text{let } (\text{rt} := \text{update } (\text{rt } \xi) (\text{sip } \xi) (0, \text{unk}, \text{val}, 1, \text{sip } \xi, \{\}))) RREP(\lambda\xi. (\text{hops } \xi, \text{dip } \xi, \text{dsn } \xi, \text{oip } \xi, \text{sip } \xi))
\end{array}
\oplus (\text{is_rerr})
```plaintext
Γ. ξ (rt := update (rt ξ) (sip ξ) (o, unk, val, 1, sip ξ, {})) []
RERR(λξ. (dests ξ, sip ξ))

② (λξ. { ξ | dip := dip }) | dip. dip ∈ qD(store ξ) \ vD(rt ξ)}
[ξ. ξ () data := hd(σqueue(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) []]
[ξ. ξ () pre := {the (prec (rt ξ) rip) | rip. rip ∈ dom (dests ξ)} []]
[ξ. ξ () dests := (λrip. if ((dests ξ) rip \̸= None \&\& the (prec (rt ξ) rip) \̸= {}))
then (dests ξ) rip else None) []]
groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)). AODV()  

③ (λξ. { ξ | dip := dip }) |
dip. dip ∈ qD(store ξ) \ vD(rt ξ) \ the (σp-flag(store ξ, dip)) = req }
[ξ. ξ () store := unsetRRF (store ξ) (dip ξ) []]
[ξ. ξ () sn := inc (sn ξ) []]
[ξ. ξ () rreqid := nrreqid (rreqs) (ip ξ) []]
[ξ. ξ () rreqs := rreqs \cup \{(ip ξ, rreqid ξ) \}}]
broadcast(λξ. rreq(0, rreqid ξ, dip ξ, sqn (rt ξ) (dip ξ), sqnf (rt ξ) (dip ξ),
ip ξ, sn ξ, ip ξ)). AODV()"

"ΓAODV PNewPkt = labelled PNewPkt ( 
  ⟨ξ. dip ξ = ip ξ⟩
  deliver(λξ. data ξ).AODV()
⑤ ⟨ξ. dip ξ \̸= ip ξ⟩
  (⟨ξ. dip ξ ∈ vD (rt ξ)⟩
unicast(λξ. the (nhop (rt ξ) (dip ξ)), λξ. pkt(data ξ, dip ξ, oip 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) []]
[ξ. ξ () pre := {the (prec (rt ξ) rip) | rip. rip ∈ dom (dests ξ)} []]
[ξ. ξ () dests := (λrip. if ((dests ξ) rip \̸= None \&\& the (prec (rt ξ) rip) \̸= {}))
then (dests ξ) rip else None) []]
groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)). AODV()  

⑥ ⟨ξ. dip ξ \̸∈ vD (rt ξ)⟩
  (⟨ξ. dip ξ ∈ iD (rt ξ)⟩
unicast(λξ. the (nhop (rt ξ) (dip ξ)), λξ. pkt(data ξ, dip ξ, oip 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) []]
[ξ. ξ () pre := {the (prec (rt ξ) rip) | rip. rip ∈ dom (dests ξ)} []]
[ξ. ξ () dests := (λrip. if ((dests ξ) rip \̸= None \&\& the (prec (rt ξ) rip) \̸= {}))
then (dests ξ) rip else None) []]
groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)). AODV()  

⑦ ⟨ξ. dip ξ \̸∈ iD (rt ξ)⟩
  )")

"ΓAODV PRreq = labelled PRreq ( 
  ⟨ξ. (oip ξ, rreqid ξ) ∈ rreqs ξ⟩
AODV()
⑧ ⟨ξ. (oip ξ, rreqid ξ) \̸∈ rreqs ξ⟩
  [ξ. ξ () rt := update (rt ξ) (oip ξ) (osn ξ, kno, val, hops ξ + 1, sip ξ, {})] []
  [ξ. ξ () rreqs := rreqs ξ \cup \{(oip ξ, rreqid ξ)\}}]
```
(ξ. dip ξ = ip ξ)
[ξ. oip ξ = ip ξ] AODV()
① (ξ. dip ξ = ip ξ)
}

/*Γ_AODV PRrep = labelled PRrep */
"""AODV PRrep = labelled PRrep (rt := update (rt ξ) (dip ξ) (dsn ξ, kno, val, hops ξ + 1, sip ξ, {}))]

(ξ. oip ξ = ip ξ) AODV()
① (ξ. oip ξ ≠ ip ξ)
(ξ. dip ξ ∈ vD (rt ξ) ∧ dip ξ ∈ vD (rt ξ))
[ξ. rt := the (addpreRT (rt ξ) (dip ξ) (the (nhop (rt ξ) (oip ξ))))]
[ξ. rt := the (addpreRT (rt ξ) (the (nhop (rt ξ) (dip ξ))))]
unicast(ξ. the (nhop (rt ξ) (oip ξ)), ξ. rrep(the (dhops (rt ξ) (dip ξ))), dip ξ, sqn (rt ξ) (dip ξ), oip ξ, ip ξ)). AODV()
② (ξ. dip ξ ∈ vD (rt ξ) ∧ dip ξ ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (oip ξ))
then Some (inc (sqn (rt ξ) rip)) else None)
[ξ. rt := invalidate (rt ξ) (dests ξ)]
[ξ. store := setRRF (store ξ) (dests ξ)]
[ξ. pre := ∪ { the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) }]
[ξ. dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ {}) then (dests ξ) rip else None)]
groupcast(ξ. pre ξ, ξ. rerr(dests ξ, ip ξ)). AODV()
\[
\begin{align*}
\text{groupcast}(\lambda \xi. \text{pre} \equiv \bigcup \{ \text{the (precs (rt \xi) rip) | rip. rip} \in \text{dom (dests \xi)} \}) \\
\text{dests} := (\lambda \text{rip. case (dests \xi) rip of } \text{None} \Rightarrow \text{None} | \text{Some rsn} \Rightarrow \text{if rip} \in \text{vD (rt \xi) \land the (nhop (rt \xi) rip) = sip \xi} \\
\text{sq (rt \xi) rip < rsn then Some rsn else None} ))
\end{align*}
\]
lemma aodv_wf [simp]:
  "wellformed $\Gamma_{AODV}$"
proof (rule, intro allI)
  fix pn pn'
  show "call(pn') $\notin$ 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 pn p
  assume "p $\in$ stermss($\Gamma_{AODV}$ 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]:
  "($\xi$, p) $\in$ $\sigma_{AODV}$ i $\Rightarrow$ kD (rt $\xi$) = {}"
  unfolding $\sigma_{AODV}$_def kD_def by simp

lemma aodv_init_sip_not_ip [simp]: "$\neg$(sip (aodv_init i) = i)" by simp

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

lemma clear_locals_sip_not_ip':
  assumes "ip $\xi$ = i"
  shows "$\neg$(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 $\Gamma_{AODV}$_sims [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
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"
"∀ 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 ⇒ 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"
"∀ 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"
"∀ 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 →
  (sqn crt oipc > osnc
   ∨ (sqn crt oipc = osnc
      ∧ the (dhops crt oipc) ≤ hopsc
      ∧ the (flag crt oipc) = val))))
  | Rrep hopsc dipc _ _ _ _ _ _ _ ipcc ⇒ (ipcc ≠ dipc →
  0 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"

"∀ 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 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) = (∀ rip ∈ dom(dests).
    rip ∈ iD(crt) ∧ the (dests rip) = sqn crt rip)"

"∀ 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)))"
2.5  Quality relations between routes

theory B_Fresher
imports B_Aodv_Data
begin

2.5.1  Net sequence numbers

On individual routes

definition
  nsqn :: "r ⇒ sqn"
where
  "nsqn r ≡ (if π4(r) = val ∨ 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 (0, dsk, flag, hops, nhip, pre) = 0"
  unfolding nsqn_r_def by clarsimp

lemma nsqn_r_val [simp]:
  "∀dsn dsk hops nhip pre. nsqn (dsn, dsk, val, hops, nhip, pre) = dsn"
  unfolding nsqn_r_def by clarsimp

lemma nsqn_r_inv [simp]:
  "∀dsn dsk hops nhip pre. nsqn (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 (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 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 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, (the (the (addpreRT rt dip npre) dip')) = nsqn, (the (rt dip'))"
unfolding addpreRT_def nsqn_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 (clarsimp split: option.split)

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 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
  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)

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 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'"Longrightarrow P r r'"
  and "nsqn r = nsqn r' ∧ π₅(r) ≥ π₅(r')"Longrightarrow P r r'"
  shows "P r r'"
  using assms unfolding fresher_def by auto

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

lemma fresher_trans [elim, trans]:
  "[ x ⊑ y; y ⊑ z ]Longrightarrow x ⊑ z"
  unfolding fresher_def by auto

lemma not_fresher_trans [elim, trans]:
  "[ ¬(x ⊑ y); ¬(z ⊑ x) ]Longrightarrow ¬(z ⊑ y)"
  unfolding fresher_def by auto

lemma fresher_dsn_flag_hops_const [simp]:
  fixes dsn dsk ds' flag hops nhip nhip' pre pre'
  shows "(dsn, dsk, flag, hops, nhip, pre) ⊑ (dsn, ds', flag, hops, nhip', pre')"
  unfolding fresher_def by (cases flag) simp_all
lemma addpre_fresher [simp]: "∀ r npre. r ⊑ (addpre r npre)"
  by clarsimp

2.5.3 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 ∧ the (dhops rt1 dip) ≥ the (dhops rt2 dip))"
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_def refresh_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" ("(_/≈_ _)") [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 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 "¬ (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 "¬ (the (rt1 ip) ⊑ the (rt2 ip))"
  shows "¬ (rt1 ⊑ ip rt2)"
  proof
    assume "the (rt1 ip) ⊑ the (rt2 ip)"
    hence "rt1 ⊑ ip rt2" ..
    with (¬ (the (rt1 ip) ⊑ the (rt2 ip))) show False ..
    qed

lemma rt_fresh_as_nsqnr:
  assumes "dip ∈ kD(rt1)"
  and "dip ∈ kD(rt2)"

202
"rt1 \approx_{\text{dip}} rt2" shows \( \text{nsqn}_r (\text{the (rt2 dip)}) = \text{nsqn}_r (\text{the (rt1 dip)}) \)

using \text{assms}(3) unfolding \text{rt\_fresh\_as\_def}

by (auto simp: \text{rt\_fresh\_as\_def2} \[ OF \langle \text{dip} \in kD(rt1) \rangle \langle \text{dip} \in kD(rt2) \rangle \]
\text{kD\_nsqn} \[ OF \langle \text{dip} \in kD(rt2) \rangle \langle \text{dip} \in kD(rt1) \rangle \]
\text{kD\_nsqn} \[ OF \langle \text{dip} \in kD(rt2) \rangle \]

\text{lemma rt\_fresher\_mapupd \[ intro! \]:}
assumes \( \text{dip} \in kD(rt) \)
and \( \text{the (rt dip)} \subseteq r \)
shows \( rt \subseteq \text{dip rt (dip \mapsto r)} \)
using \text{assms unfolding \text{rt\_fresher\_def} by simp}

\text{lemma rt\_fresher\_map_update_other \[ intro! \]:}
assumes \( \text{dip} \in kD(rt) \)
and \( \text{dip} \neq ip \)
shows \( rt \subseteq \text{dip rt (ip \mapsto r)} \)
using \text{assms unfolding \text{rt\_fresher\_def} by simp}

\text{lemma rt\_fresher\_update_other \[ simp \]:}
assumes \text{inkD: } \text{dip} \in kD(rt)
and \( \text{dip} \neq ip \)
shows \( rt \subseteq \text{dip update rt ip r} \)
using \text{assms unfolding \text{update\_def} by \text{clarsimp split: option.split} (\text{fastforce})}

\text{theorem rt\_fresher\_update \[ simp \]:}
assumes \( \text{dip} \in kD(rt) \)
and \( \text{the (dhops rt dip)} \geq 1 \)
and \( \text{update\_arg\_wf r} \)
shows \( rt \subseteq \text{dip update rt ip r} \)
proof (cases "dip = ip")
next
assume "dip = ip"
from \( \langle \text{dip} \in kD(rt) \rangle \) obtain \( \text{dsn}_n, \text{dsk}_n, f_n, \text{hops}_n, \text{nhip}_n, \text{pre}_n \)
where \( \text{rtn} \[ simp \]: \text{the (rt dip)} = (\text{dsn}_n, \text{dsk}_n, f_n, \text{hops}_n, \text{nhip}_n, \text{pre}_n) \)
by (metis \text{prod\_cases6})
with \( \langle \text{the (dhops rt dip)} \geq 1 \rangle \) and \( \langle \text{dip} \in kD(rt) \rangle \) have \( \text{hops}_n \geq 1 \)
by (metis \text{proj5\_eq\_dhops \text{projs}(4)})
from \( \langle \text{dip} \in kD(rt) \rangle \) \( \text{rtn} \) have \( \text{sqn rt dip = dsn}_n \)
and \[ simp \]: \( \text{the (dhops rt dip)} = \text{hops}_n \)
and \[ simp \]: \( \text{the (flag rt dip)} = f_n \)
by (simp add: \text{sqn\_def proj5\_eq\_dhops [symmetric] proj4\_eq\_flag [symmetric]})

from \( \langle \text{update\_arg\_wf r} \rangle \) have \( (\text{dsn}_n, \text{dsk}_n, f_n, \text{hops}_n, \text{nhip}_n, \text{pre}_n) \)
\( \subseteq \text{the ((update rt dip r) dip)} \)
proof (rule \text{wf\_r\_cases})
next
fix \text{nhip pre}
from \( \langle \text{hops}_n \geq 1 \rangle \) have "\( \text{\backslash pre}. (\text{dsn}_n, \text{dsk}_n, f_n, \text{hops}_n, \text{nhip}_n, \text{pre}_n) \)
\( \subseteq (\text{dsn}_n, \text{unk}, \text{val}, \text{Suc 0}, \text{nhip}, \text{pre}') \)"
unfolding \text{fresher\_def sqn\_def} by (cases \text{f}_n) auto
thus "(\text{dsn}_n, \text{dsk}_n, f_n, \text{hops}_n, \text{nhip}_n, \text{pre}_n)
\( \subseteq \text{the ((update rt dip (0, unk, val, Suc 0, nhip, pre) dip)} \)"
using \( \langle \text{dip} \in kD(rt) \rangle \) by - (rule \text{update\_cases\_kd}, \text{simp\_all})
next
fix \text{dsn :: sqn and hops nhip pre}
assume "\text{0 < dsn}"
show "(\text{dsn}_n, \text{dsk}_n, f_n, \text{hops}_n, \text{nhip}_n, \text{pre}_n)
\( \subseteq \text{the ((update rt dip (dsn, kno, val, hops, nhip, pre) dip)} \)"
proof (rule update_cases_kD [OF _ (dip∈kD(rt))], simp_all add: 0 < dsn)
  assume "dsn n < dsn"
  thus "\((dsn_n, dsk_n, f_n, hops_n, nhip_n, pre_n)\)\n      \subseteq (dsn, kno, val, hops, nhip, pre \cup pre_n)"
    unfolding fresher_def by auto
next
  assume "dsn n = dsn"
  and "hops < hops_n"
  thus "\((dsn, dsk_n, f_n, hops_n, nhip_n, pre_n)\)\n      \subseteq (dsn, kno, val, hops, nhip, pre \cup pre_n)"
    unfolding fresher_def nsqn._def by simp
next
  assume "dsn n = dsn"
  with \(0 < dsn\) show "\((dsn, dsk_n, inv, hops_n, nhip_n, pre_n)\)\n      \subseteq (dsn, kno, val, hops, nhip, 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: "\(\forall rip\in dom(dests). rip\in vD(rt) \land sqn rt rip < the (dests rip)\)"
  shows "rt \subseteq dip invalidate rt dests"
proof (cases "dip\in dom(dests)"
  assume "dip\in dom(dests)"
  thus ?thesis using \{dip\in kD(rt)\}
    by - (rule single_rt_fresher, simp)
next
  assume "dip\in 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 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\in kD(rt)"
  and "\(\forall rip\in dom(dests). rip\in vD(rt) \land the (dests rip) = inc (sqn rt rip)\)"
  shows "rt \equiv dip invalidate rt dests"
proof (cases "dip\in dom(dests)"
  assume "dip\in dom(dests)"
  with \{dip\in kD(rt)\} have "dip\in kD(invalidate rt dests)"
    by simp
  ultimately show ?thesis
    by rule (simp_all add: \{dip\in dom(dests)\})
next
  assume "dip\in dom(dests)"
  with assms(2) have "dip\in vD(rt)"
    and "the (dests dip) = inc (sqn rt dip)"
      by auto
  from \{dip\in vD(rt)\} have "dip\in kD(rt)" by simp
  moreover then have "dip\in kD(invalidate rt dests)" by simp
ultimately show "thesis proof (rule nsqn_rt_fresh_asI)
  from ⟨dip∈vD(rt)⟩ have "nsqn rt dip = sqn rt dip" by simp
  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)
      show "sqn rt dip = nsqn r (the (invalidate rt dests dip))" by simp
    qed
  also have "sqn rt dip = nsqn r (the (invalidate rt dests dip))"
    proof -
      from ⟨dip∈kD(invalidate rt dests)⟩ have "nsqn r (the (invalidate rt dests dip)) = nsqn (invalidate rt dests) dip"
        by (simp add: kD_nsqn)
      finally show "nsqn rt dip = nsqn (invalidate rt dests) dip".
    qed simp
  qed

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]

2.5.4 Strictly comparing routing tables

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

abbreviation
  rt_strictly_fresher_syn :: "rt ⇒ ip ⇒ rt ⇒ bool" ("(_/ ⊏ _)") [51, 999, 51] 50

where
"rt1 ⊏ 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 ⊑ rt2"
  and "¬(rt2 ⊑ rt1)"
  shows "rt1 ⊏ rt2"
    using assms unfolding rt_strictly_fresher_def'' by simp

lemma rt_strictly_fresherE' [elim]:
  assumes "rt1 ⊑ rt2"
  and "[ rt1 ⊑ rt2; ¬(rt2 ⊑ 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 ⊑ rt2"
  and "¬(rt1 ≈ rt2)"
  shows "rt1 ⊏ 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 ⊑ rt2"
  and "[ rt1 ⊑ rt2; ¬(rt1 ≈ rt2) ] ⟹ P rt1 rt2 i"
  shows "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 =
(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"
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_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"
    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(rt2)"
  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 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 "kD(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 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 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 sqn)
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, 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, 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 sqn)
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))" .
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 ** 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)"
sows "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

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 ⪰ onll Γ_{\text{AODV}} (λ(\xi, _). sn \xi ≤ sn \xi')"
by inv_cterms

lemma sequence_number_one_or_bigger:
  "paodv i ⪰ onll Γ_{\text{AODV}} (\lambda(\xi, _). 1 ≤ sn \xi)"
by (rule onll_step_to_invariantI [OF sequence_number_increases])
  (auto simp: σ_{\text{AODV}}_def)

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

lemma sequence_number_increases':
  "paodv i ⪰ (λ(\xi, _). sn \xi ≤ 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 ≤ sn \xi)"
by (rule invariant_weakenE [OF sequence_number_one_or_bigger]) auto

lemma sip_in_kD:
  "paodv i ⪰ onll Γ_{\text{AODV}} (\lambda(\xi, 1). 1 ∈ \{PAodv::7\} \cup \{PAodv::5\} \cup \{PRreq::0..PRreq::4\}
  \cup \{PRreq::0..PRreq::3\}) ⪰ sip \xi ∈ kD (rt \xi))"
by inv_cterms

lemma addpreRT_partly_welldefined:
  "paodv i ⪰ onll Γ_{\text{AODV}} (\lambda(\xi, 1). \xi ∈ \{PRreq::16..PRreq::18\} \cup \{PRreq::1..PRreq::5\}) ⪰ dip \xi ∈ kD (rt \xi))"
\( l \in \{\text{PRreq-:3}..\text{PRreq-:17}\} \rightarrow \text{oip } \xi \in kD (rt \, \xi) \)"

by inv_cterms

Proposition 7.38

Lemma includes_nhip:

"paodv \ i \models \text{onl } \Gamma_{AODV} (\lambda (\xi , 1) . \forall \text{dip} \in kD(\text{rt} \, \xi) . \text{the} (\text{nhop} (\text{rt} \, \xi) \, \text{dip}) \in kD(\text{rt} \, \xi))"

proof -

\{ fix ip and \ \xi \ : \ : \ state

assume "\forall \text{dip} \in kD (\text{rt} \, \xi) . \text{the} (\text{nhop} (\text{rt} \, \xi) \, \text{dip}) \in kD (\text{rt} \, \xi)"

hence "\forall \text{dip} \in kD (\text{rt} \, \xi).

the (\text{nhop} (\text{update} (\text{rt} \, \xi) \, \text{ip} \, (0 , \text{unk} , \text{val} , \text{Suc} \, 0 , \text{ip} , \{\})) \, \text{dip}) = \text{ip}

∨ \text{the} (\text{nhop} (\text{update} (\text{rt} \, \xi) \, \text{ip} \, (0 , \text{unk} , \text{val} , \text{Suc} \, 0 , \text{ip} , \{\})) \, \text{dip}) \in kD (\text{rt} \, \xi)"

by clarsimp (metis nhop_update_unk_val update_another)

\} note one_hop = this

\{ fix ip sip sn hops and \ \xi \ : \ : \ state

assume "\forall \text{dip} \in kD (\text{rt} \, \xi) . \text{the} (\text{nhop} (\text{rt} \, \xi) \, \text{dip}) \in kD (\text{rt} \, \xi)"

and 

\( \xi' = \xi (| \text{rt} := \text{update} (\text{rt} \, \xi) \, \text{ip} \, (\text{sn} , \text{kno} , \text{val} , \text{Suc} \, \text{hops} , \text{sip} , \{\})) |)

and 

\( \text{sip} \in kD (\text{rt} \, \xi) \)

hence 

\( (\text{the} (\text{nhop} (\text{update} (\text{rt} \, \xi) \, \text{ip} \, (\text{sn} , \text{kno} , \text{val} , \text{Suc} \, \text{hops} , \text{sip} , \{\})) \, \text{ip}) = \text{ip}

\lor \text{the} (\text{nhop} (\text{update} (\text{rt} \, \xi) \, \text{ip} \, (\text{sn} , \text{kno} , \text{val} , \text{Suc} \, \text{hops} , \text{sip} , \{\})) \, \text{ip}) \in kD (\text{rt} \, \xi)

\lor (\text{the} (\text{nhop} (\text{update} (\text{rt} \, \xi) \, \text{ip} \, (\text{sn} , \text{kno} , \text{val} , \text{Suc} \, \text{hops} , \text{sip} , \{\})) \, \text{dip}) = \text{ip}

\lor \text{the} (\text{nhop} (\text{update} (\text{rt} \, \xi) \, \text{ip} \, (\text{sn} , \text{kno} , \text{val} , \text{Suc} \, \text{hops} , \text{sip} , \{\})) \, \text{dip}) \in kD (\text{rt} \, \xi))"

by (metis kD_update_unchanged nhop_update_changed update_another)

\} note nhip_is_sip = this

show \( \text{thesis} \) by (inv_cterms inv add: onl_invariantSterms [OF aodv_wf sip_in_kD]

onl_invariantSterms [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 \models \text{onl } \Gamma_{AODV} (\lambda (\xi , 1) \cdot \forall \text{dip} \in kD(\text{rt} \, \xi) . \text{the} (\text{nhop} (\text{rt} \, \xi) \, \text{dip}) \in kD(\text{rt} \, \xi)"

\( (l = \text{PRreq-:17} \rightarrow \text{oip} \, \xi \in kD(\text{rt} \, \xi)) \land

(l = \text{PRrep-:4} \rightarrow \text{dip} \, \xi \in kD(\text{rt} \, \xi)) \land

(l = \text{PRrep-:5} \rightarrow (\text{the} (\text{nhop} (\text{rt} \, \xi) \, \text{dip} \, \xi)) \in kD(\text{rt} \, \xi)))"

(is 

"\_ \models \text{onl } \Gamma_{AODV} \, ?P")

unfolding invariant_def

proof

fix s

assume "s \in \text{reachable} (\text{paodv} \, i) \, \text{TT}"

then obtain \( \xi \, p \) where "s = (\xi , p)"

and "(\xi , p) \in \text{reachable} (\text{paodv} \, i) \, \text{TT}"

by (metis PairE)

have "\text{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 \{\text{PRreq-:16}..\text{PRreq-:18}\} \rightarrow \text{dip} \, \xi \in kD(\text{rt} \, \xi)"

and I2: "l = \text{PRreq-:17} \rightarrow \text{oip} \, \xi \in kD(\text{rt} \, \xi)"

and I3: "l \in \{\text{PRrep-:1}..\text{PRrep-:5}\} \rightarrow \text{dip} \, \xi \in kD(\text{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 \land I3

have "l = \text{PRrep-:5} \rightarrow (\text{the} \, (\text{nhop} \, (\text{rt} \, \xi) \, (\text{dip} \, \xi))) \in kD(\text{rt} \, \xi)"

by (auto dest!: invariantD [OF includes_nhip])

ultimately show "?P (\xi , 1)"

by simp

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

by simp

qed
Proposition 7.4

lemma known_destinations_increase:
  "paodv i |= onl 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 |= 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, 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 ≤ 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
    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: σAODV_def)

Proposition 7.8

lemma sender_ip_valid':

"paodv i |=A onll $\Gamma_{AODV}$ ($\lambda((\xi, \ _), a, \ _). \text{anycast}\ (\lambda m. \text{not}_Pkt m \rightarrow \text{msg}_sender m = \text{ip} \ (\xi)) a) "
by $\text{inv}_\text{cterms}$

lemma $\text{sender}_\text{ip}_\text{valid}$:
"paodv i |=A onll $\Gamma_{AODV}$ ($\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'])
(auto dest!: onll onld)

lemma $\text{received}_\text{msg}_\text{inv}$:
"paodv i |= (recvmsg P \rightarrow) onll $\Gamma_{AODV}$ ($\lambda(\xi, l). l \in \{P\text{Aodv}-:1\} \rightarrow P \ (\text{msg} \ (\xi)) "
by $\text{inv}_\text{cterms}$

Proposition 7.9

lemma $\text{sip}\_\text{not}\_\text{ip}'$:
"paodv i |= (recvmsg ($\lambda m. \text{not}_Pkt m \rightarrow \text{msg}_sender m \neq i) \rightarrow) onll $\Gamma_{AODV}$ ($\lambda((\xi, \ _). \text{sip} \ (\xi \neq \text{ip} \ (\xi)) "$
by (inv\_cterm inv add: onll\_invariant\_sterms [OF aodv\_wf received\_msg\_inv]
\text{onll\_invariant\_sterms} [OF aodv\_wf ip\_constant [THEN invariant\_restrict\_inD]]
simp add: clear\_locals\_sip\_not\_ip') clarsimp+

lemma $\text{sip}\_\text{not}\_\text{ip}$:
"paodv i |= (recvmsg ($\lambda m. \text{not}_Pkt m \rightarrow \text{msg}_sender m \neq i) \rightarrow) onll $\Gamma_{AODV}$ ($\lambda((\xi, \ _). \text{sip} \ (\xi \neq i)) "$
by (inv\_cterm inv add: onll\_invariant\_sterms [OF aodv\_wf received\_msg\_inv]
\text{onll\_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 $\text{hop}_\text{count}_\text{positive}$:
"paodv i |= onll $\Gamma_{AODV}$ ($\lambda((\xi, \ _). \forall ip \in kD (rt \ (rt) \ (\text{dhops} (rt) \ (\text{ip}) \ (1)) \geq 1) "$
by (inv\_cterm inv add: onll\_invariant\_sterms [OF aodv\_wf addpreRT\_welldefined]) auto

lemma $\text{rreq}_\text{dip}_\text{in}_\text{vD}_\text{dip}_\text{eq}_\text{ip}$:
"paodv i |= onll $\Gamma_{AODV}$ ($\lambda(\xi, l). l \in \{\text{Preq}-:16..\text{Preq}-:18\} \rightarrow \text{dip} \ (\xi) \in vD(\ (rt)) "$
\hfill ($1 \in \{\text{Preq}-:5, \text{Preq}-:6\} \rightarrow \text{dip} \ (\xi) = \text{ip} \ (\xi)$
\hfill ($1 \in \{\text{Preq}-:15..\text{Preq}-:18\} \rightarrow \text{dip} \ (\xi) \neq \text{ip} \ (\xi)$)

proof (inv\_cterm, elim conjE)

fix $1 \xi pp p'$
assume "$(\xi, pp) \in \text{reachable (paodv i)} TT"$
and "$\{\text{Preq}-:17]\{\xi, \xi) = \text{the}\ (\text{addpreRT} (\text{nt}) \ (\text{eip} \ (\xi) \ (\text{the}\ (\text{nhop}\ (\text{nt}) \ (\text{dip} \ (\xi))))}) \ (\text{pp}) "$\hfill ($\text{p'} "\text{e}\ \text{sterms} \ \Gamma_{AODV} \ (\text{pp}) "$\hfill ($\text{dip} \ (\xi) \in \text{vD} (\text{rt}) "$)
and "$1 = \text{Preq}-:17 "$\hfill ($\text{p'} "\text{e}\ \text{sterms} \ \Gamma_{AODV} \ (\text{pp}) "$\hfill ($\text{dip} \ (\xi) \in \text{vD} (\text{rt}) "$)
and "$\text{dip} \ (\text{dip} \ (\xi) \in \text{vD} (\text{rt}) "$)
from this(1-3) have "$\text{dip} \ (\xi) \in kD (\text{rt}) "$ by (auto dest: onll\_invariant\_sterms [OF aodv\_wf addpreRT\_welldefined, where l="\text{Preq}-:17"])
with (\text{dip} \ (\xi) \in \text{vD (rt)} $
show "\text{dip} \ (\xi) \in \text{vD (the}\ (\text{addpreRT} (\text{rt}) \ (\text{eip} \ (\xi) \ (\text{the}\ (\text{nhop}\ (\text{rt}) \ (\text{dip} \ (\xi)))))) "$ by simp
qed

lemma $\text{rrep}_\text{dip}_\text{in}_\text{vD}$:
"paodv i |= onll $\Gamma_{AODV}$ ($\lambda(\xi, l). l \in \{\text{Preq}-:4..\text{Preq}-:6\} \rightarrow \text{dip} \ (\xi) \in vD(\ (rt)) "$
proof (inv\_cterm)

fix $1 \xi pp p'$
assume "$(\xi, pp) \in \text{reachable (paodv i)} TT"$
and "$\{\text{Preq}-:5}\{\xi, \xi) = \text{the}\ (\text{addpreRT} (\text{nt}) \ (\text{eip} \ (\xi) \ (\text{the}\ (\text{nhop}\ (\text{nt}) \ (\text{dip} \ (\xi))))}) \ (\text{pp}) "$\hfill ($\text{p'} "\text{e}\ \text{sterms} \ \Gamma_{AODV} \ (\text{pp}) "$\hfill ($\text{dip} \ (\xi) \in \text{vD (rt)} "$)
and "$1 = \text{Preq}-:5 "$\hfill ($\text{p'} "\text{e}\ \text{sterms} \ \Gamma_{AODV} \ (\text{pp}) "$\hfill ($\text{dip} \ (\xi) \in \text{vD (rt)} "$)
and "$\text{dip} \ (\xi) \in \text{vD (rt)} "$)
from this(1-3) have "the (\text{the}\ (\text{nhop}\ (\text{rt}) \ (\text{dip} \ (\xi)))) \ (\text{pp}) "$ by simp
qed
Proposition 7.11

**Lemma anycast_msg_zhops:**

\[
\forall \text{rreqid dip dsn desk oip osn sip. } \quad \text{paodv i } \equiv \text{A onll } \Gamma_{AODV} (\lambda(\xi, a, \_). \text{anycast msg_zhops a})
\]

**Proof:**

\[
\text{proof (inv_cterms inv add:}
\begin{align*}
\text{onl_invariant_sterms [OF aodv_wf rreq_dip_in_vD_dip_eq_ip]} \\
\text{onl_invariant_sterms [OF aodv_wf rrep_dip_in_vD]} \\
\text{onl_invariant_sterms [OF aodv_wf hop_count_positive],} \\
\text{elim conjE)}
\end{align*}
\]

\[
\text{fix } l \xi \text{ a pp pp' pp'' assume } "\zeta, pp \in \text{reachable } (\text{paodv i) TT}" \\
\text{and } "\{\text{PRrep-:6} \} \text{unicast}(\lambda(\zeta. \text{the } (\text{nhop } (rt \xi) (\text{oip } \xi)), \\
\lambda(\zeta. \text{Rrep } (\text{the } (\text{dhops } (rt \xi) (\text{dip } \xi))) (\text{dip } \xi) (\text{sqn } (rt \xi) (\text{dip } \xi)) (\text{oip } \xi) (\text{ip } \xi)). \\
p' > pp' \in \text{sterms } \Gamma_{AODV} pp"
\]

\[
\text{and } "1 = \text{PRreq-:18}" \\
\text{and } "a = \text{unicast } (\text{the } (\text{nhop } (rt \xi) (\text{oip } \xi))) \\
(\text{Rrep } (\text{the } (\text{dhops } (rt \xi) (\text{dip } \xi))) (\text{dip } \xi) (\text{sqn } (rt \xi) (\text{dip } \xi)) (\text{oip } \xi) (\text{ip } \xi)). \\
\text{Suc } 0 \leq \text{the } (\text{dhops } (rt \xi) \text{ ip})"
\]

\[
\text{and } "\text{dip } \xi \in vD (rt \xi)" \\
\text{from } dip \xi \in vD (rt \xi) \text{ have } "\text{Suc } 0 \leq \text{the } (\text{dhops } (rt \xi) (\text{dip } \xi))" ..
\]

\[
\text{thus } "0 < \text{the } (\text{dhops } (rt \xi) (\text{dip } \xi))" \text{ by simp}
\]

**Next:**

\[
\text{fix } l \xi \text{ a pp pp' pp'' assume } "\zeta, pp \in \text{reachable } (\text{paodv i) TT}" \\
\text{and } "\{\text{PRreq-:6} \} \text{unicast}(\lambda(\zeta. \text{the } (\text{nhop } (rt \xi) (\text{oip } \xi)), \\
\lambda(\zeta. \text{Rrep } (\text{the } (\text{dhops } (rt \xi) (\text{dip } \xi))) (\text{dip } \xi) (\text{sqn } (rt \xi) (\text{dip } \xi)) (\text{oip } \xi) (\text{ip } \xi)). \\
p' > pp' \in \text{sterms } \Gamma_{AODV} pp"
\]

\[
\text{and } "1 = \text{PRreq-:6}" \\
\text{and } "a = \text{unicast } (\text{the } (\text{nhop } (rt \xi) (\text{oip } \xi))) \\
(\text{Rrep } (\text{the } (\text{dhops } (rt \xi) (\text{dip } \xi))) (\text{dip } \xi) (\text{sqn } (rt \xi) (\text{dip } \xi)) (\text{oip } \xi) (\text{ip } \xi)). \\
\text{Suc } 0 \leq \text{the } (\text{dhops } (rt \xi) \text{ ip})"
\]

\[
\text{and } "\text{dip } \xi \in vD (rt \xi)" \\
\text{from } dip \xi \in vD (rt \xi) \text{ have } "\text{Suc } 0 \leq \text{the } (\text{dhops } (rt \xi) (\text{dip } \xi))" ..
\]

\[
\text{thus } "\text{the } (\text{dhops } (rt \xi) (\text{dip } \xi)) = 0 \rightarrow \text{dip } \xi = \text{ip } \xi" \text{ by simp}
\]

**QED**

**Lemma hop_count_zero_oip_dip_sip:**

\[
\text{"paodv i } \equiv \text{ (recvmsg msg_zhops } \rightarrow) \text{ onll } \Gamma_{AODV} (\lambda(\xi, 1). \\
(1 \in \{\text{PAodv-:4, PAodv-:5}\} \cup \{\text{PRreq-:n|n. True}\} \rightarrow \\
(\text{hops } \xi = 0 \rightarrow oip \xi = sip \xi))
\]

\[
\wedge \\
((1 \in \{\text{PAodv-:6, PAodv-:7}\} \cup \{\text{PRrep-:n|n. True}\} \rightarrow \\
(\text{hops } \xi = 0 \rightarrow dip \xi = sip \xi)))")
\]

\[
\text{by (inv_cterms inv add: onll_invariant_sterms [OF aodv_wf received_msg_inv]) auto}
\]

**Lemma osn_rreq:**

\[
\text{"paodv i } \equiv \text{ (recvmsg rreq_rrep_sn } \rightarrow) \text{ onll } \Gamma_{AODV} (\lambda(\xi, 1). \\
1 \in \{\text{PAodv-:4, PAodv-:5}\} \cup \{\text{PRreq-:n|n. True}\} \rightarrow 1 \leq \text{ osn } \xi")
\]

\[
\text{by (inv_cterms inv add: onll_invariant_sterms [OF aodv_wf received_msg_inv]) clarsimp}
\]

**Lemma osn_rreq':**

\[
\text{"paodv i } \equiv \text{ (recvmsg } (\lambdam. \text{ rreq_rrep_sn m } \& \text{ msg_zhops m}) \rightarrow) \text{ onll } \Gamma_{AODV} (\lambda(\xi, 1). \\
1 \in \{\text{PAodv-:4, PAodv-:5}\} \cup \{\text{PRreq-:n|n. True}\} \rightarrow 1 \leq \text{ osn } \xi")
\]

**Proof:**

\[
\text{rule invariant_weakenE [OF osn_rreq])}
\]

\[
\text{fix a}
\]

\[
\text{assume } "\text{recvmsg } (\lambdam. \text{ rreq_rrep_sn m } \& \text{ msg_zhops m}) \text{ a}"
\]

\[
\text{thus } "\text{recvmsg } \text{ rreq_rrep_sn a}"
\]

213
by (cases a) simp_all

qed

lemma dsn_rrep:
"paodv i \models (\text{recvmsg req\_rrep\_sn } \to) \text{ onl } \Gamma_{AODV} (\lambda(\xi, l).
1 \in \{\text{PAodv-:6, PAodv-:7} \cup \text{PRrep-:n|n. True} \to 1 \leq \text{dsn } \xi\}"
by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]) clarsimp

lemma dsn_rrep':
"paodv i || (\text{recvmsg (} \lambda\ m. rreq\_rrep\_sn m \land \text{msg\_zhops } m) \to) \text{ onl } \Gamma_{AODV} (\lambda(\xi, l).
(l \in \{\text{PAodv-:6, PAodv-:7} \cup \text{PRrep-:n|n. True} \to 1 \leq \text{dsn } \xi\})
proof (rule invariant_weakenE [OF dsn_rrep])
fix a
assume "\text{recvmsg (} \lambda\ m. rreq\_rrep\_sn m \land \text{msg\_zhops } m) a"
thus "\text{recvmsg rreq\_rrep\_sn } a"
by (cases a) simp_all

qed

lemma hop_count_zero_oip_dip_sip':
"paodv i || (\text{recvmsg (} \lambda\ m. rreq\_rrep\_sn m \land \text{msg\_zhops } m) \to) \text{ onl } \Gamma_{AODV} (\lambda(\xi, l).
(l \in \{\text{PAodv-:4..PAodv-:5} \cup \text{PRreq-:n|n. True} \to \text{hops } \xi = 0 \to \text{oip } \xi = \text{sip } \xi\})
\land
((l \in \{\text{PAodv-:6..PAodv-:7} \cup \text{PRrep-:n|n. True} \to \text{hops } \xi = 0 \to \text{dip } \xi = \text{sip } \xi))))"
proof (rule invariant_weakenE [OF hop_count_zero_oip_dip_sip])
fix a
assume "\text{recvmsg (} \lambda\ m. rreq\_rrep\_sn m \land \text{msg\_zhops } m) a"
thus "\text{recvmsg msg\_zhops } a"
by (cases a) simp_all

qed

Proposition 7.12

lemma zero_seq_unk_hops_one':
"paodv i || (\text{recvmsg (} \lambda\ m. rreq\_rrep\_sn m \land \text{msg\_zhops } m) \to) \text{ onl } \Gamma_{AODV} (\lambda(\xi, _).
\forall dip \in kD(rt). (sqn (\text{rt } \xi dip = 0 \to sqnf (\text{rt } \xi dip) = \text{unk})
\land
((\text{sqnf } (\text{rt } \xi dip) = \text{unk } \to \text{the (dhops (rt } \xi dip) = 1}))
\land
(\text{the (dhops (rt } \xi dip) = 1 } \to \text{the (nhop (rt } \xi dip) = dip )))"
proof -
{ fix dip and \xi :: state and P
  assume "sqn (\text{invalidate (rt } \xi dip) (\text{dests } \xi)) dip = 0"
  and all: "\forall ip. sqn (\text{rt } \xi dip \leq sqn (\text{invalidate (rt } \xi dip) (\text{dests } \xi)) ip"
  and *: "sqn (\text{rt } \xi dip = 0 \to P \text{ } \xi dip"
  have "P \text{ } \xi dip"
  proof -
  from all have "sqn (\text{rt } \xi dip \leq sqn (\text{invalidate (rt } \xi dip) (\text{dests } \xi)) dip" ..
  with sqn (\text{invalidate (rt } \xi dip) (\text{dests } \xi)) dip = 0: have "sqn (\text{rt } \xi dip = 0" by simp
  thus "P \text{ } \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 \in kD(rt).
  (sqn rt dip = 0 \to \pi_3(\text{the (rt dip)}) = \text{unk})
  \land
  ((\pi_3(\text{the (rt dip)}) = \text{unk } \to \text{the (dhops rt dip) = Suc 0})
  \land
  (\text{the (dhops rt dip) = Suc 0 } \to \text{the (nhop rt dip) = dip})))
  and "hops = 0 \to sip = dip"
  and "Suc 0 \leq dsn" and "ip \neq dip \to ip \in kD(rt)"
  hence "the (dhops (update rt dip (dsn, kno, val, Suc hops, sip, {})) ip) = Suc 0 \to
  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

214
{ 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 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)
} 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_TT [OF aodv_wf addprefRT_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 (Am. rreq_rrep_sn m ∧ msg_zhops m) → onl ΓAODV (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 req rept sn →) onl Γ AODV (λξ, 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 pre1 pre2
    assume "dsk1 = unk ∨ Suc 0 ≤ dsn2"
    hence "π₃ (the (update rt sip (dsn1, dsk1, flag1, hops1, nhip1, pre1) sip)) = unk
      ∨ Suc 0 ≤ sqn (update rt sip (dsn2, dsk2, flag2, hops2, nhip2, pre2) sip)"
    unfolding update_def by (cases "dsk1 = unk") (clarsimp split: option.split)+
  }

  { 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
    }

  { fix dip dsn 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
        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_update [simp] = this

  note solve_invalidate [simp] = this
Proposition 7.13

Lemma \( \text{rreq\_rrep\_sn\_any\_step\_invariant} \):

\[
\text{paodv i} \models \text{onll} \Gamma_{AODV} (\lambda(, a, \_). \text{anycast rreq\_rrep\_sn a})
\]

Proof -

- \(\text{have sqnf\_kno: "paodv i} \models \text{onl} \Gamma_{AODV} (\lambda(, 1).\)

\[
(1 \in \{PRreq-:16..PRreq-:18\} \rightarrow \text{sqnf (rt } \xi) (\text{dip } \xi) = \text{kno}))"
\]

by \((\text{inv\_cterms inv add: onl\_invariant\_sterms\_TT [OF aodv\_wf addpreRT\_welldefined]})\)

- \(\text{have rrep\_sqn\_greater\_dsn: "paodv i} \models \text{onl} \Gamma_{AODV} (\lambda(, 1).\)

\[
(1 \in \{PRreq-:1 .. PRreq-:6\} \rightarrow 1 \leq \text{sqn (rt } \xi) (\text{dip } \xi))"
\]

by \((\text{inv\_cterms inv add: onl\_invariant\_sterms [OF aodv\_wf received\_msg\_inv]}
\text{onl\_invariant\_sterms\_TT [OF aodv\_wf addpreRT\_welldefined]}
\text{onl\_invariant\_sterms [OF aodv\_wf dsn\_rrep]})\)

(clarsimp simp: update\_kno\_dsn\_greater\_zero [simplified])

show \(?\text{thesis}\)

by \((\text{inv\_cterms inv add: onl\_invariant\_sterms\_TT [OF aodv\_wf addpreRT\_welldefined]}
\text{onl\_invariant\_sterms [OF aodv\_wf sequence\_number\_one\_or\_bigger}
\text{[THEN invariant\_restrict\_inD]]}
\text{onl\_invariant\_sterms [OF aodv\_wf kD\_unk\_or\_atleast\_one]}
\text{onl\_invariant\_sterms\_TT [OF aodv\_wf sqnf\_kno]}
\text{onl\_invariant\_sterms [OF aodv\_wf osn\_rreq]}
\text{onl\_invariant\_sterms [OF aodv\_wf dsn\_rrep]}
\text{onl\_invariant\_sterms [OF aodv\_wf rrep\_sqn\_greater\_dsn]})\)

(auto simp: proj2\_eq\_sqn)

qed

Proposition 7.14

Lemma \( \text{rreq\_rrep\_fresh\_any\_step\_invariant} \):

\[
\text{paodv i} \models \text{onll} \Gamma_{AODV} (\lambda(, a, \_). \text{anycast (rreq\_rrep\_fresh (rt } \xi)) a)
\]

Proof -

- \(\text{have rreq\_oip: "paodv i} \models \text{onl} \Gamma_{AODV} (\lambda(, 1).\)

\[
(1 \in \{PRreq-:3, PRreq-:4, PRreq-:15, PRreq-:27\} \rightarrow \text{oip } \xi \in \text{kD(rt } \xi) \text{)} \land \text{(sqn (rt } \xi) (\text{oip } \xi) > (\text{osn } \xi) \text{)} \lor \text{(sqn (rt } \xi) (\text{oip } \xi) = (\text{osn } \xi) \text{)} \land \text{the (dhops (rt } \xi) (\text{oip } \xi)) \leq \text{Suc (hops } \xi) \text{)} \land \text{the (flag (rt } \xi) (\text{oip } \xi)) = \text{val}))"
\]

proof inv\_cterms

fix \(1 \xi \_\_ pp p\')

assume "((, pp) \in \text{reachable (paodv i) TT}"

and "\{PRreq\_2\}[]\lambda. (, rt := \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} pp"

and "1' = PRreq\_3"

show "\text{osn } \xi < \text{sqn (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 \land \text{the (dhops (update (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) \land \text{the (flag (update (rt } \xi) (\text{oip } \xi) (\text{osn } \xi, \text{kno, val, Suc (hops } \xi), \text{sip } \xi, {)}\} (\text{oip } \xi)) = \text{val})"

unfolding update\_def by \((\text{clarsimp split: option\_split})\)

(auto simp: proj2\_eq\_sqn)

qed

have \( \text{rrep\_prrep: "paodv i} \models \text{onl} \Gamma_{AODV} (\lambda(, 1).\)

\[
(1 \in \{PRreq-4, PRreq-6\} \rightarrow \text{(dip } \xi \in \text{kD(rt } \xi) \land \text{the (flag (rt } \xi) (\text{dip } \xi)) = \text{val})")
\]

217
by (inv_cterms inv add: onl_invariant_sterms TT [OF aodv_wf addpreRT_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 |=A onll Γ AODV (λ((ξ, _), a, _). anycast (rerr_invalid (rt ξ)) a)"
proof -
  have dests_inv: "paodv i |=
    onl 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 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 |=
  onl 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 = 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)+

thesis

by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf dests_inv])

qed

Proposition 7.27

lemma route_tables_fresher:
"paodv i |=A (recvmsg rreq_rrep_sn →) 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_iD]]
  onl_invariant_sterms [OF aodv_wf hop_count_positive [THEN invariant_restrict_iD]]
  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_iD]])

fix ξ pp p'

assume "((ξ, pp) ∈ reachable (paodv i) (recvmsg rreq_rrep_sn)"
  and "(PRreq-:2)[λξ. ξ(rt := update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ, {}))]

218
2.7 The quality increases predicate

theory B_Quality_Increases
imports B_Aodv_Predicates B_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'"
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 ⊑ ip rt ξ'"
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"
using assms by auto
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
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" ..

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 :sn = nsqn (rt (σ' sip)) ip have "sqn (rt (σ' sip)) ip > 1" by simp 
next
assume "ip ∈ iD(rt (σ' sip))"
hence "nsqn (rt (σ' sip)) ip > sqn (rt (σ' sip)) ip" ..
with :sn = nsqn (rt (σ' sip)) ip have "nsqn (rt (σ' sip)) ip > nsqn (rt (σ' sip)) ip" by simp 
thus ?thesis .. 
qed 
qed 
thus ?thesis by (metis (mono_tags) le_cases not_le) 
qed 
with ⟨ip ∈ kD (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 
∧ (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 :: "(ipc ⇒ state) ⇒ msg ⇒ bool"
where
"msg_fresh σ m ≡
case m of Rreq hops _ _ _ _ oipc osnc sipc ⇒ osnc ≥ 1 ∧ (sipc ≠ oipc →
oipc∈kd(rt (σ sipc)) ∧ nsqn (rt (σ sipc)) oipc ≥ osnc
∧ (nsqn (rt (σ sipc)) oipc = osnc
→ (hops ≥ the (dhops (rt (σ sipc)) oipc)
∨ the (flag (rt (σ sipc)) oipc) = inv))))
| Rrep hops dipc _ _ _ _ sipc ⇒ dsn ≥ 1 ∧ (sipc ≠ dipc →
dipc∈kd(rt (σ sipc)) ∧ nsqn (rt (σ sipc)) dipc ≥ dsn
∧ (nsqn (rt (σ sipc)) dipc = dsn
→ (hops ≥ 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 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 (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))))"

"∀ 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
assume 
$sip \neq oip$

and 
$"nsqn \ ?rt oip = osn"$

show 
"the (dhops \ ?rt oip) \leq \text{hops} \lor \text{the (flag \ ?rt oip) = inv}"

proof 
(cases "oip\in\nuD(\text{rt})")

assume "oip\in\nuD(\text{rt})"

hence "nsqn \ ?rt oip = sqn \ ?rt oip" ..

with \langle nsqn \ ?rt oip = osn \rangle have "sqn \ ?rt oip = osn" by simp

with assms(1) and \langle sip \neq oip \rangle have "the (dhops \ ?rt oip) \leq \text{hops}"

by simp

thus \ ?thesis ..

next

assume "oip \notin \nuD(\text{rt})"

moreover from assms(1) and \langle sip \neq oip \rangle have "oip\in\nuD(\text{rt})" by simp

ultimately have "oip\in\nuD(\text{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 (\text{rt (\sigma sip)}) oip"

proof 
(rule nat_le_eq_or_lt)

assume "osn < sqn \ ?rt oip"

hence "osn \leq sqn \ ?rt oip - 1" by simp

also have "... \leq nsqn \ ?rt oip" by (rule sqn_nsqn)

finally show "osn \leq nsqn \ ?rt oip" .

next

assume "osn = sqn \ ?rt oip"

with assms(1) and \langle sip \neq oip \rangle have "oip\in\nuD(\text{rt})"

and "the (flag \ ?rt oip) = val"

by auto

hence "nsqn \ ?rt oip = sqn \ ?rt oip" ..

with \langle osn = sqn \ ?rt oip \rangle have "nsqn \ ?rt oip = osn" by simp

thus "osn \leq nsqn \ ?rt oip" by simp

qed

qed simp

lemma rrep_nsqn_is_fresh [simp]:

fixes \sigma \ msg hops dip dsn oip sip

assumes "rreq_rrep_fresh (rt (\sigma sip)) (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 \longrightarrow dip\in\nuD(?rt) \land sqn \ ?rt dip = dsn \land the (flag \ ?rt dip) = val" by simp

hence "sip \neq dip \longrightarrow dip\in\nuD(?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 (rt (\sigma sip)) (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\text{dom(dests)}. (rip\in\nuD(rt (\sigma sip)) \land the (dests rip) = sqn (rt (\sigma sip)) rip) \rangle"

224
by clarsimp
have "(\forall \text{rip} \in \text{dom(dests)}. (\text{rip} \in \text{kD}(\text{rt} (\sigma \text{ sip}))) \\
\wedge \text{the (dests rip)} - 1 \leq \text{nsqn (rt (\sigma \text{ sip})) rip})"

proof
fix rip
assume "rip \in \text{dom dests}"
with * have "rip \in \text{iD}(\text{rt} (\sigma \text{ sip}))" and "the (dests rip) = \text{sqn (rt (\sigma \text{ sip})) rip}" \\
by auto
from this(2) have "the (dests rip) - 1 = \text{sqn (rt (\sigma \text{ sip}) rip)} - 1" \\
by simp
also have "... \leq \text{nsqn (rt (\sigma \text{ sip})) rip}" by (rule sqn_nsqn)
finally have "the (dests rip) - 1 \leq \text{nsqn (rt (\sigma \text{ sip})) rip}".

qed
thus "msg_fresh \sigma \ ?msg" \\
by simp
qed

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 = \text{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}(\text{rt} (\sigma \text{ sip})) \wedge \text{osn} \leq \text{nsqn (rt (\sigma \text{ sip})) oip} \\
  \wedge (\text{nsqn (rt (\sigma \text{ sip})) oip} = \text{osn} \\
  \rightarrow (\text{the (dhops (rt (\sigma \text{ sip})) oip)} \leq \text{hops} \\
  \lor \text{the (flag (rt (\sigma \text{ sip})) oip)} = \text{inv})))"
  by auto
  from this(2) show ?thesis
  proof
    assume "\text{sip} = \text{oip}" with \text{(osn \geq 1)} show ?thesis by simp
  next
    assume "\text{oip} \in \text{kD}(\text{rt} (\sigma \text{ sip})) \wedge \text{osn} \leq \text{nsqn (rt (\sigma \text{ sip})) oip} \\
  \wedge (\text{nsqn (rt (\sigma \text{ sip})) oip} = \text{osn} \\
  \rightarrow (\text{the (dhops (rt (\sigma \text{ sip})) oip)} \leq \text{hops} \\
  \lor \text{the (flag (rt (\sigma \text{ sip})) oip)} = \text{inv}))" \\
  moreover from qinc have "\text{quality_increases (\sigma \text{ sip}) (\sigma' \text{ sip})}" ..
  ultimately have "\text{oip} \in \text{kD}(\text{rt} (\sigma' \text{ sip})) \wedge \text{osn} \leq \text{nsqn (rt (\sigma' \text{ sip})) oip} \\
  \wedge (\text{nsqn (rt (\sigma' \text{ sip})) oip} = \text{osn} \\
  \rightarrow (\text{the (dhops (rt (\sigma' \text{ sip})) oip)} \leq \text{hops} \\
  \lor \text{the (flag (rt (\sigma' \text{ sip})) oip)} = \text{inv}))"
  using \text{(osn \geq 1)} by (rule quality_increases_rreq_rrep_props [rotated 2])
  with \text{(osn \geq 1)} show "\text{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 "\text{msg_fresh (\sigma m)}"
  then have "\text{dsn} \geq 1" and "\text{sip} = \text{dip} \lor (\text{dip} \in \text{kD}(\text{rt} (\sigma \text{ sip})) \wedge \text{dsn} \leq \text{nsqn (rt (\sigma \text{ sip})) dip} \\
  \wedge (\text{nsqn (rt (\sigma \text{ sip})) dip} = \text{dsn} \\
  \rightarrow (\text{the (dhops (rt (\sigma \text{ sip})) dip)} \leq \text{hops} \\
  \lor \text{the (flag (rt (\sigma \text{ sip})) dip)} = \text{inv}))" \\
  by auto
  from this(2) show ?thesis
  proof
    assume "\text{sip} = \text{dip}" with \text{(dsn \geq 1)} show ?thesis by simp
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

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 oaedv_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' ⟹ labels Γ_AODV p = {PAodv-:0})"
unfolding σ_AODV'_def by simp

lemma oaedv_init_kD_empty [simp]: "((σ, p) ∈ σ_AODV' ⟹ kD (rt (σ i)) = {})"
unfolding σ_AODV'_def kD_def by simp
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 \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 \Rightarrow P' \sigma m"
  and weakenQ: "\forall \sigma m. Q \sigma m \Rightarrow Q' \sigma m"
  shows "otherwith P' I (orecvmsg Q') \sigma \sigma' a"
  proof
    fix j
    assume "j /\in I"
    with * have "P (\sigma j) (\sigma' j)" by auto
    thus "P' (\sigma j) (\sigma' j)" by (rule weakenP)
  next
    from * have "orecvmsg Q \sigma a" by auto
    thus "orecvmsg Q' \sigma a" by (erule weakenQ)
  qed

lemma orecvmessage_invariant:
  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 "\sigmamsg i = (otherwith Q \{i\} (orecvmsg P), other Q \{i\} \Rightarrow)
  onl \Gamma_AODV (\lambda(\sigma, l). l \in {PAodv-:1} \rightarrow P \sigma (msg (\sigma i)))"
  proof (inv_cterms, intro impl)
    fix \sigma s l
    assume "l = PAodv-:1 \rightarrow P \sigma (msg (\sigma i))"
    and "l = PAodv-:1"
    and "other Q \{i\} \sigma s'"
    from this(1-2) have "P \sigma (msg (\sigma i))" ..
    hence "P \sigma' (msg (\sigma i))" using (other Q \{i\} \sigma s')
    by (rule other)
    moreover from (other Q \{i\} \sigma s') have "\sigma' i = \sigma i" ..
    ultimately show "P \sigma' (msg (\sigma i))" by simp
  next
    fix \sigma s msg
    assume "otherwith Q \{i\} (orecvmsg P) \sigma s' (receive msg)"
and "σ′ i = σ i[msg := msg]")
from this(1) have "P σ msg"
    and "∀ j. j≠i \rightarrow 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 \rightarrow 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 \models_A (recvmsg rreq_rrep_sn \rightarrow) onll Γ\text{AODV} (λ((ξ, _), _, (ξ′, _)). quality_increases ξ ξ′)"
proof (rule step_invariantI)
fix s a s'
assume sr: "s \in reachable (paodv i) (recvmsg rreq_rrep_sn)"
    and tr: "(s, a, s') \in trans (paodv i)"
    and rm: "recvmsg rreq_rrep_sn a"
from sr have srTT: "s \in reachable (paodv i) TT" ..
from route_tables_fresher sr tr rm
have "onll Γ\text{AODV} (λ((ξ, _), _, (ξ′, _)). ∀ dip∈kD (rt ξ). rt ξ \sqsubseteq dip rt ξ′) (s, a, s')"
    by (rule step_invariantD)
moreover from known_destinations_increase srTT tr TT_True
have "onll Γ\text{AODV} (λ((ξ, _), _, (ξ′, _)). kD (rt ξ) \subseteq kD (rt ξ′)) (s, a, s')"
    by (rule step_invariantD)
moreover from sqns_increase srTT tr TT_True
have "onll Γ\text{AODV} (λ((ξ, _), _, (ξ′, _)). ∀ ip. sqn (rt ξ) ip \leq sqn (rt ξ′) ip) (s, a, s')"
    by (rule step_invariantD)
ultimately show "onll Γ\text{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 \models_A (otherwith quality_increases \{i\} (orecvmsg (λ_. rreq_rrep_sn)),
    other_quality_increases \{i\} \rightarrow)
    onll Γ\text{AODV} (λ((σ, _), _, (σ′, _)). ∀ j. quality_increases (σ j) (σ′ j))"
(is "\models_A (\{S, \_ \rightarrow \}_)"
proof (rule onll_ostep_invariantI, simp)
fix σ p a σ′ p′ l'
assume or: "(σ, p) \in reachable (opaodv i) ?S (other_quality_increases \{i\})"
    and ll: "l ∈ labels Γ\text{AODV} p"
    and "?S σ σ′ a" 
    and tr: "((σ, p), a, (σ′, p′)) \in oseqp_sos Γ\text{AODV} i"
    and ll': "l' ∈ labels Γ\text{AODV} p'"
from this(1-3) have "orecvmsg (λ_. rreq_rrep_sn) σ a"
    by (auto dest!: reachable_weakenE [where QS="act (recvmsg rreq_rrep_sn)"
        and QU="other_quality_increases \{i\}"]
otherwith_actionD)
with or have orw: "(σ, p) \in reachable (opaodv i) (act (recvmsg rreq_rrep_sn))
    (other_quality_increases \{i\})"
    by (erule reachable_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)"

228
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} \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 \text{oreachable (opaodv i)} (act (recvmsg rreq_rrep_sn)) (other A {i})"
and "((\sigma, p), a, (\sigma', p')) \in \text{oseqp_sos }{\Gamma_{AODV} i}"
and recv: "act (recvmsg rreq_rrep_sn) \sigma \sigma' a"

obtain l l' where "l \in \text{labels }{\Gamma_{AODV} p}" and "l' \in \text{labels }{\Gamma_{AODV} p'}" by (metis aodv_ex_label)
from "((\sigma, p), a, (\sigma', p')) \in \text{oseqp_sos }{\Gamma_{AODV} i}" and recv
have tr: "((\sigma, p), a, (\sigma', p')) \in \text{trans (opaodv i)}" by simp

have "anycast (rreq_rrep_fresh (rt (\sigma i))) a"
proof -
have "opaodv i |\ A (act (recvmsg rreq_rrep_sn), other A {i} \rightarrow)
onll \Gamma_{AODV} (\text{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} (\text{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 \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 |\ A (act (recvmsg rreq_rrep_sn), other A {i} \rightarrow)
onll \Gamma_{AODV} (\text{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} (\text{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 \text{labels }{\Gamma_{AODV} p}\} and \{l' \in \text{labels }{\Gamma_{AODV} p'}\} by auto
qed

moreover have "anycast rreq_rrep_sn a"
proof -
from or tr recv
have "onll \Gamma_{AODV} (\text{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 \text{labels }{\Gamma_{AODV} p}\} and \{l' \in \text{labels }{\Gamma_{AODV} p'}\} by auto
qed

moreover have "anycast (\lambda. \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} (\text{seqll }i (\lambda((\xi, _, a, _)). anycast (\lambda. \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 \{l \in \text{labels }{\Gamma_{AODV} p}\} and \{l' \in \text{labels }{\Gamma_{AODV} p'}\}

229
ultimately have "anycast (msg_fresh σ) a"

thus "onll Γ\textsubscript{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} →)
onl Γ\textsubscript{AODV} (λ((σ, _), _, (σ', _)). anycast (msg_fresh σ) a) ((σ, p), a, (σ', p'))"

proof (rule oreceived_msg_inv)

fix σ σ'
p l
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 |= (otherwith quality_increases {i} (orecvmsg msg_fresh),
other quality_increases {i} →)
onll Γ\textsubscript{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\textsubscript{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\textsubscript{AODV} (λ(ξ, 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 → _)"
证明 (inv_cterms inv add: oseq_step_invariant_branches [OF oquality_increases_nsqn_fresh]
aodv_wf oadv_trans]
onll oinvariant_branches [OF aodv_wf oreceived_rreq_rrep_nsqn_fresh_inv]
onll oinvariant_branches [OF aodv_wf oosn_rreq]
simp add: seqsimp
simp del: [one_nat_def, rule impI]
fix σ σ' p l
assume "(σ, p) ∈ oreachable (opaodv i) ?S ?U"
and \( l \in \text{labels} \Gamma_{AODV} \ p \)

and pre:

\[
(1 = \text{PAodv}:-4 \lor 1 = \text{PAodv}:-5 \lor 1 = \text{PReq}:-0 \lor 1 = \text{PReq}:-2) \land \text{sip} (\sigma \ i) \neq \text{oip} (\sigma \ i)
\]
\[
\rightarrow \text{oip} (\sigma \ i) \in kD (rt (\sigma (\text{sip} (\sigma \ i))))
\]
\[
\land \text{osn} (\sigma \ i) \leq \text{nsqn} (rt (\sigma (\text{sip} (\sigma \ i)))) (\text{oip} (\sigma \ i))
\]
\[
\land (\text{nsqn} (rt (\sigma (\text{sip} (\sigma \ i)))) (\text{oip} (\sigma \ i)) = \text{osn} (\sigma \ i)
\]
\[
\rightarrow (\text{the} (\text{dhops} (rt (\sigma (\text{sip} (\sigma \ i)))) (\text{oip} (\sigma \ i))) \leq \text{hops} (\sigma \ i)
\]
\[
\lor (\text{the} (\text{flag} (rt (\sigma (\text{sip} (\sigma \ i)))) (\text{oip} (\sigma \ i))) = \text{inv})
\]

and "other_quality_increases \{i\} \ σ σ'"

and hyp: "(1 = \text{PAodv}:-4 \lor 1 = \text{PAodv}:-5 \lor 1 = \text{PReq}:-0 \lor 1 = \text{PReq}:-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 "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})"

proof (cases "\text{sip} (\sigma \ i) = \text{i}'")

assume "\text{sip} (\sigma \ i) \neq \text{i}'"

from (other_quality_increases \{i\} \ σ σ')

have "quality_increases (∧ (\text{sip} (\sigma \ i')) (\text{oip} (\sigma \ i'))))")

by (rule otherE) (clarsimp simp:)

moreover from \(l \in \text{reachable} (\text{opaodv} \ i) ?S \ ?U\) \(l \in \text{labels} \Gamma_{AODV} \ p\)

have "1 \leq \text{osn} (\sigma \ i)"

by (auto dest!: onl_oinvariant_weakenD [OF oosn_rreq]

simp add: seqlsimp)

moreover from (\text{sip} (\sigma \ i) \neq \text{i}'

hyp' and pre

have "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:)

ultimately show "?thesis"

by (rule quality_increases_rreq_rrep_props)

next

assume "\text{sip} (\sigma \ i) = \text{i}'" thus "?thesis"

using "σ' \ i = \text{i}' hyp and pre by auto

qed

qed (auto elim!: quality_increases_rreq_rrep_props')

lemma odsn_rrep:

"\text{opaodv} \ i \mid (\text{otherwith quality_increases} \{i\} (\text{orecvmsg msg_fresh}),
\text{other_quality_increases} \{i\} \rightarrow)
\text{on1} \ \Gamma_{AODV} (\text{seq1} \ (\text{λ}(\xi, \ i). \ l \in \{\text{PAodv}:-6, \text{PAodv}:-7\} \cup \{\text{PReq}:-n\n\text{True} \rightarrow 1 \leq \text{dsn} \ \xi\}))"\n
by (rule oinvariant_weakenE [OF open_seq_invariant [OF dsn_rrep_initiali_aodv]])

(auto simp: seq1_onl_swap)

lemma rrep_sip:

"\text{opaodv} \ i \mid (\text{otherwith quality_increases} \{i\} (\text{orecvmsg msg_fresh}),
\text{other_quality_increases} \{i\} \rightarrow)
\text{on1} \ \Gamma_{AODV} (\text{λ}(\sigma, \ i).
(1 \in \{\text{PAodv}:-6, \text{PAodv}:-7, \text{PReq}:-0, \text{PReq}:-1\} \land \text{sip} (\sigma \ i) \neq \text{dip} (\sigma \ i))
\rightarrow \text{dip} (\sigma \ i) \in kD(rt (\sigma (\text{sip} (\sigma \ i))))
\land \text{nsqn} (rt (\sigma (\text{sip} (\sigma \ i)))) (\text{dip} (\sigma \ i)) \geq \text{dsn} (\sigma \ i)
\land (\text{nsqn} (rt (\sigma (\text{sip} (\sigma \ i)))) (\text{dip} (\sigma \ i)) = \text{dsn} (\sigma \ i)
\rightarrow (\text{hops} (\sigma \ i) \geq \text{the} (\text{dhops} (rt (\sigma (\text{sip} (\sigma \ i)))) (\text{dip} (\sigma \ i)))
\lor (\text{the} (\text{flag} (rt (\sigma (\text{sip} (\sigma \ i)))) (\text{dip} (\sigma \ i))) = \text{inv}))")\n
(is "" \mid (?S, ?U \rightarrow "")"

proof (inv_cterms inv add: oseq_step_invariant_sterns [OF quality_increases_nsqn_fresh aodv_wf

aoadv_trans]

onl_oinvariant_sterns [OF aodv_wf oreceived_rreq_rrep_nsqn_fresh_inv]

onl_oinvariant_sterns [OF aodv_wf odsn_rrep]

231
simp del: One_nat_def, rule iml)

fix σ σ' p 1
assume "(σ, p) ∈ oreachable (opaodv i) ?S ?U"
and "1 ∈ labels Γ_{AODV} p"
and pre:
"(l = PAodv-:6 ∨ l = PAodv-:7 ∨ l = PRrep-:0 ∨ l = PRrep-:1) ∧ dip (σ i) ≠ dip (σ i)
→ dip (σ i) ∈ kD (rt (σ (sip (σ i))))
∧ dns (σ i) ≤ nsqn (rt (σ (sip (σ i)))) (dip (σ i))
∧ (nsqn (rt (σ (sip (σ i)))) (dip (σ i)) = dns (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) ∧ dip (σ i) ≠ dip (σ i)"
(is "?labels ∧ dip (σ i) ≠ dip (σ i)"
from (4) have "σ i = σ i"
with hyp have hyp': "?labels ∧ dip (σ i) ≠ dip (σ i)" by simp
show "dip (σ i) ∈ kD (rt (σ (sip (σ i))))
∧ dns (σ i) ≤ nsqn (rt (σ (sip (σ i)))) (dip (σ i))
∧ (nsqn (rt (σ (sip (σ i)))) (dip (σ i)) = dns (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 "dip (σ i) = i")
assume "dip (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: dip (σ i) ≠ i)
moreover from ⟨σ, p⟩ ∈ oreachable (opaodv i) ?S ?U | 1 ∈ labels Γ_{AODV} p and hyp
have "1 ≤ dns (σ i)"
by (auto dest!: onl_invariant_weakenD [OF dsn_rrep]
simp add: seqsimp [σ' i = σ i])
moreover from "dip (σ i) ≠ i" hyp' and pre
have "dip (σ i) ∈ kD (rt (σ (sip (σ i))))
∧ dns (σ i) ≤ nsqn (rt (σ (sip (σ i)))) (dip (σ i))
∧ (nsqn (rt (σ (sip (σ i)))) (dip (σ i)) = dns (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_req_rrep_props)
next
assume "dip (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, l).
1 ∈ {PAodv-:8, PAodv-:9, PRrep-:0, PRrep-:1}
→ (∀ rip ∈ dom(dests (σ i)). rip ∈ kD (rt (σ (sip (σ i)))) ∧
the (dests (σ i) rip) - 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 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(rt (σ’ sip)) ∨ rsn - 1 ≤ nsqn (rt (σ’ sip)) rip"
proof
from \(\text{rip} \in \text{kD}(rt (\sigma \text{ sip}))\) and \(\text{quality Increases} (\sigma \text{ sip}) (\sigma’ \text{ sip})\)
show "rip ∈ kD(rt (σ’ sip))" ..
next
from \(\text{rip} \in \text{kD}(rt (\sigma \text{ sip}))\) and \(\text{quality Increases} (\sigma \text{ sip}) (\sigma’ \text{ sip})\)
have "nsqn (rt (σ’ sip)) rip ≤ nsqn (rt (σ’ sip)) rip" ..
with \(\text{rsn - 1 ≤ nsqn (rt (σ’ sip)) rip}\) show "rsn - 1 ≤ nsqn (rt (σ’ sip)) rip"
by (rule le_trans)
qed
\}
\note{partial = this}
show ?thesis
by (inv_terms inv add: oseq_step_invariant_sterms [OF equality_increases nsqn fresh aodv_wf aodv_trans]
onl_oinvvariant_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

\textbf{Lemma} prerr_guard: "paodv i |=
onl Γ_{AODV} (λ(ξ, 1). (1 = 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_terms) (clarsimp split: option.split_asm split_if_asm)

\textbf{Lemma} addpreRT_welldefined =
open_seq_invariant [OF addpreRT_welldefined initiali_aodv aodv_trans aodv_trans, simplified seql_onl_swap, THEN oinvvariant_anyact]

\textbf{Lemma} dests_vD_inc_sqn =
open_seq_invariant [OF dests_vD_inc_sqn initiali_aodv aodv_trans aodv_trans, simplified seql_onl_swap, THEN oinvvariant_anyact]

\textbf{Lemma} prerr_guard =
open_seq_invariant [OF prerr_guard initiali_aodv aodv_trans aodv_trans, simplified seql_onl_swap, THEN oinvvariant_anyact]

\textbf{Proposition 7.28}
\textbf{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 (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)). nhip 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 "nhip dip ≠ dip"
with pre have "dip∈kD(rt (σ (nhip dip)))"
and \(\text{nsqn (rt (}\sigma i))\) dip \(\le\) \(\text{nsqn (rt (}\sigma' (\text{nhop dip})\))\) dip

by auto

from qinc have qinc_nhop: "quality_increases (rt (\sigma (\text{nhop dip}))) (\sigma' (\text{nhop dip}))" ..
with \(\text{dip} \in \text{kD}(rt (\sigma (\text{nhop dip})))\) have "\(\text{dip} \in \text{kD} (rt (\sigma' (\text{nhop dip})))\)" ..

moreover have "\(\text{nsqn (rt (}\sigma i))\) dip \(\le\) \(\text{nsqn (rt (}\sigma' (\text{nhop dip})\))\) dip"

proof -
from \(\text{dip} \in \text{kD}(rt (\sigma (\text{nhop dip})))\) qinc_nhop
have "\(\text{nsqn (rt (}\sigma (\text{nhop dip}))\) dip \(\le\) \(\text{nsqn (rt (}\sigma' (\text{nhop dip})\))\) dip" ..
with \(\text{nsqn (rt (}\sigma i))\) dip \(\le\) \(\text{nsqn (rt (}\sigma (\text{nhop dip})\))\) dip show \(?\)thesis
by simp

qed

ultimately show "\(\text{dip} \in \text{kD}(rt (\sigma' (\text{nhop dip})))\)
∧ \(\text{nsqn (rt (}\sigma i))\) dip \(\le\) \(\text{nsqn (rt (}\sigma' (\text{nhop dip})\))\) dip" ..

qed

\} note basic = this

\{ fix nhop and \(\sigma, \sigma':: "\text{ip} \Rightarrow \text{state}"\)
assume pre: "\(\forall \text{dip} \in \text{kD}(rt (}\sigma i)\). nhop dip \(\ne\) dip \(\rightarrow\) \(\text{dip} \in \text{kD}(rt (}\sigma (\text{nhop dip})))\)
∧ \(\text{nsqn (rt (}\sigma i))\) dip \(\le\) \(\text{nsqn (rt (}\sigma (\text{nhop dip})\))\) dip"

and ndest: "\(\forall \text{ripic} \in \text{dom (dests (}\sigma i)\). ripic \in \text{kD} (rt (}\sigma (\text{ripic (}\sigma i)))\)
∧ the (dests (}\sigma i) ripic) - 1 \(\le\) \(\text{nsqn (rt (}\sigma (}\text{ripic (}\sigma i))))\) ripic"

and issip: "\(\forall \text{ip} \in \text{dom (dests (}\sigma i)\). nhop ip = sip (}\sigma i)\)"

and qinc: "\(\forall j. \text{quality_increases (}\sigma j (}\sigma (} j)\)"

have "\(\forall \text{dip} \in \text{kD}(rt (}\sigma i)\). nhop dip \(\ne\) dip \(\rightarrow\) \(\text{dip} \in \text{kD} (rt (}\sigma' (\text{nhop dip})))\)
∧ \(\text{nsqn (invalidate (rt (}\sigma i)) (dests (}\sigma i))\) dip \(\le\) \(\text{nsqn (rt (}\sigma' (\text{nhop dip})\))\) dip"

proof (intro ballI impI)

fix dip

assume "\(\text{dip} \in \text{kD}(rt (}\sigma i))\"
∧ nhop dip \(\ne\) dip

with pre and qinc have "\(\text{dip} \in \text{kD}(rt (}\sigma' (\text{nhop dip})))\)"
∧ "\(\text{nsqn (rt (}\sigma i))\) dip \(\le\) \(\text{nsqn (rt (}\sigma' (\text{nhop dip}))\) dip"

by (auto dest!: basic)

have "\(\text{nsqn (invalidate (rt (}\sigma i)) (dests (}\sigma i))\) dip \(\le\) \(\text{nsqn (rt (}\sigma' (\text{nhop dip}))\) dip"

proof (cases "\(\text{dip} \in \text{dom (dests (}\sigma i))\)"
assume "\(\text{dip} \in \text{dom (dests (}\sigma i))\)"
with \(\text{dip} \in \text{kD}(rt (}\sigma i))\) obtain dsn where "\(\text{dests (}\sigma i)\) dip = Some dsn" by auto
with \(\text{dip} \in \text{kD}(rt (}\sigma i))\) have "\(\text{nsqn (invalidate (rt (}\sigma i)) (dests (}\sigma i))\) dip = dsn - 1"
by (rule nsqn_invalidate_eq)

moreover have "dsn - 1 \(\le\) \(\text{nsqn (rt (}\sigma' (\text{nhop dip}))\) dip"

proof -
from \(\text{dip} = \text{Some dsn}\) have "the (\(\text{dests (}\sigma i)\) dip) = dsn" by simp
with ndest and \(\text{dip} \in \text{dom (dests (}\sigma i))\) have "\(\text{dip} \in \text{kD} (rt (}\sigma (\text{ripic (}\sigma i))))\)"
"dsn - 1 \(\le\) \(\text{nsqn (rt (}\sigma (}\text{ripic (}\sigma i))))\) dip"

by auto

moreover from issip and \(\text{dip} \in \text{dom (dests (}\sigma i))\) have "\(\text{nhop dip} = \text{sip (}\sigma i)\)" ..

ultimately have "\(\text{dip} \in \text{kD (rt (}\sigma (\text{nhop dip}))\)"
∧ "dsn - 1 \(\le\) \(\text{nsqn (rt (}\sigma (\text{nhop dip}))\) dip" by auto

with qinc show "dsn - 1 \(\le\) \(\text{nsqn (rt (}\sigma' (\text{nhop dip})\)) dip"
by simp (metis kD_nsqn_quality_increases_trans)

qed

ultimately show \(?\)thesis by simp

next
assume "\(\text{dip} \notin \text{dom (dests (}\sigma i))\)"
with \(\text{dip} \in \text{kD}(rt (}\sigma i))\)
have "\(\text{nsqn (invalidate (rt (}\sigma i)) (dests (}\sigma i))\) dip = \(\text{nsqn (rt (}\sigma (}\text{dip}))\) dip"
by (rule nsqn_invalidate_other)
with \(\text{nsqn (rt (}\sigma i))\) dip \(\le\) \(\text{nsqn (rt (}\sigma (\text{nhop dip}))\) dip) show \(?\)thesis by simp

qed

with \(\text{dip} \in \text{kD}(rt (}\sigma' (\text{nhop dip}))\) show "\(\text{dip} \in \text{kD (rt (}\sigma' (\text{nhop dip}))\)"

234
\( \land \text{nsqn (invalidate (rt } (\sigma i)) (\text{dests } (\sigma i))) \text{ dip} \leq \text{nsqn (rt } (\sigma' \text{ (nhop dip))) \text{ dip} \) ..

**qed**

\} note basic_prerr = this

\{ fix \( \sigma \sigma' :: "ip \Rightarrow \text{state}" \)

**assume a1:** "\( \forall \text{ dip} \in kD(\text{rt } (\sigma i)). \text{ the (nhop (rt } (\sigma i)) \text{ dip} \neq \text{ dip} \)

\( \rightarrow dip \in kD(\text{rt } (\sigma (\text{the (nhop (rt } (\sigma i)) \text{ dip}))) \)

\( \land \text{ nsqn (rt } (\sigma i)) \text{ dip} \leq \text{nsqn (rt } (\sigma (\text{the (nhop (rt } (\sigma i)) \text{ dip}))) \text{ dip} \)"

**and a2:** "\( \forall j. \text{ quality.increases } (\sigma j) (\sigma' j)\)"

**have** "\( \forall \text{ dip} \in kD(\text{rt } (\sigma i)). \)

\( \text{the (nhop (update (rt } (\sigma i)) (\text{sip } (\sigma i)) (0, \text{ unk, val, Suc 0, sip } (\sigma i), \{})) \text{ dip} \neq \text{ dip} \rightarrow \text{dip} \in kD(\text{rt } (\sigma' (\text{the (nhop (rt } (\sigma i)) \text{ dip})))) \)

\( \land \text{ nsqn (rt } (\sigma i)) \text{ dip} \leq \text{nsqn (rt } (\sigma' (\text{the (nhop (rt } (\sigma i)) \text{ dip})))) \text{ dip} \)

\( \) (is "\( \forall \text{ dip} \in kD(\text{rt } (\sigma i)). \ ?P \text{ dip}\)"

**proof**

**fix dip**

**assume "dip \in kD(\text{rt } (\sigma i))"**

with a1 and a2

**have** "\( \text{the (nhop (rt } (\sigma i)) \text{ dip} \neq \text{ dip} \rightarrow dip \in kD(\text{rt } (\sigma' (\text{the (nhop (rt } (\sigma i)) \text{ dip})))) \)

\( \land \text{ nsqn (rt } (\sigma i)) \text{ dip} \leq \text{nsqn (rt } (\sigma' (\text{the (nhop (rt } (\sigma i)) \text{ dip})))) \text{ dip} \)"

by - (drule(1) basic, auto)

**thus **"\( \ ?P \text{ dip}\)" by (cases "dip = sip (\sigma i)") auto

**qed**

\} note nhop_update_sip = this

\{ fix \( \sigma \sigma' \ oip \ sip \ osn \ hops\)

**assume pre:** "\( \forall \text{ dip} \in kD (\text{rt } (\sigma i)). \text{ the (nhop (rt } (\sigma i)) \text{ dip} \neq \text{ dip} \)

\( \rightarrow dip \in kD(\text{rt } (\sigma (\text{the (nhop (rt } (\sigma i)) \text{ dip}))) \)

\( \land \text{ nsqn (rt } (\sigma i)) \text{ dip} \leq \text{nsqn (rt } (\sigma (\text{the (nhop (rt } (\sigma i)) \text{ dip}))) \text{ dip} \)"

**and qinc:** "\( \forall j. \text{ quality.increases } (\sigma j) (\sigma' j)\)"

**and \( \ast : "\text{sip } \neq \text{ oip } \rightarrow oip \in kD(\text{rt } (\sigma sip))\)

\( \land \osn \leq \text{nsqn (rt } (\sigma sip)) \text{ oip} \)

\( \land (\text{nsqn (rt } (\sigma sip)) \text{ oip} = \osn \)

\( \rightarrow (\text{dhops (rt } (\sigma sip)) \text{ oip} \leq \text{hops} \)

\( \lor (\text{the (flag (rt } (\sigma sip)) \text{ oip} = \text{inv})\)"

**from pre and qinc**

**have pre':** "\( \forall \text{ dip} \in kD (\text{rt } (\sigma i)). \text{ the (nhop (rt } (\sigma i)) \text{ dip} \neq \text{ dip} \)

\( \rightarrow dip \in kD(\text{rt } (\sigma' (\text{the (nhop (rt } (\sigma i)) \text{ dip}))) \)

\( \land \text{ nsqn (rt } (\sigma i)) \text{ dip} \leq \text{nsqn (rt } (\sigma' (\text{the (nhop (rt } (\sigma i)) \text{ dip}))) \text{ dip} \)"

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} \)

\( \rightarrow oip \in kD(\text{rt } (\sigma' (\text{the (nhop (update (rt } (\sigma i)) \text{ oip} \)

\( (\text{osn, kno, val, Suc hops, sip, } \})\) \text{ oip}))) \)

\( \land \text{nsqn (update (rt } (\sigma i)) \text{ oip } (\text{osn, kno, val, Suc hops, sip, } \})\) \text{ oip} \)

\( \leq \text{nsqn (rt } (\sigma' (\text{the (nhop (update (rt } (\sigma i)) \text{ oip} \)

\( (\text{osn, kno, val, Suc hops, sip, } \})\) \text{ oip}))) \text{ oip})))\)"

(is "\( ?\text{nhop\_not\_oip } \rightarrow ?\text{oip\_in\_kD } \land ?\text{nsqn\_le\_nsqn}\)"

**proof** (rule, split update_rt_split_asm)

**assume** "\( (\sigma i) = \text{update (rt } (\sigma i)) \text{ oip } (\text{osn, kno, val, Suc hops, sip, } \})\)"

and "the (nhop (rt } (\sigma i)) \text{ oip} \neq \text{ oip}"

**with pre' show** "\( ?\text{oip\_in\_kD } \land ?\text{nsqn\_le\_nsqn}\)" by auto

**next**

**assume rtnot: "\( (\sigma i) \neq \text{update (rt } (\sigma i)) \text{ oip } (\text{osn, kno, val, Suc hops, sip, } \})\)"

and notoip: ?nhop_not_oip

**with \( \ast \text{ qinc have } ?\text{oip\_in\_kD } \)

by (clarsimp elim!: kD_quality_increases)

**moreover with \( \ast \text{ pre qinc rtnot notoip have } ?\text{nsqn\_le\_nsqn}\)

by simp (metis kD_nsqn_quality_increases_trans)

ultimately show "\( ?\text{oip\_in\_kD } \land ?\text{nsqn\_le\_nsqn}\) ..

235
\[ \text{qed update1 = this} \]

\[
\begin{array}{l}
\{ \text{fix } \sigma \sigma' \ oip \ sip \ osn \ hops \} \\
\text{assume pre: "} \forall \text{dip} \in kD (rt (\sigma i)). \text{the (nhop (rt (\sigma i)) dip) } \neq \text{ dip} \\
\text{ } \implies \text{dip} \in kD(rt (\sigma (\text{the (nhop (rt (\sigma i)) dip))})) \\
\text{ } \land \text{nsqn (rt (\sigma i))) dip } \leq \text{nsqn (rt (\sigma (\text{the (nhop (rt (\sigma i)) dip))))) dip}" \\
\text{and qinc: "} \forall j. \text{quality_increases (\sigma j) (\sigma' j)}" \\
\text{and *: "} \text{sip } \neq \text{oip } \implies \text{oip} \in kD(rt (\sigma sip)) \\
\text{ } \land \text{osn } \leq \text{nsqn (rt (\sigma sip)) oip} \\
\text{ } \land (\text{nsqn (rt (\sigma sip)) oip } = \text{osn} \\
\text{ } \implies \text{the (dhops (rt (\sigma sip)) oip) } \leq \text{hops} \\
\text{ } \lor \text{the (flag (rt (\sigma sip)) oip) } = \text{inv}" \\
\text{from pre and qinc} \\
\text{have pre': "} \forall \text{dip} \in kD (rt (\sigma i)). \text{the (nhop (update (rt (\sigma i)) dip) dip) } \neq \text{ dip} \\
\text{ } \implies \text{dip} \in kD(rt (\sigma' (\text{the (nhop (update (rt (\sigma i)) dip))})) \\
\text{ } \land \text{nsqn (rt (\sigma i))) dip } \leq \text{nsqn (rt (\sigma' (\text{the (nhop (update (rt (\sigma i)) dip))))) dip}" \\
\text{by (rule basic)} \\
\text{have } "\forall \text{dip} \in kD(rt (\sigma i)). \text{the (nhop (update (rt (\sigma i)) dip) dip) } \neq \text{ dip} \\
\text{ } \implies \text{dip} \in kD(rt (\sigma' (\text{the (nhop (update (rt (\sigma i)) dip))})) \\
\text{ } \land \text{nsqn (update (rt (\sigma i)) dip (osn, kno, val, Suc hops, sip, {})) dip } \leq \text{nsqn (rt (\sigma' (\text{the (nhop (update (rt (\sigma i)) dip)) dip))) dip}" \\
\text{(is } "\forall \text{dip} \in kD(rt (\sigma i)). _ } \implies \text{?dip_in_kD dip } \land \text{?nsqn_le_nsqn dip})" \\
\text{proof (intro ballI impI, split update_rt_split_asm)} \\
\text{fix dip} \\
\text{assume "} \text{dip} \in kD(rt (\sigma i))" \\
\text{and } "\text{the (nhop (rt (\sigma i)) dip)} } \neq \text{ dip}" \\
\text{and } "\text{rt (\sigma i) = update (rt (\sigma i)) dip (osn, kno, val, Suc hops, sip, {})}" \\
\text{with pre' show } "?\text{dip_in_kD dip } \land \text{?nsqn_le_nsqn dip}" \text{ by simp} \\
\text{next} \\
\text{fix dip} \\
\text{assume "} \text{dip} \in kD(rt (\sigma i))" \\
\text{and notdip: } "\text{the (nhop (update (rt (\sigma i)) dip) oip (osn, kno, val, Suc hops, sip, {})) dip} } \neq \text{ dip}" \\
\text{and } "\text{rt (\sigma i) } \neq \text{ update (rt (\sigma i)) dip (osn, kno, val, Suc hops, sip, {})}" \\
\text{show } "?\text{dip_in_kD dip } \land \text{?nsqn_le_nsqn dip}" \text{ by simp (cases "dip = oip")} \\
\text{assume } "\text{dip } \neq \text{oip}" \\
\text{with pre' } (\text{dip} \in kD(rt (\sigma i))) \text{ notdip} \\
\text{show } \text{?thesis by clarsimp} \\
\text{next} \\
\text{assume } "\text{dip = oip}" \\
\text{with notnot dip oip (\text{dip} \in kD(rt (\sigma i))) notdip *} \\
\text{have } "?\text{dip_in_kD dip}" \\
\text{by simp (metis kD_quality_increases) } \\
\text{moreover from } (\text{dip = oip}) \text{ rtnot qinc (\text{dip} \in kD(rt (\sigma i))) notdip *} \\
\text{have } "?\text{nsqn_le_nsqn dip}" \text{ by simp (metis kD_nsqn_quality_increases_trans) } \\
\text{ultimately show } \text{?thesis ..} \\
\text{qed} \\
\text{qed} \\
\} \text{note update2 = this} \]

\[
\text{have } "\text{opaodv i } \models (\exists S, \exists U \rightarrow ) \text{ onl } \Gamma_{AODV} (\lambda(\sigma, _). \\
\forall \text{dip } \in kD(rt (\sigma i)). \text{the (nhop (rt (\sigma i)) dip) } \neq \text{ dip} \\
\text{ } \implies \text{dip } \in kD(rt (\sigma (\text{the (nhop (rt (\sigma i)) dip))})) \\
\text{ } \land \text{nsqn (rt (\sigma i))) dip } \leq \text{nsqn (rt (\sigma (\text{the (nhop (rt (\sigma i)) dip))))) dip}" \\
\text{by (inv_cterms inv add: oseq_step_invariant_sterms [OF quality_increases_nsqn_fresh aodv_wf oaoav_trans] } \\
\text{onl_oivarient_sterms [OF aodv_wf oaddpreRT_welldefined] } \\
\text{onl_oivarient_sterms [OF aodv_wf odests_vD_inc_sqn] } \\
\text{onl_oivarient_sterms [OF aodv_wf oprerr_guard]} \\
\text{236} \]

Thus \(?\)\text{thesis}\?I \?',1\,\text{erule}(2) \ update1\?, \text{erule}(2) \ update2\)+

Thus \(?\)\text{thesis}\?I \?',1\,\text{erule}(2) \ update1\?, \text{erule}(2) \ update2\)+

Proposition 7.30

lemmas \text{okD}_unk_or_atleast_one =
\open_seq\text{invariant} \ [\text{OF} kD\_unk_or_atleast_one initial\text{i}\_aodv, \ simplified seql\_onl\_swap]

lemmas \text{ozero_seq unk hops one} =
\open_seq\text{invariant} \ [\text{OF} zero_seq\_unk_hops_one initial\text{i}\_aodv, \ simplified seql\_onl\_swap]

lemma \text{oreachable fresh okD unk or atleast one}:
fixes dip
assumes "\((\sigma, p) \in \text{oreachable (opaodv i)}\)
(\text{otherwith (op=} \{i\} (\text{orecvmsg (\(\lambda\sigma\ m. \ msg\_fresh \ \sigma\ m \wedge msg\_zhops m\)))
(\text{other quality increases} \{i\})")
and "dip\in\text{kD(rt (}\sigma\ i)\))"
shows "\(\pi_3(\text{the (rt (}\sigma\ i) dip))) = \text{unk} \vee 1 \leq \pi_2(\text{the (rt (}\sigma\ i) 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 dip\in\text{kD (rt (}\sigma\ i)\)). ?P dip"
by - (drule oinvariant\_weakenD [OF \text{okD}_unk_or_atleast_one [OF oaodv\_trans aodv\_trans]],
\text{auto dest!: onlD otherwith actionD onlD simp: seqlsimp})
with (dip\in\text{kD(rt (}\sigma\ i)\)) show \text{thesis} by simp

qed

lemma \text{oreachable fresh ozero seq unk hops one}:
fixes dip
assumes "\((\sigma, p) \in \text{oreachable (opaodv i)}\)
(\text{otherwith (op=} \{i\} (\text{orecvmsg (\(\lambda\sigma\ m. \ msg\_fresh \ \sigma\ m \wedge msg\_zhops m\)))
(\text{other quality increases} \{i\})")
and "dip\in\text{kD(rt (}\sigma\ i)\))"
shows "sqn (\text{rt (}\sigma\ i)\) dip = 0 \rightarrow sqnf (\text{rt (}\sigma\ i)\) dip = \text{unk}
\wedge \text{the (dhops (rt (}\sigma\ i)\) dip} = 1
\wedge \text{the (nhop (rt (}\sigma\ i)\) dip} = 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 dip\in\text{kD (rt (}\sigma\ i)\)). ?P dip"
by - (drule oinvariant\_weakenD [OF \text{ozero seq unk hops one} [OF oaodv\_trans aodv\_trans]],
\auto dest!: onlD otherwith actionD onlD simp: seqlsimp})
with (dip\in\text{kD(rt (}\sigma\ i)\)) show \text{thesis} by simp

qed

lemma seq\_nhop quality increases':
shows "\text{opaodv i} \models (\text{otherwith (op=} \{i\})
\text{orecvmsg (\(\lambda\sigma\ m. \ msg\_fresh \ \sigma\ m \wedge msg\_zhops m\)))},
\text{other quality increases} \{i\} \rightarrow)
\text{onl }\Gamma_{AODV} (\lambda(\sigma, \_). \forall dip. \text{let nhip = the (nhop (rt (}\sigma\ i)\) dip}
proof -

have weaken:
"\forall I Q R P. p \models (otherwith quality_increases I (orecmsg Q), other quality_increases I \rightarrow) P 
\Rightarrow p \models (otherwith (op=) I (orecmsg (\lambda \sigma. Q \circ \sigma \circ R \circ \sigma m)), 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 i) dip) "
\land (the (nhop (rt \sigma i)) dip) \neq dip
\rightarrow rt \sigma i \square dip rt \sigma (the (nhop (rt \sigma i)) dip))"

and ow: "?S i \sigma \sigma' a"

have "\forall dip. dip \in VD(rt \sigma i)
\land dip \in VD (rt \sigma i) (the (nhop (rt \sigma i)) dip))
\land (the (nhop (rt \sigma i) dip)) \neq dip
\rightarrow rt \sigma i \square 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 i) dip)"

and a4: "(the (nhop (rt \sigma i)) dip) \neq dip"

from ow have "\forall j. j \neq i \rightarrow j = \sigma' j" by auto

show "rt \sigma i \square dip rt \sigma (the (nhop (rt \sigma i)) dip))"

proof (cases "(the (nhop (rt \sigma i)) dip) = i")

assume "(the (nhop (rt \sigma i)) dip) = i"

with \{ dip \in VD(rt \sigma i) \} have "dip \in VD (rt \sigma i) (the (nhop (rt \sigma i)) dip))" by simp

with a1 a2 a4 have "rt \sigma i \square dip rt \sigma (the (nhop (rt \sigma i)) dip))" by simp

with (the (nhop (rt \sigma i) dip)) = i have "rt \sigma i \square dip rt \sigma i" by simp

hence False by simp

thus \?thesis ..

next

assume "(the (nhop (rt \sigma i)) dip) \neq i"

with \{ j. j \neq i \rightarrow j = \sigma' j \}

have *: "\sigma (the (nhop (rt \sigma i)) dip) = \sigma' (the (nhop (rt \sigma i)) dip)" by simp

with \{ dip \in VD (rt \sigma i) (the (nhop (rt \sigma i)) dip))) \} have "dip \in VD (rt \sigma i (the (nhop (rt \sigma i)) dip))" by simp

with a1 a2 a4 have "rt \sigma i \square 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 i) dip)"
\land (the (nhop (rt \sigma i) dip) \neq dip
\rightarrow rt \sigma i \square dip rt (the (nhop (rt \sigma i) dip)))"

and ow: "?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 i) (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, \}) dip)"
\square 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, \})"
\[ \square_{\text{dip}} \text{rt} (\sigma' (\text{the (nhop (update (rt (\sigma i))) sip (0, unk, val, Suc 0, sip, {})))) dip)\]

proof (cases "dip = sip")
assume "dip = sip"
with \(\text{the (nhop (update (rt (\sigma i))) sip (0, unk, val, Suc 0, sip, {})) dip} \neq dip\)
have False by simp
thus \(?\text{thesis} \).
next
assume [simp]: "dip \neq sip"
from a2 have "dip\in\nuD(rt (\sigma i)) \lor dip = sip"
  by (rule \nuD_update_val)
with \(\text{dip} \neq dip\) have "dip\in\nuD(rt (\sigma i))" by simp
moreover from a3 have "dip\in\nuD(rt (\sigma' (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) \subseteq_{\text{dip}} rt (\sigma' (the (nhop (rt (\sigma i)) dip)))"
  using a1 ow by - (drule(1) basic, simp)
with \(\text{dip} \neq dip\) show \(?\text{thesis} \)
  by - (erule rt_strictly_fresher_update_other, simp)
qed
qed

} note update_0_unk = this

{ fix \(\sigma') \text{ nhop}
assume pre: "\(\forall dip. \ \text{dip}\in\nuD(rt (\sigma i)) \land dip\in\nuD(rt (\sigma (\text{nhop dip})))) \land \text{nhop dip} \neq dip
\rightarrow rt (\sigma i) \subseteq_{\text{dip}} rt (\sigma (\text{nhop dip})))"
  and ow: "?S i \sigma' \ a"
have "\(\forall dip. \ \text{dip}\in\nuD (\text{invalidate (rt (\sigma i)) (dests (\sigma i)))}
\land dip\in\nuD (rt (\sigma' (\text{nhop dip})))) \land \text{nhop dip} \neq dip
\rightarrow rt (\sigma i) \subseteq_{\text{dip}} rt (\sigma' (\text{nhop dip})))"
proof clarify
fix dip
assume "dip\in\nuD (\text{invalidate (rt (\sigma i)) (dests (\sigma i)))}"
  and "dip\in\nuD(rt (\sigma' (\text{nhop dip})))"
  and "\text{nhop dip} \neq dip"
from this(1) have "dip\in\nuD (rt (\sigma i))" by (clarsimp dest!: \nuD_invalidate_vD_not_dests)
moreover from ow have "\(\forall j. \ j \neq i \rightarrow \sigma j = \sigma' j\)" by auto
ultimately have "rt (\sigma i) \subseteq_{\text{dip}} rt (\sigma' (\text{nhop dip}))"
  using pre \(\text{dip}\in\nuD (rt (\sigma' (\text{nhop dip}))) \land \text{nhop dip} \neq dip\)
by metis
with \(\forall j. \ j \neq i \rightarrow \sigma j = \sigma' j\) show "rt (\sigma i) \subseteq_{\text{dip}} rt (\sigma' (\text{nhop dip}))"
by (metis rt_strictly_fresher_irefl)
qed

} note invalidate = this

{ fix \(\sigma) \text{ a} \sigma' \text{ dip oip osn sip hops i}
assume pre: "\(\forall dip. \ \text{dip}\in\nuD (rt (\sigma i))
\land dip\in\nuD (rt (\sigma (\text{the (nhop (rt (\sigma i)) dip)))))
\land (\text{the (nhop (rt (\sigma i)) dip)} \neq dip
\rightarrow rt (\sigma i) \subseteq_{\text{dip}} rt (\sigma (\text{the (nhop (rt (\sigma i)) dip)))))"
  and ow: "?S i \sigma \ a"
  and "Suc 0 \leq osn"
and a6: "\text{sip} \neq oip \rightarrow oip\in\nuD (rt (\sigma sip))"
\land dip\in\nuD (update (rt (\sigma i)) oip (osn, kno, val, Suc hops, sip, {}))"
\land (\text{the (nhop (update (rt (\sigma i)) oip (osn, kno, val, Suc hops, sip, {})) dip)}) \neq dip
\rightarrow update (rt (\sigma i)) oip (osn, kno, val, Suc hops, sip, {})
\subseteq_{\text{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 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 {ω} have a5: "∀j. j ≠ i → σ j = σ' j" by auto
  show "update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})
    = rt (σ i)" 
    by (cases "?rt1 = rt (σ i)"
      proof
        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 simp
        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. σ j = σ' j" by clarsimp
        ultimately have "rt (σ i) = rt (σ (the (nhop (σ i)) dip))"
          by simp
        hence "rt (σ i) = 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) = rt (σ (the (nhop (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
  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 \rightarrow 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))
    have "Suc hops ≤ osn" by simp
    with nsqn (rt (σ sip)) oip = osn \rightarrow the (dhops (rt (σ sip)) oip) ≤ hops
    show False by simp
    next
    assume "the (flag (rt (σ i)) oip) = inv"
    with oip ∈ vD (rt (σ sip))
    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 "sqn (rt (σ i)) oip = osn"
  also from Suc 0 ≤ osn
  have "Suc hops < osn" by simp
  with nsqn (rt (σ sip)) oip = osn
  have "Suc hops ≤ osn" by simp
  with Suc hops ≤ the (dhops (rt (σ i)) oip)
  show 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 oip ∈ vD (rt (σ sip))
  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 (osn, kno, val, Suc hops, i, π7 (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 oip ∈ iD (rt (σ' sip))
  have False by simp
  from a5: f.distinct(1) 1D_flag_is_inv
  with oip ∈ iD (rt (σ sip))
  have "the (flag (rt (σ sip)) oip) = inv" by auto
  with oip ∈ iD (rt (σ sip))
  have "nsqn (rt (σ sip)) oip < nsqn (rt (σ' sip)) oip"
  unfolding update_def
  by (clarsimp dest: kD_Some)
next
  from 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 show ?thesis using ⟨dip = oip⟩ by simp
    qed
  ultimately show ?thesis
    by (rule rt_strictly_fresher_ltI)
next
  assume osneq: "osn = nsqn (rt (σ' sip)) ∧ 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))" ..
    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 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 (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
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 Γ_{AODV}
  (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]]
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 AODV p"
and pre: "∀ dip. 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
∀ dip ∈ kD (rt (σ i)). sqn (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
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 \( ?\text{nhip} \neq \text{dip} \) show False ..

qed

also have "... = \text{nsqn (rt (\( \sigma \) i)) dip}"
  by (rule \text{vD_nsqn_sqn [OF dip\in\text{vD(rt (\( \sigma \) i))}, THEN sym]})

also have "... \leq \text{nsqn (rt (\( \sigma \) ?nhip)) dip}"
  by (rule nsqns)

also have "... \leq \text{sqn (rt (\( \sigma \) ?nhip)) dip}"
  by (rule nsqn_sqn)

finally have "0 < \text{sqn (rt (\( \sigma \) ?nhip)) dip}".

have "rt (\( \sigma \) i) \sqsubseteq dip rt (\( \sigma \) ?nhip)"
proof (cases "dip\in\text{vD(rt (\( \sigma \) i))}")
  assume "dip\in\text{vD(rt (\( \sigma \) ?nhip))}"
  with pre "dip\in\text{vD(rt (\( \sigma \) i))}" and ?nhip \neq dip
  have "rt (\( \sigma \) i) \sqsubseteq dip 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 ?thesis
  using "dip\in\text{kD(rt (\( \sigma \) ?nhip))}"
  by (rule strictly_fresher_quality_increases_right)

next
  assume "dip\in\text{kD(rt (\( \sigma \) ?nhip))}"
  with "dip\in\text{iD(rt (\( \sigma \) ?nhip))}" have "... = sqn (rt (\( \sigma \) ?nhip)) dip - 1" ..

also from "dip\in\text{iD(rt (\( \sigma \) ?nhip))}"
  have "... = sqn (rt (\( \sigma \) ?nhip)) dip - 1" ..

also have "... \leq sqn (rt (\( \sigma \) ?nhip)) dip"
proof -
  from \( \forall j. \text{quality_increases (\( \sigma \) j) (\( \sigma \) j')} \)
    have "\text{quality_increases (\( \sigma \) ?nhip) (\( \sigma \) ?nhip')}" ..

  hence "\text{sqn (rt (\( \sigma \) ?nhip)) dip} \leq sqn (rt (\( \sigma \) ?nhip)) dip" ..

  with 0 < sqn (rt (\( \sigma \) ?nhip)) dip show ?thesis by auto

qed

also have "... = nsqn (rt (\( \sigma \) ?nhip)) dip"
proof (rule \text{vD_nsqn_sqn [THEN sym]})
  from "dip\in\text{vD(rt (\( \sigma \) ?nhip))}") and \( \sigma \) i = \( \sigma \) i
  show "dip\in\text{vD(rt (\( \sigma \) ?nhip))}" by simp

qed

finally have "nsqn (rt (\( \sigma \) i)) dip < nsqn (rt (\( \sigma \) ?nhip)) dip" ..

moreover from "dip\in\text{vD(rt (\( \sigma \) ?nhip))}) and \( \sigma \) i = \( \sigma \) i
  have "dip\in\text{kD(rt (\( \sigma \) ?nhip))}" by auto

ultimately show "rt (\( \sigma \) i) \sqsubseteq dip rt (\( \sigma \) ?nhip)"
  using "dip\in\text{kD(rt (\( \sigma \) i))}" by - (rule rt_strictly_fresher_ltI)

qed

with \( \sigma \) i = \( \sigma \) i show "rt (\( \sigma \) i) \sqsubseteq dip rt (\( \sigma \) (the (\text{nhop (rt (\( \sigma \) i)) dip})))
  by simp

qed

thus ?thesis unfolding Let_def .

qed

lemma seq_compare_next_hop:
fixes w
shows "\text{opaodv i \sqsubseteq (otherwith (op=) \{i\} (\text{orecmsg msg_fresh}), \text{other quality_increases \{i\} \rightarrow})
  global (\( \lambda \sigma. \forall dip. \text{let nhip = the (\text{nhop (rt (\( \sigma \) i)) dip})
    in dip \in \text{kD(rt (\( \sigma \) i))} \land \text{nhip} \neq dip \rightarrow
dip \in \text{kD(rt (\( \sigma \) nhip))}}
  \land \text{nsqn (rt (\( \sigma \) i)) dip} \leq \text{nsqn (rt (\( \sigma \) nhip)) dip}"
"
by (rule oinvariant_weakenE [OF seq_compare_next_hop']) (auto dest!: onlD)

lemma seq_nhop_quality_increases:
  shows "opadvs i | (otherwith (op=) {i})
    (orevmsg (λ σ 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

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 = (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' 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 "∀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) +)"

end
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 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 nhip) ⊑ 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 } 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:
  \(\text{fixes } \sigma s a \sigma' s' \text{ assumes } (\sigma, s, a, (\sigma', s')) \in \text{oparp_sos i (oseqp_sos } \Gamma_{AODV} i) (\text{seqp_sos } \Gamma_{QM SG})\) 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 \(\text{opaodv i } (\text{seq_nhop_quality_increases})\)
proof (rule lift_into_qmsg [OF seq_nhop_quality_increases])
  \(\text{show } \text{opaodv i } (\text{seq_nhop_quality_increases})\)
proof (rule ostep_invariant_weakenE [OF olocal_quality_increases], simp_all)

next
  fix \(t \cdot ((nat \Rightarrow \text{state}) \times (\text{state, msg, pseqp, pseqp label}) seqp), \text{msg seq_action}) \text{transition}\)
assume \(\text{onll } \Gamma_{AODV} (\lambda ((\sigma, _), _, (\sigma', _)). \text{quality_increases } (\sigma i) (\sigma' i)) t\)
thus \(\text{quality_increases } (\text{fst } (\text{fst } t) i) (\text{fst } (\text{snd } (\text{snd } t)) i)\)
by (cases t) (clarsimp dest!: onllD, metis aodv_ex_label)

next
  fix \(\sigma \sigma' a \text{ assume } \text{otherwith } (\text{op=} \{i\}) (\text{orecvmsg } (\lambda _. \text{rreq_rrep_sn}) \sigma, \text{other } (\lambda _ _. \text{True}) \{i\} \rightarrow)\)
  \(\text{by } (\text{rule weaken_otherwith, auto)}\)
  qed

qed auto

lemma par_rreq_rrep_sn_quality_increases:
  \(\text{opaodv i (rreq_rrep_sn_quality_increases)}\)
proof -
  \(\text{have } \text{opaodv i } (\text{rreq_rrep_sn_quality_increases})\)
by (rule ostep_invariant_weakenE [OF rreq_rrep_sn_quality_increases], simp_all)

hence \(\text{opaodv i (rreq_rrep_sn_quality_increases)}\)
by (rule lift_step_into_qmsg_statelessassm) simp_all
thus \(?\text{thesis by rule auto}\)
qed

lemma par_rreq_rrep_snqsn_fresh_any_step:
  \(\text{opaodv i (rreq_rrep_snqsn_fresh_any_step)}\)
proof -
  \(\text{have } \text{opaodv i (rreq_rrep_snqsn_fresh_any_step)}\)
by (rule ostep_invariant_weakenE [OF rreq_rrep_snqsn_fresh_any_step], simp_all)

fix t
assume "onll Γ\textit{AODV} (λ((σ, _), a, _). anycast (msg\_fresh σ) a) t" 
thus "globala (λ(σ, a, σ'). anycast (msg\_fresh σ) a) t"
by (cases t) (clarsimp dest!: onllD, metis aodv\_ex\_label)
qed

hence "opaodv i ∥ i qmsg :=A (λσ _. orecvmsg (λ_. rreq\_rrep\_sn) σ, other (λ_. True) i) \rightarrow 
globala (λ(σ, a, _). anycast (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) \rightarrow 
globala (λ(σ, a, _). anycast msg\_zhops a)"
proof - 
from anycast\_msg\_zhops initial\_i\_aodv oaodv\_trans aodv\_trans
have "opaodv i ∥ i qmsg :=A (act TT, other (λ_. True) i) \rightarrow 
seqll i (onll Γ\textit{AODV} (λ(_, a, _). anycast msg\_zhops a))"
by (rule open\_seq\_step\_invariant)
hence "opaodv i ∥ i qmsg :=A (λσ _. orecvmsg (λ_. rreq\_rrep\_sn) σ, other (λ_. True) i) \rightarrow 
globala (λ(_, a, _). anycast msg\_zhops a)"
proof (rule ostep\_invariant\_weakenE)
fix t :: "(((nat ⇒ state) × (state, msg, pseqp, pseqp label) seqp), msg seq_action) transition"
assume "seqll i (onll Γ\textit{AODV} (λ(_, a, _). anycast msg\_zhops a)) t"
thus "globala (λ(_, a, _). anycast msg\_zhops a)" by (cases t) (clarsimp dest!: seqllD 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) \rightarrow 
globala (λ(_, a, _). anycast msg\_zhops 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 (oparp\_sos i (oseqp\_sos Γ\textit{AODV} i) (seqp\_sos Γ\textit{QM SG}))"
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) ⇒ 
(\{i\} \rightarrow 
  (oparps\_sos i (oseqps\_sos Γ\textit{AODV} i) (seqps\_sos Γ\textit{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\_quality\_increases:
shows "(i : opaodv i ∥ i qmsg := R) ⇒ 
  (\{i\} \rightarrow 
    global (λσ. ∀dip. let nhip = the (nhop (rt σ i)) dip 
      in dip \in vD (rt σ i) \cap vD (rt σ nhip) \land nhip ≠ dip 
      → (rt σ i) \sqsubseteq dip (rt σ nhip)))"
by (rule node\_lift [OF par\_nhop\_quality\_increases]) auto

lemma node\_rreq\_rrep\_nsqn\_fresh\_any\_step:
shows "(i : opaodv i ∥ i qmsg := R) ⇒ 
(\{i\} \rightarrow 
  global (λσ. ∀dip. let nhip = the (nhop (rt σ i)) dip 
    in dip \in vD (rt σ i) \cap vD (rt σ nhip) \land nhip ≠ dip 
    → (rt σ i) \sqsubseteq dip (rt σ nhip)))"
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) = A
(λσ _, oarrivemsg (λ._ rreq_rrep_sn) σ), other (λ_ _. True) {i} \rightarrow
)globala (λ(λ_, 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) = A
(λσ _, oarrivemsg (λ._ rreq_rrep_sn) σ), other (λ_ _. True) {i} \rightarrow
)globala (λ(λ(λ_, a, σ'). a \neq \tau \rightarrow σ' i = σ i)"
proof (rule ostep_invariantI, simp (no_asm), rule impI)
fix σ \zeta a σ'
assume or: "(σ, ζ) \in ooreachable ((i : opaodv i (i qmsg : R))
(λσ _, oarrivemsg (λ._ rreq_rrep_sn) σ), other (λ_ _. True) {i})" 
and tr: "((σ, ζ), a, (σ', ζ')) \in 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, i, NodeS i p R), a, (σ', ζ'))
\in 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

2.11.3 Lift to partial networks

lemma arrive_rreq_rrep_nsqn_fresh_inc_sn [simp]:
assumes "oarrivemsg (λσ m. 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 : p) = =
(ootherwith (op=) (net_tree_ips p)
(oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
other quality_increases (net_tree_ips p) \rightarrow)
globala (λσ. ∀ i\in net_tree_ips p. ∀ dip.
let nhip = the (nhop (rt (σ i)) dip
in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip \neq dip
\rightarrow (rt (σ i)) ⊏ dip (rt (σ nhip)))"
by (rule pnet_lift [OF node_nhop_quality_increases])
fix i R
have "(i : opaodv i (i qmsg : R) = A
(λσ _, oarrivemsg (λ._ rreq_rrep_sn) σ), other (λ_ _. True) {i} \rightarrow)
)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' s'
assume or: "(σ, s) \in ooreachable ((i : opaodv i (i qmsg : R))
(λσ _, oarrivemsg (λ._ rreq_rrep_sn) σ), other (λ_ _. True) {i})"
and tr: "((σ, s), a, (σ', s')) \in trans ((i : opaodv i (i qmsg : R))"
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_rreq_rrep_nsqn_fresh_any_step])
moreover from or tr am have "castmsg (msg_zhops) 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) = A
(λσ _, oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m) σ,
249
other quality_increases \{i\} \to \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 \colon \text{opaodv} \ i \ (i \ qmsg : \ R)) \models \ _\ 
(\lambda \ _, \_. \ oarrivemsg \ (\lambda \ m. \text{msg\_fresh} \, \sigma \, m \land \text{msg\_zhops} \, m) \, \sigma,
\text{other quality_increases} \{i\} \to \text{globa} \ (\lambda (\sigma, a, \_).
a = \tau \land \exists \ d. \ a \neq i: \text{deliver}(d) \to \sigma \ i = \sigma' \ i)"

by (rule ostep\_invariant\_weakenE [OF node\_silent\_change\_only]) auto

next
fix \ i \ R
show \ "(i \colon \text{opaodv} \ i \ (i \ qmsg : \ R)) \models \ _\ 
(\lambda \ _, \_. \ oarrivemsg \ (\lambda \ m. \text{msg\_fresh} \, \sigma \, m \land \text{msg\_zhops} \, m) \, \sigma,
\text{other quality_increases} \{i\} \to \text{globa} \ (\lambda (\sigma, a, \sigma').
a = \tau \lor (\exists \ d. \ a = i: \text{deliver}(d) \to \text{quality\_increases}(\sigma \ i) \ (\sigma' \ i))"

by (rule ostep\_invariant\_weakenE [OF node\_quality\_increases]) auto

qed simp_all

2.11.4 Lift to closed networks

lemma onet\_nhop\_quality\_increases:
shows \ "oclosed \ (\text{opnet} \ (\lambda i. \text{opaodv} \ i \ (i \ qmsg) \ p)
\models \ (\lambda \ _, \_. \ True, \text{other quality\_increases}(\text{net\_tree\_ips} \, \ p) \to)
\text{globa} \ (\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)) \land \text{vD}(\text{rt}(\sigma \ nhip)) \land \text{nhip} \neq dip
\to (\text{rt}(\sigma \ i)) \sqsubseteq \ dip \ (\text{rt}(\sigma \ nhip)))"

(is \ "_ \models (_, \ ?U \to) \ ?inv")
proof (rule inclosed\_closed)
from opnet\_nhop\_quality\_increases
show \ "\text{opnet} \ (\lambda i. \text{opaodv} \ i \ (i \ qmsg) \ p)
\models \ (\text{otherwith}(op=) \ (\text{net\_tree\_ips} \, p) \ \text{in\closed}, \ ?U \to) \ ?inv"
proof (rule oinvariant\_weakenE)
fix \ \sigma \ \sigma' :: \ "ip \Rightarrow \ state"
and \ a :: "msg \ node\_action"
assume \ "otherwith(op=) (\text{net\_tree\_ips} \, p) \ \text{in\closed} \ \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 \ a"
proof (cases a)
fix \ ii \ ni \ ms
assume \ "a = ii\ =\ ni: \text{arrive}(ms)"
moreover with \ "\text{in\closed} \ \sigma \ a"
obtain \ d \ di \ where \ "ms = \text{newpkt}(d, \ di)"
by (cases ms) auto
ultimately show \ ?thesis \ by \ simp
qed simp\_all
qed

2.11.5 Transfer into the standard model

interpretation aodv\_openproc: openproc \paodv \ opaodv \ id
where "aodv\_openproc\_initmissing = \text{init\_missing}"
proof -
show \ "\text{openproc} \ \paodv \ \opaodv \ \id"
proof unfold\_locales
fix \ i :: \ ip
have \ "\{\sigma, \zeta. \ (\sigma \ i, \zeta) \in \sigma_{AODV} \land (\forall j. \ j \neq i \to \sigma \ j \in \text{fst} ' \sigma_{AODV} \, j)\} \subseteq \sigma_{AODV}'"
unfolding \sigma_{AODV}\_def \sigma_{AODV}'\_def
proof (rule equalityD1)
show \[\forall p. \{ (\sigma, \zeta) \mid (\sigma i, \zeta) \in \{(f i, p)\} \land (\forall j. j \neq i \rightarrow \sigma j \in \text{fst} \ ' \{(f j, p)\}) \} = \{(f, p)\}\]
by (rule set_eqI) auto
qed
thus \[\{ (\sigma, \zeta) \mid (\sigma i, \zeta) \in \{f p. \{(f i, p)\} \land (\forall j. j \neq i \rightarrow \sigma j \in \text{fst} \ ' \{(f j, p)\}) \} = \{(f, p)\}\]
by simp
next
show \[\forall j. \text{init} \ (\text{paodv} i) \neq \{\}\]
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 \[((\sigma, a, \sigma')) \in \text{trans} \ (\text{paodv} i)\]
then obtain q q'
where \[s = (\sigma i, q)\]
and \[s' = (\sigma' i, q')\]
and \[(((\sigma i, q), a, (\sigma' i, q')) \in \text{trans} \ (\text{paodv} i)\]
by (cases s, cases s') auto
from this(3) have \[(((\sigma, q), a, (\sigma', q')) \in \text{trans} \ (\text{opaodv} i)\]
by simp (rule open_seqp_action \[\text{OF aodv_wf}\])
with \(\langle s = (\sigma i, q) \rangle\) and \(\langle s' = (\sigma' i, q') \rangle\)
show \[(((\sigma, \text{snd} \ (\text{id} s)), a, (\sigma', \text{snd} \ (\text{id} s'))) \in \text{trans} \ (\text{opaodv} i)\]
by simp
qed
then interpret \(\text{op}: \text{openproc} \ \text{paodv} \ \text{opaodv} \ \text{id}\).
have \(\text{simp}\): \[\forall i. (\text{SOME} \ x. x \in \{(\text{fst} \ \text{id} p) \ ' \ \text{init} \ (\text{paodv} i)\} = \text{aodv_init} i\]
unfolding \(\sigma_{AODV\_def}\) by simp
hence \[\forall i. \text{openproc.initmissing} \ \text{paodv} \ \text{id} \ i = \text{initmissing} \ i\]
unfolding \(\text{op.initmissing_def}\) \(\text{op.someinit_def}\) \(\text{initmissing_def}\)
by (auto split: option.split)
thus \[\text{openproc.initmissing} \ \text{paodv} \ \text{id} \ i = \text{initmissing}\] ..
qed
interpretation \(\text{aodv_openproc_par_qmsg}: \text{openproc_parq} \ \text{paodv} \ \text{opaodv} \ \text{id} \ \text{qmsg}\)
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 \(\text{opq}: \text{openproc_parq} \ \text{paodv} \ \text{opaodv} \ \text{id} \ \text{qmsg}\).
have \(\text{imp}\): \[\forall \sigma. \text{openproc.initmissing} \ (\lambda i. \text{paodv} \ i \ ((\text{qmsg}) \ (\lambda(p, q). (\text{fst} \ (\text{id} p), \text{snd} \ (\text{id} p), q))) \ \sigma = \text{initmissing} \ \sigma\]
unfolding \(\text{opq.initmissing_def}\) \(\text{opq.someinit_def}\) \(\text{initmissing_def}\)
unfolding \(\sigma_{AODV\_def}\) \(\sigma_{QMSG\_def}\) by (clarsimp cong: option.case_cong)
thus \"\text{openproc.initmissing} \ (\lambda i. \text{paodv} \ i \ ((\text{qmsg}) \ (\lambda(p, q). (\text{fst} \ (\text{id} p), \text{snd} \ (\text{id} p), q))) \ P \ \sigma = \text{netglobal} \ P \ \sigma\"
unfolding \(\sigma_{QMSG\_def}\)
by (clarsimp cong: option.case_cong)
simp del: \text{One_nat_def}
simp add: \text{fst_initmissing_netgmap_default_aodv_init_netlift}
[symmetric, unfolded \(\text{initmissing_def}\)]
thus \"\text{openproc.netglobal} \ (\lambda i. \text{paodv} \ i \ ((\text{qmsg}) \ (\lambda(p, q). (\text{fst} \ (\text{id} p), \text{snd} \ (\text{id} p), q))) = \text{netglobal}\"
by auto
qed
lemma \text{net_nhop_quality_increases}:
assumes \"\text{wf_net_tree} \ n\"

shows "closed (pnet (λi. paodv i ∥ 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 ∥ 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
  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) |= 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 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 ⇒ 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 "π₆ ≡ λ(_, _, _, _, nhip). nhip"

lemma projs [simp]:
  "π₂(dsn, dsk, flag, hops, nhip) = dsn"
  "π₃(dsn, dsk, flag, hops, nhip) = dsk"
  "π₄(dsn, dsk, flag, hops, nhip) = flag"
  "π₅(dsn, dsk, flag, hops, nhip) = hops"
  "π₆(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 ] ⇒ P (π₃ x)"
  by (rule k.induct)

lemma proj4_pred [intro]: "[ P val; P inv ] ⇒ P (π₄ x)"
  by (rule f.induct)

lemma proj6_pair_snd [simp]:
  fixes dsn' r
  shows "π₆ (dsn', snd (r)) = π₆(r)"
  by (cases r) simp

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

type_synonym rt = "ip → r"

syntax
  "_Sigma_route" :: "rt ⇒ ip → r" ("σ_route(_, _)"")

translations
  "σ_route(rt, dip)" => "rt dip"

definition sqn :: "rt ⇒ ip ⇒ sqn"
  where "sqn rt dip ≡ case σ_route(rt, dip) of Some r ⇒ π₂(r) | None ⇒ 0"

definition sqnf :: "rt ⇒ ip ⇒ k"
  where "sqnf rt dip ≡ case σ_route(rt, dip) of Some r ⇒ π₃(r) | None ⇒ 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))"

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

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

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 \(\text{vD_iD_gives_kD [simp]}:\)
\(\forall rt. ip \in \text{vD rt} \Rightarrow ip \in \text{kD rt}\)
\(\forall rt. ip \in \text{iD rt} \Rightarrow ip \in \text{kD rt}\)
unfolding \(\text{kD_is_vD_and_iD}\) by simp_all

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

lemma \(\text{kD_None [dest]}:\)
fixes dip rt
assumes \(\text{"dip \notin kD rt}\)
shows \(\sigma_{\text{route}(rt, dip)} = \text{None}\)
using assms unfolding \(\text{kD_def}\) by (metis (mono_tags) mem_Collect_eq)

lemma \(\text{vD_Some [dest]}:\)
fixes dip rt
assumes \(\text{"dip \in vD rt}\)
shows \(\exists dsn dsk hops nhip.\)
\(\sigma_{\text{route}(rt, dip)} = \text{Some (dsn, dsk, val, hops, nhip)}\)
using assms unfolding \(\text{vD_def}\) by simp

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

lemma \(\text{iD_Some [dest]}:\)
fixes dip rt
assumes \(\text{"dip \in iD rt}\)
shows \(\exists dsn dsk hops nhip.\)
\(\sigma_{\text{route}(rt, dip)} = \text{Some (dsn, dsk, inv, hops, nhip)}\)
using assms unfolding \(\text{iD_def}\) by simp

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

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

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

lemma kD_but_not_vD_is_iD [elim]:
  fixes ip \text{rt}
  assumes "\( \text{ip} \in \text{kD}(\text{rt}) \)"
  and "\( \text{ip} \not\in \text{vD}(\text{rt}) \)"
  shows "\( \text{ip} \in \text{iD}(\text{rt}) \)"
proof -
  from \( \langle \text{ip} \in \text{kD}(\text{rt}) \rangle \) obtain dsn dsk f hops nhop
  where rtip: "\( \text{rt} \text{ip} = \text{Some (dsn, dsk, f, hops, nhop)} \)"
  by (metis kD_Some)
  from \( \langle \text{ip} \not\in \text{vD}(\text{rt}) \rangle \) 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 \( \langle \text{ip} \in \text{kD}(\text{rt}) \rangle \) show "\( \text{ip} \in \text{vD}(\text{rt}) \)" ..
    qed
  with rtip have "\( \text{the (flag rt ip)} = \text{inv} \)"
  by (simp)
  with \( \langle \text{ip} \in \text{kD}(\text{rt}) \rangle \) show "\( \text{ip} \in \text{iD}(\text{rt}) \)" ..
  qed

lemma vD_or_iD [elim]:
  fixes ip \text{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 \( \langle \text{ip} \in \text{kD}(\text{rt}) \rangle \) 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 proj5_eq_dhops: "\( \forall \text{dip rt}. \text{dip} \in \text{kD}(\text{rt}) \Rightarrow \pi_5(\text{the (rt dip)}) = \text{the (dhops rt dip)} \)"
unfolding sqn_def by (drule kD_Some) clarsimp

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

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

lemma kD_sqnf_is_proj3 [simp]:
  "\( \forall \text{ip rt}. \text{ip} \in \text{kD}(\text{rt}) \Rightarrow \text{sqnf rt ip} = \pi_3(\text{the (rt ip)}) \)"
unfolding sqnf_def by auto

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

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

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

lemma 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 "\( [ \text{ip} \neq \text{ip}'; \text{ip} \in \text{kD}(\text{rt})] \Rightarrow \text{P rt ip ip}' \)"
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 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 \rightarrow (hops = 1)"

unfolding update_arg_wf_def by auto
have "P (dsn, dsk, flg, 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 \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
  \[ [ \neg \in kD(rt) ] \] \implies update rt ip r = rt (ip \mapsto r)"
  \[ [ \in kD(rt); sqn rt ip < \pi_2(r) ] \] \implies update rt ip r = rt (ip \mapsto r)"
  \[ [ \in kD(rt); sqn rt ip = \pi_2(r);
      \text{the (dhops rt ip)} > \pi_5(r) ] \] \implies update rt ip r = rt (ip \mapsto r)"
  \[ [ \in kD(rt); sqn rt ip = \pi_2(r);
      \text{flag rt ip} = \text{Some inv} ] \] \implies update rt ip r = rt (ip \mapsto r)"
  \[ [ \in kD(rt); \pi_3(r) = unk; (\pi_2(r) = 0) = (\pi_3(r) = unk) ] \implies update rt ip r = rt (ip \mapsto nr')"\n  \[ [ \in kD(rt); sqn rt ip \geq \pi_2(r); \pi_3(r) = kno;
      \text{sqn rt ip} = \pi_2(r) \implies \text{the (dhops rt ip)} \leq \pi_5(r) \land \text{the (flag rt ip)} = \text{val} ] \] \implies update rt ip r = rt (ip \mapsto s)"

proof -
  assume "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 "ip \in kD(rt)"
  and "sqn rt ip < \pi_2(r)"
  from this(1) obtain dsn dsk flg fl hops nhip
    where "rt ip = Some (dsn, dsk, fl, hops, nhip)"
    by (metis kD_Some)
  with (sqn rt ip < \pi_2(r)) show "update rt ip r = rt (ip \mapsto r)"
  unfolding update_def s_def by auto
next
  assume "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 (sqn rt ip = \pi_2(r)) and (the (dhops rt ip) > \pi_5(r))
show "update \(rt\) \(ip\) \(r\) = \(rt\) (\(ip\) \(\mapsto\) \(r\))"

unfolding \(\text{update}\_\text{def}\) \(s\_\text{def}\) by auto

next

assume "\(ip \in kD(rt)\)"
and "sqn \(rt\) \(ip\) = \(\pi_2(r)\)"
and "flag \(rt\) \(ip\) = Some inv"

from \(\text{this}(1)\) obtain \(dsn\) \(dsk\) \(fl\) \(hops\) \(nhip\)
where "\(rt\) \(ip\) = Some (\(dsn\), \(dsk\), \(fl\), \(hops\), \(nhip\))"
by (metis \(kD\_Some\))

with "\(\text{sqn } rt \ ip = \pi_2(r)\)" and "\(\text{flag } rt \ ip = \text{Some } inv\)"
show "update \(rt\) \(ip\) \(r\) = \(rt\) (\(ip\) \(\mapsto\) \(r\))"
unfolding \(\text{update}\_\text{def}\) \(s\_\text{def}\) by auto

next

assume "\(ip \in kD(rt)\)"
and "\(\pi_3(r) = unk\)"
and "\((\pi_2(r) = 0) = (\pi_3(r) = unk)\)"

from \(\text{this}(1)\) obtain \(dsn\) \(dsk\) \(fl\) \(hops\) \(nhip\)
where "\(rt\) \(ip\) = Some (\(dsn\), \(dsk\), \(fl\), \(hops\), \(nhip\))"
by (metis \(kD\_Some\))

with "\((\pi_2(r) = 0) = (\pi_3(r) = unk)\)" and "\(\pi_3(r) = unk\)"
show "update \(rt\) \(ip\) \(r\) = \(rt\) (\(ip\) \(\mapsto\) \(nr\))"
unfolding \(\text{update}\_\text{def}\) \(nr'\_\text{def}\) \(s\_\text{def}\)
by (cases \(r\)) simp

next

assume "\(ip \in kD(rt)\)"
and "\(\pi_3(r) = unk\)"
and "\((\pi_2(r) = 0) = (\pi_3(r) = unk)\)"

from \(\text{this}(1)\) obtain \(dsn\) \(dsk\) \(fl\) \(hops\) \(nhip\)
where "\(rt\) \(ip\) = Some (\(dsn\), \(dsk\), \(fl\), \(hops\), \(nhip\))"
by (metis \(kD\_Some\))

with otherassms: "\(\text{sqn } rt \ ip \geq \pi_2(r)\)"
and "\(\pi_3(r) = kno\)"
and "\(\pi_3(r) = \text{unk} \Rightarrow \text{the (dhops rt ip)} \leq \pi_5(r) \land \text{the (flag rt ip)} = \text{val}\)"

from \(\text{this}(1)\) obtain \(dsn\) \(dsk\) \(fl\) \(hops\) \(nhip\)
where "\(rt\) \(ip\) = Some (\(dsn\), \(dsk\), \(fl\), \(hops\), \(nhip\))"
by (metis \(kD\_Some\))

with otherassms show "update \(rt\) \(ip\) \(r\) = \(rt\) (\(ip\) \(\mapsto\) \(s\))"
unfolding \(\text{update}\_\text{def}\) \(s\_\text{def}\) by auto

qed

lemma \(\text{update}\_cases\) [elim]:

assumes "\((\pi_2(r) = 0) = (\pi_3(r) = unk)\)"
and \(c1\): "\([[ip \notin kD(rt)] \Rightarrow P (rt (ip \mapsto \(r\)))\]"

and \(c2\): "\([[ip \in kD(rt); \text{sqn } rt \ ip \geq \pi_2(r)]\]
\[\Rightarrow P (rt (ip \mapsto \(r\)))]"

and \(c3\): "\([[ip \in kD(rt); \text{sqn } rt \ ip = \pi_2(r); \text{the (dhops rt ip)} \geq \pi_5(r)]\]
\[\Rightarrow P (rt (ip \mapsto \(r\)))]"

and \(c4\): "\([[ip \in kD(rt); \text{sqn } rt \ ip = \pi_2(r); \text{the (flag rt ip)} = \text{inv}]]
\[\Rightarrow P (rt (ip \mapsto \(r\)))]"

and \(c5\): "\([[ip \in kD(rt); \pi_3(r) = unk]]
\[\Rightarrow P (rt (ip \mapsto (\pi_3(\text{the } \sigma_{\text{route}}(rt, ip)), \pi_3(r),
\pi_4(r), \pi_5(r), \pi_6(r)))))]"

and \(c6\): "\([[ip \in kD(rt); sqn \ ip \geq \pi_2(r); \pi_3(r) = kno; \text{sqn } rt \ ip = \pi_3(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{the } \sigma_{\text{route}}(rt, ip))))""

shows "\((P (\text{update } rt \ ip \ r)))"

proof (cases "\(ip \in kD(rt)\")"

assume "\(ip \notin kD(rt)\)"

with \(c1\) show \(?\)thesis
by simp

next

assume "\(ip \in kD(rt)\)"

moreover then obtain \(dsn\) \(dsk\) \(fl\) \(hops\) \(nhip\)
where \(\text{rteq: } \text{"\(rt\) } \text{ip} = \text{Some (\(dsn\), \(dsk\), \(fl\), \(hops\), \(nhip\))}"
by (metis \(kD\_Some\))

moreover obtain \(dsn'\) \(dsk'\) \(fl'\) \(hops'\) \(nhip'\)
where \(\text{req: } \text{"\(r = (dsn', dsk', fl', hops', nhip')\)}"
by (cases \(r\)) metis
ultimately show \( \text{?thesis} \)

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)"
c2: "sqn rt ip < \pi_2(r) = \Rightarrow P (rt (ip \mapsto r))"
c3: "[sqn rt ip = \pi_2(r); the (dhops rt ip) > \pi_5(r)]
\Rightarrow P (rt (ip \mapsto r))"
c4: "[sqn rt ip = \pi_2(r); the (flag rt ip) = inv]
\Rightarrow P (rt (ip \mapsto r))"
c5: "\pi_3(r) = \text{unk} \Rightarrow P (rt (ip \mapsto (\pi_2(\sigma_{route}(rt, ip)), \pi_3(r),
\pi_4(r), \pi_5(r), \pi_6(r)))))"
c6: "[sqn rt ip \geq \pi_2(r); \pi_3(r) = \text{kno};
sqn rt ip = \pi_2(r) \Rightarrow the (dhops rt ip) \leq \pi_5(r) \wedge the (flag rt ip) = \text{val}]
\Rightarrow P (rt (ip \mapsto 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) \Rightarrow the (dhops rt ip) \leq \pi_5(r) \wedge the (flag rt ip) = \text{val}"
thus "P (rt (ip \mapsto the (rt ip))))"
by (rule c6)

qed (simp add: \(<ip \in kD(rt)>\))

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 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: "\(\forall v. rt dip = Some v \implies update rt dip (dsn, dsk, flag, hops, nhip)
\neq rt dip \mapsto (the (rt dip))\)"

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

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
  assumes "(dsn = 0) = (dsk = unk)"
  shows "sqn rt dip ≤ sqn (update rt rip (dsn, dsk, val, hops, nhip)) dip"
  proof (rule update_cases)
    show "(π3 (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 sqn_def
  by (clarsimp split: option.split) auto

lemma dhops_update [intro]:
  fixes rt dsn dsk flag hops ip rip nhip
  assumes \("\forall ip∈kD rt. \text{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 ds n ds k fl g h o p s n h i p
  assumes "ip ≠ dip"
  shows "dhops (update rt dip (ds n, ds k, fl g, h o p s, n h i p)) ip = dhops rt ip"
  using asms unfolding update_def
  by (clarsimp split: option.split)

lemma sqn_update_same [simp]:
  "∀ rt ip ds n ds k fl g h o p s n h i p. sqn (rt(ip ↦ v)) ip = π_2(v)"
  unfolding sqn_def by simp

lemma dhops_update_changed [simp]:
  fixes rt dip ds n ds k fl g h o p s n h i p
  assumes "rt ≠ update rt dip (ds n, ds k, fl g, h o p s, n h i p)"
  shows "the (dhops (update rt dip (ds n, ds k, fl g, h o p s, n h i p)) dip) = hops"
  using asms unfolding update_def
  by (clarsimp split: option.split_asm option.split split_if_asm) auto

lemma nhop_update_unk_val [simp]:
  "∀ rt dip ds n ds k fl g h o p s ip. the (nhop (update rt dip (ds n, unk, fl g, h o p s, ip)) dip) = ip"
  unfolding update_def by (clarsimp split: option.split)

lemma nhop_update_changed [simp]:
  fixes rt dip ds n ds k fl g h o p s sip
  assumes "update rt dip (ds n, ds k, fl g, h o p s, sip) ≠ rt"
  shows "the (nhop (update rt dip (ds n, ds k, fl g, h o p s, sip)) dip) = sip"
  using asms unfolding update_def
  by (clarsimp split: option.splits split_if_asm) auto

lemma update_rt_split_asm:
  "∀ rt ip ds n ds k fl g h o p s sip. P (update rt ip (ds n, ds k, fl g, h o p s, sip))
  = (∼(rt = update rt ip (ds n, ds k, fl g, h o p s, sip) ∧ ∼P rt
  ∨ rt ≠ update rt ip (ds n, ds k, fl g, h o p s, sip)
  ∧ ∼P (update rt ip (ds n, ds k, fl g, h o p s, sip))))"
  by auto

lemma sqn_update [simp]: "∀ rt dip ds n ds k fl g h o p s sip. rt ≠ update rt dip (ds n, ds k, fl g, h o p s, sip)
  ⇒ sqn (update rt dip (ds n, ds k, fl g, h o p s, sip)) dip = ds n"
  unfolding update_def by (clarsimp split: option.split split_if_asm) auto

lemma sqnf_update [simp]: "∀ rt dip ds n ds k fl g h o p s sip.
  rt ≠ update rt dip (ds n, ds k, fl g, h o p s, sip)
  ⇒ sqnf (update rt dip (ds n, ds k, fl g, h o p s, sip)) dip = ds k"
  unfolding sqnf_def
  by (clarsimp split: option.splits split_if_asm) auto

lemma update_kno_dsn_greater_zero:
  "∀ rt dip ip ds n h o p s. 1 ≤ ds n ⇒ 1 ≤ (sqn (update rt dip (ds n, kno, val, h o p s, ip)) dip)"
  unfolding update_def
  by (clarsimp split: option.splits)

lemma proj3_update [simp]: "∀ rt dip ds n ds k fl g h o p s sip.
  rt ≠ update rt dip (ds n, ds k, fl g, h o p s, sip)
  ⇒ π_3 (the (update rt dip (ds n, ds k, fl g, h o p s, sip)) dip) = ds k"
  unfolding update_def sqnf_def
  by (clarsimp split: option.splits split_if_asm) auto

lemma nhop_update_changed_kno_val [simp]: "∀ rt dip ds n ds k fl g h o p s n h i p.
  rt ≠ update rt ip (ds n, kno, val, h o p s, n h i p)
  ⇒ the (nhop (update rt ip (ds n, kno, val, h o p s, n h i p)) ip) = n h i p"
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
  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.
  dip ∈ vD(update rt dip’ (dsn, dsk, val, 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 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)

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 pre 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 ξ) (dests ξ)) dip = sqn (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 nhop_def 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:
  "∀ 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
  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 ∈ 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)

3.1.5 Route Requests

Generate a fresh route request identifier.

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

3.1.6 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]))"

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 \[\sigma_p\text{-flag} :: \text{store} \Rightarrow \text{ip} \mapsto p\] ("\sigma_p\text{-flag}'(_, _)"")
where \[\sigma_p\text{-flag}(\text{store}, \text{dip}) \equiv \text{map\_option} \ \text{fst} \ (\text{store} \ \text{dip})\]

definition \[\text{unsetRRF} :: \text{store} \Rightarrow \text{ip} \Rightarrow \text{store}\]
where \[\text{unsetRRF} \ \text{store} \ \text{dip} \equiv \text{case} \ \text{store} \ \text{dip} \ \text{of} \]
\[\begin{array}{l}
\text{None} \Rightarrow \text{store} \\
| \text{Some} \ (p, q) \Rightarrow \text{store} \ (\text{dip} \mapsto \text{(noreq, q)})
\end{array}\]

definition \[\text{setRRF} :: \text{store} \Rightarrow (\text{ip} \mapsto \text{sqn}) \Rightarrow \text{store}\]
where \[\text{setRRF} \ \text{store} \ \text{dests} \equiv \lambda \ \text{dip}. \ \text{if} \ \text{dests} \ \text{dip} = \text{None} \ \text{then} \ \text{store} \ \text{dip} \ \text{else} \ \text{map\_option} \ (\lambda (_, q). \ \text{(req, q)}) \ (\text{store} \ \text{dip})\]

### 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. \(\sigma_p\text{-flag}(\text{store}, \text{dip})\) 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

### 3.2 AODV protocol messages

theory \textit{C_Aodv_Message}
imports \textit{C_Gtobcast}
begin

datatype \textit{msg} =
\begin{array}{l}
\text{Rreq} \ \text{nat} \ \text{rreqid} \ \text{ip} \ \text{sqn} \ \text{k} \ \text{ip} \ \text{sqn} \ \text{ip} \\
| \text{Rrep} \ \text{nat} \ \text{ip} \ \text{sqn} \ \text{ip} \\
| \text{Rerr} \ "\text{ip} \Rightarrow \text{sqn}" \ \text{ip} \\
| \text{Newpkt} \ \text{data} \ \text{ip} \\
| \text{Pkt} \ \text{data} \ \text{ip} \ \text{ip}
\end{array}

instantiation \textit{msg} :: \textit{msg}
begin

definition \text{newpkt\_def} \ [\text{simpl}]: "\text{newpkt} \equiv \lambda (d, dip) \ . \ \text{Newpkt} \ d \ dip"

definition \text{eq\_newpkt\_def} : "\text{eq\_newpkt} \ m \equiv \text{case} \ m \ \text{of} \ \text{Newpkt} \ d \ dip \Rightarrow \text{True} \ | \ _ \Rightarrow \text{False}"

instance by intro\_classes (\text{simpl add: eq\_newpkt\_def})
end

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.

definition \[\text{rreq} :: \text{nat} \times \text{rreqid} \times \text{ip} \times \text{sqn} \times \text{k} \times \text{ip} \times \text{sqn} \times \text{ip} \Rightarrow \text{msg}\]
where \[\text{rreq} \equiv \lambda (\text{hops}, \text{rreqid}, \text{dip}, \text{dsn}, \text{dsk}, \text{oip}, \text{osn}, \text{sip}). \]
\[\text{Rreq} \ \text{hops} \ \text{rreqid} \ \text{dip} \ \text{dsn} \ \text{dsk} \ \text{oip} \ \text{osn} \ \text{sip}\]

lemma \text{rreq\_simp} \ [\text{simpl}]:
\[\text{rreq}(\text{hops}, \text{rreqid}, \text{dip}, \text{dsn}, \text{dsk}, \text{oip}, \text{osn}, \text{sip}) = \text{Rreq} \ \text{hops} \ \text{rreqid} \ \text{dip} \ \text{dsn} \ \text{dsk} \ \text{oip} \ \text{osn} \ \text{sip}\]
unfolding \text{rreq\_def} by \text{simpl}
definition \texttt{rrep} :: \texttt{nat} $\times$ \texttt{ip} $\times$ \texttt{sqn} $\times$ \texttt{ip} $\Rightarrow$ \texttt{msg}
\[
\begin{align*}
\text{where } \texttt{rrep} &\equiv \lambda (\text{hops}, \text{dip}, \text{dsn}, \text{oip}, \text{sip}). \text{Rrep hops dip dsn oip sip} \\
\end{align*}
\]

lemma \texttt{rrep_simp} [simp]:
\[
\text{rrep(hops, dip, dsn, oip, sip) = Rrep hops dip dsn oip sip}
\]
unfolding \texttt{rrep_def} by simp

definition \texttt{rerr} :: \texttt{(ip $\rightarrow$ sqn) $\times$ ip} $\Rightarrow$ \texttt{msg}
\[
\begin{align*}
\text{where } \texttt{rerr} &\equiv \lambda (\text{dests}, \text{sip}). \text{Rerr dests sip} \\
\end{align*}
\]

lemma \texttt{rerr_simp} [simp]:
\[
\text{rerr(dests, sip) = Rerr dests sip}
\]
unfolding \texttt{rerr_def} by simp

lemma \texttt{not_eq_newpkt_rreq} [simp]:
\[
\neg \text{eq_newpkt (Rreq hops rreqid dip dsn dsk oip osn sip)}
\]
unfolding \texttt{eq_newpkt_def} by simp

lemma \texttt{not_eq_newpkt_rrep} [simp]:
\[
\neg \text{eq_newpkt (Rrep hops dip dsn oip sip)}
\]
unfolding \texttt{eq_new pkt_def} by simp

lemma \texttt{not_eq_newpkt_rerr} [simp]:
\[
\neg \text{eq_newpkt (Rerr dests sip)}
\]
unfolding \texttt{eq_newpkt_def} by simp

lemma \texttt{not_eq_newpkt_pkt} [simp]:
\[
\neg \text{eq_newpkt (Pkt d dip sip)}
\]
unfolding \texttt{eq_newpkt_def} by simp

definition \texttt{pkt} :: \texttt{data} $\times$ \texttt{ip} $\times$ \texttt{ip} $\Rightarrow$ \texttt{msg}
\[
\begin{align*}
\text{where } \texttt{pkt} &\equiv \lambda (\text{d, dip, sip}). \text{Pkt d dip sip} \\
\end{align*}
\]

lemma \texttt{pkt_simp} [simp]:
\[
\text{pkt(d, dip, sip) = Pkt d dip sip}
\]
unfolding \texttt{pkt_def} by simp

end

3.3 The AODV protocol

theory \texttt{C_Aodv}
imports \texttt{C_Aodv_Data C_Aodv_Message}
\texttt{"../../../AWN/AWN_SOS_Labels" "../../../AWN/AWN_Invariants"}
begin

3.3.1 Data state

record \texttt{state} =
\[
\begin{align*}
\text{ip} &\colon \text{"ip"} \\
\text{sn} &\colon \text{"sqn"} \\
\text{rt} &\colon \text{"rt"} \\
\text{rreqs} &\colon \text{(ip $\times$ rreqid) set} \\
\text{store} &\colon \text{"store"} \\
\text{msg} &\colon \text{"msg"} \\
\text{data} &\colon \text{"data"} \\
\text{dests} &\colon \text{(ip $\rightarrow$ sqn)} \\
\text{rreqid} &\colon \text{"rreqid"} \\
\text{dip} &\colon \text{"ip"} \\
\text{oip} &\colon \text{"ip"} \\
\text{hops} &\colon \text{"nat"} \\
\text{dsn} &\colon \text{"sqn"} \\
\text{dsk} &\colon \text{"k"} \\
\text{osn} &\colon \text{"sqn"} \\
\text{sip} &\colon \text{"ip"} \\
\end{align*}
\]

267
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
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' ∧
ξ' = (ξ | 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 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_rrep ξ ⟹ ip ξ' = ip ξ"
"ξ' ∈ is_rreq ξ ⟹ ip ξ' = ip ξ"
"ξ' ∈ is_rerr 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_rrep ξ ⟹ sn ξ' = sn ξ"
"ξ' ∈ is_rreq ξ ⟹ sn ξ' = sn ξ"
"ξ' ∈ is_rerr 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_rrep ξ ⟹ rt ξ' = rt ξ"

269
"\( \xi' \in \text{is_rrep} \xi \implies \text{rt} \xi' = \text{rt} \xi '' \)
"\( \xi' \in \text{is_rreq} \xi \implies \text{rt} \xi' = \text{rt} \xi '' \)
"\( \xi' \in \text{is_pkt} \xi \implies \text{rt} \xi' = \text{rt} \xi '' \)
"\( \xi' \in \text{is_newpkt} \xi \implies \text{rt} \xi' = \text{rt} \xi '' \)

unfolding \text{is_msg_defs}
by (cases "msg \xi ", clarsimp+)+

lemma \text{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 \text{is_msg_defs}
by (cases "msg \xi ", clarsimp+)+

lemma \text{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 \text{is_msg_defs}
by (cases "msg \xi ", clarsimp+)+

lemma \text{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 \text{is_msg_defs}
by (cases "msg \xi ", clarsimp+)+

3.3.3 The protocol process

datatype pseqp =
| PAodv
| PNewPkt
| PPkt
| PRreq
| PRrep
| PRerr

fun nat_of_seqp :: "pseqp \Rightarrow \text{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]: "\( 11 \leq 12 = (\text{nat_of_seqp} 11 \leq \text{nat_of_seqp} 12) \)"

definition less_seqp [iff]: "\( 11 < 12 = (\text{nat_of_seqp} 11 < \text{nat_of_seqp} 12) \)"

instance ..
end

abbreviation AODV
where
"\text{AODV} \equiv \lambda _. \ [\text{clearlocals}] \ \text{call}(\text{PAodv})"

abbreviation PKT
where
"\text{PKT args} \equiv"
let (data, dip, oip) = args
(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')().
① (is_newpkt) NEWPKT(λξ. (data ξ, dip ξ))
② (is_pkt) PKT(λξ. (data ξ, dip ξ, oip ξ))
③ (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))
④ (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))
⑤ (is_rerr)
[ξ. ξ () rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ)]
RERR(λξ. (dests ξ, sip ξ))
)
⑥ (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)).
⑦ (λξ. { 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) ]
⑧ (λξ. { ξ () rt := invalidate (rt ξ) (dests ξ) }]
⑨ (λξ. { ξ () store := setRRF (store ξ) (dests ξ) }]
broadcast(λξ. rerr(dests ξ, ip ξ)). AODV()
⑩ (λξ. { ξ () dip := dip } | dip. dip ∈ qD(store ξ) − vD(rt ξ) ∧ the (σ_p-flag(store ξ, dip)) = req )
[ξ. ξ () store := unsetRRF (store ξ) (dip ξ) ]]
\[ \gamma \text{AODV} \text{PNewPkt} = \text{labelled PNewPkt} (\xi) \]
\[ \text{deliver}(\lambda \xi. \text{data } \xi).\text{AODV}\(\)\]
\[ \text{broadcast}(\lambda \xi. \text{rreq(0, rreq } \xi, \text{ dip } \xi, \text{ sqn } (r t \xi) (\text{dip } \xi), \text{ sqnf } (r t \xi) (\text{dip } \xi), \text{ ip } \xi, \text{ sn } \xi, \text{ ip } \xi)).\text{AODV}(\)\]

/ "\text{\gamma AODV}\ P\text{Pkt} = \text{labelled PPkt} (\xi) \]
\[ \text{deliver}(\lambda \xi. \text{data } \xi).\text{AODV}\(\)\]
\[ \text{broadcast}(\lambda \xi. \text{rerr(dests } \xi, \text{ ip } \xi)).\text{AODV}\(\)\]

/ "\text{\gamma AODV} \ PR\text{req} = \text{labelled PRreq} (\xi) \]
\[ \text{broadcast}(\lambda \xi. \text{rreq(0, rreq } \xi, \text{ dip } \xi, \text{ sqn } (r t \xi) (\text{dip } \xi), \text{ sqnf } (r t \xi) (\text{dip } \xi), \text{ ip } \xi, \text{ sn } \xi, \text{ ip } \xi)).\text{AODV}(\)\]

\[ [\xi. \xi \circ \text{sn} := \text{inc } (\text{sn } \xi)]\]
\[ [\xi. \xi \circ \text{rreqid} := \text{nrreqid } (\text{rreqs } \xi) (\text{ip } \xi)]\]
\[ [\xi. \xi \circ \text{rreqs} := \text{rreqs } \xi \cup \{(\text{ip } \xi, \text{rreqid } \xi)\}]\]
\[ \text{broadcast}(\lambda \xi. \text{rreq(0, rreq } \xi, \text{ dip } \xi, \text{ sqn } (r t \xi) (\text{dip } \xi), \text{ sqnf } (r t \xi) (\text{dip } \xi), \text{ ip } \xi, \text{ sn } \xi, \text{ ip } \xi)).\text{AODV}(\)\]
broadcast(\(\lambda \xi. \text{rreq}(\text{hops} \, \xi + 1, \text{rreqid} \, \xi, \text{dip} \, \xi, \max (\text{sqn} (\text{rt} \, \xi) (\text{dip} \, \xi)), \text{dsn} \, \xi, \text{oip} \, \xi, \text{osn} \, \xi, \text{ip} \, \xi))\).

\[\text{AODV()}\]

"\text{\(\Gamma_{\text{AODV}}\) PRrep = labelled PRrep (}
\langle \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) \rangle
\}
\}

\[\langle \xi. \xi \mid \text{rt} := \text{update} (\text{rt} \, \xi) (\text{dip} \, \xi) (\text{dsn} \, \xi, \text{kno}, \text{val}, \text{hops} \, \xi + 1, \text{sip} \, \xi) \rangle \]
\[\text{AODV()\}
\]
\[\oplus \langle \xi. \text{oip} \, \xi \neq \text{ip} \, \xi \rangle\]
\[\text{AODV()\}
\]
\[\langle \xi. \text{oip} \, \xi \in vD (\text{rt} \, \xi)\]
\[\text{unicast}(\lambda \xi. \text{the} (\text{nhop} (\text{rt} \, \xi) (\text{oip} \, \xi)), \lambda \xi. \text{rrep}(\text{hops} \, \xi + 1, \text{dip} \, \xi, \text{dsn} \, \xi, \text{oip} \, \xi, \text{ip} \, \xi)). \text{AODV()}\]
\[\text{AODV()}\]
\[\rangle\]
\[\rangle\]
\[\rangle\]
\[\text{\(\oplus \langle \xi. \text{oip} \, \xi \notin vD (\text{rt} \, \xi)\rangle\}
\]
\[\text{AODV()}\]

"\text{\(\Gamma_{\text{AODV}}\) PRerr = labelled PRerr (}
\langle \xi. \text{dests} := (\lambda \text{rip}. \text{if} (\text{rip} \in vD (\text{rt} \, \xi) \land \text{nhop} (\text{rt} \, \xi) \text{rip} = \text{nhop} (\text{rt} \, \xi) (\text{oip} \, \xi))
\text{then Some (inc (\text{sqn} (\text{rt} \, \xi) \text{rip})) else None)} \rangle
\}
\]
\[\langle \xi. \xi \mid \text{rt} := \text{invalidate} (\text{rt} \, \xi) (\text{dests} \, \xi) \rangle\]
\[\langle \xi. \xi \mid \text{store} := \text{setRRF} (\text{store} \, \xi) (\text{dests} \, \xi)\rangle\]
\[\text{unicast}(\lambda \xi. \text{the} (\text{nhop} (\text{rt} \, \xi) (\text{ip} \, \xi)), \lambda \xi. \text{rerr}(\text{hops} \, \xi + 1, \text{dip} \, \xi, \text{dsn} \, \xi, \text{oip} \, \xi, \text{ip} \, \xi)). \text{AODV()}\]
\[\text{AODV()}\]
\[\rangle\]
\[\rangle\]
\[\rangle\]
\[\text{\(\oplus \langle \xi. \text{dests} \, \xi \neq \text{Map.empty}\rangle\}
\]
\[\text{broadcast}(\lambda \xi. \text{rerr}(\text{dests} \, \xi, \text{ip} \, \xi)). \text{AODV()}\]
\[\text{AODV()}\]

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

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

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

declare \( \Gamma_{AODV} \_\text{skeleton}.\text{simps} \) simp del, code del
lemmas \( \Gamma_{AODV} \_\text{skeleton}\_\text{simps} \) simp, code = \( \Gamma_{AODV} \_\text{skeleton}.\text{simps} \) simp del, code del

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 \( \sigma_{AODV} :: \text{ip} \Rightarrow \text{(state \times (state, msg, pseqp, pseqp label) seqp)} \text{ set} \) where "\( \sigma_{AODV} i \equiv \{\text{aodv_init } i, \Gamma_{AODV} \ PAodv\} \)"

abbreviation paodv as "\text{ip} \Rightarrow \text{(state \times (state, msg, pseqp, pseqp label) 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 } (\text{paodv } i) = \text{seqp_sos } \Gamma_{AODV}"
  by simp

lemma aodv_control_within [simp]: "\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 aodv_wf [simp]: "\text{wellformed } \Gamma_{AODV}"
  proof (rule, intro allI)
    fix \( pn \) \( pn' \)
    show "\text{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 "\( \exists p \ l. \ P \ l \ p \ \Rightarrow \ Q\)"
  shows "\Q"
  using assms by (metis aodv_ex_label)

lemma aodv_simple_labels [simp]: "\text{simple_labels } \Gamma_{AODV}"
  proof
    fix \( p \)
    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

274
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 (Rreq hops dip dsn oip sip) = sip"
"∀ dests sip. msg_sender (Rreq dests sip) = sip"
"∀ d dip sip. msg_sender (Rreq 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
lemma msg_zhops_simps [simp]:
"\(\forall hops\ rreqid\ dip\ dsn\ dsk\ oip\ osn\ sip.\)
msg_zhops (Rreq hops rreqid dip dsn dsns oip osn sip) = (hops = 0 \(\longrightarrow\) oip = sip)"
"\(\forall hops\ dip\ dsn\ oip\ sip.\) msg_zhops (Rrep hops dip dsn oip sip) = (hops = 0 \(\longrightarrow\) dip = sip)"
"\(\forall d\ dip.\) msg_zhops (Newpkt d dip) = True"
"\(\forall d\ dip\ sip.\) msg_zhops (Pkt d dip sip) = True"
unfolding msg_zhops_def by simp_all

definition rreq_rrep_sn :: "msg \\Rightarrow\ bool"
where
"rreq_rrep_sn m \equiv \begin{cases} 
Rreq _ _ _ _ _ _ osnc _ \Rightarrow osnc \geq 1 
| Rrep _ _ dsnc _ \Rightarrow dsnc \geq 1 
| _ \Rightarrow True 
\end{cases}"

lemma rreq_rrep_sn_simps [simp]:
"\(\forall hops\ rreqid\ dip\ dsn\ dsk\ oip\ osn\ sip.\)
rreq_rrep_sn (Rreq hops rreqid dip dsn dsk oip osn sip) = (osn \geq 1)"
"\(\forall hops\ dip\ dsn\ oip\ sip.\) rreq_rrep_sn (Rrep hops dip dsn oip sip) = (dsn \geq 1)"
"\(\forall dests\ sip.\) rreq_rrep_sn (Rerr dests sip) = True"
"\(\forall d\ dip.\) rreq_rrep_sn (Newpkt d dip) = 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 \begin{cases} 
Rreq hopsc _ _ _ _ oipc osnc ipcc \Rightarrow (ipcc \neq oipc \longrightarrow 
\begin{array}{l}
\text{oipc} \in kD(crt) \wedge 
\text{sqn} \text{ crt oipc > osnc} \\
\text{\vee (sqn} \text{ crt oipc = osnc} \\
\text{\wedge (dhopsc} \text{ crt oipc} \leq \text{hopsc} \\
\text{\wedge (flag} \text{ crt oipc} = \text{val}))
\end{array}) \\
| Rrep hopsc dip dsnc _ ipcc \Rightarrow (ipcc \neq dip \longrightarrow 
\begin{array}{l}
\text{ipcc} \in kD(crt) \\
\text{sqn crt dip = dsnc} \\
\text{\wedge (dhopsc} \text{ crt dip} = \text{hopsc} \\
\text{\wedge (flag} \text{ crt dip} = \text{val})
\end{array}) \\
| _ \Rightarrow True 
\end{cases}"

lemma rreq_rrep_fresh [simp]:
"\(\forall hops\ rreqid\ dip\ dsn\ dsk\ oip\ osn\ sip.\)
rreq_rrep_fresh crt (Rreq hops rreqid dip dsn dsns oip osn sip) = 
(hopsc \neq oip \longrightarrow oip \in kD(crt) \\
\begin{array}{l}
\text{\vee (sqn} \text{ crt oip > osnc} \\
\text{\wedge (dhopsc} \text{ crt oip} \leq \text{hopsc} \\
\text{\wedge (flag} \text{ crt oip} = \text{val}))
\end{array})" 
"\(\forall hops\ dip\ dsn\ oip\ sip.\) rreq_rrep_fresh crt (Rrep hops dip dsn oip sip) = 
(sip \neq dip \longrightarrow dip \in kD(crt) \\
\begin{array}{l}
\text{\wedge sqn} \text{ crt dip = dsnc} \\
\text{\wedge (dhopsc} \text{ crt dip} = \text{hopsc} \\
\text{\wedge (flag} \text{ crt dip} = \text{val})
\end{array})" 
"\(\forall dests\ sip.\) rreq_rrep_fresh crt (Rerr dests sip) = True"
"\(\forall d\ dip.\) rreq_rrep_fresh crt (Newpkt d dip) = True"
"\(\forall d\ dip\ sip.\) rreq_rrep_fresh crt (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 \begin{cases} 
Rerr destsc _ \Rightarrow (\forall ripc \in dom(destsc). \\
ripc \in kD(crt)) \wedge (\text{destsc} \ ripc = \text{sqn} \text{ crt ripc}) \\
| _ \Rightarrow True 
\end{cases}"

lemma rerr_invalid [simp]:
"\(\forall hops\ rreqid\ dip\ dsn\ dsk\ oip\ osn\ sip.\)
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 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 ≡ (λs. 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 ⇒ sqn"
where
"nsqn_r r ≡ if π_4(r) = val ∨ π_2(r) = 0 then π_2(r) else (π_2(r) - 1)"

lemma nsqn_r_def':
"nsqn_r r = (if π_4(r) = inv then π_2(r) - 1 else π_2(r))"
unfolding nsqn_r_def by simp

lemma nsqn_r_zero [simp]:
"∀dsn dsk flag hops nhip. nsqn_r (0, dsk, flag, hops, nhip) = 0"
unfolding nsqn_r_def by clarsimp

lemma nsqn_r_val [simp]:
"∀dsn dsk hops nhip. nsqn_r (dsn, dsk, val, hops, nhip) = dsn"
unfolding nsqn_r_def by clarsimp
lemma nsqn\textsubscript{\text{r}}\_inv \text{[simp]}: 
"\forall dsn\ dsk\ hops\ nhip. \text{nsqn} (dsn, dsk, inv, hops, nhip) = dsn - 1"
unfolding nsqn\_def by clarsimp

lemma nsqn\textsubscript{\text{r}}\_lte_dsn \text{[simp]}: 
"\forall dsn\ dsk\ flag\ hops\ nhip. \text{nsqn} (dsn, dsk, flag, hops, nhip) \leq dsn"
unfolding nsqn\_def by clarsimp

On routes in routing tables

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

lemma nsqn\textsubscript{\text{r}}\_def: 
"\forall rt\ dip. \text{nsqn}_r rt dip = (if flag rt dip = \text{Some } \text{val} \lor \text{sqn rt dip} = 0 
then \text{sqn rt dip} else \text{sqn rt dip - 1})"
unfolding nsqn\_def sqn\_def by (clarsimp split: option.split) auto

lemma not\_in\_kD\textsubscript{\text{nsqn}} \text{[simp]}:
assumes "dip /\in kD(rt)"
shows "\text{nsqn} rt dip = 0"
using assms unfolding nsqn\_def by simp

lemma kD\_nsqn: 
assumes "dip \in kD(rt)"
shows "\text{nsqn} rt dip = \text{nsqn}_r(\text{the } (\sigma_{\text{route}}(rt, dip)))"
using assms \[THEN kD_Some\] unfolding nsqn\_def by clarsimp

lemma nsqn\textsubscript{\text{r}}\_r\_flag\_pred \text{[simp, intro]}: 
fixes dsn\ dsk\ flag\ hops\ nhip
assumes "P (\text{nsqn}_r (dsn, dsk, \text{val}, hops, nhip))"
and "P (\text{nsqn}_r (dsn, dsk, inv, hops, nhip))"
shows "P (\text{nsqn}_r (dsn, dsk, flag, hops, nhip))"
using assms by (cases flag) auto

lemma sqn\_nsqn: 
"\forall rt\ dip. \text{sqn} rt dip - 1 \leq \text{nsqn} rt dip"
unfolding sqn\_def nsqn\_def by (clarsimp split: option.split)

lemma nsqn\_sqn: "\text{nsqn} rt dip \leq \text{sqn} rt dip"
unfolding sqn\_def nsqn\_def by (cases "rt dip") auto

lemma val\_nsqn\_sqn \text{[elim, simp]}: 
assumes "ip\in\text{kd}(rt)"
and "the (flag rt ip) = \text{val}"
shows "\text{nsqn} rt ip = \text{sqn} rt ip"
using assms unfolding nsqn\_sqn\_def by auto

lemma vD\_nsqn\_sqn \text{[elim, simp]}: 
assumes "ip\in\text{vD}(rt)"
shows "\text{nsqn} rt ip = \text{sqn} rt ip"
proof -
  from \(ip\in\text{vD}(rt)\) have "ip\in\text{kd}(rt)"
  and "the (flag rt ip) = \text{val}" by auto
  thus \?thesis ..
qed

lemma inv\_nsqn\_sqn \text{[elim, simp]}: 
assumes "ip\in\text{kd}(rt)"
and "the (flag rt ip) = \text{inv}"
shows "\text{nsqn} rt ip = \text{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 \(?thesis\) ..
qed

lemma $nsqn_update_changed_kno_val$ [simp]: "\(\forall rt ip dsn dsk hops nhip. rt \neq \text{update rt ip (dsn, kno, val, hops, nhip)} \Rightarrow nsqn (update rt ip (dsn, kno, val, hops, nhip)) ip = dsn\)"
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 $nsqn_update_other$ [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 "\(\text{dests dip} = \text{Some rsn}\)"
shows "\(nsqn (invalidate rt dests) dip = rsn - 1\)"
using assms
proof -
  from assms obtain dsk hops nhip pre
    where "\(\text{invalidate rt dests dip} = \text{Some (rsn, dsk, inv, hops, nhip)}\)"
    unfolding invalidate_def
    by auto
  moreover from \(\text{dip} \in kD(rt)\) have "\(\text{dip} \in kD(\text{invalidate rt dests})\)" by simp
  ultimately show \(?thesis\)
    using \(\text{dests dip} = \text{Some rsn}\) by simp
qed

lemma $nsqn_invalidate_other$ [simp]:
assumes "\(dip \in kD(rt)\)"
  and "\(\text{dip} \notin \text{dom dests}\)"
shows "\(nsqn (invalidate rt dests) dip = nsqn rt dip\)"
using assms
by (clarsimp simp add: kD_nsqn)

3.5.2 Comparing routes

definition
  $fresher :: \ r \Rightarrow \ r \Rightarrow \ \text{bool}$ ("\(_/\ \sqsubseteq\ _/\)" [51, 51] 50)
where
  "\(fresher r r' \equiv ((\text{nsqn, } r < \text{nsqn, } r') \lor (\text{nsqn, } r = \text{nsqn, } r' \land \pi_5(r) \geq \pi_5(r'))))\)"

lemma $fresherI1$ [intro]:
assumes "\(\text{nsqn, } r < \text{nsqn, } r'\)"
  shows "\(r \sqsubseteq r'\)"
unfolding $fresher_def$ using assms by simp

lemma $fresherI2$ [intro]:
assumes "\(\text{nsqn, } r = \text{nsqn, } r'\)"
  and "\(\pi_5(r) \geq \pi_5(r')\)"
  shows "\(r \sqsubseteq r'\)"
unfolding $fresher_def$ using assms by simp

lemma $fresherI$ [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 fresher_def using assms.

lemma fresherE [elim]:
assumes "\( r \sqsubseteq r' \)"
and "\( \text{nsqn}_r r < \text{nsqn}_r r' \implies P \; r \; r' \)"
and "\( \text{nsqn}_r r = \text{nsqn}_r r' \land \pi_5(r) \geq \pi_5(r') \implies P \; r \; r' \)"

shows "\( P \; r \; r' \)"
using assms unfolding fresher_def by auto

lemma fresher_refl [simp]: "\( r \sqsubseteq r \)"
unfolding fresher_def by simp

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

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

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

3.5.3 Comparing routing tables

definition
\( \text{rt} \_ \text{fresher} :: \cdot \text{ip} \Rightarrow \text{rt} \Rightarrow \text{rt} \Rightarrow \text{bool} \cdot \)
where
\( \text{rt} \_ \text{fresher} \equiv \lambda \text{dip} \; \text{rt} \; \text{rt}'. \; \text{(the (\sigma_{\text{route}}(\text{rt}, \text{dip}))) \sqsubseteq \text{(the (\sigma_{\text{route}}(\text{rt}', \text{dip}))}}) \)

abbreviation
\( \text{rt} \_ \text{fresher} \_ \text{syn} :: \cdot \text{rt} \Rightarrow \text{ip} \Rightarrow \text{rt} \Rightarrow \text{bool} \cdot (\cdot / \sqsubseteq \cdot) \cdot [51, 999, 51] 50 \cdot \)
where
\( \text{rt}1 \sqsubseteq i \text{rt}2 \equiv \text{rt} \_ \text{fresher} \; i \; \text{rt}1 \; \text{rt}2 \cdot \)

lemma rt_fresher_def':
\( (\text{rt}_1 \sqsubseteq i \text{rt}_2) = (\text{nsqn}_r (\text{the (rt}_1 i)) < \text{nsqn}_r (\text{the (rt}_2 i)) \lor \text{nsqn}_r (\text{the (rt}_1 i)) = \text{nsqn}_r (\text{the (rt}_2 i)) \land \pi_5 (\text{the (rt}_2 i)) \leq \pi_5 (\text{the (rt}_1 i))) \)
unfolding rt_fresher_def fresher_def by (rule refl)

lemma single_rt_fresher [intro]:
assumes "\( \text{the (rt}_1 \text{ip}) \sqsubseteq \text{the (rt}_2 \text{ip}) \)"
shows "\( \text{rt}_1 \sqsubseteq \text{i} \text{rt}_2 \)"
using assms unfolding rt_fresher_def .

lemma rt_fresher_single [intro]:
assumes "\( \text{rt}_1 \sqsubseteq \text{i} \text{rt}_2 \)"
shows "\( \text{the (rt}_1 \text{ip}) \sqsubseteq \text{the (rt}_2 \text{ip}) \)"
using assms unfolding rt_fresher_def .

lemma rt_fresher_def2:
assumes "\( \text{dip} \in \text{kD(} \text{rt}_1) \)"
and "\( \text{dip} \in \text{kD(} \text{rt}_2) \)"
shows "\( (\text{rt}_1 \sqsubseteq \text{dip \; rt}_2) = (\text{nsqn} \; \text{rt}_1 \; \text{dip} < \text{nsqn} \; \text{rt}_2 \; \text{dip} \lor (\text{nsqn} \; \text{rt}_1 \; \text{dip} = \text{nsqn} \; \text{rt}_2 \; \text{dip} \land \text{the (dhops} \; \text{rt}_1 \; \text{dip}) \geq \text{the (dhops} \; \text{rt}_2 \; \text{dip})) \)"
using assms unfolding rt_fresher_def fresher_def by (simp add: kD_nsqn proj5_eq_dhops)

lemma rt_fresherI1 [intro]:
assumes "\( \text{dip} \in \text{kD(} \text{rt}_1) \)"

\( \)
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 ] ⇒ 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 "rt ≈ 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)"
  shows "rt1 ⊑ dip rt2"
  unfolding rt_fresher_def2 [OF assms(1-2)] using assms(3) by simp
and "the (rt1 dip) ⊑ the (rt2 dip)"
and "the (rt2 dip) ⊑ the (rt1 dip)"
sows "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 "π_5(the (rt dip)) = π_5(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]:

282
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_nsqr:
  assumes "dip ∈ kD(rt1)"
  and "dip ∈ kD(rt2)"
  and "rt1 ≈_dip rt2"
  shows "nsqnr (the (rt2 dip)) = nsqnr (the (rt1 dip))"
using assms(3)
unfolding rt_fresh_as_def
by (auto simp: rt_fresher_def2 [OF ⟨dip ∈ kD(rt1)⟩ ⟨dip ∈ kD(rt2)⟩]
  rt_fresher_def2 [OF ⟨dip ∈ kD(rt2)⟩ ⟨dip ∈ kD(rt1)⟩]
  kD_nsqnr [OF ⟨dip ∈ kD(rt1)⟩]
  kD_nsqnr [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, dsk, fn, hops, nhip
  where rtN [simp]: "the (rt dip) = (dsn, dsk, fn, hops, nhip)"
  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 (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
\( \langle \text{dsn} \in kD(rt) \rangle \) \( \text{rtn} \) 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])+

theorem rt_fresher_invalidate [simp]:
assumes "\( \text{dip} \in kD(rt) \)"
and indests: "\( \forall \text{rip} \in \text{dom(dests)}. \text{rip} \in vD(rt) \) \land sqn rt rip < the (dests rip)" shows "rt \( \subseteq \text{dip} \) invalidate rt dests"
proof (cases "\( \text{dip} \in \text{dom(dests)} \)"
assume "\( \text{dip} \notin \text{dom(dests)} \)"
thus \( \langle \text{dip} \in kD(rt) \rangle \) \( \text{rtn} \) 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 "\( \langle \text{dsn}, \text{dsn}, \text{f}, \text{hops}, \text{nhip} \rangle \) \( \text{rtn} \) 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])+

next
assume "\( \text{dsn} = dsn \)"
and "hops < hops \( n \)"
thus "\( \langle \text{dsn}, \text{dsn}, \text{f}, \text{hops}, \text{nhip} \rangle \) \( \text{rtn} \) 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])+

next
assume "\( \text{dsn} = dsn \)"
with "\( 0 < dsn \)"
show "\( \langle \text{dsn}, \text{dsn}, \text{inv}, \text{hops}, \text{nhip} \rangle \) \( \text{rtn} \) 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])+

next
assume "\( \text{dsn} = dsn \)"
with "\( 0 < dsn \)"
show "\( \langle \text{dsn}, \text{dsn}, \text{inv}, \text{hops}, \text{nhip} \rangle \) \( \text{rtn} \) 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])+

qed

with \( \langle \text{dhops rt dip} \rangle \geq 1 \) and \( \langle \text{dip} \in kD(rt) \rangle \) have "hops \( n \geq 1 \)"
by (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])+
lemma nsqn\_invalidate [simp]:
  assumes "dip\in kD(rt)"
  and "dip\in 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 rip\in dom(dests). rip\in vD(rt) \land the (dests rip) = inc (sqn rt rip)"
  shows "rt \approx dip invalidate rt dests"
proof (cases "dip\in dom(dests)"")
  assume "dip/\in dom(dests)"
  with ⟨dip\in kD(rt)⟩ have "dip\in kD(invalidate rt dests)" by simp
  with ⟨dip\in kD(rt)⟩ show ?thesis by rule (simp_all add: ⟨dip/\in dom(dests)⟩)
next
  assume "dip\in dom(dests)"
  with assms(2) have "sqn rt dip = nsqn, (the (invalidate rt dests dip))" by auto
  moreover then 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, (the (invalidate rt dests dip))" proof -
      from ⟨dip\in kD(rt)⟩ have "nsqn, (the (invalidate rt dests dip)) = the (dests dip) - 1" by simp
      with ⟨dip\in dom(dests)⟩ by (rule nsqn\_invalidate)
      show "sqn rt dip = nsqn, (the (invalidate rt dests dip))" by simp
    qed
    also have "nsqn, (the (invalidate rt dests dip)) = nsqn (invalidate rt dests) dip" by (simp add: kD\_nsqn)
    finally show "nsqn rt dip = nsqn (invalidate rt dests) dip" .
    qed simp
  qed

lemmas rt\_fresher\_inc\_invalidate [simp] = rt\_fresh\_as\_inc\_invalidate [THEN rt\_fresh\_asD1]

3.5.4 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 \subseteq 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 \subseteq ip rt2 \equiv rt\_strictly\_fresher i rt1 rt2"

lemma rt\_strictly\_fresher\_def'':
  "rt1 \subseteq ip rt2 = ((rt1 \subseteq dip rt2) \land \neg (rt1 \subseteq dip rt1))" unfolding rt\_strictly\_fresher\_def rt\_fresh\_as\_def by auto

lemma rt\_strictly\_fresherI' [intro]:
  assumes "rt1 \subseteq dip rt2"
  and "\neg (rt2 \subseteq dip rt1)"
  shows "rt1 \subseteq dip 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) ] \implies 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) ] \implies P rt1 rt2 i"
  shows "P rt1 rt2 i"
using assms(1) unfolding rt_strictly_fresher_def
by (erule(1) assms(2))

lemma rt_strictly_fresher_def':
  "rt1 ⊏_i 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 (rt1 i)) > \pi_5(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 ⊏_i 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 \( rt1 \sqsubseteq_{\text{dip}} rt2 \): have "rt1 \sqsubseteq_{\text{dip}} rt2" 
and "\neg (rt2 \sqsubseteq_{\text{dip}} rt1)"
unfolding rt_strictly_fresher_def'' by auto
from this(1) and \( rt2 \sqsubseteq_{\text{dip}} rt3 \) have "rt1 \sqsubseteq_{\text{dip}} rt3" ..

moreover from \( \neg (rt2 \sqsubseteq_{\text{dip}} rt1) \) have "\neg (rt3 \sqsubseteq_{\text{dip}} rt1)"
proof (rule contrapos_nn)
  assume "rt3 \sqsubseteq_{\text{dip}} rt1"
  with \( rt2 \sqsubseteq_{\text{dip}} rt3 \) show "rt2 \sqsubseteq_{\text{dip}} rt1" ..
qed

ultimately show "rt1 \sqsubseteq_{\text{dip}} rt3"
unfolding rt_strictly_fresher_def'' by auto
qed

lemma rt_fresher_trans_rt_strictly_fresher' [elim, trans]:
assumes "rt1 \sqsubseteq_{\text{dip}} rt2"
and "rt2 \sqsubseteq_{\text{dip}} rt3"
sows "rt1 \sqsubseteq_{\text{dip}} rt3"
proof -
from \( rt2 \sqsubseteq_{\text{dip}} rt3 \): have "rt2 \sqsubseteq_{\text{dip}} rt3"
and "\neg (rt3 \sqsubseteq_{\text{dip}} rt2)"
unfolding rt_strictly_fresher_def'' by auto
from \( rt1 \sqsubseteq_{\text{dip}} rt2 \): and this(1) have "rt1 \sqsubseteq_{\text{dip}} rt3" ..
moreover from \( \neg (rt3 \sqsubseteq_{\text{dip}} rt2) \) have "\neg (rt3 \sqsubseteq_{\text{dip}} rt1)"
proof (rule contrapos_nn)
  assume "rt3 \sqsubseteq_{\text{dip}} rt1"
  thus "rt3 \sqsubseteq_{\text{dip}} rt2" using \( rt1 \sqsubseteq_{\text{dip}} rt2 \) ..
qed

ultimately show "rt1 \sqsubseteq_{\text{dip}} rt3"
unfolding rt_strictly_fresher_def'' by auto
qed

lemma rt_fresher_imp_nsqn_le:
assumes "rt1 \sqsubseteq_{\text{ip}} rt2"
and "ip \in kD rt1"
and "ip \in kD rt2"
sows "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"
sows "rt1 \sqsubseteq_{\text{dip}} rt2"
proof
  from assms show "rt1 \sqsubseteq_{\text{dip}} rt2" ..
next
  show "\neg (rt1 \approx_{\text{dip}} rt2)"
proof
    assume "rt1 \approx_{\text{dip}} rt2"
    hence "rt2 \sqsubseteq_{\text{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
  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))"
says "rt1 ⊏ rt2"

using assms unfolding rt_strictly_fresher_def' by (auto simp add: kD_nsqn)

lemma invalidate_rtsf_left [simp]:
"\[dests dip rt rt'. dests dip = None \implies (invalidate rt dests ▦ dip rt') = (rt ▦ dip rt')\]"

unfolding invalidate_def rt_strictly_fresher_def'
by (rule iffI) (auto split: option.split_asm)

lemma vD_invalidate_rtsf_left [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!]:
"\[dip ip rt rt'. [ dip \neq ip; rt ▦ dip rt' ] \implies update rt ip r ▦ dip rt'\]"

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 simp
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 (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 (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, 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 ** 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)"
sows "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

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

by inv_cterms

lemma rrep_1_update_changes:
"paodv i |= onl \( \Gamma_{AODV} (\lambda(\xi, l). l \in \{PAodv-:7\} \cup \{PAodv-:5\} \cup \{PRrep-:0..PRrep-:1\} \cup \{PRreq-:0..PRreq-:3\}) \rightarrow rrep_1_update_changes (\xi) \)"

by inv_cterms

Proposition 7.38

lemma includes_nhip:
"paodv i |= onl \( \Gamma_{AODV} (\lambda(\xi, l). \forall dip \in kD (rt \ (\xi)). \text{the (nhop (rt \ (\xi)) dip)}) \)"

proof -
{ fix ip and \( \xi, \xi' \) :: state
  assume \( \forall dip \in kD (rt \ (\xi)). \text{the (nhop (rt \ (\xi)) dip)}) \)
  hence \( \forall dip \in kD (rt \ (\xi)). \text{the (nhop (rt \ (\xi)) dip)}) \)
    by clarsimp (metis nhop_update_unk_val update_another)
} note one_hop = this

{ fix ip sip sn hops and \( \xi, \xi' \) :: state
  assume \( \forall dip \in kD (rt \ (\xi)). \text{the (nhop (rt \ (\xi)) dip)}) \)
  and \( \xi' = \xi (|rt := update (rt \ (\xi)) ip (0, unk, val, Suc 0, ip))) \)
  and \( \in kD (rt \ (\xi)) \)\)
  hence \( \forall dip \in kD (rt \ (\xi)). \text{the (nhop (rt \ (\xi)) dip)}) \)
    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 |= onl \( \Gamma_{AODV} (\lambda((\xi, _,), (_, \xi')). kD (rt \ (\xi)) \subseteq kD (rt \ (\xi'))\)"

by (inv_cterms simp add: subset_insertI)

Proposition 7.5

lemma rreqs_increase:
"paodv i |= onl \( \Gamma_{AODV} (\lambda((\xi, _,), (_, \xi')). rreqs \ (\xi) \subseteq rreqs \ (\xi'))\)"

by (inv_cterms simp add: subset_insertI)

lemma dests_bigger_than_sqn:
"paodv i |= onl \( \Gamma_{AODV} (\lambda(\xi, l). l \in \{PAodv-:15..PAodv-:17\} \cup \{PPkt-:7..PPkt-:9\} \cup \{PRreq-:9..PRreq-:11\} \cup \{PRreq-:17..PRreq-:19\} \cup \{PRrep-:8..PRrep-:10\} \cup \{PRerr-:1..PRerr-:4\} \cup \{PRerr-:6\}) \rightarrow (\forall ip \in \text{dom(dests (\xi))). ip \in kD (rt (\xi)) \land sqn (rt (\xi)) ip \leq \text{the (dests (\xi) ip)})\)"

proof -
  have sqninv:
    "\( \forall dests \ rt rsn ip. \ [
      \forall ip \in \text{dom(dests)). ip \in kD (rt (\xi)) \land sqn (rt (\xi)) ip \leq \text{the (dests (\xi) ip)})
    \Rightarrow (\forall ip \in \text{dom(dests (\xi))). ip \in kD (rt (\xi)) \land sqn (rt (\xi)) ip \leq \text{the (dests (\xi) ip)))\)"

  by (rule sqn_invalidate_in_dests [THEN eq_imp_le], assumption)

  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 |=A 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
       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 dests_bigger_than_sqn]
   simp add: solve_invalidate)
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 |=A onll Γ_AODV (λ((ξ, _, a, _). ancast (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, _). ancast (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 |= onl \( \Gamma_{AODV} \) (\( \lambda(\xi, \_). \forall ip \in kD (rt (\xi)). \text{the (dhops (rt (\xi) ip))} \geq 1)\)"
by (inv_cterms) auto

lemma rreq_dip_in_vD_dip_eq_ip:
  "paodv i |= onl \( \Gamma_{AODV} \) (\( \lambda(\xi, _). \forall ip \in kD (rt (\xi)). \text{the (dhops (rt (\xi) ip)))} \geq 1)"
by (inv_cterms) auto

Proposition 7.11

lemma anycast_msg_zhops:
  "\( \\forall rreqid dip dsn dsk oip osn sip. \) paodv i ||= onl \( \Gamma_{AODV} \) (\( \lambda(_, a, \_). \) 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]])
  (inv_cterms inv add: 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 reachable (paodv i) TT"
  and "\( \{PRreq-:14\} \) \{PRrep-:n|n. True\} \rightarrow (\{PAodv-:4..PAodv-:5\} \cup \{PRreq-:n|n. True\} \rightarrow (dhops (rt (\xi))) (oip (\xi)) (ip (\xi)).
  p' \in \pi' \in \pi's terms \( \Gamma_{AODV} \) pp"
  and "l = PRreq-:14"
  and "a = unicast (the (nhop (rt (\xi) (oip (\xi)))) (dhops (rt (\xi)) (dip (\xi))) (sbn (rt (\xi) (dip (\xi)))) (oip (\xi)) (ip (\xi)))"
  and *: "\( \forall ip \in kD (rt (\xi)). Suc 0 \leq \text{the (dhops (rt (\xi)) ip)})".
  from dip \( \xi \in vD (rt (\xi))\) have "dip \( \xi \in kD (rt (\xi))\)"
  and (rule vD_iD_gives_kD(1))
  with * have "Suc 0 \leq \text{the (dhops (rt (\xi)) (dip (\xi)))}" ..
  thus "0 < the (dhops (rt (\xi)) (dip (\xi)))" by simp
qed

lemma hop_count_zero_oip_dip_sip:
  "paodv i |= (recvmsg msg_zhops \rightarrow) onl \( \Gamma_{AODV} \) (\( \lambda(\_, a, \_). \) anycast msg_zhops a)"
proof
  (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv])
  (cases a) simp all
  by

lemma osn_rreq:
  "paodv i |= (recvmsg rreq_rrep_sn \rightarrow) onl \( \Gamma_{AODV} \) (\( \lambda(\xi, l). \) l \in \{PAodv-:4, PAodv-:5\} \cup \{PRreq-:n|n. True\} \rightarrow (dhops (rt (\xi))) (oip (\xi)) (ip (\xi)))"
by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv])

lemma osn_rreq':
  "paodv i |= (recvmsg (\( \lambdam. rreq_rrep_sn \rightarrow \) msg_zhops m) \rightarrow) onl \( \Gamma_{AODV} \) (\( \lambda(\xi, l). \) l \in \{PAodv-:4, PAodv-:5\} \cup \{PRreq-:n|n. True\} \rightarrow (dhops (rt (\xi))) (oip (\xi)) (ip (\xi)))"
proof
  (rule invariant_weakenE [OF osn_rreq])
  fix a
  assume "recvmsg (\( \lambdam. rreq_rrep_sn \rightarrow \) msg_zhops m) a"
  thus "recvmsg (\( \lambdam. rreq_rrep_sn \rightarrow \) msg_zhops m) a"
  by (cases a) simp_all
qed

lemma dsn_rrep:
  "paodv i |= (recvmsg rreq_rrep_sn \rightarrow) onl \( \Gamma_{AODV} \) (\( \lambda(\xi, l). \) l \in \{PAodv-:6, PAodv-:7\} \cup \{PRreq-:n|n. True\} \rightarrow (dhops (rt (\xi))) (oip (\xi)) (ip (\xi)))"
by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv])

lemma dsn_rrep':

292
"paodv i |= (recvmsg (\m. rreq_rrep_sn m \& msg_zhops m) \rightarrow) onl \Gamma_{AODV} (\lambda(\xi, 1). \\
\{l \in \{PAodv-\:4, PAodv-\:6, \} \cup \{PRreq-\:n | n. True\} \rightarrow 1 \leq dsn \xi\}"

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) \rightarrow) onl \Gamma_{AODV} (\lambda(\xi, l). \\
(\xi \in \{PAodv-\:4..PAodv-\:5\} \cup \{PRreq-\:n | n. True\} \rightarrow \\
\pi_3 (the (rt dip)) = unk \rightarrow the (dhops (ht dip)) = Suc 0 \rightarrow \\
the (nhop (rt dip)) = dip\))"

proof (rule invariant_weakenE [OF hop_count_zero_oip_dip_sip])
fix a
assume "recvmsg (\m. 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 (\m. rreq_rrep_sn m \& msg_zhops m) \rightarrow) onl \Gamma_{AODV} (\lambda(\xi, l). \\
\forall dip \in kD(rt). (sqn (rt dip) = 0 \rightarrow sqnf (rt dip) = unk) \\
\& (sqnf (rt dip) = unk \rightarrow the (dhops (rt dip)) = Suc 0) \\
\& (the (dhops (rt dip)) = Suc 0 \rightarrow the (nhop (rt dip)) = dip)\))"

proof -
{ 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 dip) = 0 \rightarrow sqnf (rt dip) = unk)"
  and *: "sqn (rt dip) = 0 \rightarrow P \xi dip"
  have "P \xi dip" by (rule *)
  note sqn_invalidate_zero [elim!] = 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 "\(\forall dip \in kD(rt). (sqn rt dip = 0 \rightarrow \pi_3(\text{the (rt dip)}) = unk) \land (\pi_3(\text{the (rt dip)}) = unk \rightarrow (dhops rt dip) = Suc 0) \land (\text{the (dhops rt dip)} = Suc 0 \rightarrow (nhop rt dip) = dip)\)"
  and "Suc 0 \leq dsn"
  and "ip \neq dip \rightarrow ip \in kD(rt)"
  hence "sqn (update rt dip (dsn, kno, Suc hops, sip)) ip = 0 \rightarrow (\pi_3 (\text{the (update rt dip (dsn, kno, Suc hops, sip) ip)}) = unk)"
} by (rule update_cases, auto simp add: sqn_def dest!: bspec)
} note prreq_ok3 [simp] = this

{ fix rt sip
  assume "\(\forall dip \in kD rt. (sqn (update rt sip (0, unk, val, Suc 0, sip)) dip = 0 \rightarrow (\pi_3 (\text{the (update rt sip (0, unk, val, Suc 0, sip) dip)}) = unk) \land (\pi_3 (\text{the (update rt sip (0, unk, val, Suc 0, sip) dip)}) = unk \rightarrow (dhops (update rt sip (0, unk, val, Suc 0, sip) dip)) = Suc 0) \land (\text{the (dhops (update rt sip (0, unk, val, Suc 0, sip) dip)}) = Suc 0 \rightarrow (\text{the (nhop (update rt sip (0, unk, val, Suc 0, sip) dip)}) = dip)}\)"
  by (rule update_cases) simp_all
} note prreq_ok4 [simp] = this

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

have prreq_ok6 [simp]: "\(\forall sip rt. sqn (update rt sip (0, unk, val, Suc 0, sip)) sip = 0 \rightarrow (\pi_3 (\text{the (update rt sip (0, unk, val, Suc 0, sip) sip)}) = unk)\)"
  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 (\lambda m. rreq_rrep_sn m \land msg_zhops m) \rightarrow onl \Gamma_{AODV} (\lambda(\xi, _). \forall dip \in kD(\xi rt). (sqn (\xi rt dip) = 0 \rightarrow (sqnf (\xi rt dip) = unk \land \text{the (dhops (\xi rt dip)}) = 1 \land \text{the (nhop (\xi rt dip)}) = dip))))" 
  by (rule invariant_weakenE [OF zero_seq_unk_hops_one']) auto

lemma kD_unk_or_atleast_one:
  "paodv i ||= (recvmsg (\lambda m. rreq_rrep_sn) \rightarrow onl \Gamma_{AODV} (\lambda(\xi, 1). \forall dip \in kD(\xi rt). (\pi_3 (\text{the (\xi rt dip)}) = unk \lor 1 \leq \pi_2 (\text{the (\xi rt dip)})))" 
  proof - 
  { fix sip rt dsn1 dsn2 dsk1 dsk2 flag1 flag2 hops1 hops2 nhip1 nhip2 
    assume "dsk1 = unk \lor Suc 0 \leq dsn2"
    hence "\(\pi_3 (\text{the (update rt sip (dsn1, dsk1, flag1, hops1, nhip1) sip)}) = unk \lor Suc 0 \leq sqn (\text{update rt sip (dsn2, dsk2, flag2, hops2, nhip2) sip})\)"
      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")

  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
  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_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)

  qed

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)"

295
Proposition 7.14

lemma rreq_rrep_fresh_any_step_invariant:
  "paodv i ⇑η onl Γ (λ((ξ, _), a, _). anycast (rreq_rrep_fresh (rt ξ)) a)"

proof -
  have rreq_oip: "paodv i ⇑η onl Γ (λ((ξ, l). (l ∈ {PRreq-:2, PRreq-:3, PRreq-:4, PRreq-:13, PRreq-:21} −→ oip ξ ∈ kD(rt ξ) ∧ sqn (rt ξ) (oip ξ) > (osn ξ) ∨ (sqn (rt ξ) (oip ξ) = (osn ξ) ∧ the (dhops (rt ξ) (oip ξ)) ≤ Suc (hops ξ) ∧ the (flag (rt ξ) (oip ξ)) = val)))" 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])
  show ?thesis
    by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf rrep_prrep])
  qed

Proposition 7.15

lemma rerr_invalid_any_step_invariant:
  "paodv i ⇑η onl Γ (λ((ξ, _), a, _). anycast (rerr_invalid (rt ξ)) a)"

proof -
  have dests_inv: "paodv i ⇑η onl Γ (λ((ξ, l'). pp p') " by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf rrep_prrep])
  show ?thesis
    by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf rrep_prrep])
  qed

296
→ (∀ ip∈dom(dests ξ). ip∈vD(rt ξ)))
∧ (l ∈ {PAodv=16..PAodv=17}
   ∪ {PPkt=8..PPkt=9}
   ∪ {PRreq=10..PRreq=11}
   ∪ {PRreq=18..PRreq=19}
   ∪ {PRrep=9..PRrep=10}
   ∪ {PRerr=2..PRerr=4} → (∀ ip∈dom(dests ξ). ip∈iD(rt ξ)
∧ the(dests ξ ip) = sqn(rt ξ ip))
∧ (l = PPkt=12 → 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 |=
onl ΓAODV (λ(ξ, l). (l ∈ {PAodv=15, PPkt=7, PRreq=9, PRreq=17, PRrep=8}
   → (∀ 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)+

Proposition 7.27

lemma route_tables_fresher:
"paodv i |=A (recvmsg rreq_rrep_sn → onll ΓAODV (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 invariant_restrict_inD])
fix ξ pp p'
assume "(ξ, pp) ∈ reachable (paodv i) (recvmsg rreq_rrep_sn)"
and "{PRreq=2}[λξ. ξ[rt := update(rt ξ)(oip ξ)(osn ξ, kno, val, Suc(hops ξ), sip ξ)]
   p' ∈ sterms ΓAODV pp"
and "Suc 0 ≤ osn ξ"
and *: "∀ ip∈kD(rt ξ). Suc 0 ≤ (dhops(rt ξ) ip)"
show "∀ ip∈kD(rt ξ). rt ξ ⊑ dip update(rt ξ)(oip ξ)(osn ξ, kno, val, Suc(hops ξ), sip ξ)"
proof
fix ip
assume "ip∈kD(rt ξ)"
moreover with * have "1 ≤ (dhops(rt ξ) ip)" by simp
moreover from "Suc 0 ≤ osn ξ"
have "update_arg_wf (osn ξ, kno, val, Suc(hops ξ), sip ξ)" ..
ultimately show "rt ξ ⊑ dip update(rt ξ)(oip ξ)(osn ξ, kno, val, Suc(hops ξ), sip ξ)"
by (rule rt_fresher_update)
qed
next
fix ξ pp p'
assume "(ξ, pp) ∈ reachable (paodv i) (recvmsg rreq_rrep_sn)"
and "{PRrep-1}\{λξ. \((rt := update (rt ξ) (dip ξ) (dsn ξ, kno, val, Suc (hops ξ), sip ξ))\)}
p' ∈ stterms 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 ⊑ dip 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 ⊑ dip 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∈kD(rt ξ) ⇒ dip∈kD(rt ξ')" and "[dip∈kD(rt ξ); 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_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 ⊑ dip 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'
shows "R dip 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)"
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 "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 "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)
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 "I ≤ 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∈iD(rt (σ' sip))"
  hence "the (flag (rt (σ' sip)) ip) = inv" ..
  with :sn = nsqn (rt (σ' sip)) ip 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)) ip⟩ show ?thesis
    by simp

next

  assume "ip∈kD(rt (σ' sip))"
  hence "the (flag (rt (σ' sip)) ip) = inv" ..
  with ⟨sn = nsqn (rt (σ' sip)) ip⟩ show ?thesis
    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)
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))
lemma msg_fresh [simp]:
"∀ hops rreqid dip dsn dsk oip osn sip.
  \(msg\_fresh\ \sigma \ (Rreq\ hops\ rreqid\ dip\ dsn\ dsk\ oip\ osn\ sip) = \)
  \((osn \geq 1 \land (sip \neq oip \rightarrow oip \in kD(rt \ (\sigma \ sip))) \land
  nsqn (rt \ (\sigma \ sip)) \ oip \geq osn \land
  (nsqn (rt \ (\sigma \ sip)) \ oip = osn \rightarrow (hops \geq \text{the (dhops (rt \ (\sigma \ sip)) oip))}
  \lor \text{the (flag (rt \ (\sigma \ sip)) oip) = inv))))"
"∀ hops dip dsn oip sip. \(msg\_fresh\ \sigma \ (Rrep\ hops\ dip\ dsn\ oip\ sip) = \)
  \((dsn \geq 1 \land (sip \neq dip \rightarrow dip \in kD(rt \ (\sigma \ sip)) \land
  nsqn (rt \ (\sigma \ sip)) \ dip \geq dsn \land
  (nsqn (rt \ (\sigma \ sip)) \ dip = dsn \rightarrow (hops \geq \text{the (dhops (rt \ (\sigma \ sip)) dip))}
  \lor \text{the (flag (rt \ (\sigma \ sip)) dip) = inv))))"
"∀ dests sip. \(msg\_fresh\ \sigma \ (Rerr\ dests\ sip) = \)
  \((\forall ripc \in \text{dom(dests)}. (ripc \in kD(rt \ (\sigma \ sip)) \land
  \text{the (dests ripc) - 1 \leq nsqn (rt \ (\sigma \ sip)) ripc}))
  \lor \text{the (flag (rt \ (\sigma \ sip)) ripc) = inv))))"
"∀ d dip. \(msg\_fresh\ \sigma \ (Newpkt\ d\ dip) = True""
"∀ d dip sip. \(msg\_fresh\ \sigma \ (Pkt\ d\ dip\ sip) = True"
unfolding msg_fresh_def by simp_all

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) \ σ\ m =⇒ recvmsg rreq\_rrep\_sn\ m"
by (cases m) simp_all

lemma rreq_nsqn_is_fresh [simp]:
fixes \( \sigma\ msg\ hops\ rreqid\ dip\ dsn\ dsk\ oip\ osn\ sip\)
assumes "rreq\_rrep\_fresh\ (rt \ (\sigma \ 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\ \sigma\ (Rreq\ hops\ rreqid\ dip\ dsn\ dsk\ oip\ osn\ sip)"
(is  "msg\_fresh\ \sigma\ ?msg")
proof -
  let ?rt = "rt \ (\sigma \ sip)"
  from assms(2) have "1 \leq osn" by simp
  thus ?thesis
  unfolding msg_fresh_def
  proof (simp only: msg.case, intro conjI impI)
    assume "sip \neq oip"
    with assms(1) show "oip \in kD(?rt)" by simp
  next
    assume "sip \neq oip"
    and "nsqn \ ?rt\ oip = osn"
    show "the (dhops \ ?rt\ oip) \leq hops \lor \text{the (flag \ ?rt\ oip) = inv}"
    proof (cases "oip \in vD(?rt)"
      assume "oip \in vD(?rt)"
      hence "nsqn \ ?rt\ oip = sqn \ ?rt\ oip" ..
      with assms(1) and \( \text{sqn \ ?rt\ oip = osn}" by simp
      with assms(1) and \( \text{the (dhops \ ?rt\ oip) \leq hops}" by simp
      thus ?thesis ..
    next
    assume "oip \in vD(?rt)"
    moreover from assms(1) and \( \text{the (dhops \ ?rt\ oip) \leq hops}" by simp
    ultimately have "oip \in iD(?rt)" by auto
    hence "the (flag \ ?rt\ oip) = inv" ..
thus \( \text{thesis} \).

qed

next

assume "rip \neq \text{op}"

with assms(1) have "osn \leq sqn \rt \text{op}" by auto

thus "osn \leq \text{nsqn} (\rt (\sigma \text{sip})) \text{op}" proof (rule nat_le_eq_or_lt)

assume "osn < sqn \rt \text{op}"

hence "osn \leq sqn \rt \text{op} - 1" by simp

also have "... \leq \text{nsqn} \rt \text{op}" by (rule sqn_nsqn)

finally show "osn \leq \text{nsqn} \rt \text{op}".

next

assume "osn = sqn \rt \text{op}"

with assms(1) and \( \sigma \text{sip} \neq \text{op} \) have "oip \in kD(\rt)"

and "the (flag \rt \text{op}) = val" by auto

hence "\text{nsqn} \rt \text{op} = sqn \rt \text{op}".

thus "osn \leq \text{nsqn} \rt \text{op}" by simp

qed

qed simp

qed

lemma rrep_nsqn_is_fresh [simp]:

fixes \sigma \ msg \ hops \ dip \ dsn \ op \ sip

assumes "rreq_rrep_fresh (\rt (\sigma \text{sip})) (Rrep \ hops \ dip \ dsn \ op \ sip)"

and "rreq_rrep_sn (Rrep \ hops \ dip \ dsn \ op \ sip)"

shows "msg_fresh \sigma (Rrep \ hops \ dip \ dsn \ op \ sip)"

(is "msg_fresh \sigma \ ?msg")

proof -

let \rt = "rt (\sigma \text{sip})"

from assms have "\text{op} \neq \text{dip} \rightarrow \text{dip} \in kD(\rt) \land \text{sqn} \rt \text{dip} = \text{dsn} \land \text{the (flag \rt \text{dip}) = val}"

by simp

hence "\text{op} \neq \text{dip} \rightarrow \text{dip} \in kD(\rt) \land \text{nsqn} \rt \text{dip} \geq \text{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 (\rt (\sigma \text{sip})) (\text{Rerr} \ dests \ \text{sip})"

shows "msg_fresh \sigma (\text{Rerr} \ dests \ \text{sip})"

(is "msg_fresh \sigma \ ?msg")

proof -

let \rt = "rt (\sigma \text{sip})"

from assms have \( \forall \text{rip} \in \text{dom(dests)}. (\text{rip} \in kD(\rt) \land \text{sqn} \rt \text{dip} = \text{dsn} \land \text{the (dests rip) = sqn (\rt \text{dip}) rip})""

by clarsimp

have \( \forall \text{rip} \in \text{dom(dests)}. (\text{rip} \in kD(\rt) \land \text{the (dests rip) = sqn (\rt \text{dip}) rip}))""

by clarsimp

proof

fix rip

assume "rip \in \text{dom \ dests}"

with \* have "rip \in kD(\rt (\sigma \text{sip}))" and "\text{the (dests rip) = sqn (\rt (\sigma \text{sip}) rip)}"

by auto

from this(2) have "\text{the (dests rip)} - 1 = \text{sqn (\rt (\sigma \text{sip}) rip} - 1" by simp

also have "... \leq \text{nsqn (\rt (\sigma \text{sip}) rip)" by (rule sqn_nsqn)

finally have "\text{the (dests rip)} - 1 \leq \text{nsqn (\rt (\sigma \text{sip}) rip)".

with \(\text{rip} \in kD(\rt (\sigma \text{sip}))

show "\text{rip} \in kD(\rt (\sigma \text{sip})) \land \text{the (dests rip)} - 1 \leq \text{nsqn (\rt (\sigma \text{sip}) rip)"

by clarsimp

303
thus \[ \text{"msg\_fresh } \sigma \text{ ?msg\"} \]
by simp

qed

lemma quality\_increases\_msg\_fresh [elim]:
assumes qinc: \[ \forall j. \text{quality\_increases } (\sigma j) (\sigma' j) \]
and \[ \text{"msg\_fresh } \sigma j \text{\"} \]
shows \[ \text{"msg\_fresh } \sigma' j \text{\"} \]
using assms(2)
proof (cases m)
fix hops rreqid dip dsn oip osn sip
assume [simp]: \[ m = \text{Rreq hops rreqid dip dsn oip osn sip} \]
and \[ \text{"msg\_fresh } \sigma m \text{\"} \]
then have \[ \text{osn } \geq 1 \]
and \[ \text{"sip = oip } \lor (\text{\textcircled{1}} \text{\textcircled{1}} \text{\textcircled{1}}} \]
moreover from qinc have \[ \text{"quality\_increases } (\sigma j) (\sigma' j) \text{\"} \]
ultimately have \[ \text{osn } \geq 1 \]
by auto
from this(2) show \[ ?thesis \]
proof
assume \[ \text{"sip = oip } \]
next
assume \[ \text{"sip = oip } \]
moreover from qinc have \[ \text{"quality\_increases } (\sigma j) (\sigma' j) \text{\"} \]
ultimately have \[ \text{osn } \geq 1 \]
using \[ \text{osn } \geq 1 \]
with \[ \text{osn } \geq 1 \]
by clarsimp
qed
next
fix hops dip dsn oip sip
assume [simp]: \[ m = \text{Rrep hops dip dsn oip sip} \]
and \[ \text{"msg\_fresh } \sigma m \text{\"} \]
then have \[ \text{dsn } \geq 1 \]
and \[ \text{\textcircled{1}} \text{\textcircled{1}} \text{\textcircled{1}}} \]
moreover from qinc have \[ \text{"quality\_increases } (\sigma j) (\sigma' j) \text{\"} \]
ultimately have \[ \text{dsn } \geq 1 \]
using \[ \text{dsn } \geq 1 \]
with \[ \text{dsn } \geq 1 \]
by clarsimp
qed
next
fix dests sip
assume \[ simp \]: \( m = \text{Rerr dests sip} \)
and \( \text{msg\_fresh} \ \sigma \ m \)
then have \*: \( \forall \text{rip} \in \text{dom(dests)}. \ \text{rip} \in \text{kD}(\text{rt (\sigma sip)}) \wedge \text{the (dests rip)} - 1 \leq \text{nsqn (rt (\sigma sip)) rip} \)
by \( \text{simp} \)
have \( \forall \text{rip} \in \text{dom(dests)}. \ \text{rip} \in \text{kD}(\text{rt (\sigma' sip)}) \wedge \text{the (dests rip)} - 1 \leq \text{nsqn (rt (\sigma' sip)) rip} \)
proof
fix \text{rip}
assume \( \text{rip} \in \text{dom(dests)} \)
with \* have \( \text{rip} \in \text{kD}(\text{rt (\sigma sip)}) \wedge \text{the (dests rip)} - 1 \leq \text{nsqn (rt (\sigma sip)) rip} \)
by \( \text{(drule(1) bspec, clarsimp)+} \)
moreover from \( \text{qinc} \) have "quality\_increases (\sigma sip) (\sigma' sip)" by \( \text{simp} \)
ultimately show "\( \text{rip} \in \text{kD}(\text{rt (\sigma' sip)}) \wedge \text{the (dests rip)} - 1 \leq \text{nsqn (rt (\sigma' sip)) rip} \) ..
qed
thus \( \text{?thesis by simp} \)
qed \( \text{simp\_all} \)
end

3.8 The ‘open’ AODV model

definition \( \sigma_{AODV}' :: \frac{\text{(ip \Rightarrow state) \times (state, msg, pseqp, pseqp label) seqp)} set}{\lambda i. \text{aodv}\_init i, \Gamma_{AODV \ PAodv}}\)
 abbreviation \( \text{opaodv} :: \frac{\text{ip \Rightarrow (ip \Rightarrow state) \times (state, msg, pseqp, pseqp label) seqp, msg seq\_action) automaton}}{\frac{i}{\text{|init = } \sigma_{AODV}'}, \frac{\text{trans = oseqp\_sos} \ \Gamma_{AODV} i}{\frac{i}{}}}\)

lemma \( \text{initial\_aodv [intro!, simp]}: \frac{\text{initial\_i (init (opaodv i)) (init (paodv i))}}{\text{\frac{i}{init = } \sigma_{AODV}'}\ \frac{\text{\wedge trans = oseqp\_sos \ \Gamma_{AODV} i}}{\frac{i}{}}}\)

lemma \( \text{oaodv\_control\_within [simp]}: \frac{\text{control\_within \Gamma_{AODV} (init (opaodv i))}}{\text{\frac{i}{\text{\wedge trans = oseqp\_sos \ \Gamma_{AODV} i}}\ \frac{i}{}}}\)

lemma \( \text{oaodv\_init\_kd\_empty [simp]}: \frac{\text{\sigma, p) \in \sigma_{AODV}' \Longrightarrow \text{kd (rt (\sigma i)) = {}}}\}{\text{\frac{\text{\wedge labels \Gamma_{AODV} p = {PAodv=-0}}}{\text{\frac{\text{\wedge trans = oseqp\_sos \ \Gamma_{AODV} i}}{\text{\frac{i}{}}}}}}\}

lemma \( \text{oaodv\_init\_vd\_empty [simp]}: \frac{\text{\sigma, p) \in \sigma_{AODV}' \Longrightarrow \text{vd (rt (\sigma i)) = {}}}\}{\text{\frac{\text{\wedge trans = oseqp\_sos \ \Gamma_{AODV} i}}{\text{\frac{i}{}}}}}}\)

lemma \( \text{oaodv\_trans: trans (opaodv i) = oseqp\_sos \ \Gamma_{AODV} i}}\)

declare 
oseq\_invariant\_ctermsI \( \text{[OF aodv\_wf aodv\_control\_within aodv\_simple\_labels aodv\_trans, cterms\_intros]}\)
oseq\_step\_invariant\_ctermsI \( \text{[OF aodv\_wf aodv\_control\_within aodv\_simple\_labels aodv\_trans, cterms\_intros]}\)
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 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 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 −→ 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:
"opaodv i \_ \Rightarrow (recvmsg rreq_rrep_sn) onll \Gamma_{AODV} (\lambda((\xi, _), _, (\xi', _)). \text{quality_increases}\ (\xi')")
proof (rule step_invariantI)
  fix s a s'
  assume sr: "s \in \text{reachable (opaodv i) (recvmsg rreq_rrep_sn)}"
  and tr: "(s, a, s') \in \text{trans (opaodv i)}"
  and rm: "recvmsg rreq_rrep_sn a"
  from sr have srTT: "s \in \text{reachable (opaodv i) TT}"
  ..
  from route_tables_fresher sr tr rm
  have "onll \Gamma_{AODV} (\lambda((\xi, _), _, (\xi', _)). \forall dip \in kD \ (rt \xi) \sqsubseteq 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', _)). kD \ (rt \xi) \subseteq 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', _)). \forall ip. \text{sqn} \ (rt \xi) \leq \text{sqn} \ (rt \xi')) (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 oaodv_trans aodv_trans, simplified seqll_onll_swap]

lemma quality_increases:
"opaodv i \_ \Rightarrow (act (recvmsg rreq_rrep_sn), other A \{i\} \rightarrow) onll \Gamma_{AODV} (\lambda((\sigma, _), a, (\sigma', _)). \forall j. \text{quality_increases}\ (\sigma j) (\sigma' j))"
(is "/= (\Rightarrow (\Rightarrow (\Rightarrow (\Rightarrow \Rightarrow \Rightarrow \Rightarrow \Rightarrow))")
proof (rule onll_ostep_invariantI, simp)
  fix \sigma p l a \sigma' p' l'
  assume or: "((\sigma, p) \in \text{oreachable (opaodv i) ?S (other A \{i\})")"
  and ll: "l \in \text{labels \Gamma_{AODV} p}"
  and ?S \sigma \sigma' a
  and tr: "((\sigma, p), a, (\sigma', p')) \in \text{oseqp_sos \Gamma_{AODV} i}"
  and ll': "l' \in \text{labels \Gamma_{AODV} p'}"
  from this(1-3) have "orecvmsg (\lambda_. rreq_rrep_sn) \sigma a"
    by (auto dest!: oreachable_weakenE [where QS="act (recvmsg rreq_rrep_sn)"
      and QU="other A \{i\}"
    ] otherwith_actionD)
  with or have orw: "((\sigma, p) \in \text{oreachable (opaodv i) (act (recvmsg rreq_rrep_sn)) (other A \{i\})")"
    by - (erule oreachable_weakenE, auto)
  with tr ll ll' and orecvmsg (\lambda_. 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 "\forall j. \text{quality_increases}\ (\sigma j) (\sigma' j)"
    by (auto dest!: otherwith_syncD)
  qed

lemma rreq_rrep_nsqn_fresh_any_step_invariant:
"opaodv i \_ \Rightarrow (act (recvmsg rreq_rrep_sn), other A \{i\} \rightarrow) onll \Gamma_{AODV} (\lambda((\sigma, _), a, _). \text{anycast (msg_fresh \sigma) a})"
proof (rule onll_ostep_invariantI, simp del: act_simp)
  fix \sigma p a \sigma' p'
  assume or: "((\sigma, p) \in \text{oreachable (opaodv i) (act (recvmsg rreq_rrep_sn)) (other A \{i\})")"
    and "((\sigma, p), a, (\sigma', p')) \in \text{oseqp_sos \Gamma_{AODV} i}"
  and recv: "act (recvmsg rreq_rrep_sn) \sigma a"
  obtain l l' where "l \in \text{labels \Gamma_{AODV} p}" and "l' \in \text{labels \Gamma_{AODV} p'}"
    by (metis aodv_ex_label)
have "anycast (rreq_rrep_fresh (rt (σ i))) a"
proof -
  have "opaodv i \models_A (act (recvmsg rreq_rrep_sn), other A {i} \to)
    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 iniadv, 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 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 iniadv, 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 (msg_fresh σ) a"
proof -
  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

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 \models (otherwith quality_increases {i} (orecvmsg msg_fresh),
other quality_increases \{1\} →
onl \Gamma_{AODV} (\lambda(\sigma, 1). 1 :\{PAodv-=1\} → msg\_fresh \sigma (msg (\sigma i)))

proof (rule oreceived_msg_inv)
fix \sigma \sigma' m
assume #: "msg\_fresh \sigma m"
and "other quality_increases \{1\} \sigma \sigma'"
from this(2) have "\forall j. 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 i\{msg := m\}) m"
proof (cases m)
fix dests sip
assume "m = Rerr dests sip"
with msg\_fresh \sigma m: show ?thesis by auto
qed auto

lemma quality_increases\_nsqn\_fresh:
"opaodv i \models (\text{otherwith quality\_increases \{1\} (orecmsg msg\_fresh)},
other quality_increases \{i\} \→
onl \Gamma_{AODV} (\lambda(\sigma, _, _). \sigma\_{\text{other quality\_increases \{i\}}})). \forall j. quality\_increases (\sigma j) (\sigma' j)"
by (rule ostep\_invariant\_weakenE [OF quality\_increases])

lemma oosn\_rreq:
"opaodv i \models (\text{otherwith quality\_increases \{1\} (orecmsg msg\_fresh)},
other quality_increases \{i\} \→
onl \Gamma_{AODV} (\lambda(\sigma, _, _). \sigma\_{\text{other quality\_increases \{i\}}})). \forall j. quality\_increases (\sigma j) (\sigma' j)"
by (rule oinvariant\_weakenE [OF open\_seq\_invariant [OF oosn\_rreq initiali\_aodv]])
(auto simp: seql\_onl\_swap)

lemma rreq\_sip:
"opaodv i \models (\text{otherwith quality\_increases \{1\} (orecmsg msg\_fresh)},
other quality_increases \{i\} \→
onl \Gamma_{AODV} (\lambda(\sigma, 1). (l \in \{PAodv-=4, PAodv-=5, PRreq-=0, PRreq-=2\} \land sip (\sigma i) \neq oip (\sigma i))
\rightarrow oip (\sigma i) \in kD (rt (\sigma (sip (\sigma i))))
\land nsqn (rt (\sigma (sip (\sigma i)))) (oip (\sigma i)) \geq osn (\sigma i)
\land (nsqn (rt (\sigma (sip (\sigma i)))) (oip (\sigma i)) = osn (\sigma i)
\rightarrow (hops (\sigma i) \geq \text{the \(d\)hops (rt (\sigma (sip (\sigma i)))) (oip (\sigma i)))
\lor \text{the \(f\)lag (rt (\sigma (sip (\sigma i)))) (oip (\sigma i))) = \text{inv}))")
(is "\_ \models (?S, ?U \rightarrow \_)")
proof (inv\_cterms inv add: oseq\_step\_invariant\_sterms [OF quality\_increases\_nsqn\_fresh]
\text{aodv\_wf oaoadv\_trans]}
onl \text{oinvariant\_sterms [OF aodv\_wf oreceived\_rreq\_rrep\_nsqn\_fresh\_inv]
\text{oinvariant\_sterms [OF aodv\_wf oosn\_rreq]
simp add: seql\_simp
simp del: One\_nat\_def, rule implI)
fix \sigma \sigma' p l
assume "(\sigma, p) \in \text{oreachable \(opaodv\) i} ?S ?U"
and "l \in \text{labels \(aodv\) p}"
and pre:
"(l = PAodv-=4 \lor l = PAodv-=5 \lor l = PRreq-=0 \lor l = PRreq-=2) \land sip (\sigma i) \neq oip (\sigma i)
\rightarrow oip (\sigma i) \in kD (rt (\sigma (sip (\sigma i))))
\land osn (\sigma i) \leq nsqn (rt (\sigma (sip (\sigma i)))) (oip (\sigma i))
\land (nsqn (rt (\sigma (sip (\sigma i)))) (oip (\sigma i)) = osn (\sigma i)
\rightarrow (\text{the \(d\)hops (rt (\sigma (sip (\sigma i)))) (oip (\sigma i))) \leq \text{hops (\sigma i)}
\lor \text{the \(f\)lag (rt (\sigma (sip (\sigma i)))) (oip (\sigma i))) = \text{inv})")
and "other quality\_increases \{1\} \sigma \sigma'"
and hyp: "(l=PAodv-=4 \lor l=PAodv-=5 \lor l=PRreq-=0 \lor l=PRreq-=2) \land sip (\sigma' i) \neq oip (\sigma' i)"
(is "\text{\(label\)s} \land sip (\sigma' i) \neq oip (\sigma' i)")
from this(4) have "\sigma' i = \sigma i" ..
with hyp have hyp': "\text{\(label\)s} \land sip (\sigma i) \neq oip (\sigma 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 oinvariant_weakenE [OF open_seq_invariant [OF dsn_rrep initiali_aodv])

by (auto simp: seql_onl_swap)

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:
"oapdvi i = (otherwith quality_increases {i} (orecvmsg msg_fresh),
other quality increases {i} →)

onl Γ_AODV (seq i (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: oseq-onl_swap)


lemma rrep_sip:
"oapdvi i = (otherwith quality_increases {i} (orecvmsg msg_fresh),
other quality increases {i} →)

onl Γ_AODV (λ(σ, 1).

(1 ∈ {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)
→ (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 aodv_trans]

onl_oainvariant_sterms [OF aodv_wf oreceived_rreq_rrep_nsqn_fresh_inv]

onl_oainvariant_sterms [OF aodv_wf odsn_rrep]

simp del: One_nat_def, rule impl)

fix σ σ' p l

assume "σ, p) ∈ oreachable (oapdvi i) ?S ?U"

and "l ∈ labels Γ_AODV p"

and pre:

"(1 = PAodv-:6 ∨ 1 = PAodv-:7 ∨ 1 = PRrep-:0 ∨ 1 = 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)
→ (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) ∧ sip (σ' i) ≠ dip (σ' i)"
(is "\(\text{labels} \land \text{sip} (\sigma' i) \neq \text{dip} (\sigma i)\)"

from this(4) have "\(\sigma' i = \sigma i\)"

with hyp have hyp': "\(\text{labels} \land \text{sip} (\sigma i) \neq \text{dip} (\sigma i)\)" by simp

show "\(\text{dip} (\sigma' i) \in kD (\text{rt} (\sigma' (\text{sip} (\sigma' i))))\)

\(\land \text{dsn} (\sigma' i) \leq \text{nsqn} (\text{rt} (\sigma' (\text{sip} (\sigma' i)))) (\text{dip} (\sigma' i))\)

\(\land (\text{nsqn} (\text{rt} (\sigma' (\text{sip} (\sigma' i)))) (\text{dip} (\sigma' i)) = \text{dsn} (\sigma' i))\)

\(\rightarrow \) the \(\text{(dhops} (\text{rt} (\sigma' (\text{sip} (\sigma' i)))) (\text{dip} (\sigma' i)) \leq \text{hops} (\sigma' i)\)

\(\lor \) the \(\text{(flag} (\text{rt} (\sigma' (\text{sip} (\sigma' i)))) (\text{dip} (\sigma' i)) = \text{inv})\)"

proof (cases "\(\text{sip} (\sigma i) = i\)"

assume "\(\text{sip} (\sigma i) \neq i\)"

from \(\text{other quality increases} \{i\} \sigma \sigma'\)

have "\(\text{quality increases} (\sigma (\text{sip} (\sigma i))) (\sigma' (\text{sip} (\sigma' i)))\)"

by (rule otherE) \(\text{(clarsimp simp:} \langle \text{sip} (\sigma i) \neq i\rangle)\)

moreover from \(\langle\sigma, p\rangle \in \text{oreachable} \ (\text{opaodv} i) ?S ?U \ i \in \text{labels} \Gamma_{\text{AODV}} p\) and hyp

have "\(1 \leq \text{dsn} (\sigma' i)\)"

by (auto dest!: onl_oinvariant_weakenD [OF odsn_rrep]

simp add: sqsli simp: \(\sigma' i = \sigma i\)):\)

moreover from \(\text{\(\text{sip} (\sigma i) \neq i\) hyp' and pre}\)

have "\(\text{dip} (\sigma' i) \in kD (\text{rt} (\sigma (\text{sip} (\sigma i))))\)

\(\land \text{dsn} (\sigma' i) \leq \text{nsqn} (\text{rt} (\sigma (\text{sip} (\sigma i)))) (\text{dip} (\sigma' i))\)

\(\land (\text{nsqn} (\text{rt} (\sigma (\text{sip} (\sigma i)))) (\text{dip} (\sigma' i)) = \text{dsn} (\sigma' i))\)

\(\rightarrow \) the \(\text{(dhops} (\text{rt} (\sigma (\text{sip} (\sigma i)))) (\text{dip} (\sigma' i)) \leq \text{hops} (\sigma' i)\)

\(\lor \) the \(\text{(flag} (\text{rt} (\sigma (\text{sip} (\sigma i)))) (\text{dip} (\sigma' i)) = \text{inv})\)"

by (auto simp: \(\sigma' i = \sigma i\)):

ultimately show \?thesis

by (rule quality_increases_rreq_rrep_props)

next

assume "\(\text{sip} (\sigma i) = i\)" thus \?thesis

using \(\sigma' i = \sigma i\) hyp and pre by auto

qed

qed (auto simp add: sqsli simp elim!: quality_increases_rreq_rrep_props')

lemma \text{rerr_sip}:

\("\text{opaodv} i \vdash \text{(otherwith quality increases} \{i\} \text{orecvmsg msg_fresh),}

\text{other quality increases} \{i\} \rightarrow\)

\text{onl} \Gamma_{\text{AODV}} (\lambda(\sigma, 1).)

\(1 \in \{\text{PAodv}-:8, \text{PAodv}-:9, \text{PRerr}-:0, \text{PRerr}-:1\}\)

\(\rightarrow \langle \forall \text{rip} \in \text{dom(dests} (\sigma i)). \text{rip} \in kD (\text{rt} (\sigma (\text{sip} (\sigma i)))) \land

\text{the} (\text{dests} (\sigma i) \text{ rip} - 1 \leq \text{nsqn} (\text{rt} (\sigma (\text{sip} (\sigma i)))) \text{ rip})\rangle\)"

(is \("_ \vdash (??S, ??U \rightarrow) _\)"

proof -

\{ fix \text{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 \text{rip} \in \text{dom dests}. \text{rip} \in kD (\text{rt} (\sigma \text{ sip}))

\land \text{the} (\text{dests} \text{ rip} - 1 \leq \text{nsqn} (\text{rt} (\sigma \text{ sip})) \text{ rip})\)"

and "\text{dests} \text{ rip} = \text{Some} \text{ rsn}\"

from this(3) have "\text{rip} \in \text{dom dests}" by auto

with * and \(\text{dests} \text{ rip} = \text{Some} \text{ rsn}\) have "\text{rip} \in kD (\text{rt} (\sigma \text{ sip}))"

and "\text{rsn} - 1 \leq \text{nsqn} (\text{rt} (\sigma \text{ sip})) \text{ rip}" by (auto dest!: bspec)

from qinc have "\text{quality increases} (\sigma \text{ sip}) (\sigma' \text{ sip})" ..

have "\text{rip} \in kD (\text{rt} (\sigma' \text{ sip})) \land \text{rsn} - 1 \leq \text{nsqn} (\text{rt} (\sigma' \text{ sip})) \text{ rip}" proof

from \(\text{rip} \in kD (\text{rt} (\sigma \text{ sip}))\) and \(\text{quality increases} (\sigma \text{ sip}) (\sigma' \text{ sip})\)

show "\text{rip} \in kD (\text{rt} (\sigma' \text{ sip}))" ..

next

from \(\text{rip} \in kD (\text{rt} (\sigma \text{ sip}))\) and \(\text{quality increases} (\sigma \text{ sip}) (\sigma' \text{ sip})\)

have "\text{nsqn} (\text{rt} (\sigma \text{ sip})) \text{ rip} \leq \text{nsqn} (\text{rt} (\sigma' \text{ sip})) \text{ rip}" ..

with \(\text{rsn} - 1 \leq \text{nsqn} (\text{rt} (\sigma \text{ sip})) \text{ rip}\) show "\text{rsn} - 1 \leq \text{nsqn} (\text{rt} (\sigma' \text{ sip})) \text{ rip}" by (rule le_trans)

qed

\} note partial = \text{this}

show \?thesis
by (inv_cterms inv add: oseq_step_invariant_sterms [OF equality_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 conjI)
(clarsimp simp del: One_nat_def split: split_if_asm option.split_asm, erule(2) partial)+
qed

lemma prerr_guard: "paodv i \|= 
onl \Gamma_{\text{AODV}} (\lambda(\xi, 1). (1 = \text{PRerr-}1
\rightarrow (\forall ip:\text{dom}(\text{dests }\xi). ip:\text{vD}(rt \xi)
\wedge \text{the}(\text{nhop}(rt \xi) ip) = sip \xi
\wedge \text{sqn}(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 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\} (\text{orecmsg msg_fresh}),
\text{other quality_increases }\{i\} \rightarrow \text{onl } \Gamma_{\text{AODV}} (\lambda(\sigma, _). 
\forall dip. \text{let } nhip = \text{the}(\text{nhop}(rt (\sigma i)) dip)
\rightarrow dip \in \text{kD}(rt (\sigma (nhop dip))) \wedge \text{nsqn}(rt (\sigma i)) dip \le\text{nsqn}(rt (\sigma (nhop dip)) dip)"
(is "_ \|= (?S, ?U \rightarrow ) _")
proof -
{ fix nhop and \sigma \sigma' :: "ip \Rightarrow \text{state}"
assume pre: "\forall dip:\text{kD}(rt (\sigma i)). \text{nhop dip} \neq dip \rightarrow
dip:\text{kD}(rt (\sigma (\text{nhop dip}))) \wedge \text{nsqn}(rt (\sigma i)) dip \le\text{nsqn}(rt (\sigma (\text{nhop dip}))) dip"
and qinc: "\forall j. \text{quality_increases}(\sigma j) (\sigma' j)"
have "\forall dip:\text{kD}(rt (\sigma i)). \text{nhop dip} \neq dip \rightarrow
dip:\text{kD}(rt (\sigma (\text{nhop dip}))) \wedge \text{nsqn}(rt (\sigma i)) dip \le\text{nsqn}(rt (\sigma' (\text{nhop dip}))) dip"
proof (intro ballI impI)
fix dip
assume "dip:\text{kD}(rt (\sigma i))"
and "\text{nhop dip} \neq dip"
with pre have "dip:\text{kD}(rt (\sigma (\text{nhop dip})))"
and "\text{nsqn}(rt (\sigma i)) dip \le\text{nsqn}(rt (\sigma (\text{nhop dip}))) dip"
by auto
from qinc have qinc_nhop: "\text{quality_increases}(\sigma (\text{nhop dip})) (\sigma' (\text{nhop dip}))" ..
with \text{dip}\in\text{kD}(rt (\sigma (\text{nhop dip}))) have "\text{dip}\in\text{kD}(rt (\sigma' (\text{nhop dip})))" ..
moreover have "\text{nsqn}(rt (\sigma i)) dip \le\text{nsqn}(rt (\sigma' (\text{nhop dip}))) dip"
proof -
from \text{dip}\in\text{kD}(rt (\sigma (\text{nhop dip}))) qinc_nhop
have "\text{nsqn}(rt (\sigma (\text{nhop dip}))) dip \le\text{nsqn}(rt (\sigma' (\text{nhop dip}))) dip" ..
with \text{nsqn}(rt (\sigma i)) dip \le\text{nsqn}(rt (\sigma (\text{nhop dip}))) dip show ?thesis
by simp
qed
ultimately show "dip:\text{kD}(rt (\sigma' (\text{nhop dip})))
\wedge \text{nsqn}(rt (\sigma i)) dip \le\text{nsqn}(rt (\sigma' (\text{nhop dip}))) dip" ..
qed
} note basic = this
\begin{verbatim}
{ fix nhop and \( \sigma \sigma' :: "ip \Rightarrow state" \)
assume pre: "\( \forall \text{dip} \in kD(rt (\sigma i)). \text{nhop dip} \neq \text{dip} \Rightarrow \text{dip} \in kD(rt (\text{nhop dip})) \)
\wedge nsqn rt (\sigma i) dip \leq nsqn rt (\text{nhop dip})\)
and ndest: "\( \forall \text{ripc} \in dom \text{dests (\sigma i)}. \text{ripc} \in kD rt (\sigma (\text{sip (\sigma i)}))\)
\wedge the (\text{dests (\sigma i) ripc}) - 1 \leq nsqn rt (\sigma (\text{sip (\sigma i)}))\) ripc"
and issip: "\( \forall \text{ip} \in dom \text{dests (\sigma i)}. \text{nhop dip} = \text{sip (\sigma i)}\)"
and qinc: "\( \forall j. \text{quality_increases (\sigma j) (\sigma')}\)"

have "\( \forall \text{dip} \in kD rt (\sigma i). \text{nhop dip} \neq \text{dip} \Rightarrow \text{dip} \in kD (rt (\sigma' (\text{nhop dip})))\)
\wedge nsqn (invalidate (\text{rt (\sigma i)}) (\text{dests (\sigma i)})) dip \leq nsqn (rt (\sigma' (\text{nhop dip}))) dip" proof
\begin{proof}
\begin{itemize}
\item Assume "\( \text{dip} \in kD (rt (\sigma i))\)" and "\( \text{nhop dip} \neq \text{dip} \)"
\item with pre and qinc have "\( \text{dip} \in kD (rt (\sigma' (\text{nhop dip})))\)"
\item and "\( \text{nsqn (rt (\sigma i)) dip} \leq \text{nsqn (rt (\sigma' (\text{nhop dip}))) dip}\)"
\end{itemize}
by (auto dest!: basic)
\end{proof}

have "\( \text{nsqn (invalidate (\text{rt (\sigma i)}) (\text{dests (\sigma i)}) dip} \leq \text{nsqn (rt (\sigma' (\text{nhop dip}))) dip}\)"
proof (cases "\( \text{dip} \in kD (\text{dests (\sigma i)})\)"
assume "\( \text{dip} \in kD (\text{dests (\sigma i)})\)"
with \( \text{dip} \in kD (rt (\sigma i))\) obtain \( \text{dsn}\) where "\( \text{dests (\sigma i)} dip = \text{dsn}\)"
by auto
with \( \text{dip} \in kD (rt (\sigma i))\) have "\( \text{nsqn (invalidate (\text{rt (\sigma i)}) (\text{dests (\sigma i)}) dip = dsn - 1}\)"
by (rule nsqn_invalidate_eq)
moreover have "\( dsn - 1 \leq \text{nsqn (rt (\sigma' (\text{nhop dip}))) dip}\)"
proof -
from \( \text{dests (\sigma i)} dip = \text{dsn}\) have "\( \text{the (dests (\sigma i)} dip = \text{dsn} by simp\)"
with ndest and \( \text{dip} \in kD (\text{dests (\sigma i)})\) have "\( \text{dip} \in kD (rt (\sigma (\text{sip (\sigma i)})))\)"
\( \text{dsn - 1} \leq \text{nsqn (rt (\sigma (\text{sip (\sigma i)}))) dip}\)"
by auto
moreover from issip and \( \text{dip} \in kD (\text{dests (\sigma i)})\) have "\( \text{nhop dip} = \text{sip (\sigma i)}\) ..
ultimately have "\( \text{dip} \in kD (rt (\sigma' (\text{nhop dip})))\)"
and "\( dsn - 1 \leq \text{nsqn (rt (\sigma' (\text{nhop dip}))) dip}\)" by auto
with qinc show "\( dsn - 1 \leq \text{nsqn (rt (\sigma' (\text{nhop dip}))) dip}\)"
by simp (metis kD_nsqn_quality_increases_trans)
\end{proof}

\end{verbatim}
\[
dip \quad (is \quad \forall dip \in kD (rt (\sigma i)). \ ?P dip)
\]

**proof**

fix dip

assume "dip \in kD (rt (\sigma i))"

with a1 and a2

have "(the (nhop (rt (\sigma i)) dip) \neq dip \implies 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

} note nhop_update_sip = this

{ fix \sigma' oip sip osn hops

assume pre: "\forall dip \in kD (rt (\sigma i)). the (nhop (rt (\sigma i)) dip) \neq dip
\implies 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 qinc: "\forall j. quality_increases (\sigma j) (\sigma' j)"

and *: "sip \neq oip \implies oip \in kD (rt (\sigma sip))
\land osn \leq nsqn (rt (\sigma sip)) oip
\land (nsqn (rt (\sigma sip)) oip = osn
\implies the (dhops (rt (\sigma sip)) oip) \leq hops
\lor the (flag (rt (\sigma sip)) oip) = inv)"

from pre and qinc

have pre': "\forall dip \in kD (rt (\sigma i)). the (nhop (rt (\sigma i)) dip) \neq dip
\implies 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
\implies oip \in kD (rt (\sigma' (the (nhop (update (rt (\sigma i)) 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
\land (nsqn (update (rt (\sigma i)) oip (osn, kno, val, Suc hops, sip)) oip)))) oip)"

(is "\ ?nhop_not_oip \implies ?oip_in_kD \land ?nsqn_le_nsqn")

proof (rule, split update rt_split_asm)

assume "rt (\sigma i) = 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 \land \ ?nsqn_le_nsqn" ..

qed

} note update1 = this

{ fix \sigma' oip sip osn hops

assume pre: "\forall dip \in kD (rt (\sigma i)). the (nhop (rt (\sigma i)) dip) \neq dip
\implies 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 qinc: "\forall j. quality_increases (\sigma j) (\sigma' j)"

and *: "sip \neq oip \implies oip \in kD (rt (\sigma sip))
\land osn \leq nsqn (rt (\sigma sip)) oip
\land (nsqn (rt (\sigma sip)) oip = osn
\implies the (dhops (rt (\sigma sip)) oip) \leq hops
\lor the (flag (rt (\sigma sip)) oip) = inv)"

from pre and qinc

have pre': "\forall dip \in kD (rt (\sigma i)). the (nhop (rt (\sigma i)) dip) \neq dip
\implies 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 "∀ 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 byclarsimp
next
  assume "dip = oip"
  with rtnot ginc (dip ∈ kD(rt (σ i))) notdip *
  have "?dip_in_kD dip"
    by simp (metis kD_quality_increases)
  moreover from (dip = oip) rtnot ginc (dip ∈ kD(rt (σ 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 =
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 ∈ 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
gqed

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 ∈ 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
gqed

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:
"∀ 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 "\sigma \sigma' a dip sip i"

and a3: "\sigma \sigma' dip \in \nuD (\sigma (\nu t (\sigma i)))"

and a4: "\sigma (\nu t (\sigma i)) dip \neq dip"

show "\sigma (\nu t (\sigma i)) dip = \sigma (\nu t (\sigma i)) dip"

proof

\sigma (\nu t (\sigma i)) dip \neq dip

using a1 a2 a4 have "\sigma (\nu t (\sigma i)) dip \neq dip" by auto

have "\sigma (\nu t (\sigma i)) dip = \sigma (\nu t (\sigma i)) dip"

hence False by simp

thus ?thesis ..

next

assume "\sigma (\nu t (\sigma i)) dip \neq dip"

have "\sigma (\nu t (\sigma i)) dip = \sigma (\nu t (\sigma i)) dip"

hence False by simp

thus ?thesis ..

next

assume "\sigma (\nu t (\sigma i)) dip = dip"

have "\sigma (\nu t (\sigma i)) dip = dip"

hence False by simp

thus ?thesis ..

next

assume simp": "dip = dip"

have False by simp

thus ?thesis ..

next

assume simp": "dip \neq dip"

have False by simp

thus ?thesis ..

next

assume simp": "dip = dip"

have False by simp

thus ?thesis ..

next

assume simp": "dip \neq dip"

have False by simp

thus ?thesis ..
\{ 
fix \sigma \ a  \sigma' nhop 
assume pre: 
\[ \forall \text{dip. dip} \in vD(\text{rt (\sigma i)}) \wedge \text{dip} \in vD(\text{rt (\sigma (\text{nhop dip}))}) \wedge \text{nhop dip} \neq \text{dip} \rightarrow \text{rt (\sigma i) \cap dip} \text{rt (\sigma (\text{nhop dip}))} \]
and ow: "?? i \ sigma \ sigma' a"

have 
\[ \forall \text{dip. dip} \in vD (\text{invalidate (rt (\sigma i))) (dests (\sigma i))) \wedge \text{dip} \in vD (\text{rt (\sigma' (\text{nhop dip}))}) \wedge \text{nhop dip} \neq \text{dip} \rightarrow \text{rt (\sigma i) \cap dip} \text{rt (\sigma' (\text{nhop dip}))} \]

proof clarify
\hspace{1em} fix dip
\hspace{2em} assume "dip \in vD (\text{invalidate (rt (\sigma i))) (dests (\sigma i)))"
\hspace{3em} and "dip \in vD (\text{rt (\sigma' (\text{nhop dip}))})"
\hspace{4em} and "\text{nhop dip} \neq \text{dip}"
\hspace{1em} from this(1) have "dip \in vD (\text{rt (\sigma i))}"
\hspace{2em} by (clarsimp dest!: vD_invalidate_vD_not_dests)
\hspace{1em} moreover from ow have "\forall \text{j. j} \neq \text{i} \rightarrow \text{j} = \text{\sigma'} \text{j}" by auto
\hspace{1em} ultimately have "rt (\sigma i) \cap dip \text{rt (\sigma (\text{nhop dip}))}"
\hspace{2em} using pre \text{dip} \in vD (\text{rt (\sigma' (\text{nhop dip}))}) \text{\wedge nhop dip} \neq \text{dip}"
\hspace{3em} by metis
\hspace{4em} with \forall \text{j. j} \neq \text{i} \rightarrow \text{j} = \text{\sigma'} \text{j} show "rt (\sigma i) \cap dip \text{rt (\sigma' (\text{nhop dip}))}"
\hspace{5em} by (metis rt_strictly_fresher_i refl)
qed
\}

} 

note invalidate = this

\{ 
fix \sigma \ a  \sigma' dip oip osn sip hops i 
assume pre: 
\[ \forall \text{dip. dip} \in vD (\text{rt (\sigma i)}) \wedge \text{dip} \in vD (\text{rt (\sigma (\text{the (nhop (\text{rt (\sigma i))}) dip))}) \wedge \text{the (nhop (\text{rt (\sigma i))}) dip} \neq \text{dip} \rightarrow \text{rt (\sigma i) \cap dip} \text{rt (\sigma (\text{the (nhop (\text{rt (\sigma i))}) dip))})} \]
\hspace{1em} and ow: "?? i \ sigma \ sigma' a"
\hspace{2em} and "\text{Suc 0} \leq \text{osn}"
\hspace{3em} and a6: "\text{sip} \neq \text{oip} \rightarrow \text{oip} \in kD (\text{rt (\sigma sip)}) \land \text{osn} \leq \text{nsqn (rt (\sigma sip))} \land \text{(nsqn (rt (\sigma sip)) oip = osn} \rightarrow \text{the (dhops (rt (\sigma sip)) oip) \leq hops} \lor \text{the (flag (rt (\sigma sip)) oip) = inv)}"
\hspace{1em} and after: "\sigma' i = \sigma i (\text{rt := update (rt (\sigma i)) oip (osn, kno, val, Suc hops, sip)})" 
\hspace{2em} have "\forall \text{dip. dip} \in vD (\text{update (rt (\sigma i)) oip (osn, kno, val, Suc hops, sip)}) \wedge \text{dip} \in vD (\text{rt (\sigma') (the (nhop (update (rt (\sigma i)) oip (osn, kno, val, Suc hops, sip)) dip)))} \land \text{the (nhop (update (rt (\sigma i)) oip (osn, kno, val, Suc hops, sip)) dip) \neq dip} \rightarrow \text{update (rt (\sigma i)) oip (osn, kno, val, Suc hops, sip)} \cap dip \text{rt (\sigma' (the (nhop (update (rt (\sigma i)) oip (osn, kno, val, Suc hops, sip)) dip))})"

proof clarify
\hspace{1em} fix dip
\hspace{2em} assume a2: "dip \in vD (\text{update (rt (\sigma i)) oip (osn, kno, val, Suc (hops, sip))})"
\hspace{3em} and a3: "dip \in vD (\text{rt (\sigma') (the (nhop (update (rt (\sigma i)) oip (osn, kno, val, Suc hops, sip)) dip)))}"
\hspace{4em} and a4: "\text{the (nhop (update (rt (\sigma i)) oip (osn, kno, val, Suc hops, sip)) dip) \neq dip}"
\hspace{1em} from ow have a5: "\forall \text{j. j} \neq \text{i} \rightarrow \text{j} = \text{\sigma'} \text{j}" by auto
\hspace{2em} show "update (rt (\sigma i)) oip (osn, kno, val, Suc hops, sip) \cap dip rt (\sigma' (the (nhop (update (rt (\sigma i)) oip (osn, kno, val, Suc hops, sip)) dip)))" by (cases "?rt1 = rt (\sigma i)"
\hspace{3em} assume nochange [simp]: "update (rt (\sigma i)) oip (osn, kno, val, Suc hops, sip) = rt (\sigma i)"
\hspace{4em} from after have "\sigma' i = \sigma i" by simp
\hspace{5em} with a5 have "\forall \text{j. j} = \text{\sigma'} \text{j}" by metis
\hspace{3em} from a2 have "dip \in vD (rt (\sigma i))" by simp
\}
moreover from a3 have "dip ∈ vD(rt (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 (∀ 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 ∈ kD (rt (σ i))" by auto
moreover with a3 a5 after and (dip ≠ oip)
  have "dip ∈ kD(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 ∈ kD(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 ∈ kD(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 \( \text{osn} \leq \text{nsqn} (\text{rt} (\sigma \text{ sip})) \text{ oip} \)
    have "... \leq \text{nsqn} (\text{rt} (\sigma \text{ i})) \text{ oip} " by simp
also have "... \leq \text{sqn} (\text{rt} (\sigma \text{ i})) \text{ oip}"
    by (rule nsqn_sqn)
finally have "\text{sqn} (\text{rt} (\sigma \text{ i})) \text{ oip} < \text{sqn} (\text{rt} (\sigma \text{ i})) \text{ oip} ".
hence False by simp
thus "(\lambda a. \text{if } a = \text{ oip} \text{ then Some (osn, kno, val, Suc hops, i) } \\
    \text{else rt (\sigma \text{ i}) a)} = \text{rt (\sigma \text{ i})}" ..
next
assume "\text{sqn} (\text{rt} (\sigma \text{ i})) \text{ oip} = \text{osn}"
    and "\text{Suc hops} < \text{the (dhops (rt (\sigma \text{ i})) \text{ oip})}"
from this(1) and \( \text{oip} \in \text{vD (rt (\sigma \text{ sip}))} \) have "\text{nsqn (rt (\sigma \text{ i})) oip} = \text{osn}"
    by simp
with \( \text{nsqn} (\text{rt} (\sigma \text{ sip})) \text{ oip} = \text{osn} \longrightarrow \) \( \text{the (dhops (rt (\sigma \text{ sip})) \text{ oip})} \leq \text{hops} \) have "\text{the (dhops (rt (\sigma \text{ i})) \text{ oip})} \leq \text{hops} " by simp
with \( \text{Suc hops} < \text{the (dhops (rt (\sigma \text{ i})) \text{ oip})} \) have False by simp
thus "(\lambda a. \text{if } a = \text{ oip} \text{ then Some (osn, kno, val, Suc hops, i) } \\
    \text{else rt (\sigma \text{ i}) a)} = \text{rt (\sigma \text{ i})}" ..
next
assume "\text{the (flag (rt (\sigma \text{ i})) \text{ oip})} = \text{inv}"
with \( \text{the (flag (rt (\sigma \text{ sip})) \text{ oip})} = \text{val} \) have False by simp
thus "(\lambda a. \text{if } a = \text{ oip} \text{ then Some (osn, kno, val, Suc hops, i) } \\
    \text{else rt (\sigma \text{ i}) a)} = \text{rt (\sigma \text{ i})}" ..
next
from \( \text{oip} \in \text{kD(rt (\sigma \text{ sip}))} \)
    show "(\lambda a. \text{if } a = \text{ oip} \text{ then Some (the (\text{rt (\sigma \text{ i}) \text{ oip)})) } \\
    \text{else rt (\sigma \text{ i}) a)} = \text{rt (\sigma \text{ i})}" by (auto dest!: kD_Some)
qed
with change have False ..
thus ?thesis ..
qed
next
assume \( \text{oip} \in \text{I}(\text{rt (\sigma \text{ sip}))} \)
with \( \text{the (flag (\text{rt (\sigma \text{ i})}) \text{ oip})} = \text{val} \) and a5 have "\text{sip} = \text{i}"
    by (metis f.distinct(1) iD_flag_is_inv)
from \( \text{oip} \in \text{I}(\text{rt (\sigma \text{ sip}))} \) have "\text{the (flag (\text{rt (\sigma \text{ i})}) \text{ oip})} = \text{inv} " by auto
with \( \text{sip} = \text{i} ; \text{Suc 0} \leq \text{osn} \) change after \( \text{oip} \in \text{I}(\text{rt (\sigma \text{ sip}))} \)
    have "\text{nsqn (rt (\sigma \text{ sip})) \text{ oip} < nsqn (rt (\sigma \text{ ' sip})) \text{ oip}} "
    unfolding update_def
    by (clarsimp split: option.split_asm split_if_asm)
(auto simp: sqn_def)
with \( \text{osn} \leq \text{nsqn (rt (\sigma \text{ sip})) \text{ oip} } \) have "\text{osn < nsqn (rt (\sigma \text{ sip})) \text{ oip}} 
    by simp
thus ?thesis ..
qed
thus ?thesis
proof
    assume osnlt: "\text{osn} < \text{nsqn (rt (\sigma \text{ ' sip})) \text{ oip}} 
from \( \text{dip} \in \text{kd (rt (\sigma \text{ i}))} \) and \( \text{dip} = \text{oip} \) have "\text{dip} \in \text{kd (rt (\sigma \text{ ' sip}))} " by simp
moreover from a3 have "\text{dip} \in \text{kd (rt (\sigma \text{ ' sip}))} " by simp
moreover have "\text{nsqn ?rt1 dip} < \text{nsqn (rt (\sigma \text{ ' sip})) dip}"
    proof -
      have "\text{nsqn ?rt1 dip} = \text{osn} 
        by (simp add: \text{dip} = \text{oip} \text{ nsqn_update_changed_kno_val [OF change [THEN not_sym]]})
also have "... < \text{nsqn (rt (\sigma \text{ sip})) \text{ oip}} " using osnlt .
also have "... = \text{nsqn (rt (\sigma \text{ sip})) \text{ oip}} " by (simp add: change)
finally show ?thesis
      using \text{dip} = \text{oip} 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 ∈ 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 (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 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 (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

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} \rightarrow)
  onl Γ_{AODV}
  (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)"
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_o invariant weakenD [OF seq_compare_next_hop'])

from or and 11 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_o invariant 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 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 (σ ?nhip)) dip"
by (rule nsqns)
also have "... ≤ sqn (rt (σ ?nhip)) dip"
by (rule sqns)
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 ⟨∀ 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 ∉ νD(rt (σ ?nhip))"
with \(\langle dip ∉ kD(rt (σ ?nhip)) \rangle\) 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 \(\langle dip ∉ iD(rt (σ ?nhip)) \rangle\)
also have "... = nsqn (rt (σ ?nhip)) dip - 1" ..
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)) 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 \(\langle dip ∉ vD(rt (σ’ (the (nhop (rt (σ’ i)) dip)))\rangle\) and \(σ’ i = σ i\)
also have "... < sqn (rt (σ’ ?nhip)) dip" proof
from \(∀ j. quality_increases (σ j) (σ’ j)\)
have "quality_increases (σ i) (σ’ i)" ..
hence "∀ ip. sqn (rt (σ i)) ip ≤ sqn (rt (σ’ i)) dip" by auto
hence "sqn (rt (σ i)) dip ≤ sqn (rt (σ’ i)) dip" ..
with \(0 < sqn (rt (σ i)) dip\) show ?thesis by auto qed
finally have "nsqn (rt (σ i)) dip < nsqn (rt (σ’ ?nhip)) dip".
moreover from \(\langle dip ∉ νD(rt (σ’ (the (nhop (rt (σ’ i)) dip)))\rangle\) and \(σ’ i = σ i\)
ultimately show "rt (σ i) ⊏ rt (σ’ ?nhip)" using \(\langle dip ∉ kD(rt (σ i)) \rangle\) by - (rule rt_strictly_fresher_ltI)
with \(σ’ i = σ i\) show "rt (σ i) ⊏ 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 "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
Given the state of a network $\sigma$, 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 "$(ip, ip') \in rt_graph \sigma dip" shows "$ip \neq dip \land (\exists r. rt (\sigma ip) = r \land (\exists dsn dsk hops. r dip = Some (dsn, dsk, val, hops, ip')))"
using assms unfolding rt_graph_def by auto

**lemma rt_graph_vD [dest]:**

$\forall ip ip' \sigma dip. (ip, ip') \in rt_graph \sigma dip \Rightarrow dip \in vD(rt (\sigma ip))$
unfolding rt_graph_def vD_def by auto

**lemma rt_graph_vD_trans [dest]:**

$\forall ip ip' \sigma dip. (ip, ip') \in (rt_graph \sigma dip)\Rightarrow dip \in vD(rt (\sigma ip))$
by (erule converse_tranclE) auto

**lemma rt_graph_not_dip [dest]:**

$\forall ip ip' \sigma dip. (ip, ip') \in rt_graph \sigma dip \Rightarrow ip \neq dip$
unfolding rt_graph_def by auto

**lemma rt_graph_not_dip_trans [dest]:**

$\forall ip ip' \sigma dip. (ip, ip') \in (rt_graph \sigma dip)\Rightarrow ip \neq dip$
by (erule converse_tranclE) auto

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

**lemma rt_graph_nhip_is_nhop [dest]:**

$\forall ip ip' \sigma dip. (ip, ip') \in rt_graph \sigma dip \Rightarrow ip' = \text{the (nhop (rt (\sigma ip)) dip)}$
unfolding rt_graph_def by auto

**theorem inv_to_loop_freedom:**

assumes "\(\forall i dip. \text{let nhip = the (nhop (rt (\sigma i)) dip)}\)

in dip \in vD (rt (\sigma i)) \land vD (rt (\sigma nhip)) \land nhip \neq dip \rightarrow (rt (\sigma i)) \sqsubseteq dip (rt (\sigma 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: "\(\forall ip dip. \text{let nhip = the (nhop (rt (\sigma ip)) dip)}\)

in dip \in vD (rt (\sigma ip)) \land vD(rt (\sigma nhip)) \land nhip \neq dip \rightarrow rt (\sigma ip) \sqsubseteq dip rt (\sigma nhip)"

\{ fix ip ip' 
assume "\((ip, ip') \in (rt_graph \sigma dip)\)\" 
and "\(\text{dip} \in vD(rt (\sigma ip'))\)" 
and "\(\text{ip'} \neq \text{dip}\)"
hence "\(rt (\sigma ip) \sqsubseteq dip rt (\sigma ip')\)"
proof induction
fix nhip 
assume "\((ip, nhip) \in rt_graph \sigma dip\)" 
and "\(\text{dip} \in vD(rt (\sigma nhip))\)" 
and "\(\text{nhip} \neq \text{dip}\)"
from \((ip, nhip) \in rt_graph \sigma dip\) have "\(\text{dip} \in vD(rt (\sigma ip))\)" 
and "\(\text{nhip} = \text{the (nhop (rt (\sigma ip)) dip)}\)"
by auto
from \(\text{dip} \in vD(rt (\sigma ip))\) and \(\text{dip} \in vD(rt (\sigma nhip))\)
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: "[(nhip = the (nhop (rt (σ nhip)) dip)]
and (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

also have "rt (σ nhip) ⊑ dip rt (σ nhip')"
proof -
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)

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

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 s a s'
  assumes "((s, a), (s', a')) ∈ oparp_sos i (oseqp_sos Γ_AODV i) (seqp_sos Γ_QMSG)"
  and "a ≠ τ"
  shows "a' = s"
using assms by (rule qmsg_no_change_on_send_or_receive)

lemma par_nhop_quality_increases:
  shows "opaodv i (λ qmsg ⇒ (otherwith (op=) {i}) (orecvmsg (λ σ m.
msg_fresh σ m ∧ msg_zhops m)),
other quality_increases {i} →)
  end

325
lemma par_rreq_rrep_sn_quality_increases:

proof (rule ostep_invariant_weakenE [OF olocal_quality_increases], simp_all)

next

proof

|  shows 
|  shows 

|  by (erule weaken_otherwith, auto)

|  next

|  proof
|  proof

|  have
|  have

|  hence
|  hence

|  thus
|  thus

|  from
|  from

|  have
|  have

|  thus
|  thus

|  qed
|  qed

lemma par_rreq_rrep_sn_quality_increases:

proof -

have "opaqdv i ≒_A (otherwith (op=) {i})

globala (^iσ => | λσ. ∀dip. let nhp = the (nhop (rt (σ i)) dip)
in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhp)) ∧ nhp ≠ dip
→ (rt (σ i) ⊆ dip (rt (σ nhp))))"

proof (rule lift_into_qmsg [OF seq_nhop_quality_increases])

show "opaqdv i ≒_A (otherwith (op=) {i})

(orecvmsg (λm. 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 "call 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)

qed

lemma par_rreq_rrep_snqsn_fresh: any_step:

shows "opaqdv i ≒_A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ), other (λ_. True) {i} ⇒
globala (λ(σ, a, σ'). anycast (msg_fresh σ) a)"

proof -

have "opaqdv i ≒_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 olocal_quality_increases])

assume "call Γ_AODV (λ((σ, _, a, _)). anycast (msg_fresh σ) a) t"

thus "globala (λ(σ, a, σ'). anycast (msg_fresh σ) a) t"

by (cases t) (clarsimp dest!: onllD, metis aodv_ex_label)

qed auto

hence "opaqdv i ≒_A (λσ _. (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 par_anycast_msg_zhops:

shows "opaqdv i ≒_A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ), other (λ_. True) {i} ⇒
globala (λ(_, a, _). anycast msg_zhops a)"

proof -

from anycast_msg_zhops initial_aodv oadv_trans aodv_trans

have "opaqdv i ≒_A (act TT, other (λ_. True) {i} ⇒)

seqll i (onll Γ_AODV (λ(_, a, _). anycast msg_zhops a))"

by (rule open_seq_step_invariant)

hence "opaqdv i ≒_A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ), other (λ_. True) {i} ⇒

proof (rule ostep_invariant_weakenE)
  fix t :: "(((nat ⇒ state) × (state, msg, pseqp, pseqp label) seqp), msg seq_action) transition"
  assume "seqll i (onll Γ AODV (λ(_, a, _). anycast msg_zhops a)) t"
  thus "globala (λ(_, a, _). anycast msg_zhops a) t"
  by (cases t) (clarsimp dest!: seqllD 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} →
globala (λ(_, a, _). 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 "((σ, NodeS i P R), a, (σ', NodeS i' P' R')) ∈ onode_sos
  (oparp_sos i (oseqp_sos Γ AODV) (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 ⟩ o | =
        (otherwith (op=) {i})
        (arrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
        other_quality_increases (i) ⟩
        → (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 ⟩ o | =
        A (λσ _. oarrivemsg (λ_. rreq_rrep_sn) σ,
        other (λ_ _ True) {i} →
globala (λ(_, a, _). 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 ⟩ o | =
        A (λσ _ oarrivemsg (λ_. rreq_rrep_sn) σ,
        other (λ_ _ True) {i} →
globala (λ(_, a, _). castmsg (msg_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 (λ_. rreq_rrep_sn) σ,
        other (λ_ _ True) {i} →
globala (λ(_, 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 (λ_. rreq_rrep_sn) σ,
        other (λ_ _ True) {i} →
globala (λ(_ _, a, _). 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⟩ ∈ ooreachable (i : opaodv i ⟨⟨ i qmsg : R ⟩ o | =
        A (λσ _ oarrivemsg (λ_. rreq_rrep_sn) σ),
        other (λ_ _ True) {i} →
globala (λ(_ _, a, _). a ≠ τ → σ' i = σ i)"
  and tr: "((σ, ζ), a, (σ', ζ')) ∈ trans (i : opaodv i ⟨⟨ i qmsg : R ⟩ o | =
        A (λσ _ oarrivemsg (λ_. rreq_rrep_sn) σ),
        other (λ_ _ True) {i} →
globala (λ(_ _, a, _). a ≠ τ → σ' i = σ i)"
  and "a ≠ τ" from or obtain p R where "ζ = NodeS i p R" by (rule 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

3.11.3 Lift to partial networks

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

lemma opnet_nhop_quality_increases:
    shows "opnet (λi. opaodv i (⟨⟨i qmsg⟩⟩ o | = A (λσ m. oarrivemsg (λσ m. 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 pnet_lift [OF node_nhop_quality_increases])
      fix i R
      have "(i : opaodv i ⟨⟨i qmsg : R⟩⟩o = A (λσ m. oarrivemsg (λσ m. rreq_rrep_sn) σ, other (λ_ True) {i} →) globala (λ(σ, a, σ').
        castmsg (λm. msg_fresh σ m ∧ msg_zhops m) a)"
        by (rule ostep_invariantI, simp (no_asm))
      fix σ s a σ' s'
      assume or: "((σ, s) ∈ ooreachable ((i : opaodv i ⟨⟨i qmsg : R⟩⟩o)) (λσ m. oarrivemsg (λσ m. rreq_rrep_sn) σ), other (λ_ True) {i})" and tr: "(((σ, s), a, (σ', s')) ∈ trans ((i : opaodv i ⟨⟨i qmsg : R⟩⟩o))" 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_rreq_rrep_nsqn_fresh_any_step])
      moreover from or tr am have "castmsg (msg_zhops) 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

next
fix i R
show "(i : opaodv i ⟨⟨i qmsg : R⟩⟩o = A (λσ m. oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m) σ, other quality_increases {i} →) globala (λ(σ, a, σ').
      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 m. oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m) σ, other quality_increases {i} →) globala (λ(σ, a, σ').
      a ≠ τ ∧ (∀i d. a ≠ i:deliver(d)) → σ i = σ' i)"
    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 m. oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m) σ, other quality_increases {i} →) globala (λ(σ, a, σ').
      a = τ ∨ (∃d. a = i:deliver(d)) → quality_increases (σ i) (σ' 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 "onet_nhop_quality_increases = (λ_i. opaodv i ⟨⟨i qmsg⟩⟩ p
    \to (λ_ _ _ _ True, other quality_increases (net_tree_ips p) −→
    \to (rt (σ i)) ⊏ dip (rt (σ nhip)))" (is "_ = (?U → ?inv)"
proof (rule inclosed_closed)
  from opnet_nhop_quality_increases
  show "opnet (λ_i. 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 simp_all
  qed
  qed

proof (rule inclosed_closed)
  from opnet_nhop_quality_increases
  show "opnet (λ_i. 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 simp_all
  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, ⊥. (σ i, ⊥) ∈ σ_{AODV} i ∧ (∀ j. j ≠ i −→ σ j ∈ fst ' σ_{AODV} j) \subseteq σ_{AODV}"'
    unfolding σ_{AODV}_def σ_{AODV'}_def
  proof (rule equalityD1)
    show "∀ f p. (σ i, ⊥) ∈ {(f i, p)} ∧ (∀ j. j ≠ i
      −→ σ j ∈ fst ' {f j, p})\} = {(f, p)}" by (rule set_eqI) auto
  qed
  thus "{ σ, ⊥ | σ ⊥ s. s ∈ init (paodv i)
    ∧ (σ i, ⊥) = id s
    ∧ (∀ j. j ≠ i −→ σ j ∈ (fst o id) ' init (paodv j)) \} \subseteq 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') ∈ 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.netglobal (λi. paodv i ⟨⟨ qmsg) (λ(p, q). (fst (id p), snd (id p), q)) 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 ⟨⟨ 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 (λi. paodv i ⟨⟨ 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 ⟨⟨ qmsg) n) ∣|= netglobal (λσ. ∀ i ∈ net_tree_ips n. ∀ dip. 
let nhp = the (nhop (rt (σ i)) dip)
in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhp)) ∧ nhp ≠ dip 
→ (rt (σ i)) ⊏ dip (rt (σ nhp)))"
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],
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

3.11.6 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 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 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
imports 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" ("\(π_2\)")
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 \text{nat}" ("\pi_5")
where \( \pi_5 \equiv \lambda(_, _, _, hops, _, _). hops \)

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

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

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]: 
\[
[ [ P kno; P unk ] \] \Rightarrow P (\pi_3 x)
\]
by (rule k.induct)

lemma proj4_pred [intro]: 
\[
[ [ P val; P inv ] \] \Rightarrow 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

4.1.3 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_{\text{route}}(_, _, _)"")

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

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

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

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

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

abbreviation nhop :: "rt \Rightarrow ip \Rightarrow \text{ip}"
where \( \text{nhop} \equiv \text{map_option } \pi_6 (\sigma_{\text{route}}(rt, dip)) \)”
abbreviation \texttt{precs} :: "\(rt \Rightarrow ip \rightarrow ip\ \text{set}\)"
  where \(\text{precs} \ rt \ dip \equiv \text{map\_option} \ pi7 \ (\sigma\text{route}(\rt, \ dip))\)

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

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

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

lemma \texttt{kD} \_\texttt{is_vD} \_\texttt{and_iD}: \("kD \ rt = vD \ rt \cup iD \ rt\"
  unfolding \texttt{kD} \_\texttt{def} \ \texttt{vD} \_\texttt{def} \ \texttt{iD} \_\texttt{def} \ \text{by auto}\)

lemma \texttt{vD} \_\texttt{iD} \_\texttt{gives_kD} [simp]:
  "\(\forall \text{ip } \text{rt}. \text{ip} \in vD \text{ rt} \implies \text{ip} \in kD \text{ rt}\)"
  "\(\forall \text{ip } \text{rt}. \text{ip} \in iD \text{ rt} \implies \text{ip} \in kD \text{ rt}\)"
  unfolding \texttt{kD} \_\texttt{is_vD} \_\texttt{and_iD} \ \text{by simp_all}\)

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

lemma \texttt{kD} \_\texttt{None} [dest]:
  \(\text{fixes dip rt}\)
  assumes \("dip \notin kD \ rt\)"
  shows \("\sigma\text{route}(\rt, \ dip) = \text{None}\)"
  using \texttt{assms} unfolding \texttt{kD} \_\texttt{def} \ \text{by (metis (mono_tags) mem\_Collect_eq)}\)

lemma \texttt{vD} \_\texttt{Some} [dest]:
  \(\text{fixes dip rt}\)
  assumes \("dip \in vD \ rt\)"
  shows \("\exists dsn \ dsk \ \text{hops nhip pre}. \sigma\text{route}(\rt, \ dip) = \text{Some (dsn, dsk, val, hops, nhip, pre)}\)"
  using \texttt{assms} unfolding \texttt{vD} \_\texttt{def} \ \text{by simp}\)

lemma \texttt{vD} \_\texttt{empty} [simp]: \("vD \text{ Map.empty} = \{\}\"
  unfolding \texttt{vD} \_\texttt{def} \ \text{by simp}\)

lemma \texttt{iD} \_\texttt{Some} [dest]:
  \(\text{fixes dip rt}\)
  assumes \("dip \in iD \ rt\)"
  shows \("\exists dsn \ dsk \ \text{hops nhip pre}. \sigma\text{route}(\rt, \ dip) = \text{Some (dsn, dsk, inv, hops, nhip, pre)}\)"
  using \texttt{assms} unfolding \texttt{iD} \_\texttt{def} \ \text{by simp}\)

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

lemma \texttt{inv_is_iD} [elim]:
  \(\text{fixes ip rt}\)
  assumes \("ip \in kD(\rt)\)"
  \(\text{and } \"\text{the (flag rt ip) = inv}\)"
  shows \("ip \in iD(\rt)\)"
  using \texttt{assms} unfolding \texttt{iD} \_\texttt{def} \ \text{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 pre
    where rtip: "rt ip = Some (dsn, dsk, f, hops, nhop, pre)"
    by (metis kD_Some)
  from ⟨ip /∈ vD(rt)⟩ have "f ≠ val"
    proof (rule contrapos_nn)
      assume "f = val"
      with rtip have "the (flag rt ip) = val" by simp
    qed
  with rtip have "the (flag rt ip) = inv" by simp
  with ⟨ip ∈ kD(rt)⟩ show "ip ∈ iD(rt)" ..
qed

lemma vD_or_iD [elim]:
  fixes ip rt
  assumes "ip ∈ kD(rt)" and "ip ∈ vD(rt)" ⇒ P rt ip" and "ip ∈ iD(rt)" ⇒ P rt ip"
  shows "P rt ip"
proof -
  from ⟨ip ∈ kD(rt)⟩ have "ip ∈ vD(rt) ∪ iD(rt)"
    by (simp add: kD_is_vD_and_iD)
  thus ?thesis by (auto elim: assms(2-3))
qed

lemma proj5_eq_dhops: "∀ dip rt. dip ∈ kD(rt) ⇒ π_5 the (rt dip) = the (dhops rt dip)"
  unfolding sqn_def by (drule kD_Some) clarsimp
lemma proj4_eq_flag: "∀ dip rt. dip ∈ kD(rt) ⇒ π_4 the (rt dip) = the (flag rt dip)"
  unfolding sqn_def by (drule kD_Some) clarsimp
lemma proj2_eq_sqn: "∀ dip rt. dip ∈ kD(rt) ⇒ π_2 the (rt dip) = sqn rt dip"
  unfolding sqn_def by (drule kD_Some) clarsimp
lemma kD_sqnf_is_proj3 [simp]:
  "∀ ip rt. ip ∈ kD(rt) ⇒ sqnf rt ip = π_3 the (rt ip)"
  unfolding sqnf_def by auto

lemma vD_flag_val [simp]:
  "∀ dip rt. dip ∈ vD rt ⇒ the (flag rt dip) = val"
  unfolding vD_def by clarsimp
lemma kD_update [simp]:
  "∀ rt nip v. kD (rt nip v) = 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' ∨ 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 "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

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, and(addpre (dsn', v) pre)) = addpre (dsn, v) pre]\nunfolding addpre_def by clarsimp

lemma proj2_addpreRT [simp]:
fixes ip rt ip' npre
assumes "ip\(\in\)kD rt"
and "ip'\(\in\)kD rt"
shows "\(\pi_2\)\((the \ (the \ (addpre\RT \ rt \ ip' \ npre) \ ip)) = \pi_2\((the \ (rt \ ip))\)"
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 "\(\pi_3\)\((the \ (the \ (addpre\RT \ rt \ ip' \ npre) \ ip)) = \pi_3\((the \ (rt \ ip))\)"
using assms [THEN kD_Some] unfolding addpreRT_def by clarsimp

lemma proj5_addpreRT [simp]:
"\(\pi_5\)\((the \ (the \ (addpre\RT \ rt \ dip \ npre) \ ip)) = \pi_5\((the \ (rt \ ip))\)"
unfolding addpreRT_def by auto

lemma flag_addpreRT [simp]:
fixes rt pre ip dip
assumes "dip\(\in\)kD rt"
shows "flag \((the \ (addpre\RT \ 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 \((the \ (addpre\RT \ 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 \((the \ (addpre\RT \ 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 \((the \ (addpre\RT \ 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 \((the \ (addpre\RT \ 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 \((the \ (addpre\RT \ 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 \implies sqnf (the (addpreRT rt dip dip pp)) dip = sqnf (rt dip dip pp)"
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 \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 /∈ 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 /∈ kD rt"
  shows "sqn rt dip = 0"
using assms unfolding sqn_def by simp

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

lemma update_arg_wf_gives_cases: 
  "\r. update_arg_wf r \implies (\pi_2 (r) = 0) = (\pi_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]:
  "\n hops nhip pre. Suc 0 \leq n \Rightarrow 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 \Rightarrow 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 $wf3$: "$dsk = unk \rightarrow (hops = 1)$"

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 \( \langle flag = val \rangle \) 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
qed

definition update :: "rt ⇒ ip ⇒ r ⇒ rt"
where
\[ \text{update } rt \ ip \ r \equiv \begin{cases} \sigma_{\text{route}}(rt, ip) \Rightarrow rt (ip \mapsto r) & \text{if } σ_{\text{route}}(rt, ip) = \text{None} \\ \sigma_{\text{route}}(rt, ip) \Rightarrow rt (ip \mapsto \text{addpre } r (\pi_7(s))) & \text{else if } σ_{\text{route}}(rt, ip) = \text{Some } s \\ \text{if } \pi_2(s) < \pi_2(r) \text{ then } rt (ip \mapsto \text{addpre } r (\pi_7(s))) & \text{else if } \pi_3(r) = \text{unk} \\ \text{else if } \pi_2(s) = \pi_2(r) \wedge (\pi_5(s) > \pi_5(r) \lor \pi_4(s) = \text{inv}) & \text{else if } \pi_3(r) = \text{unk} \\ \text{then } rt (ip \mapsto \text{addpre } r (\pi_7(s))) & \text{then } rt (ip \mapsto \text{addpre } s (\pi_7(r))) \\ \text{else } rt (ip \mapsto \text{addpre } r (\pi_7(s))) & \text{else } rt (ip \mapsto \text{addpre } s (\pi_7(r))) \\ \end{cases} \]

lemma update_simps [simp]:
fixes $r$ $s$ $nr$ $nr'$ $ns$ $rt$ $ip$
defines "$s \equiv \text{the } σ_{\text{route}}(rt, ip)"
and "$nr \equiv \text{addpre } r (\pi_7(s))"
and "$nr' \equiv (\pi_2(s), \pi_3(nr), \pi_4(nr), \pi_5(nr), \pi_6(nr), \pi_7(nr))"
and "$ns \equiv \text{addpre } s (\pi_7(r))"
shows
\[ \begin{array}{ll}
[ ip / \notin kD(rt) ] & \Rightarrow \text{update } rt \ ip \ r = rt (ip \mapsto r) \\
[ ip \in kD(rt); \text{sqn rt ip} < \pi_2(r) ] & \Rightarrow \text{update } rt \ ip \ r = rt (ip \mapsto nr) \\
[ ip \in kD(rt); \text{sqn rt ip} = \pi_2(r); \text{the (dhops rt ip)} > \pi_5(r) ] & \Rightarrow \text{update } rt \ ip \ r = rt (ip \mapsto nr) \\
[ ip \in kD(rt); \text{sqn rt ip} = \pi_2(r); \text{flag rt ip} = \text{Some } inv ] & \Rightarrow \text{update } rt \ ip \ r = rt (ip \mapsto nr) \\
[ ip \in kD(rt); \pi_3(r) = \text{unk}; (\pi_2(r) = 0) = (\pi_3(r) = \text{unk})] & \Rightarrow \text{update } rt \ ip \ r = rt (ip \mapsto nr') \\
[ ip \in kD(rt); \text{sqn rt ip} \geq \pi_2(r); \pi_3(r) = \text{kno}; \text{sqn rt ip} = \pi_2(r) \Rightarrow \text{the (dhops rt ip)} \leq \pi_5(r) \wedge \text{the (flag rt ip)} = \text{val} ] & \Rightarrow \text{update } rt \ ip \ r = rt (ip \mapsto ns) \\
\end{array} \]

proof -
assume "ip /∈ kD(rt)"
hence "σ_{\text{route}}(rt, ip) = \text{None} " ..
thus "update rt ip r = rt (ip ⇒ r)"
unfolding update_def by simp
next
assume "ip \in kD(rt)"
and "\text{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 ⇒ 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 "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 = (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 auto
next
assume "ip ∈ kD(rt)"
and "sqn rt ip ≥ π₂(r)"
and "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: "[\[ \[ ip \notin kD(rt) \] \] \] = 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 \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)
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 ⟨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 ↦ addpre r (π7(πroute(rt, ip)))))
and c3: "[sqn rt ip = π2(r); the (dhops rt ip) > π5(r)]
⇒ P (rt (ip ↦ addpre r (π7(πroute(rt, ip)))))"
and c4: "[sqn rt ip = π2(r); the (flag rt ip) = inv]
⇒ P (rt (ip ↦ addpre r (π7(πroute(rt, ip)))))"
and c5: "π3(r) = unk ⇒ P (rt (ip ↦ (π2(πroute(rt, ip)), π3(r), π4(r), π5(r), π6(r), π7(adpre r (π7(πroute(rt, ip)))))))"
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 ↦ addpre (π7(πroute(rt, ip)))) (π7(r))))"
shows "(P (update rt ip r))"
using assms(1) proof (rule update_cases)
assume "sqn rt ip < π2(r)"
thus "P (rt(ip ↦ addpre r (π7(πroute(rt, ip))))))" by (rule c2)
next
assume "sqn rt ip = π2(r)"
and "the (dhops rt ip) > π5(r)"
thus "P (rt(ip ↦ addpre r (π7(πroute(rt, ip))))))" by (rule c3)
next
assume "sqn rt ip = π2(r)"
and "the (flag rt ip) = inv"
thus "P (rt(ip ↦ addpre r (π7(πroute(rt, ip))))))" by (rule c4)
next
assume "π3(r) = unk"
thus "P (rt (ip ↦ (π2(πroute(rt, ip)), π3(r), π4(r), π5(r), π6(r), π7(adpre r (π7(πroute(rt, ip)))))))" 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 ↦ addpre (π7(πroute(rt, ip)))) (π7(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: "\(v. rt\ dip = Some v \Rightarrow\)
  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
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 dip 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: "\(\forall ip\in\text{kD rt}. the (dhops rt ip) ≥ 1\)"
  and ip: "(ip = rip ∧ Suc 0 ≤ hops) ∨ (ip ≠ rip ∧ ip\in\text{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 \in \text{kD rt}") (drule(1) bspec, auto)
next
  assume "ip ≠ rip ∧ ip\in\text{kD rt}" thus ?thesis
  using ex unfolding update_def
  by (cases "rip\in\text{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 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 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, kno, flg, hops, sip, {})
  ⇒ 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 dip 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 dsk flg hops sip.
unfolding update_def sqnf_def
by (clarsimp split: option.splits split_if_asm) auto

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

lemma flag_update [simp]: "\(rt \neq \text{update} \ rt \ dip (dsn, \text{kno}, \text{flg}, \text{hops}, \text{sip}, \{}\)
\[\implies \text{the} (\text{flag} (\text{update} \ rt \ dip (dsn, \text{kno}, \text{flg}, \text{hops}, \text{sip}, \{})) \ dip) = \text{flg}\]
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 \text{kD} \ rt"
shows "\text{flag} \ rt \ ip = \text{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, \text{hops}, nhip, \text{pre})"
shows "\text{dip} \in \text{kD}(rt)"
proof -
have "\text{dip} \in \text{kD}(\text{update} \ rt \ dip (dsn, dsk, flag, \text{hops}, nhip, \text{pre}))" by simp
with assms show ?thesis by simp
qed

lemma nhop_update [simp]: "\(rt \neq \text{update} \ rt \ dip (dsn, dsk, flag \text{ hops} \text{sip})\)
\[\implies \text{the} (\text{nhop} (\text{update} \ rt \ dip (dsn, dsk, flag, \text{hops}, \text{sip}, \{})) \ dip) = \text{sip}\]
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 \text{dip}"
shows "\text{sqn} (\text{update} \ rt \ dip (dsn, dsk, flag, \text{hops}, \text{nhip}, \text{pre})) \ ip = \text{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 \text{dip}"
shows "\text{sqnf} (\text{update} \ rt \ dip (dsn, dsk, flag, \text{hops}, \text{nhip}, \text{pre})) \ ip = \text{sqnf} \ rt \ ip"
using assms unfolding update_def sqnf_def
by (clarsimp split: option.splits) auto

lemma vD_update_val [dest]:
"\(\text{dip} \in \text{vD} (\text{update} \ rt \ dip' (dsn, dsk, \text{val}, \text{hops}, nhip, \text{pre})) \implies (\text{dip} \in \text{vD}(rt) \lor \text{dip}=\text{dip}')\)"
unfolding update_def vD_def by (clarsimp split: option.split_asm split_if_asm)

Invalidating route entries

definition invalidate :: "rt \Rightarrow (ip \Rightarrow \text{sqn}) \Rightarrow rt"
where "\text{invalidate} \ rt \ dests \equiv"
\[\lambda \text{ip}. \text{case} (rt \ ip, \text{dests} \ ip) \ of\]
\[(\text{None}, \_ ) \Rightarrow \text{None}\]
lemma proj3_invalidate [simp]:
"\( \forall \text{ dip. } \pi_3(\text{the } (\text{invalidate rt dests} \text{ dip})) = \pi_3(\text{the } (\text{rt dip})) \)"
unfolding invalidate_def by (clarsimp split: option.split)

lemma proj5_invalidate [simp]:
"\( \forall \text{ dip. } \pi_5(\text{the } (\text{invalidate rt dests} \text{ dip})) = \pi_5(\text{the } (\text{rt dip})) \)"
unfolding invalidate_def by (clarsimp split: option.split)

lemma proj6_invalidate [simp]:
"\( \forall \text{ dip. } \pi_6(\text{the } (\text{invalidate rt dests} \text{ dip})) = \pi_6(\text{the } (\text{rt dip})) \)"
unfolding invalidate_def by (clarsimp split: option.split)

lemma proj7_invalidate [simp]:
"\( \forall \text{ dip. } \pi_7(\text{the } (\text{invalidate rt dests} \text{ dip})) = \pi_7(\text{the } (\text{rt dip})) \)"
unfolding invalidate_def by (clarsimp split: option.split)

lemma invalidate_kD_inv [simp]:
"\( \forall \text{ rt dests. } kD (\text{invalidate rt dests}) = kD \text{ rt} \)"
unfolding invalidate_def kD_def by (simp split: option.split)

lemma invalidate_sqn:
fixes rt dip dests
assumes "\( \forall \text{ rsn. dests dip = Some rsn } \rightarrow \text{ sqn rt dip } \leq \text{ rsn} \)"
shows "\( \text{ sqn rt dip } \leq \text{ sqn } (\text{invalidate rt dests} \text{ dip}) \)"
proof
  (cases "\( \text{ dip } \notin \text{ kD(rt)} \)"
    assume "\( \neg \text{ dip } \notin \text{ kD(rt)} \)"
    hence "\( \text{ dip } \in \text{ kD(rt)} \)" by simp
    then obtain dsn dsk flag hops nhip pre where "\( \text{ rt dip } = \text{ Some (dsn, dsk, flag, hops, nhip, pre)} \)"
      by (metis kD_Some)
    with assms show "\( \text{ sqn rt dip } \leq \text{ sqn } (\text{invalidate rt dests} \text{ dip}) \)"
      by (cases "\( \text{ dests dip } \)" (auto simp add: invalidate_def sqn_def))
  qed

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 "\( \text{ sqn } (\text{invalidate rt dests}) \text{ ipa } = \text{ rsn} \)"
unfolding invalidate_def sqn_def
using assms(1) assms(2) [THEN kD_Some]
by clarsimp

lemma dhops_invalidate [simp]:
"\( \forall \text{ dip. } \text{the } (\text{dhops } (\text{invalidate rt dests} \text{ dip})) = \text{the } (\text{dhops rt dip}) \)"
unfolding invalidate_def by (clarsimp split: option.split)

lemma sqnf_invalidate [simp]:
"\( \forall \text{ dip. } \text{sqnf } (\text{invalidate } (\text{rt } \xi) \text{ (dests } \xi)) \text{ dip } = \text{sqnf } (\text{rt } \xi) \text{ dip} \)"
unfolding sqnf_def invalidate_def by (clarsimp split: option.split)

lemma nhop_invalidate [simp]:
"\( \forall \text{ dip. } \text{the } (\text{nhop } (\text{invalidate } (\text{rt } \xi) \text{ (dests } \xi)) \text{ dip }) = \text{the } (\text{nhop } (\text{rt } \xi) \text{ dip}) \)"
unfolding invalidate_def by (clarsimp split: option.split)

lemma invalidate_other [simp]:
fixes rt dests dip
assumes "\( \text{ dip } \notin \text{ dom(dests)} \)"
shows "\( \text{ invalidate rt dests dip } = \text{ rt dip} \)"
using assms unfolding invalidate_def
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:
  "∀ 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 ⇒ 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 ∈ 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 ∈ 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)

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) ∈ rreqs} ∪ {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" ("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 ⇒ (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)"

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. 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

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 ip

instance by intro_classes

end

347
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 × bool ⇒ msg"
  where "rreq ≡ λ(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 :: "nat × ip × sqn × 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 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

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 × 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 ≡ (|
  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_not_globals [simp]:
  "ip (clear_locals ξ) = ip ξ"
4.3.2 Auxiliary 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' handled' ⇒
   | _ ⇒ {}"

**lemma is_rreq_asm [dest!]:**

assumes "ξ' ∈ is_rreq ξ"

shows "∃hops' rreqid' dip' dsn' dsk' oip' osn' sip' handled'.
    msg ξ = Rreq hops' rreqid' dip' dsn' dsk' oip' osn' sip' handled' ∧
    handled := handled'\}"

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' ⇒
   { ξ\{hops := hops', dip := dip', dsn := dsn', oip := oip', sip := 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 =
lemma \texttt{is_msg_inv_ip} [simp]:

\[
\text{\(\xi' \in \text{is\_rerr} \xi \Rightarrow ip \xi' = ip \xi\)}
\]

\[
\text{\(\xi' \in \text{is\_rrep} \xi \Rightarrow ip \xi' = ip \xi\)}
\]

\[
\text{\(\xi' \in \text{is\_rreq} \xi \Rightarrow ip \xi' = ip \xi\)}
\]

\[
\text{\(\xi' \in \text{is\_pkt} \xi \Rightarrow ip \xi' = ip \xi\)}
\]

\[
\text{\(\xi' \in \text{is\_newpkt} \xi \Rightarrow ip \xi' = ip \xi\)}
\]

unfolding \texttt{is_msg_defs}

by (cases "msg \(\xi\)", clarsimp+)+

lemma \texttt{is_msg_inv_sn} [simp]:

\[
\text{\(\xi' \in \text{is\_rerr} \xi \Rightarrow sn \xi' = sn \xi\)}
\]

\[
\text{\(\xi' \in \text{is\_rrep} \xi \Rightarrow sn \xi' = sn \xi\)}
\]

\[
\text{\(\xi' \in \text{is\_rreq} \xi \Rightarrow sn \xi' = sn \xi\)}
\]

\[
\text{\(\xi' \in \text{is\_pkt} \xi \Rightarrow sn \xi' = sn \xi\)}
\]

\[
\text{\(\xi' \in \text{is\_newpkt} \xi \Rightarrow sn \xi' = sn \xi\)}
\]

unfolding \texttt{is_msg_defs}

by (cases "msg \(\xi\)", clarsimp+)+

lemma \texttt{is_msg_inv_rt} [simp]:

\[
\text{\(\xi' \in \text{is\_rerr} \xi \Rightarrow rt \xi' = rt \xi\)}
\]

\[
\text{\(\xi' \in \text{is\_rrep} \xi \Rightarrow rt \xi' = rt \xi\)}
\]

\[
\text{\(\xi' \in \text{is\_rreq} \xi \Rightarrow rt \xi' = rt \xi\)}
\]

\[
\text{\(\xi' \in \text{is\_pkt} \xi \Rightarrow rt \xi' = rt \xi\)}
\]

\[
\text{\(\xi' \in \text{is\_newpkt} \xi \Rightarrow rt \xi' = rt \xi\)}
\]

unfolding \texttt{is_msg_defs}

by (cases "msg \(\xi\)", clarsimp+)+

lemma \texttt{is_msg_inv_rreqs} [simp]:

\[
\text{\(\xi' \in \text{is\_rerr} \xi \Rightarrow rreqs \xi' = rreqs \xi\)}
\]

\[
\text{\(\xi' \in \text{is\_rrep} \xi \Rightarrow rreqs \xi' = rreqs \xi\)}
\]

\[
\text{\(\xi' \in \text{is\_rreq} \xi \Rightarrow rreqs \xi' = rreqs \xi\)}
\]

\[
\text{\(\xi' \in \text{is\_pkt} \xi \Rightarrow rreqs \xi' = rreqs \xi\)}
\]

\[
\text{\(\xi' \in \text{is\_newpkt} \xi \Rightarrow rreqs \xi' = rreqs \xi\)}
\]

unfolding \texttt{is_msg_defs}

by (cases "msg \(\xi\)", clarsimp+)+

lemma \texttt{is_msg_inv_store} [simp]:

\[
\text{\(\xi' \in \text{is\_rerr} \xi \Rightarrow store \xi' = store \xi\)}
\]

\[
\text{\(\xi' \in \text{is\_rrep} \xi \Rightarrow store \xi' = store \xi\)}
\]

\[
\text{\(\xi' \in \text{is\_rreq} \xi \Rightarrow store \xi' = store \xi\)}
\]

\[
\text{\(\xi' \in \text{is\_pkt} \xi \Rightarrow store \xi' = store \xi\)}
\]

\[
\text{\(\xi' \in \text{is\_newpkt} \xi \Rightarrow store \xi' = store \xi\)}
\]

unfolding \texttt{is_msg_defs}

by (cases "msg \(\xi\)", clarsimp+)+

lemma \texttt{is_msg_inv_sip} [simp]:

\[
\text{\(\xi' \in \text{is\_pkt} \xi \Rightarrow sip \xi' = sip \xi\)}
\]

\[
\text{\(\xi' \in \text{is\_newpkt} \xi \Rightarrow sip \xi' = sip \xi\)}
\]

unfolding \texttt{is_msg_defs}

by (cases "msg \(\xi\)", clarsimp+)+

4.3.3 The protocol process

datatype pseqp =

\begin{itemize}
  \item \texttt{PAodv}
  \item \texttt{PNewPkt}
  \item \texttt{PPkt}
  \item \texttt{PRreq}
  \item \texttt{PRrep}
  \item \texttt{PRerr}
\end{itemize}

fun nat_of_seqp :: "pseqp ⇒ nat"

351
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
call(PPkt)"

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

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

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

abbreviation RERR
where
  "RERR args ≡
  [ξ. let (dests, sip) = args ξ in
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)"
Γ. ξ (rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ, {}) )]
RREQ(λξ. (hops ξ, rreqid ξ, dip ξ, sn ξ, oip ξ, osn ξ, sip ξ, handled ξ))

> (is_rrep)
[ξ. ξ (rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ, {}) )]
RREP(λξ. (hops ξ, dip ξ, dn ξ, oip ξ, sip ξ))

> (is_deliver)
[ξ. ξ (rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ, {}) )]
RREP(λξ. (dests ξ, sip ξ))

> (λξ. { ξ | dip := dip }) l dip. dip ∈ qD(store ξ) ∩ vD(rt ξ) )]
κ. ξ | data := hd(queue(store ξ, dip ξ)]
unicast(λξ. the (nhop (rt ξ) (dip ξ)), λξ. pkt(data ξ, dip ξ, ip ξ)).
κ. ξ | store := the (dip (dip ξ) (store ξ)]
AODV()

> (λξ. { ξ | dests := (λrip. if (rip ∈ vD (rt ξ) ∩ nhop (rt ξ) rip = nhop (rt ξ) (sip ξ))
then Some (inc (sqn (rt ξ) rip)) else None) )]
κ. ξ | rt := invalidate (rt ξ) (dests ξ)]
κ. ξ | store := setRRF (store ξ) (dests ξ)]
κ. ξ | pre := \{ the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) \}]
κ. ξ | dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ {})
then (dests ξ) rip else None) )]
groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)). AODV()"

> "\(Γ_{AODV}\) PNewPkt = labelled PNewPkt (κ. dip ξ = ip ξ)
deliver(λξ. data ξ).AODV()"

> "\(Γ_{AODV}\) PPkt = labelled PPkt (κ. dip ξ = ip ξ)
deliver(λξ. data ξ).AODV()"

> "\(Γ_{AODV}\) PPkt = labelled PPkt (κ. dip ξ ≠ ip ξ)
(κ. dip ξ ∈ vD (rt ξ))
unicast(λξ. the (nhop (rt ξ) (dip ξ)), λξ. pkt(data ξ, dip ξ, oip 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) )]
κ. ξ | rt := invalidate (rt ξ) (dests ξ)]
κ. ξ | store := setRRF (store ξ) (dests ξ)]
κ. ξ | pre := \{ the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) \}]
κ. ξ | dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ {})
then (dests ξ) rip else None) )]
groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)).AODV()"

> (ξ. dip ξ ∈ iD (rt ξ))
groupcast(λξ. the (precs (rt ξ) (dip ξ)), λξ. rerr([dip ξ ← sqn (rt ξ) (dip ξ)],
ip ξ)).AODV()"

> (ξ. dip ξ ∉ iD (rt ξ))
AODV()"

)"
Γ\AODV: \text{PRreq} = \text{labelled} \text{PRreq}
\begin{align*}
\{. \text{(oip } \xi, \text{ rreqid } \xi) \in \text{rreqs} \}
\end{align*}
\AODV()

\oplus \{. \text{(oip } \xi, \text{ rreqid } \xi) \notin \text{rreqs} \}
\begin{align*}
\{. \xi \in \text{rt} := \text{update} \ (\text{rt } \xi) \ (\text{oip } \xi) \ (\text{osn } \xi, \text{kno}, \text{val}, \text{hops } \xi + 1, \text{sip } \xi, \{\}) \}
\end{align*}
\AODV()

\oplus \{. \text{handled } \xi = \text{False} \}
\begin{align*}
(\xi. \text{ dip } \xi = \text{ip } \xi)
\begin{align*}
\{. \xi \in \text{sn} := \max \ (\text{sn } \xi) \ (\text{dsn } \xi) \}
\end{align*}
\text{unicast}(\lambda \xi. \text{ the } (\text{nhop } (\text{rt } \xi) \ (\text{oip } \xi)). \lambda \xi. \text{ rrep}(\text{0}, \text{ dip } \xi, \text{sn } \xi, \text{oip } \xi, \text{ip } \xi)).
\text{broadcast}(\lambda \xi. \text{ rreq}(\text{hops } \xi + 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()} \}
\end{align*}
\oplus \{. \text{dip } \xi \notin \text{ip } \xi \}
\begin{align*}
\{. \text{dip } \xi \in \text{vD } (\text{rt } \xi) \land \text{dsn } \xi \leq \text{sqn } (\text{rt } \xi) \ (\text{dip } \xi) = \text{kno} \)
\{. \xi \in \text{rt} := \text{the } (\text{addpreRT } (\text{rt } \xi) \ (\text{dip } \xi) \ {\text{sqn}} \ (\text{rt } \xi)) \}
\{. \xi \in \text{store} := \text{setRRF } (\text{store } \xi) \ (\text{dests } \xi) \}
\{. \xi \in \text{pre} := \bigcup \ {\text{the } (\text{precs } (\text{rt } \xi) \ (\text{dip } \xi))} \}
\{. \xi \in \text{dests} := (\lambda \text{rip. if } ((\text{dests } \xi) \ (\text{rreqid } \xi) \ (\text{ip } \xi)) \}
\{. \text{dip } \xi = \text{ip } \xi \}
\text{broadcast}(\lambda \xi. \text{ rreq}(\text{hops } \xi + 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()} \}
\oplus \{. \text{dip } \xi \notin \text{vD } (\text{rt } \xi) \lor \text{sqn } (\text{rt } \xi) \ (\text{dip } \xi < \text{dsn } \xi \lor \text{sqn} f (\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, \text{True})).
\text{AODV()} \}
\oplus \{. \text{handled } \xi = \text{True} \}
\text{broadcast}(\lambda \xi. \text{ rreq}(\text{hops } \xi + 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()} \}
\}
\)

Γ\AODV: \text{PRrep} = \text{labelled} \text{PRrep}
\begin{align*}
\{. \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, \{\}) \}
\end{align*}
\AODV()

\oplus \{. \text{oip } \xi \neq \text{ip } \xi \}
\begin{align*}
\{. \text{oip } \xi = \text{ip } \xi \}
\end{align*}
\AODV()

\oplus \{. \text{oip } \xi \neq \text{ip } \xi \}
\begin{align*}
\{. \text{oip } \xi \in \text{vD } (\text{rt } \xi) \}
\end{align*}
\AODV()
\[\begin{align*}
| \text{Lemma} \quad \Gamma & \text{ would be:} \\
\text{where} \\
| \text{show} \quad \Gamma & \text{ AODV()}
\end{align*}\]

\[\begin{align*}
\text{fun} \quad \Gamma_{\text{AODV-skeleton}} & \quad | \text{simp, code} = \Gamma_{\text{AODV-skeleton}} \quad | \text{simpdel, codesdel}\n\text{where} \\
| \text{lemma} \quad \Gamma_{\text{AODV-skeleton_wf}} & \quad | \text{simpdel, codesdel}\n\text{lemma} \quad \Gamma_{\text{AODV-skeleton.simps}} & \quad | \text{simpdel, codesdel}\n\text{lemma} \quad \text{aodv_proc_cases} & \quad | \text{dest}\n\end{align*}\]
\[ p \in \text{ctermsl} \left( \Gamma_{AODV} \text{PRrep} \right) \lor \]
\[ p \in \text{ctermsl} \left( \Gamma_{AODV} \text{PRerr} \right) \]

using \text{assms}
by \text{(cases \(pn\)) simp_all}

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

where "\( \sigma_{AODV} i \equiv \{(\text{aodv_init i, } \Gamma_{AODV} \text{PAodv})\} \)

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

where "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_within} \Gamma_{AODV} (\text{init} (paodv i))"

unfolding \( \sigma_{AODV} \_\text{def} \) by \text{(rule control_withinI) (auto simp del: \( \Gamma_{AODV} \_\text{simps} \))}

**lemma** aodv_wf [simp]: "\text{wellformed} \Gamma_{AODV}"

proof \text{(rule, intro allI)}
fix \(pn\) \(pn'\)
show "\(\text{call}(pn') \notin \text{stermsl} (\Gamma_{AODV} pn)\)"
by \text{(cases \(pn\)) simp_all}
qed

lemmas aodv_labels_not_empty [simp] = labels_not_empty [OF aodv_wf]

**lemma** aodv_ex_label [intro]: "\(\exists 1. 1 \in \text{labels} \Gamma_{AODV} p\)"

by \text{(metis aodv_labels_not_empty all_not_in_conv)}

**lemma** aodv_ex_labelE [elim]:
assumes "\(\forall 1 \in \text{labels} \Gamma_{AODV} p. P \quad p\)"
and "\(\exists p \; 1. \; P \quad p \quad \Rightarrow \; Q\)"
shows "Q"

using \text{assms} by \text{(metis aodv_ex_label)}

**lemma** aodv_simple_labels [simp]: "\text{simple_labels} \Gamma_{AODV}"

proof
fix \(pn\) \(p\)
assume "\(p \in \text{subterms}(\Gamma_{AODV} pn)\)"
thus "\(\exists 1. \text{labels} \Gamma_{AODV} p = \{1\}\)"
by \text{(cases \(pn\)) simp_all cong: seqp_congs | elim disjE)+
qed

**lemma** \( \sigma_{AODV} \_\text{labels} \) [simp]: "\((\xi, p) \in \sigma_{AODV} i \quad \Rightarrow \quad \text{labels} \Gamma_{AODV} p = \{PAodv-:0\}\)"

unfolding \( \sigma_{AODV} \_\text{def} \) by simp

**lemma** aodv_init_kD_empty [simp]: "\((\xi, p) \in \sigma_{AODV} i \quad \Rightarrow \quad \text{kD} (\text{rt} \xi) = \{\}\)"

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 \text{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 \text{assms} unfolding \( \sigma_{AODV} \_\text{def} \) 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 seq_congs [cong]

Configure the main invariant tactic for AODV.

declare 
  Γ_AODV_sims [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

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 _ _ _ _ _ _ _ _ _ _ oipc _ sipc _ ⇒ hopsc = 0 → oipc = sipc |
  Rrep hopsc dip _ _ _ _ _ _ _ _ sipc _ ⇒ hopsc = 0 → dip = sipc |
  _ ⇒ True"

definition 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 _ _ _ _ osnc _ ⇒ osnc ≥ 1 |
  _ ⇒ True"

end
lemma rreq_rrep_sn_simps [simp]:
  "∀ hops reqid dip dsn dsk oip osn sip handled.
    rreq_rrep_sn (Rreq hops reqid dip dsn dsk oip osn sip handled) = (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"
  "∀ dip sip. rreq_rrep_sn (Newpkt d dip) = True"
  "∀ 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 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 reqid dip dsn dsk oip osn sip handled.
    rreq_rrep_fresh crt (Rreq hops reqid dip dsn dsk oip osn sip handled) = (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 sip. rreq_rrep_fresh crt (Newpkt d dip) = True"
  "∀ 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 reqid dip dsn dsk oip osn sip handled.
    rerr_invalid crt (Rreq hops reqid dip dsn dsk oip osn sip handled) = True"
  "∀ hops dip dsn oip sip. rerr_invalid crt (Rrep hops dip dsn oip sip) = True"
  "∀ dests sip. rerr_invalid crt (Rerr dests sip) = (∀ rip ∈ dom(dests).
    rip ∈ iD(crt) ∧ the (dests rip) = sqn crt rip)"
  "∀ dip. rerr_invalid crt (Newpkt d dip) = True"
  "∀ 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 (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 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 ≡ (λs. 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 ⇒ sqn"
  where
  "nsqn r ≡ if \(\pi_4(r) = \text{val}\) \lor \(\pi_2(r) = 0\) then \(\pi_2(r)\) else \((\pi_2(r) - 1)\)"

lemma nsqnr_def':
  "nsqn r = (if \(\pi_4(r) = \text{inv}\) 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 ⇒ ip ⇒ sqn"
  where
  "nsqn ≡ λrt dip. case \(\sigma_{\text{route}}(rt, dip)\) 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 r (the (σroute(rt, dip)))"
using assms [THEN kD_Some] unfolding nsqn_def by clarsimp

lemma nsqnr_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 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_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 ≠ 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)

4.5.2 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 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 ⊑ 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

4.5.3 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 fresher_def by (simp add: kD_nsqn proj5_eq_dhops)

lemma rt_fresherI1 [intro]:
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 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 ] ⇒ 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 ≈_{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 "¬(\(rt1 \sqsubseteq_{ip} rt2\))"
proof
  assume "\(rt1 \sqsubseteq_{ip} rt2\)"
  hence "\(\text{the } (rt1 \text{ ip}) \subseteq \text{the } (rt2 \text{ ip})\)" ..
  with "¬(\(\text{the } (rt1 \text{ ip}) \subseteq \text{the } (rt2 \text{ ip})\))" 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 "\(\neg (rt1 \sqsubseteq_{ip} rt2)\)"
  shows "\(\neg (\text{the } (rt1 \text{ ip}) \sqsubseteq \text{the } (rt2 \text{ ip}))\)"
proof
  assume "\(\text{the } (rt1 \text{ ip}) \sqsubseteq \text{the } (rt2 \text{ ip})\)"
  hence "\(rt1 \sqsubseteq_{ip} rt2\)" ..
  with "\(\neg (rt1 \sqsubseteq_{ip} rt2)\)" show False ..
qed

lemma rt_fresher_mapupd [intro!]:
  assumes "\(\text{dip }\in kD(rt1)\)"
  and "\(\text{dip }\in kD(rt2)\)"
  and "\(rt1 \approx_{\text{dip}} rt2\)"
  shows "\(\text{nsqn, } (\text{the } (rt2 \text{ dip})) = \text{nsqn, } (\text{the } (rt1 \text{ dip}))\)"
using assms(3) unfolding rt_fresher_as_def
by (auto simp: rt_fresher_def2 [OF \(\text{dip }\in kD(rt1)\) \(\text{dip }\in kD(rt2)\)]
rt_fresher_def2 [OF \(\text{dip }\in kD(rt2)\) \(\text{dip }\in kD(rt1)\)]
kD_nsqn [OF \(\text{dip }\in kD(rt1)\)]
kD_nsqn [OF \(\text{dip }\in kD(rt2)\)]]

lemma rt_fresher_map_update_other [intro!]:
  assumes "\(\text{dip }\in kD(rt)\)"
  and "\(\text{the } (rt \text{ dip}) \subseteq r\)"
  shows "\(\text{rt }\subseteq_{\text{dip}} rt(dip \mapsto r)\)"
using assms unfolding rt_fresher_def by simp

lemma rt_fresher_update_other [simp]:
  assumes "\(\text{dip }\in kD(rt)\)"
  and "\(\text{dip }\neq ip\)"
  shows "\(\text{rt }\subseteq_{\text{dip}} update rt ip r\)"
using assms unfolding update_def
by (clarsimp split: option.split) (fastforce)

theorem rt_fresher_update [simp]:
  assumes "\(\text{dip }\in kD(rt)\)"
  and "\(\text{the } (dhops rt dip) \geq 1\)"
  and "\(\text{update_arg_wf } r\)"
  shows "\(\text{rt }\subseteq_{\text{dip}} update rt ip r\)"
proof (cases "\(\text{dip }= ip\)"
  assume "\(\text{dip }\neq ip\)" with \(\text{dip }\in kD(rt)\) show \(\text{?thesis\) by (rule rt_fresher_update_other)
next
  assume "\(\text{dip }= ip\)"

  from \(\text{dip }\in kD(rt)\) obtain \(dsn_n, dsk_n, f_n, hops_n, nhip_n, pre_n\)\n  where \(\text{rtn }\text{[simp]}: \"(\text{the } (dhops 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 \(\text{dip }\in kD(rt)\) have "\(hops_n \geq 1\)"

365
by (metis proj5_eq_dhops proj4(4))
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 ⟨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⟩
show "(dsn, dsk_n, inv, hops_n, nhip_n, pre_n)
≤ (dsn, kno, val, hops, nhip, pre ∪ pre_n)"
unfolding fresher_def by simp
qed
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∈kD(dests)"
assume "dip∈kD(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)⟩ 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))" by simp
  qed
  also from ⟨dip∈kD(invalidate rt dests)⟩
  have "nsqn_r (the (invalidate rt dests dip)) = nsqn (invalidate rt dests dip)" 
  by (simp add: kD_nsqn)
  ultimately show "nsqn rt dip = nsqn (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 (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) ∧ ¬(rt1 ≈ dip rt2)"

abbreviation
  rt_strictly_fresher_syn :: "rt ⇒ ip ⇒ rt ⇒ bool" ("(_/ ⊏/ _)" [51, 999, 51] 50)
  where 
  "rt1 ⊏ i rt2 ≡ rt_strictly_fresher i rt1 rt2"

lemma rt_strictly_fresher_def'':
  "rt1 ⊏ i rt2 = ((rt1 ⊑ i rt2) ∧ ¬(rt2 ⊑ i 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; ¬(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 \texttt{rt\_fresher\_trans\_rt\_strictly\_fresher} [elim, trans]:
assumes "\texttt{rt1} \sqsubseteq \texttt{dip} \texttt{rt2}"
and "\texttt{rt2} \sqsubseteq \texttt{dip} \texttt{rt3}"
shows "\texttt{rt1} \sqsubseteq \texttt{dip} \texttt{rt3}"
proof -
from \texttt{rt1} \sqsubseteq \texttt{dip} \texttt{rt2} have "\texttt{rt1} \sqsubseteq \texttt{dip} \texttt{rt2}"
and "\neg (\texttt{rt2} \sqsubseteq \texttt{dip} \texttt{rt1})"  
unfolding \texttt{rt\_strictly\_fresher\_def''} by auto
from this(1) and \texttt{rt2} \sqsubseteq \texttt{dip} \texttt{rt3} have "\texttt{rt1} \sqsubseteq \texttt{dip} \texttt{rt3}" ..
moreover from \texttt{\neg (rt2 \sqsubseteq dip rt1)} have "\texttt{\neg (rt3 \sqsubseteq dip rt1)}"
proof (rule contrapos_nn)
assume "\texttt{rt3} \sqsubseteq \texttt{dip} \texttt{rt1}"
with \texttt{rt2} \sqsubseteq \texttt{dip} \texttt{rt3} show "\texttt{rt2} \sqsubseteq \texttt{dip} \texttt{rt1}" ..
qed
ultimately show "\texttt{rt1} \sqsubseteq \texttt{dip} \texttt{rt3}"
unfolding \texttt{rt\_strictly\_fresher\_def''} by auto
qed

lemma \texttt{rt\_fresher\_trans\_rt\_strictly\_fresher'} [elim, trans]:
assumes "\texttt{rt1} \sqsubseteq \texttt{dip} \texttt{rt2}"
and "\texttt{rt2} \sqsubseteq \texttt{dip} \texttt{rt3}"
shows "\texttt{rt1} \sqsubseteq \texttt{dip} \texttt{rt3}"
proof -
from \texttt{rt2} \sqsubseteq \texttt{dip} \texttt{rt3} have "\texttt{rt2} \sqsubseteq \texttt{dip} \texttt{rt3}"
and "\neg (\texttt{rt3} \sqsubseteq \texttt{dip} \texttt{rt2})"  
unfolding \texttt{rt\_strictly\_fresher\_def''} by auto
from \texttt{rt1} \sqsubseteq \texttt{dip} \texttt{rt2} and this(1) have "\texttt{rt1} \sqsubseteq \texttt{dip} \texttt{rt3}" ..
moreover from \texttt{\neg (rt3 \sqsubseteq dip rt2)} have "\texttt{\neg (rt1 \sqsubseteq dip rt2)}"
proof (rule contrapos_nn)
assume "\texttt{rt3} \sqsubseteq \texttt{dip} \texttt{rt1}"
thus "\texttt{rt3} \sqsubseteq \texttt{dip} \texttt{rt2}" using \texttt{rt1} \sqsubseteq \texttt{dip} \texttt{rt2} ..
qed
ultimately show "\texttt{rt1} \sqsubseteq \texttt{dip} \texttt{rt3}"
unfolding \texttt{rt\_strictly\_fresher\_def''} by auto
qed

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

lemma \texttt{rt\_strictly\_fresher\_ltI} [intro]:
assumes "\texttt{dip} \in \texttt{kD}(\texttt{rt1})"
and "\texttt{dip} \in \texttt{kD}(\texttt{rt2})"
and "\texttt{nsqn} \texttt{rt1} \texttt{dip} < \texttt{nsqn} \texttt{rt2} \texttt{dip}"
shows "\texttt{rt1} \sqsubseteq \texttt{dip} \texttt{rt2}"
proof
from assms show "\texttt{rt1} \sqsubseteq \texttt{dip} \texttt{rt2}" ..
next
show "\neg (\texttt{rt1} \approx_{\texttt{dip}} \texttt{rt2})"  
proof
assume "\texttt{rt1} \approx_{\texttt{dip}} \texttt{rt2}"
hence "\texttt{rt2} \sqsubseteq \texttt{dip} \texttt{rt1}" ..
hence "\texttt{nsqn} \texttt{rt2} \texttt{dip} \leq \texttt{nsqn} \texttt{rt1} \texttt{dip}"
using \texttt{dip} \in \texttt{kD}(\texttt{rt2}) \texttt{dip} \in \texttt{kD}(\texttt{rt1})
by (rule \texttt{rt\_fresher\_imp\_nsqn\_le})
with \( \text{nsqn rt1 dip < nsqn rt2 dip} \) show "False"
by simp
qed
qed

lemma rt_strictly_fresher_eqI [intro]:
assumes "i \( \in \) kD(rt1)"
and "i \( \in \) kD(rt2)"
and "\text{nsqn rt1 i = nsqn rt2 i}"
and "\( \pi_5(\text{the (rt2 i)}) < \pi_5(\text{the (rt1 i)}) \)"
shows "\( \text{rt1 } \sqsubset^i \text{rt2} \)"
using assms unfolding rt_strictly_fresher_def' by (auto simp add: kD_nsqn)

lemma invalidate_rtstrf_left [simp]:
"\( \\forall \text{dests dip rt rt'}. \text{dests dip = None } \Rightarrow (\text{invalidate rt dests } \sqsubset^\text{dip} \text{rt'}) = (\text{rt } \sqsubset^\text{dip} \text{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 \( \in \) vD(invalidate rt1 dests)"
shows "(\text{invalidate rt1 dests } \sqsubset^\text{dip} \text{rt2}) = (\text{rt1 } \sqsubset^\text{dip} \text{rt2})"
proof (cases "dip \( \in \) \text{dom(dests)}")
assume "dip \notin \text{dom(dests)}"
hence "dip \notin vD(invalidate rt1 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!]:
"\( \\forall \text{ip rt rt'}. [ \text{dip } \neq \text{ip}; \text{rt } \sqsubset^\text{dip} \text{rt'} ] \Rightarrow \text{update rt ip r } \sqsubset^\text{dip} \text{rt'} "
unfolding rt_strictly_fresher_def'
by clarsimp

lemma addpreRT_strictly_fresher [simp]:
assumes "dip \( \in \) kD(rt)"
shows "(\text{the (addpreRT rt dip npre) } \sqsubset^\text{ip} \text{rt2}) = (\text{rt } \sqsubset^\text{ip} \text{rt2})"
using assms unfolding rt_strictly_fresher_def'
by clarsimp

lemma lt_sqn_imp_update_strictly_fresher:
assumes "dip \( \in \) 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 "\text{update rt dip (osn, kno, val, hops, nhip, {}) } \sqsubset^\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 } = \text{osn}"
by (rule nsqn_update_changed_kno_val)
with \( \text{dip } \in vD(\text{rt2 nhip}) \)
have "\text{nsqn (the (update rt dip (osn, kno, val, hops, nhip, {})) dip)} = \text{osn}"
by (simp add: kD_nsqn)
also have "\text{osn } < \text{sqn (rt2 nhip) dip}" by (rule *)
also have "\text{sqn (rt2 nhip) dip } = \text{nsqn (the (rt2 nhip dip))}"
unfolding nsqn_def using \( \text{dip } \in vD(\text{rt2 nhip}) \)
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) } < \text{nsqn (the (rt2 nhip dip))} ".

370
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 "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 (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, 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 ** 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

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 ||= onl 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':**

```plaintext
\( \text{paodv } i \vdash (\lambda ((\xi, \_), \_). (\xi', \_)) \cdot \text{sn } \xi \leq \text{sn } \xi') \)
```

by (rule step_invariant_weakenE [OF sequence_number_increases]) (auto dest!: onllD)

**Lemma sequence_number_one_or_bigger':**

```plaintext
\( \text{paodv } i \vdash (\lambda ((\xi, \_). 1 \leq \text{sn } \xi) \)
```

by (rule invariant_weakenE [OF sequence_number_one_or_bigger]) auto

**Lemma sip_in_kD':**

```plaintext
\( \text{paodv } i \vdash \text{onl } \Gamma_{AODV} (\lambda (\xi, 1). (1 \in \{ \text{PAodv} = 7 \cup \{ \text{PAodv} = 5 \} \cup \{ \text{PRrep} = 0..\text{PRrep} = 1 \} \) \cup \{ \text{PRreq} = 0..\text{PRreq} = 3 \} \rightarrow \text{sip } \xi \in \text{kD } (rt \xi))) \)
```

by inv_cterms

**Lemma rrep_1_update_changes':**

```plaintext
\( \text{paodv } i \vdash \text{onl } \Gamma_{AODV} (\lambda (\xi, 1). (1 = \text{PRrep} = 1 \rightarrow \text{rt } \xi \neq \text{update } (rt \xi) (\text{dip } \xi) (dsn \xi, \text{kno, val, hops } \xi + 1, \text{sip } \xi, \{\})))) \)
```

by inv_cterms

**Lemma addpreRT_partly_welldefined':**

```plaintext
\( \text{paodv } i \vdash \text{onl } \Gamma_{AODV} (\lambda (\xi, 1). (1 \in \{ \text{PRreq} = 18..\text{PRreq} = 20 \} \cup \{ \text{PRrep} = 2..\text{PRrep} = 6 \} \rightarrow \text{dip } \xi \in \text{kD } (rt \xi)) \wedge (l \in \{ \text{PRreq} = 3..\text{PRreq} = 19 \} \rightarrow \text{oip } \xi \in \text{kD } (rt \xi))) \)
```

by inv_cterms

**Proposition 7.38**

**Lemma includes_nhip:**

```plaintext
\( \text{paodv } i \vdash \text{onl } \Gamma_{AODV} (\lambda (\xi, 1). \forall \text{dip } \in \text{kD } (rt \xi). \text{the } (\text{nhop } (rt \xi) \text{ dip } ) \in \text{kD } (rt \xi))) \)
```

proof -

{ fix ip and \( \xi, \xi' : \text{ state} \)
  assume \( \forall \text{dip } \in \text{kD } (rt \xi). \text{ the } (\text{nhop } (rt \xi) \text{ dip } ) \in \text{kD } (rt \xi)\)
  and \( \xi' = \xi[(\text{rt } := \text{ update } (rt \xi) \text{ ip } 0, \text{ unk, val, Suc } 0, \text{ ip, } \{\})] \)
  hence \( \forall \text{dip } \in \text{kD } (rt \xi). \)
    the (nhop (update (rt \xi) ip 0, unk, val, Suc 0, ip, {})) dip = ip
    \lor the (nhop (update (rt \xi) ip 0, unk, val, Suc 0, ip, {})) dip \in \text{kD } (rt \xi)\)
  by clarsimp (metis nhop_update_unk_val_update_another)
}

note one_hop = this

{ fix ip sip sn hops and \( \xi, \xi' : \text{ state} \)
  assume \( \forall \text{dip } \in \text{kD } (rt \xi). \text{ the } (\text{nhop } (rt \xi) \text{ dip } ) \in \text{kD } (rt \xi)\)
  and \( \xi' = \xi[(\text{rt } := \text{ update } (rt \xi) \text{ ip } 0, \text{ unk, val, Suc hops, sip, } \{\})] \)
  and \( \text{sip } \in \text{kD } (rt \xi)\)
  hence \( (\text{the } (\text{nhop } (rt \xi) \text{ ip } 0, \text{ unk, val, Suc hops, sip, } \{\}) \text{ dip } ) = \text{ip} \)
  \lor (\text{the } (\text{nhop } (rt \xi) \text{ ip } 0, \text{ unk, val, Suc hops, sip, } \{\}) \text{ dip } ) \in \text{kD } (rt \xi)\)
    \wedge (\forall \text{dip } \in \text{kD } (rt \xi). \)
    the (nhop (update (rt \xi) ip 0, unk, val, Suc hops, sip, {})) dip = ip
    \lor the (nhop (update (rt \xi) ip 0, unk, val, Suc hops, sip, {})) dip \in \text{kD } (rt \xi)\)
  by (metis kD_update_unchanged nhop_update_unchanged update_another)
}

note nhop_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 nhop_is_sip)

qed

**Proposition 7.22: needed in Proposition 7.4**

**Lemma addpreRT_welldefined':**

```plaintext
\( \text{paodv } i \vdash \text{onl } \Gamma_{AODV} (\lambda (\xi, 1). (1 \in \{ \text{PRreq} = 18..\text{PRreq} = 20 \} \rightarrow \text{dip } \xi \in \text{kD } (rt \xi)) \wedge \)
\( (l = \text{PRreq} = 19 \rightarrow \text{oip } \xi \in \text{kD } (rt \xi)) \wedge \)
\( (l = \text{PRreq} = 5 \rightarrow \text{dip } \xi \in \text{kD } (rt \xi)) \wedge \)
\( (l = \text{PRreq} = 6 \rightarrow (\text{the } (\text{nhop } (rt \xi) \text{ dip } ) \in \text{kD } (rt \xi)))\)
```

(is \( - \vdash \text{onl } \Gamma_{AODV} ?P\))

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 l
assume "l ∈ labels Γ_{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 aodv_wf addpreRT_welldefined])
moreover from ⟨(ξ, p) ∈ reachable (paodv i) TT⟩ ⟨l ∈ labels Γ_{AODV} p⟩
and I3
have "l = PRrep-:6 −→ (the (nhop (rt ξ) (dip ξ))) ∈ kD(rt ξ)"
by (auto dest!: invariantD [OF includes_nhip])
ultimately show "?P (ξ, l)"
by simp
qed
with ⟨s = ((ξ, p))⟩
show "onl Γ_{AODV} ?P s"
by simp
qed

Proposition 7.4

lemma known_destinations_increase:
"paodv i |= onl 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 |= onl Γ_{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:
"∀ 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 |= onl Γ_{AODV} (λ((ξ, _), _, (ξ', _)). ∀ ip. sqn (rt ξ) ip ≤ sqn (rt ξ') ip)"

proof -
{ fix \( \xi :: \text{state} \)
  assume *: "\( \forall ip \in \text{dom}(\text{dests } \xi). \ ip \in kD \text{ (rt } \xi) \ \land \ \text{sqn (rt } \xi) \ \text{ip} \leq \text{the (dests } \xi \text{ ip}) \)"
  have "\( \forall ip. \ \text{sqn (rt } \xi) \ \text{ip} \leq \text{sqn (invalidate (rt } \xi) \ (\text{dests } \xi) \ ) \text{ ip} \)"
proof
  fix ip
  from * have "ip \notin \text{dom}(\text{dests } \xi) \lor \text{sqn (rt } \xi) \ \text{ip} \leq \text{the (dests } \xi \text{ ip})" by simp
  thus "\( \text{sqn (rt } \xi) \ \text{ip} \leq \text{sqn (invalidate (rt } \xi) \ (\text{dests } \xi) \ ) \text{ ip} \)"
  by (metis \text{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 \( \Gamma_{AODV} \ (\lambda(\xi, \_). \ \text{ip } \xi = i) \)"
by (inv_cterms simp add: \( \sigma_{AODV} \_\text{def} \))

Proposition 7.8

lemma sender_ip_valid':
"paodv i \|= _A onll \( \Gamma_{AODV} \ (\lambda((\xi, \_), a, \_). \text{ ancast (} \lambda m. \ \text{not_Pkt m } \rightarrow \text{msg_sender m } = \text{ip } \xi \text{ a)} \))"
by inv_cterms

lemma sender_ip_valid:
"paodv i \|= _A onll \( \Gamma_{AODV} \ (\lambda((\xi, \_), a, \_). \text{ ancast (} \lambda m. \ \text{not_Pkt m } \rightarrow \text{msg_sender m } = \text{i} \text{ a)} \))"
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 \rightarrow) onl \( \Gamma_{AODV} \ (\lambda(\xi, l). \ l \in \{\text{PAodv-:1} \} \rightarrow P (msg \ \xi)) \)"
by inv_cterms

Proposition 7.9

lemma sip_not_ip':
"paodv i \|= (recvmsg (\lambda m. \ \text{not_Pkt m } \rightarrow \text{msg_sender m } \neq \text{i}) \rightarrow) 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. \ \text{not_Pkt m } \rightarrow \text{msg_sender m } \neq \text{i}) \rightarrow) onl \( \Gamma_{AODV} \ (\lambda(\xi, \_). \ \text{sip } \xi \neq \text{i}) \)"
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 \( \Gamma_{AODV} \ (\lambda(\xi, \_). \ \forall ip \in kD \text{ (rt } \xi) \ \land \ \text{the (dhops (rt } \xi) \text{ ip} ) \geq 1) \)"
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 \( \Gamma_{AODV} \ (\lambda(\xi, l). \ l \in \{\text{PRreq-:18..PRreq-:21} \} \rightarrow \text{dip } \xi \in vD(\text{rt } \xi)) \land (l \in \{\text{PRreq-:6, PRreq-:7} \} \rightarrow \text{dip } \xi = \text{ip } \xi) \land (l \in \{\text{PRreq-:17..PRreq-:21} \} \rightarrow \text{dip } \xi \neq \text{ip } \xi)) \)"
proof (inv_cterms, elim conjE)
  fix i \ l pp p'
  assume "(\xi, pp) \in \text{reachable (paodv i) TT}"
and \(\{PRreq-:19\}[\lambda \xi. \xi() = \text{the (addpreRT (rt \xi) (oip \xi) \{the (nhop (rt \xi) (dip \xi)))\} \ p'} \in \text{sterms } \Gamma_{AODV} pp\]
and \("1 = PRreq-:19"
and \("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-:19"])
with \("dip \xi \in vD (rt \xi)"
show \("dip \xi \in vD (\text{the (addpreRT (rt \xi) (oip \xi) \{the (nhop (rt \xi) (dip \xi)))\})\)" by simp
qed

Proposition 7.11
lemma anycast_msg_zhops:
"\{rreqid dip dsn dsb oip osn sip. \paodv p \models \Gamma_{AODV} (\lambda (\xi, 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-:20\} \text{unicast (the (nhop (rt \xi) (oip \xi)))}
\lambda \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 \("1 = 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). Suc 0 \leq \text{the (dhops (rt \xi) ip)}"
and \("dip \xi \in vD (rt \xi)"
from \("dip \xi \in vD (rt \xi)"
have \("dip \xi \in kD (rt \xi)"
by (rule vD_iD_gives_kD(1))
with \(*\) have \("\forall \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:
"\paodv p \models \Gamma_{AODV} (\lambda (\xi, l). (l \in \{PAodv-:4..PAodv-:5\} \cup \{PRreq-:n n. True\} \rightarrow
\text{hops } \xi = 0 \rightarrow \text{oip } \xi = \text{sip } \xi)\)
A\((\xi, l) \in \{PAodv-:6..PAodv-:7\} \cup \{PRrep-:n n. True\} \rightarrow
\text{hops } \xi = 0 \rightarrow \text{dip } \xi = \text{sip } \xi)))\"
by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]) auto

lemma osn_rreq:
"\paodv p \models \Gamma_{AODV} (\lambda (\xi, l). (l \in \{PAodv-:4, PAodv-:5\} \cup \{PRreq-:n n. True\} \rightarrow 1 \leq \text{osn } \xi)\)
by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]) clarsimp

lemma osn_rreq':
"\paodv p \models \Gamma_{AODV} (\lambda (\xi, l). (l \in \{PAodv-:4, PAodv-:5\} \cup \{PRreq-:n n. True\} \rightarrow 1 \leq \text{osn } \xi)\)
by (rule invariant_weakenE [OF osn_rreq])
fix a
assume \("\text{recmsg } (\lambda m. \text{rreq_rrep_sn } m \land \text{msg_zhops } m) \ a\"
thus \("\text{recmsg } \text{rreq_rrep_sn } a\"
by (cases a) simp_all
qed

lemma dsn_rrep:
"\paodv p \models \Gamma_{AODV} (\lambda (\xi, l). (l \in \{PAodv-:6, PAodv-:7\} \cup \{PRrep-:n n. True\} \rightarrow 1 \leq \text{dsn } \xi)\)
by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]) clarsimp
lemma dsn_rrep':
"paodv i \vdash (recvmsg (\lambda m. rreq_rrep_sn m \land msg_zhops m) \rightarrow onl} \Gamma_{AODV} (\lambda (\xi, 1). l \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 \vdash (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)) ∧ ((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 \vdash (recvmsg (\lambda m. rreq_rrep_sn m \land msg_zhops m) \rightarrow onl} \Gamma_{AODV} (\lambda (\xi, _). \forall dip \in kD(rt \xi). (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))"
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 \implies P \xi dip"
have "P \xi dip" proof -
from all have "sqn (rt \xi) dip \leq sqn (invalidate (rt \xi) (dests \xi)) dip" ..
with \sqn (invalidate (rt \xi) (dests \xi)) dip = 0 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 \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

}
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 \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"
  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) = unk"
  by (rule update_cases, auto simp add: sqn_def dest!: bspec)
} note prreq_ok3 [simp] = this

{ fix rt sip
  assume "\(\forall dip \in 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))"\
  hence "\(\forall dip \in kD rt. (sqn (update rt sip (0, unk, val, Suc 0, sip, {})) dip = 0 \rightarrow\
             \pi_3 (the (update rt sip (0, unk, val, Suc 0, sip, {})) dip) = unk) \land\
             (\pi_3 (the (update rt sip (0, unk, val, Suc 0, sip, {})) dip) = unk \rightarrow (the (dhops (update rt sip (0, unk, val, Suc 0, sip, {})) dip) = Suc 0) \land\
             (the (dhops (update rt sip (0, unk, val, Suc 0, sip, {})) dip) = Suc 0 \rightarrow (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 sip rt. (\pi_3 (the (update rt sip (0, unk, val, Suc 0, sip, {})) sip) = unk \rightarrow (the (dhops (update rt sip (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 sip (0, unk, val, Suc 0, sip, {})) sip = 0 \rightarrow (\pi_3 (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_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 onl \Gamma_{AODV} (\lambda (\xi, _).\n          \forall dip \in kD(rt \xi). (sqn (rt \xi) dip = 0 \rightarrow (sqnf (rt \xi) dip = unk 
                             \land (the (dhops (rt \xi) dip) = Suc 0 
                             \land (the (nhop (rt \xi) 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 onl \Gamma_{AODV} (\lambda (\xi, 1).\n          \forall dip \in kD(rt \xi). (\pi_3 (the (rt \xi dip)) = unk \lor 1 \leq \pi_2 (the (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 (the (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)"

  377
unfolding \texttt{update\_def} by (cases "dsk1 = unk") \texttt{(clarsimp split: option.split)+}

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

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

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

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

\texttt{proof}

\texttt{fix dip}

\texttt{assume "dip \in kD(rt)"}

\texttt{thus "?prop dip"}

\texttt{proof (cases "dip = sip")}

\texttt{assume "dip = sip"}

\texttt{with ** show ?thesis}

\texttt{by simp}

next

\texttt{assume "dip \neq dip"}

\texttt{with \texttt{\langle dip \in kD(rt)\rangle allkd} show ?thesis}

\texttt{by simp}

\texttt{qed}

\texttt{qed}

\} note \texttt{solve\_update [simp] = this}

\{ fix dip rt dests

\texttt{assume \*: "\(\forall ip \in \text{dom(dests)}. ip \in kD(rt) \land \text{sqn rt ip} \leq \text{the (dests ip)}\)"

\texttt{and **: "\(\forall ip \in kD(rt). \pi_3\text{(the (rt ip))} = \text{unk} \lor \text{Suc 0} \leq \text{sqn rt ip}\)"

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

\texttt{proof}

\texttt{fix dip}

\texttt{assume "dip \in kD(rt)"}

\texttt{with ** have "\texttt{\pi_3\text{(the (rt dip))} = \text{unk} \lor \text{Suc 0} \leq \text{sqn rt dip}}" ..

\texttt{thus "\texttt{\pi_3\text{(the (rt dip))} = \text{unk} \lor \text{Suc 0} \leq \text{sqn (invalidate rt dests) dip}}"

\texttt{proof}

\texttt{assume "\texttt{\pi_3\text{(the (rt dip))} = \text{unk}}" \texttt{thus ?thesis ..

next

\texttt{assume "\texttt{Suc 0} \leq \text{sqn rt dip}"

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

\texttt{proof (cases "\texttt{dip \in \text{dom(dests)}\})"

\texttt{assume "dip \in \text{dom(dests)}"}

\texttt{with \* have "\texttt{sqn rt dip} \leq \text{the (dests dip)}" by simp}

\texttt{with \texttt{\langle Suc 0 \leq sqn rt dip\rangle have "\texttt{Suc 0} \leq \text{the (dests dip)}" by simp}

\texttt{with \texttt{\langle dip \in \text{dom(dests)}\rangle \langle dip \in kD(rt)\rangle \texttt{[THEN kD_Some]} show ?thesis}

\texttt{unfolding invalidate_def sqn_def by auto}

next

\texttt{assume "dip \notin \text{dom(dests)}"}

\texttt{with \texttt{\langle Suc 0 \leq sqn rt dip\rangle \langle dip \in kD(rt)\rangle \texttt{[THEN kD_Some]} show ?thesis}

\texttt{unfolding invalidate_def sqn_def by auto}

\texttt{qed}

\texttt{thus ?thesis by (rule disjI2)

\texttt{qed}

\texttt{qed}

\} note \texttt{solve\_invalidate [simp] = this}

\texttt{show ?thesis}

\texttt{by (inv\_cters inv add: onl\_invariant\_sterms\_TT [OF aodv\_wf addpreRT\_welldefined]

\texttt{onl\_invariant\_sterms [OF aodv\_wf dests\_biggerRT\_welldefined]

\texttt{[THEN invariant\_restrict\_inD]]

\texttt{onl\_invariant\_sterms [OF aodv\_wf osn\_rrreq]

\texttt{onl\_invariant\_sterms [OF aodv\_wf dsn\_rrrep]

\texttt{simp add: proj3\_inv proj2\_eq\_sqn)}

\texttt{qed}
Proposition 7.13

Lemma 7.13

```
"paodv i \vdash_A (recvmsg rreq_rrep_sn a) onll \Gamma_{AODV} (\lambda(_, a, _). anycast rreq_rrep_sn a)"
```

Proof -

```
have sqnf_kno: "paodv i \vdash onll \Gamma_{AODV} (\lambda(\xi, 1). (l \in \{PRreq-:18..PRreq-:20\} \longrightarrow sqnf (rt \xi) (dip \xi) = kD))"

by (inv_cterms inv add: onll_invariant_sterms_TT [OF aodv_wf addpreRT_welldefined])

show ?thesis

by (inv_cterms inv add: onll_invariant_sterms_TT [OF aodv_wf sequence_number_one_or_bigger]
      THEN invariant_restrict_inD)

onll_invariant_sterms [OF aodv_wf sqnf_kno]

onll_invariant_sterms [OF aodv_wf osn_rreq]

onll_invariant_sterms [OF aodv_wf sequence_number_one_or_bigger]

(auto simp: proj2_eq_sqn)
```

qed

Proposition 7.14

Lemma 7.14

```
"paodv i \vdash_A onll \Gamma_{AODV} (\lambda(\xi, 1). \_ anycast (rreq_rrep_fresh (rt \xi)) a)"
```

Proof -

```
have rreq_oip: "paodv i \vdash onll \Gamma_{AODV} (\lambda(\xi, 1). (l \in \{PRreq-:3..PRreq-:9\} \cup \{PRreq-:17, PRreq-:30, PRreq-:32\} 
               \longrightarrow oip \xi \in kD(rt \xi) 
               \land (sqn (rt \xi) (oip \xi) > (osn \xi)) 
               \lor (sqn (rt \xi) (oip \xi) = (osn \xi) 
               \land (dhops (rt \xi) (oip \xi)) \leq Suc (hops \xi) 
               \land (flag (rt \xi) (oip \xi) = val))))" 

proof inv_cterms

fix 1 l' pp p' 

assume "((\xi, pp) \in reachable (paodv i) TT" 

and "\{PRreq-:2\}\{\xi\}. \xi[rt := update (rt \xi) (oip \xi) (osn \xi, kD, Suc (hops \xi), sip \xi, \{}\}] p' \in terms \Gamma_{AODV} pp"

and "l' = PRreq-:3"

show "osn \xi < sqn (update (rt \xi) (oip \xi) (osn \xi, kD, Suc (hops \xi), sip \xi, \{}\}) (oip \xi) 
      \lor (sqn (update (rt \xi) (oip \xi) (osn \xi, kD, Suc (hops \xi), sip \xi, \{}\}) (oip \xi) = osn \xi 
      \land (dhops (update (rt \xi) (oip \xi) (osn \xi, kD, Suc (hops \xi), sip \xi, \{}\}) (oip \xi) 
      \leq Suc (hops \xi) 
      \land (flag (update (rt \xi) (oip \xi) (osn \xi, kD, 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 \vdash onll \Gamma_{AODV} (\lambda(\xi, 1). 
                        (l \in \{PRreq-:2..PRreq-:7\} \longrightarrow (dip \xi \in kD(rt \xi) 
                        \land sqn (rt \xi) (dip \xi) = dsn \xi 
                        \land (dhops (rt \xi) (dip \xi)) = Suc (hops \xi) 
                        \land (flag (rt \xi) (dip \xi)) = val 
                        \and (nhop (rt \xi) (dip \xi)) \in kD (rt \xi))))"

by (inv_cterms inv add: onll_invariant_sterms [OF aodv_wf rrep_1_update_changes]
      onll_invariant_sterms [OF aodv_wf sip_in_kD])

have rreq_oip_kD: "paodv i \vdash onll \Gamma_{AODV} (\lambda(\xi, 1). (l \in \{PRreq-:3..PRreq-:28\} \longrightarrow oip \xi \in kD(rt \xi))))"

by (inv_cterms inv add: onll_invariant_sterms_TT [OF aodv_wf addpreRT_welldefined])

have rreq_dip_kD_oip_sqn: "paodv i \vdash onll \Gamma_{AODV} (\lambda(\xi, 1). 
                        (l \in \{PRreq-:18..PRreq-:21\} 
                        \longrightarrow (dip \xi \in kD(rt \xi) 
                        \land (sqn (rt \xi) (oip \xi) > (osn \xi) 
                        \lor (sqn (rt \xi) (oip \xi) = (osn \xi) 
                        \land (dhops (rt \xi) (oip \xi)) \leq Suc (hops \xi) 
                        \land (flag (rt \xi) (oip \xi) = val))))"

379
Proposition 7.15

Lemma \textit{rerr_invalid_any_step_invariant}:

\begin{verbatim}
" paodv i ||= \Gamma\ AODV (\lambda((\xi, _), a, _). anycast (rerr_invalid (rt \xi)) a)"
\end{verbatim}

Proof -

\begin{verbatim}
have dests_inv: " paodv i ||= \Gamma\ AODV (\lambda((\xi, l). (l \in \{PAodv\-:15, PPkt\-:7, PRreq\-:11, PRreq\-:24, PRrep\-:10, PRerr\-:1\} \\
\rightarrow (\forall ip \in dom(dests \xi). ip \in vD(rt \xi))) \\
\wedge (l \in \{PAodv\-:16..PAodv\-:19\} \\
\cup \{PPkt\-:8..PPkt\-:11\} \\
\cup \{PRreq\-:12..PRreq\-:15\} \\
\cup \{PRreq\-:25..PRreq\-:28\} \\
\cup \{PRrep\-:11..PRrep\-:14\} \\
\cup \{PRerr\-:2..PRerr\-:5\} ightarrow (\forall ip \in dom(dests \xi). ip \in iD(rt \xi)) \\
\wedge \text{the(dests} \xi ip) = inc (sqn (rt \xi) ip)) \\
\wedge (l = PPkt\-:14 \rightarrow dip \xi \in iD(rt \xi))"
\end{verbatim}

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 \(\text{dests}\) 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 \textit{dests_vD_inc_sqn}:

\begin{verbatim}
" paodv i ||= \Gamma\ AODV (\lambda((\xi, l). (l \in \{PAodv\-:15, PPkt\-:7, PRreq\-:11, PRreq\-:24, PRrep\-:10, PRerr\-:1\} \\
\rightarrow (\forall ip \in dom(dests \xi). ip \in vD(rt \xi) \wedge \text{the(dests} \xi ip) = inc (sqn (rt \xi) ip))) \\
\wedge (l = PPkt\-:14 \rightarrow dip \xi \in iD(rt \xi))"
\end{verbatim}

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 7.27

Lemma \textit{route_tables_fresher}:

\begin{verbatim}
" paodv i ||= \Gamma\ AODV (\lambda((\xi, _), (\xi', _)). \forall dip \in kD(rt \xi). (rt \xi \in dip rt \xi')"
\end{verbatim}

proof (inv_cterms inv add:

\begin{verbatim}
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]]
\end{verbatim}
fix \( \xi \) pp p'
assume "(\( \xi \), pp) \in \text{reachable (paodv i) (recvmsg rreq_rrep_sn)}"
and "\{\{\text{Prreq}\rightarrow2\}\} \lambda \xi. \xi[rt := \text{update (rt} \xi) (\text{dip} \xi) (\text{osn} \xi, \text{kno, val, Suc (hops} \xi), \text{sip} \xi, \{\})]\] 
p' \in \text{sterms } \Gamma_{AODV} \text{ pp}"
and "\text{Suc 0} \leq \text{osn} \xi"
and*: "\( \forall \text{ip} \in kD \text{ (rt} \xi) \). \text{Suc 0} \leq \text{the (dhops (rt} \xi) \text{ ip)}"
show "\( \forall \text{ip} \in kD \text{ (rt} \xi) \). \text{rt} \xi \sqsubseteq_{ip} \text{update (rt} \xi) (\text{dip} \xi) (\text{osn} \xi, \text{kno, val, Suc (hops} \xi), \text{sip} \xi, \{\})"
p
proof
fix ip
assume "\text{ip} \in kD \text{ (rt} \xi)"
moreover with * have "1 \leq \text{the (dhops (rt} \xi) \text{ ip)}" by simp
moreover from :Suc 0 \leq osn \xi;
have "\text{update_arg_wf (osn} \xi, \text{kno, val, Suc (hops} \xi), \text{sip} \xi, \{\})" ..
ultimately show "\( \forall \text{ip} \in kD \text{ (rt} \xi) \). \text{rt} \xi \sqsubseteq_{ip} \text{update (rt} \xi) (\text{dip} \xi) (\text{osn} \xi, \text{kno, val, Suc (hops} \xi), \text{sip} \xi, \{\})"
by (rule rule_fresher_update)
qed

next
fix ip
assume "(\( \xi \), pp) \in \text{reachable (paodv i) (recvmsg rreq_rrep_sn)}"
and "\{\{\text{Prreq}\rightarrow2\}\} \lambda \xi. \xi[rt := \text{update (rt} \xi) (\text{dip} \xi) (\text{dsn} \xi, \text{kno, val, Suc (hops} \xi), \text{sip} \xi, \{\})]\] 
p' \in \text{sterms } \Gamma_{AODV} \text{ pp}"
and "\text{Suc 0} \leq \text{dsn} \xi"
and*: "\( \forall \text{ip} \in kD \text{ (rt} \xi) \). \text{Suc 0} \leq \text{the (dhops (rt} \xi) \text{ ip)}"
show "\( \forall \text{ip} \in kD \text{ (rt} \xi) \). \text{rt} \xi \sqsubseteq_{ip} \text{update (rt} \xi) (\text{dip} \xi) (\text{dsn} \xi, \text{kno, val, Suc (hops} \xi), \text{sip} \xi, \{\})"
p
proof
fix ip
assume "\text{ip} \in kD \text{ (rt} \xi)"
moreover with * have "1 \leq \text{the (dhops (rt} \xi) \text{ ip)}" by simp
moreover from :Suc 0 \leq dsn \xi;
have "\text{update_arg_wf (dsn} \xi, \text{kno, val, Suc (hops} \xi), \text{sip} \xi, \{\})" ..
ultimately show "\( \forall \text{ip} \in kD \text{ (rt} \xi) \). \text{rt} \xi \sqsubseteq_{ip} \text{update (rt} \xi) (\text{dip} \xi) (\text{dsn} \xi, \text{kno, val, Suc (hops} \xi), \text{sip} \xi, \{\})"
by (rule rule_fresher_update)
qed

end

4.7 The quality increases predicate

theory D_Quality_Increases
imports D_Aodv_Predicates D_Fresher
begin

definition quality_increases :: "state \Rightarrow state \Rightarrow bool"
where "quality_increases \( \xi \) \( \xi' \) \equiv (\forall \text{ip} \in kD \text{ (rt} \xi). \text{dip} \in kD \text{ (rt} \xi' \text{)} \land \text{rt} \xi \sqsubseteq_{ip} \text{rt} \xi')
\land (\forall \text{sqn (rt} \xi \text{) dip} \leq \text{sqn (rt} \xi\text{'}) \text{ dip})"

lemma quality_increasesI [intro!]:
assumes "(\( \forall \text{dip. dip} \in kD \text{ (rt} \xi) \Rightarrow dip \in kD \text{ (rt} \xi')\)"
and "(\( \forall \text{dip. [ dip} \in kD \text{ (rt} \xi); dip \in kD \text{ (rt} \xi') ] \Rightarrow rt \xi \sqsubseteq_{ip} rt \xi')\)"
and "(\( \forall \text{dip. sqn (rt} \xi \text{) dip} \leq \text{sqn (rt} \xi\text{'}) \text{ dip}\)"
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 "\( \text{dip} \in kD \text{ (rt} \xi) \)"
and "(\[ dip \in kD \text{ (rt} \xi'); rt \xi \sqsubseteq_{ip} rt \xi'; sqn (rt} \xi \text{) dip} \leq \text{sqn (rt} \xi\text{'}) \text{ dip} ] \Rightarrow \text{R dip} \xi \text{'}"
shows "\( \text{R dip} \xi \text{'})"

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 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"
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" by simp

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)) ip⟧ show ?thesis by simp

next

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

thus ?thesis by (metis (mono_tags) le_cases not_le)

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 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 rreqid dip dsn dsk oip osn sip handled.
  msg_fresh σ (Rreq hops rreqid dip dsn dsk oip osn sip handled) =
  (osn ≥ 1 ∧ (sip ≠ oip → 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))))"
lemma msg_fresh [simp]: "∀ hops dip dsn oip sip.
  msg_fresh σ (Rrep hops dip dsn oip sip) =
  (dsn ≥ 1 ∧ (sip ≠ dip → 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))))"
lemma msg_fresh [simp]: "∀ 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))"
lemma msg_fresh [simp]: "∀ d dip.
  msg_fresh σ (Newpkt d dip) = True"
lemma msg_fresh [simp]: "∀ 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_msg_fresh [simp]: fixes σ msg rreqid dip dsn ds dsk oip osn sip handled
  assumes "rreq_rrep_fresh (rt (σ sipc)) (Rreq hops rreqid dip dsn ds dsk oip osn sip handled)"
  and "rreq_rrep_sn (Rreq hops rreqid dip dsn ds dsk oip osn sip handled)"
  shows "msg_fresh σ (Rreq hops rreqid dip dsn ds dsk oip osn sip handled)"
  (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 "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"
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 = \text{"rt (\( \sigma \) sip)"} \)
from assms have \( \text{*: "(\( \forall \) rip\( \in \) dom(dests). (rip\( \in \)D(rt (\( \sigma \) sip))) \land \text{the (dests rip) = sqn (rt (\( \sigma \) sip)) rip))"} \)
by clarsimp
have 
\( \text{"(\( \forall \) rip\( \in \) dom(dests). (rip\( \in \)D(rt (\( \sigma \) sip))) \land \text{the (dests rip) - 1 \leq nsqn (rt (\( \sigma \) sip)) rip)"} \)
proof
fix rip
assume "rip \in dom dests"
with \( \text{*} \) have "rip\( \in \)D(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 \text{rip\( \in \)D(rt (\( \sigma \) sip))} \rangle \)
show "rip\( \in \)D(rt (\( \sigma \) sip)) \land the (dests rip) - 1 \leq nsqn (rt (\( \sigma \) sip)) rip" by clarsimp
qed thus "msg_fresh \( \sigma \) ?msg"
by simp
qed

lemma quality_increases_msg_fresh [elim]:
assumes qinc: \( \forall j. \text{quality_increases (\( \sigma \) j) (\( \sigma \)' j)} \)
and "msg_fresh \( \sigma \) m"
shows "msg_fresh \( \sigma \)' m"
proof (cases m)
fix hops rreqid dip dsn dsk oip osn sip handled
assume [simp]: "m = Rreq hops rreqid dip dsn dsk oip osn sip handled"
and "msg_fresh \( \sigma \) m"
then have "osn \geq 1" and "sip = oip \lor (oip\( \in \)D(rt (\( \sigma \) sip))) \land osn \leq nsqn (rt (\( \sigma \) sip)) oip
\land (nsqn (rt (\( \sigma \) sip)) oip = osn
\rightarrow (the (dhops (rt (\( \sigma \) sip)) oip) \leq hops
\lor the (flag (rt (\( \sigma \) sip)) oip) = inv)))"
by auto
from this(2) show ?thesis
proof
assume "sip = oip" with \( \langle \text{osn \geq 1} \rangle \) show ?thesis by simp
next
assume "oip\( \in \)D(rt (\( \sigma \) sip))) \land osn \leq nsqn (rt (\( \sigma \) sip)) oip
\land (nsqn (rt (\( \sigma \) sip)) oip = osn
\rightarrow (the (dhops (rt (\( \sigma \) sip)) oip) \leq hops
\lor the (flag (rt (\( \sigma \) sip)) oip) = inv)))"
moreover from qinc have "quality_increases (\( \sigma \) sip) (\( \sigma \)' sip)" ..
ultimately have "oip\( \in \)D(rt (\( \sigma \)' sip))) \land osn \leq nsqn (rt (\( \sigma \)' sip)) oip
\land (nsqn (rt (\( \sigma \)' sip)) oip = osn
\rightarrow (the (dhops (rt (\( \sigma \)' sip)) oip) \leq hops
\lor the (flag (rt (\( \sigma \)' sip)) oip) = 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 "msg_fresh \( \sigma \)' m"
by (clarsimp)
qed
next
fix hops dip dsn oip sip
assume [simp]: "m = Rrep hops dip dsn oip sip"
and "msg_fresh \( \sigma \) m"
then have "dsn \geq 1" and "sip = dip \lor (dip\( \in \)D(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)))"
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

4.8 The ‘open’ AODV model

theory D_OAodv
imports D_Aodv "./.../AWN/OAWN_SOS_Labels" "./.../AWN/OAWN_Convert"
beginn
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 aodv_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' =⇒ labels Γ_AODV p = {PAodv :-0}"  
  unfolding σ_AODV'_def by simp
lemma aodv_init_kD_empty [simp]:
"(\sigma, p) \in AODV' \Rightarrow kD (rt (\sigma i)) = {}"
unfolding AODV'_def kD_def by simp

lemma oaodv_init_vD_empty [simp]:
"(\sigma, p) \in AODV' \Rightarrow vD (rt (\sigma i)) = {}"
unfolding AODV'_def vD_def by simp

lemma oaodv_trans: "trans (opaodv i) = oseqp_sos \Gamma_{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

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 m. P \sigma m \Rightarrow P' \sigma m"
and weakenQ: "\forall m. Q \sigma m \Rightarrow Q' \sigma m"
shows "otherwith P' I (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 "orecvmsg Q \sigma a" by auto
thus "orecvmsg Q' \sigma a" by rule (erule weakenQ)
qed

lemma orceived_msg_inv:
assumes other: "\forall \sigma \sigma'. [ P \sigma m; other Q \{i\} \sigma \sigma' ] \Rightarrow P \sigma' m"
and local: "\forall \sigma. P \sigma m \Rightarrow P (\sigma (i := \sigma i[msg := m])) m"
shows "opaodv i \models (otherwith Q \{i\} (orecvmsg P), other Q \{i\} \rightarrow)\nonly \Gamma_{AODV} (\lambda(\sigma, l). l \in \{PAodv-:1\} \rightarrow P \sigma (msg (\sigma i)))"
proof (inv_cterms, intro impl)
fix \sigma \sigma' l
assume "l = PAodv-:1 \rightarrow P \sigma (msg (\sigma i))"
and "l = PAodv-:1"
and "other Q \{i\} \sigma "
from this(1-2) have "P \sigma (msg (\sigma i))" ..
hence "P \sigma' (msg (\sigma i))" using other Q \{i\} \sigma \sigma' by (rule other)
moreover from other Q \{i\} \sigma \sigma' have "\sigma' i = \sigma i" ..
ultimately show "P \sigma' (msg (\sigma' i))" by simp
fix \( \sigma \sigma' \) msg
assume "otherwith \( Q \{i\} \) (orecvmsg \( P \) \( \sigma \sigma' \) (receive msg))"
and "\( \sigma' \) \( i = \sigma \) \{msg := msg\}"
from this(1) have \( P \sigma \) msg
and "\( \forall \ j \neq i \rightarrow Q (\sigma j) (\sigma' j) \)" by auto
from this(1) have "\( P (\sigma (i := \sigma \{msg := msg\})) msg \)" by (rule local)
thus "\( P \sigma' \) msg"
proof (rule other)
from \( \langle \sigma' \) \( i = \sigma \) \{msg := msg\} \rangle 
and \( \langle \forall \ j \neq i \rightarrow Q (\sigma j) (\sigma' j) \rangle \)
show "other \( Q \{i\} (\sigma (i := \sigma \{msg := msg\}))) \sigma'" 
by - (rule otherI, auto)
qed

(Qualify increases)

lemma local_quality_increases:
"(recvmsg rreq_rrep_sn \( \rightarrow \)) onll \( \Gamma_{AODV} \) (\( \lambda((\xi, \_), \_, (\xi', \_)). \) quality_increases \( \xi \xi' \))"
proof (rule step_invariantI)
fix \( s a s' \)
assume sr: "\( s \in \) reachable (opaodv \( i \)) (recvmsg rreq_rrep_sn)"
and tr: "\( (s, a, s') \in \) trans (opaodv \( i \))"
and rm: "recvmsg rreq_rrep_sn \( a \)"
from sr have srTT: "\( s \in \) reachable (opaodv \( i \)) TT" ..
from route_tables_fresher sr tr rm
have "onll \( \Gamma_{AODV} \) (\( \lambda((\xi, \_), \_, (\xi', \_)). \) \( \forall \) dip \in kD (rt \( \xi \)) rt \( \xi \) \( \subseteq \) dip rt \( \xi' \) \( ) (s, a, s')\)"
by (rule step_invariantD)
moreover from known_destinations_increase srTT tr TT_True
have "onll \( \Gamma_{AODV} \) (\( \lambda((\xi, \_), \_, (\xi', \_)). \) kD (rt \( \xi \)) \( \subseteq \) kD (rt \( \xi' \)) \( ) (s, a, s')\)"
by (rule step_invariantD)
moreover from sqns_increase srTT tr TT_True
have "onll \( \Gamma_{AODV} \) (\( \lambda((\xi, \_), \_, (\xi', \_)). \) \( \forall \) ip. sqn (rt \( \xi \)) \( \leq \) sqn (rt \( \xi' \)) \( ip \) \( ) (s, a, s')\)"
by (rule step_invariantD)
ultimately show "onll \( \Gamma_{AODV} \) (\( \lambda((\xi, \_), \_, (\xi', \_)). \) 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 oaodv_trans aodv_trans, simplified seqll_onll_swap]

lemma equality_increases:
"opaodv \( i \models (\text{recvmsg } \_ \_ \rightarrow) \) onll \( \Gamma_{AODV} \) (\( \lambda((\xi, \_), \_, (\xi', \_)). \) \( \forall j. \) quality_increases \( (\sigma j) \) \( (\sigma' j) \))"
(is "\( \models (\_ \_ \rightarrow) \)"
proof (rule onll_ostep_invariantI, simp)
fix \( \sigma p \) \( \sigma' p' \) 1'
assume or: "\( (\sigma, p) \in \) reachable (opaodv \( i \)) \( ?S \) (other quality_increases \( \{i\} \))"
and 11: "\( l \in \) labels \( \Gamma_{AODV} p\)"
and ?S \( \sigma \) \( \sigma' \) a"
and tr: "\( ((\sigma, p), a, (\sigma', p')) \in \) oseqp_sos \( \Gamma_{AODV} i\)"
and 11: "\( l' \in \) labels \( \Gamma_{AODV} p'\)"
from this(1-3) have "\( \text{recvmsg } (\_ \_ \_ \rightarrow) (\sigma a) \)"
by (auto dest!: reachable_weakenE [where QS="act (recvmsg rreq_rrep_sn)"
and QU="other quality_increases \( \{i\} \)"
otherwith_actionD]
with or have orw: "\( (\sigma, p) \in \) reachable (opaodv \( i \)) (act (recvmsg rreq_rrep_sn)) (other quality_increases \( \{i\} \))"
by - (erule reachable_weakenE, auto)
with tr ll ll' and \( \text{recvmsg} (\lambda_. \text{rreq_rrep_sn}) \sigma a \) have "quality_increases (\sigma i) (\sigma' i)"
by (drule onll_ostep_invariantD [OF olocal_quality_increases], auto simp: seqll_def)
with (\forall S \sigma \sigma' a. show "\forall 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\} \rightarrow)
onll \Gamma_{\text{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), a, (\sigma', p')) \in \text{oseqp_sos} \Gamma_{\text{AODV}} i"
and recv: "act (recvmsg rreq_rrep_sn) \sigma \sigma' a"
obtain l l'
where l \in labels \Gamma_{\text{AODV}} p
and l' \in labels \Gamma_{\text{AODV}} p'
by (metis aodv_ex_label)
from "((\sigma, p), a, (\sigma', p')) \in \text{oseqp_sos} \Gamma_{\text{AODV}} i"
have tr: "((\sigma, p), a, (\sigma', p')) \in \text{trans} (opaodv i)"
proof (rule ostep_invariantD)
have "anycast (rreq_rrep_fresh (rt (\sigma i))) a"
proof -
have "opaodv i \_A (act (recvmsg rreq_rrep_sn), other A \{i\} \rightarrow)
onll \Gamma_{\text{AODV}} \text{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_{\text{AODV}} \text{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_{\text{AODV}} p and l'\in labels \Gamma_{\text{AODV}} p'
by auto
qed
moreover have "anycast (rerr_invalid (rt (\sigma i))) a"
proof -
have "opaodv i \_A (act (recvmsg rreq_rrep_sn), other A \{i\} \rightarrow)
onll \Gamma_{\text{AODV}} \text{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_{\text{AODV}} \text{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_{\text{AODV}} p and l'\in labels \Gamma_{\text{AODV}} p'
by auto
qed
moreover have "anycast (rreq_rrep_sn a)
proof -
from or tr recv
have "onll \Gamma_{\text{AODV}} \text{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 adv_trans,
simplified seqll_onll_swap])
thus \?thesis
using l\in labels \Gamma_{\text{AODV}} p and l'\in labels \Gamma_{\text{AODV}} p'
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_{\text{AODV}} \text{seqll i} (\lambda(((\xi, _), a, _). anycast (\lambda m. \text{not_Pkt} m \rightarrow \text{msg_sender} m = i) a))"
by (rule ostep_invariant_weakenE [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 l\in labels \Gamma_{\text{AODV}} p and l'\in labels \Gamma_{\text{AODV}} p'
by auto
qed

391
open_seq_step_invariant [OF sender_ip_valid initiali_aodv, simplified seqll_onll_swap]) auto

thus thesis using or recv (l∈labels Γ_{AODV} p) and (l'∈labels Γ_{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
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, _). 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} →)
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} →)
onll Γ_{AODV} (seq l (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} →)
onll Γ_{AODV} (λ(ξ, l). l ∈ {PAodv-:4, PAodv-:5, PRreq-:0, PRreq-:2} ∧ sip (σ i) ≠ oip (σ i) ∧ nsqn (rt (σ (sip (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 oadv_trans]
onlloinvariant_sterms [OF aodv_wf oreceived_rreq_rrep_nsqn_fresh_inv]
onlloinvariant_sterms [OF aodv_wf oosn_rreq]
simp add: seqllsimp

392
\begin{verbatim}
fix σ σ' p l
assume "(σ, p) ∈ oreachable (opaodv i) ?S ?U"
and "l ∈ labels Γ_{AODV} p"
and pre:

\( (1 = \text{Paodv}-:4 \lor 1 = \text{Paodv}-:5 \lor 1 = \text{PRreq}-:0 \lor 1 = \text{PRreq}-:2) \land \text{sip} (σ i) \neq \text{oip} (σ i) \)
\( \rightarrow \text{oip} (σ i) \in kd (rt (σ (\text{sip} (σ i)))) \land \text{dsn} (σ i) \leq \text{nsqn} (rt (σ (\text{sip} (σ i)))) (\text{oip} (σ i)) \land (\text{nsqn} (rt (σ (\text{sip} (σ i)))) (\text{oip} (σ i))) = \text{dsn} (σ i) \rightarrow \text{the (dhops (rt (σ (\text{sip} (σ i)))) (\text{oip} (σ i)))} \leq \text{hops} (σ i) \lor \text{the (flag (rt (σ (\text{sip} (σ i)))) (\text{oip} (σ i)))} = \text{inv}" \)

and "other quality_increases \{i\} σ σ'"
and hyp: "(1=\text{Paodv}-:4 \lor 1=\text{Paodv}-:5 \lor 1=\text{PRreq}-:0 \lor 1=\text{PRreq}-:2) \land \text{sip} (σ' i) \neq \text{oip} (σ' i)"
(is "\text{labels} \land \text{sip} (σ' i) \neq \text{oip} (σ' i)"
from (4(4)) have "σ' i = σ i"
with hyp have hyp'": "\text{labels} \land \text{sip} (σ i) \neq \text{oip} (σ i)" by simp
show "\text{oip} (σ i) \in kd (rt (σ (\text{sip} (σ i)))) \land \text{dsn} (σ i) \leq \text{nsqn} (rt (σ (\text{sip} (σ i)))) (\text{oip} (σ i)) \land (\text{nsqn} (rt (σ (\text{sip} (σ i)))) (\text{oip} (σ i))) = \text{dsn} (σ i) \rightarrow \text{the (dhops (rt (σ (\text{sip} (σ i)))) (\text{oip} (σ i)))} \leq \text{hops} (σ i) \lor \text{the (flag (rt (σ (\text{sip} (σ i)))) (\text{oip} (σ i)))} = \text{inv}" proof (cases "\text{sip} (σ i) = i")
assume "\text{sip} (σ i) \neq i"
from (other quality_increases \{i\} σ σ')
have "\text{quality_increases} (σ (\text{sip} (σ i))) (σ' (\text{sip} (σ i)))"
by (rule otherE) (clarsimp simp: \text{sip} (σ i) \neq i)
moreover from (σ, p) ∈ oreachable (opaodv i) ?S ?U l ∈ labels Γ_{AODV} p and hyp have "1 ≤ \text{dsn} (σ (σ i))"
by (auto dest!: \text{onl_oninvariant_weakenD} [OF \text{dsn_rreq} simp add: seqlsimp \text{σ' i} = \text{σ i}])
moreover from \text{sip} (σ i) \neq i hyp' and pre
have "\text{oip} (σ i) \in kd (rt (σ (\text{sip} (σ i)))) \land \text{dsn} (σ i) \leq \text{nsqn} (rt (σ (\text{sip} (σ i)))) (\text{oip} (σ i)) \land (\text{nsqn} (rt (σ (\text{sip} (σ i)))) (\text{oip} (σ i))) = \text{dsn} (σ i) \rightarrow \text{the (dhops (rt (σ (\text{sip} (σ i)))) (\text{oip} (σ i)))} \leq \text{hops} (σ i) \lor \text{the (flag (rt (σ (\text{sip} (σ i)))) (\text{oip} (σ i)))} = \text{inv}"
by (auto simp: \text{σ' i} = \text{σ i})
ultimately show \text{thesis}
by (rule quality_increases_rreq_rrep_props)
next
assume "\text{sip} (σ i) = i" thus \text{thesis}
using (\text{σ' i} = \text{σ i}) hyp and pre by auto
qed

lemma odsn_rrep:

\begin{verbatim}
"\text{opaodv i} \models (\text{otherwith quality_increases} \{i\} (\text{orecvmsg msg_fresh}), \text{other quality_increases} \{i\} \rightarrow)
\text{onl} Γ_{AODV} (\text{seql} \chi \lambda(λ, 1). l \in \{\text{Paodv}-:6, \text{Paodv}-:7, \text{PRrep}-:0, \text{PRrep}-:1\} \land \text{dsn} (σ i) \neq \text{dip} (σ i) \rightarrow \text{dip} (σ i) \in kd(rt (σ (\text{dip} (σ i)))) \land \text{nsqn} (rt (σ (\text{dip} (σ i)))) (\text{dip} (σ i)) \geq \text{dsn} (σ i) \land (\text{nsqn} (rt (σ (\text{dip} (σ i)))) (\text{dip} (σ i))) = \text{dsn} (σ i) \rightarrow (\text{hops} (σ i) \geq \text{the (dhops (rt (σ (\text{dip} (σ i)))) (\text{dip} (σ i)))) \lor \text{the (flag (rt (σ (\text{dip} (σ i)))) (\text{dip} (σ i)))) = \text{inv})")
(is "\models (\text{dsn, \text{∀U}} \rightarrow \_"
proof (inv_cterms inv add: oseq_step_invariant_sterns [OF \text{equality_increases_nsqn_fresh aodv_wf}
\end{verbatim}
lemma rerr_sip:

\[(\text{auto simp add: seqlsimp elim!: quality_increases_rreq_rrep_props'})\]

```
proof (cases "\(\sigma = \sigma'\)" ..
with hyp have hyp': "\(!\text{labels} \land \text{dip} (\sigma i) \neq \text{dip} (\sigma i)\)" by simp
show "\(\text{dip} (\sigma i) \in \text{kD (rt (\sigma' (\text{dip} (\sigma' i))))}\)"
\(\land\) \(\text{dsn (\sigma i) \leq \text{nsqn (rt (\sigma' (\text{dip} (\sigma' i)))) (dip (\sigma i))}\)"
\(\land\) \(\text{nsqn (rt (\sigma' (\text{dip} (\sigma' i)))) (dip (\sigma i)) = \text{dsn (\sigma i)}\)"
\(\rightarrow\) the (\(\text{dhops (rt (\sigma' (\text{dip} (\sigma' i)))) (dip (\sigma i))\) \leq \(\text{hops (\sigma i)}\)"
\(\lor\) \(\text{the (flag (rt (\sigma' (\text{dip} (\sigma' i)))) (dip (\sigma i)) = \text{inv})}\)
proof (cases "\(\text{dip} (\sigma i) = \text{i}\)"
assume "\(\text{dip} (\sigma i) \neq \text{i}\)"
from this(4) have "\(\sigma i = \sigma\)" ..
with hyp have hyp': "\(!\text{labels} \land \text{dip} (\sigma i) \neq \text{dip} (\sigma i)\)" by simp
show "\(\text{dip} (\sigma i) \in \text{kD (rt (\sigma' (\text{dip} (\sigma' i))))}\)"
\(\land\) \(\text{dsn (\sigma i) \leq \text{nsqn (rt (\sigma' (\text{dip} (\sigma' i)))) (dip (\sigma i))}\)"
\(\land\) \(\text{nsqn (rt (\sigma' (\text{dip} (\sigma' i)))) (dip (\sigma i)) = \text{dsn (\sigma i)}\)"
\(\rightarrow\) the (\(\text{dhops (rt (\sigma' (\text{dip} (\sigma' i)))) (dip (\sigma i))\) \leq \(\text{hops (\sigma i)}\)"
\(\lor\) \(\text{the (flag (rt (\sigma' (\text{dip} (\sigma' i)))) (dip (\sigma i)) = \text{inv})}\)
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
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 oquality_increases_nsqn_fresh aodv_wf oadpv_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 \|= onl Γ AODV (λξ. (\lambda(ξ, l). (l = PRerr-:1
  \rightarrow (∀ ip ∈ dom(dests ξ). ip ∈ D(rt ξ)
    \land the (nhop (rt ξ) ip) = sip ξ
    \land 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 oadpv_trans aodv_trans, simplified seql_onl_swap, THEN oinvariant_anyact]

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

lemmas oprerr_guard =
  open_seq_invariant [OF prerr_guard initiali_aodv oadpv_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} \rightarrow) onl Γ AODV (λ(σ, _).
      \forall dip. let nhp = the (nhop (rt (σ i)) dip)
      in dip ∈ kD(rt (σ i)) \land nhp \neq dip \rightarrow
      dip ∈ kD(rt (σ nhp)) \land nsqn (rt (σ i)) dip ≤ nsqn (rt (σ nhp)) dip)"
(is \_ \|= (\_S, \_U \rightarrow \_))"
proof -

{ fix nhop and σ σ' :: "ip \Rightarrow state"
  assume pre: "\forall dip ∈ kD(rt (σ i)). nhop dip \neq dip \rightarrow
dip ∈ kD(rt (σ (nhop dip))) \land nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (nhop dip))) dip"
  and qinc: "∀ j. quality_increases (σ j) (σ' j)"
  have "\forall dip ∈ kD(rt (σ i)). nhop dip \neq dip \rightarrow
dip ∈ kD(rt (σ' (nhop dip))) \land 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_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 have \thesis
by simp
qed
ultimately show "dip∈kD(rt (σ' (nhop 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)))"
and ndest: "∀ripc∈dom (dests (σ i)). ripc ∈ kD (rt (σ (sip (σ i)))))"
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 have \thesis by simp


qed

with \( \text{dip} \in \text{kD} \)

\[
\begin{align*}
\text{show} & \quad \text{"dip} \in \text{kD} \ (\text{rt (\sigma (\text{nhop dip}))}) \\
& \quad \wedge \text{nsqn (invalidate (rt (\sigma i)) (dests (\sigma i))) dip} \leq \text{nsqn (rt (\sigma (\text{nhop dip})))} \text{ dip} \\
\end{align*}
\]

qed

\}

\text{note basic_perr = this}

{ fix \( \sigma' \) :: "ip \( \Rightarrow \) state"
  assume a1: "\( \forall \text{dip} \in \text{kD} \) (rt (\sigma i)). \ (the (nhop (rt (\sigma i))) dip) \neq dip \\
  \quad \rightarrow \ \text{dip} \in \text{kD} (\ \text{rt (\sigma (the (nhop (rt (\sigma i) dip))))}) \\
  \quad \wedge \ \text{nsqn (rt (\sigma i)) dip} \leq \text{nsqn (rt (\sigma (the (nhop (rt (\sigma i) dip))))}) \text{ dip}"

  and a2: "\( \forall j. \text{quality_increases (\sigma j (\sigma') j)}"\n
  have "\( \forall \text{dip} \in \text{kD} \) (rt (\sigma i))

  \quad (\text{the (nhop (update (rt (\sigma i))) (sip (\sigma i)))}) \ (0, unk, val, Suc 0, sip (\sigma i), \{\}) dip) \neq dip \rightarrow \\
  \quad \text{dip} \in \text{kD} (\text{rt (\sigma' (the (nhop (update (rt (\sigma i))) (sip (\sigma i)))))} \\
  \quad (0, unk, val, Suc 0, sip (\sigma i), \{\}) dip))) \\
  \quad \wedge \ \text{nsqn (update (rt (\sigma i)) (sip (\sigma i)))} \ (0, unk, val, Suc 0, sip (\sigma i), \{\}) dip \\
  \quad \leq \text{nsqn (rt (\sigma' (the (nhop (update (rt (\sigma i))) (sip (\sigma i)))))} \\
  \quad (0, unk, val, Suc 0, sip (\sigma i), \{\}) dip)))" \\

  (is "\( \forall \text{dip} \in \text{kD} \) (rt (\sigma i)). ?P dip")

proof

fix dip

assume "dip \in \text{kD} (rt (\sigma i))"

with a1 and a2

  have "\( \text{the (nhop (rt (\sigma i) dip)) \neq dip} \rightarrow \text{dip} \in \text{kD} (\text{rt (\sigma' (the (nhop (rt (\sigma i) dip))))}) \\
  \quad \wedge \ \text{nsqn (rt (\sigma i)) dip} \leq \text{nsqn (rt (\sigma' (the (nhop (rt (\sigma i) dip))))}) \text{ dip}" by - (drule(1) basic, auto)

thus "?P dip" by (cases "\text{dip} = \text{sip (\sigma i)}") auto

qed

{ fix \( \sigma' oip \) oip sip osn hops

assume pre: "\( \forall \text{dip} \in \text{kD} \) (rt (\sigma i)). \ (the (nhop (rt (\sigma i) dip)) \neq dip \\
  \quad \rightarrow \ \text{dip} \in \text{kD} (\text{rt (\sigma (the (nhop (rt (\sigma i)) dip))))} \\
  \quad \wedge \ \text{nsqn (rt (\sigma i)) dip} \leq \text{nsqn (rt (\sigma (the (nhop (rt (\sigma i)) dip))))} \text{ dip}" and qinc: "\( \forall j. \text{quality_increases (\sigma j (\sigma') j)}" and *: "\( \text{sip} \neq \text{oip} \rightarrow \text{oip} \in \text{kD (rt (\sigma sip))} \\
  \quad \wedge \ \text{osn} \leq \text{nsqn (rt (\sigma sip)) oip} \\
  \quad \wedge \text{nsqn (rt (\sigma sip)) oip = osn} \\
  \quad \rightarrow \ (\text{the (dhops (rt (\sigma sip))) oip} \leq \text{hops} \\
  \quad \quad \vee \ \text{the (flag (rt (\sigma sip))) oip = inv})"

from pre and qinc

have pre': "\( \forall \text{dip} \in \text{kD} \) (rt (\sigma i)). \ (the (nhop (rt (\sigma i) dip)) \neq dip \\
  \quad \rightarrow \ \text{dip} \in \text{kD} (\text{rt (\sigma' (the (nhop (rt (\sigma i) dip))))}) \\
  \quad \wedge \ \text{nsqn (rt (\sigma i)) dip} \leq \text{nsqn (rt (\sigma' (the (nhop (rt (\sigma i) dip))))}) \text{ dip}" by (rule basic)

have "(\text{the (nhop (update (rt (\sigma i))) oip (osn, kno, val, Suc hops, sip, \{\}) oip}) \neq oip \\
  \quad \rightarrow \ \text{oip} \in \text{kD (rt (\sigma' (the (nhop (update (rt (\sigma i) oip (osn, kno, val, Suc hops, sip, \{\}) oip)))))} \\
  \quad \wedge \ \text{nsqn (update (rt (\sigma i))) oip (osn, kno, val, Suc hops, sip, \{\}) oip} \\
  \quad \leq \text{nsqn (rt (\sigma' (the (nhop (update (rt (\sigma i) oip (osn, kno, val, Suc hops, sip, \{\}) oip)))) oip"}

(is "?\text{nhop_not_oip} \rightarrow ?\text{oip_in_kD} \land ?\text{nsqn_le_nsqn}")

proof (rule, split update rt_split_asm)

assume "rt (\sigma i) = update (rt (\sigma i)) oip (osn, kno, val, Suc hops, sip, \{\})" and "the (nhop (rt (\sigma i) oip) \neq oip"

with pre' show "?\text{oip_in_kD} \land ?\text{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: ?\text{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)). (nhop (rt (σ i)) dip) ≠ dip
    → dip ∈ kD (rt (σ (nhop (rt (σ i)) dip)))
    ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (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)). (nhop (rt (σ i)) dip) ≠ dip
    → dip ∈ kD (rt (σ' (nhop (rt (σ i)) dip)))
    ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (nhop (rt (σ i)) dip))) dip"
  by (rule basic)
  have "∀ dip ∈ kD (rt (σ i)).
    (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})) dip) ≠ dip
    → dip ∈ kD (rt (σ' (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' (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_trans)
  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)). (nhop (rt (σ i)) dip) ≠ dip
  → dip ∈ kD (rt (σ (nhop (rt (σ i)) dip)))
  ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (nhop (rt (σ i)) dip))) dip"
by (inv_cterms inv add: oseq_step_invariant_sterms [OF quality_increases_nsqn_fresh aodv_wf oaodv_trans]

398
thus \( \text{thesis} \) unfolding \text{Let_def} by auto

dqed

Proposition 7.30

lemmas \( \text{okD}_\text{unk_or_atleast_one} \) =
  \text{open_seq_invariant} \[ \text{OF kD}_\text{unk_or_atleast_one} \ \text{initiali_aodv}, \]
  \text{simplified seql_onl_swap}

lemmas \( \text{ozero_seq}_\text{unk_hops_one} \) =
  \text{open_seq_invariant} \[ \text{OF zero_seq}_\text{unk_hops_one} \ \text{initiali_aodv}, \]
  \text{simplified seql_onl_swap}

lemma \( \text{oreachable_fresh}_\text{okD}_\text{unk_or_atleast_one} \): 
  fixes dip 
  assumes "\((\sigma, p) \in \text{oreachable (opaodv i)}\) 
  \((\text{otherwith (op=} \{i\} (\text{orecvmsg} (\lambda \sigma \ m. \ \text{msg_fresh} \ \sigma \ m \\
  \land \ \text{msg_zhops} \ m))) \)
  \((\text{other_quality_increases \{}i\})\)"
  and "\(\text{dip} \in \text{kD} (\text{rt (\sigma i)})\)"
  shows "\(\pi_3 (\text{the (rt (\sigma i) dip)}) = \text{unk} \lor 1 \leq \pi_2 (\text{the (rt (\sigma i) dip)})\)"
  (is "\(?P \ \text{dip}\)"

proof - 
  have "\(\exists l. l \in \text{labels} \ \Gamma_{AODV} \ p\)" by (metis aodv_ex_label)
  with \text{assms(1)} have "\(\forall \ \text{dip} \in \text{kD} (\text{rt (\sigma i)}). \ ?P \ \text{dip}\)"
  by - (drule \text{oinvariant_weakenD} \ [OF \text{okD}_\text{unk_or_atleast_one} \ [OF oaodv_trans aodv_trans]],
    \text{auto dest!: otherwith_actionD onlD simp: seqlsimp})
  with \(\text{dip} \in \text{kD} (\text{rt (\sigma i)})\) show \text{thesis} by simp

dqed

lemma \( \text{oreachable_fresh}_\text{ozero_seq}_\text{unk_hops_one} \): 
  fixes dip 
  assumes "\((\sigma, p) \in \text{oreachable (opaodv i)}\) 
  \((\text{otherwith (op=} \{i\} (\text{orecvmsg} (\lambda \sigma \ m. \ \text{msg_fresh} \ \sigma \ m \\
  \land \ \text{msg_zhops} \ m))) \)
  \((\text{other_quality_increases \{}i\})\)"
  and "\(\text{dip} \in \text{kD} (\text{rt (\sigma i)})\)"
  shows "\(\text{sqn (rt (\sigma i) dip)} = 0 \rightarrow \text{sqnf (rt (\sigma i) dip)} = \text{unk} \land \text{the (dhops (rt (\sigma i) dip)} = 1 \land \text{the (nhop (rt (\sigma i) dip)} = \text{dip}\)"
  (is "\(?P \ \text{dip}\)"

proof - 
  have "\(\exists l. l \in \text{labels} \ \Gamma_{AODV} \ p\)" by (metis aodv_ex_label)
  with \text{assms(1)} have "\(\forall \ \text{dip} \in \text{kD} (\text{rt (\sigma i)}). \ ?P \ \text{dip}\)"
  by - (drule \text{oinvariant_weakenD} \ [OF \text{ozero_seq}_\text{unk_hops_one} \ [OF oaodv_trans aodv_trans]],
    \text{auto dest!: onlD otherwith_actionD simp: seqlsimp})
  with \(\text{dip} \in \text{kD} (\text{rt (\sigma i)})\) show \text{thesis} by simp

dqed

lemma \( \text{seq_nhops}_\text{quality_increases}' \):
  shows "\(\text{opaodv i} \models (\text{otherwith (op=} \{i\})"
\[
(\text{orecvmsg} (\lambda \sigma \text{ m. msg\_fresh } \sigma \text{ m } \land \text{msg\_zhops m})), \\
\text{other quality\_increases } (i) \rightarrow \\
onl \Gamma_{AODV} (\lambda(\sigma, _). \forall \text{ dip. let nhip = the (nhop (rt (\sigma i))) dip} \\
in dip \in \text{ vD (rt (\sigma i)) } \cap \text{ vD (rt (\sigma nhip))} \\
\land \text{ nhip } \neq \text{ dip} \\
\quad \rightarrow \text{ (rt (\sigma i)) \text{ dip} (rt (\sigma nhip)))}
\]

(is "_ |= (?S i, _ \rightarrow ) _")

proof -

have weaken:

"\\\forall p \text{ Q R P. } p \models (\text{otherwith quality\_increases } I \text{ (orecvmsg Q), other quality\_increases } I \rightarrow ) P \\
\quad \rightarrow p \models (\text{otherwith (op=}) I \text{ (orecvmsg} (\lambda \sigma \text{ m. Q } \sigma \text{ m } \land \text{R } \sigma \text{ m})), \text{ other quality\_increases } I \rightarrow ) P"

by auto

\{ fix i a and \sigma \sigma' :: "ip \Rightarrow state"

assume a1: "\forall \text{ dip. dip\in vD(rt (\sigma i))} \\
\land \text{ dip\in vD(rt (\sigma (the (nhop (rt (\sigma i)) dip)))} \\
\land \text{ (the (nhop (rt (\sigma i)) dip)) } \neq \text{ dip} \\
\quad \rightarrow \text{ rt (\sigma i) dip rt (\sigma (the (nhop (rt (\sigma i)) dip)))}

and ow: "?S i \sigma \sigma' a"

have "\forall \text{ dip. dip\in vD(rt (\sigma i))} \\
\land \text{ dip\in vD rt (\sigma' (the (nhop (rt (\sigma i)) dip)))} \\
\land \text{ (the (nhop (rt (\sigma i)) dip)) } \neq \text{ dip} \\
\quad \rightarrow \text{ rt (\sigma i) 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 ow have "\\\forall j \neq i \rightarrow j = \sigma j" by auto

show "rt (\sigma i) dip rt (\sigma' (the (nhop (rt (\sigma i)) dip)))" by auto

proof (cases "(the (nhop (rt (\sigma i)) dip)) = i")

assume "(the (nhop (rt (\sigma i)) dip)) = i"

with \(\text{ dip } \in \text{ vD(rt (\sigma i))}) \) have "\text{ dip } \in \text{ vD(rt (\sigma (the (nhop (rt (\sigma i)) dip)))})" by simp

with a1 a2 a4 have "rt (\sigma i) dip rt (\sigma (the (nhop (rt (\sigma i)) dip)))" by simp

hence False by simp

thus ?thesis ..

next

assume "(the (nhop (rt (\sigma i)) dip)) \neq i"

with \(\\\forall j \neq i \rightarrow j = \sigma j\) have *: "\sigma (the (nhop (rt (\sigma i)) dip)) = \sigma' (the (nhop (rt (\sigma i)) dip))" by simp

with \(\text{ dip } \in \text{ vD(rt (\sigma i))}) \) have "\text{ dip } \in \text{ vD(rt (\sigma (the (nhop (rt (\sigma i)) dip)))})" by simp

with a1 a2 a4 have "rt (\sigma i) dip rt (\sigma (the (nhop (rt (\sigma i)) dip)))" by simp

with * show ?thesis by simp

qed

note basic = this

\{ fix \sigma \sigma' a dip sip i 

assume a1: "\forall \text{ dip. dip\in vD(rt (\sigma i))} \\
\land \text{ dip\in vD(rt (\sigma (the (nhop (rt (\sigma i)) dip))))} \\
\land \text{ (the (nhop (rt (\sigma i)) dip)) } \neq \text{ dip} \\
\quad \rightarrow \text{ rt (\sigma i) dip rt (\sigma (the (nhop (rt (\sigma i)) dip)))}

and ow: "?S i \sigma \sigma' a"

have "\forall \text{ dip. dip\in vD(update (rt (\sigma i)) sip (0, unk, val, Suc 0, sip, {}}) \\
\land \text{ dip\in vD(rt (\sigma' (the (nhop (update (rt (\sigma i)) sip (0, unk, val, Suc 0, sip, {}}) dip))))} \\
\land \text{ (the (nhop (update (rt (\sigma i)) sip (0, unk, val, Suc 0, sip, {}}) dip)) } \neq \text{ dip} \\
\quad \rightarrow \text{ update (rt (\sigma i)) sip (0, unk, val, Suc 0, sip, {}}) \\
\quad (\text{ 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 (σ i)) sip (0, unk, val, Suc 0, sip, {}))"  
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

} 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 (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 (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
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 a2 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)))"

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 (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 ∀ 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

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"

assume "sqn (rt (σ i)) oip < osn"
also from (Suc 0 ≤ osn) show "0 < osn" by simp

next
assume "sqn (rt (σ i)) oip = osn"
and (Suc hops < the (dhops (rt (σ i)) oip))
from this(1) 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 "Suc hops < the (dhops (rt (σ i)) oip)"
and (Suc 0 ≤ osn) show "0 < osn" by simp

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
assume "oip∈kD(rt (σ sip))"
and (Suc 0 ≤ osn) show False by simp
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)

next
assume "the (flag (rt (σ sip)) oip) = inv" by auto
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
qed
by (clarsimp split: option.split_asm split_if_asm)
(auto simp: sqn_def)
with \osn \leq \nsqn (\rt (\sigma (\sip))) \oip\ have \"\osn < \nsqn (\rt (\sigma (\sip))) \oip\"
by simp
thus ?thesis ..
qed
thus ?thesis
proof
assume osnlit: \"\osn < \nsqn (\rt (\sigma (\sip))) \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: \dip = \oip\ nsqn_update_changed_kno_val [OF change [THEN not_sym]])
  also have \"\ldots \leq \nsqn (\rt (\sigma (\sip))) \oip\" using osnlit .
  also have \"\ldots = \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 (\sigma (\sip))) \oip \land (\dhops (\rt (\sigma (\sip))) \oip) \leq \hops\"
have \"\oip \in kD(?rt1)\" by simp
moreover from a3 \dip = \oip\ 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 (\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 (\sigma (\sip))) \oip = \nsqn (?rt2 oip) \oip\"
  by (simp add: change)
  finally show ?thesis .
qed
moreover have \"\pi_5(?rt2 oip oip) < \pi_5(?rt1 oip)\"
proof
- from osneq have \"\dhops (\rt (\sigma (\sip))) \oip \leq \hops\" ..
  moreover from \oip \in vD(\rt (\sigma (\sip))) have \"\oip \in kD(\rt (\sigma (\sip)))\" by auto
  ultimately have \"\pi_5(?rt2 oip oip) \leq \hops\"
  by (auto simp add: proj5_eq_dhops)
  also from change after have \"\hops < \pi_5(\rt (\sigma (\i)) \oip)\"
  by (simp add: proj5_eq_dhops) (metis dhops_update_changed lessI)
  finally have \"\pi_5(\rt (\sigma (\sip)) \oip) < \pi_5(\rt (\sigma (\i)) \oip)\" .
  with change after show ?thesis by simp
qed
ultimately have \"?rt1 \sqsubseteq_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\}) (orecvmsg (\lambda \sigma \\. \msg_fresh (\sigma \m) \land \msg_zhops \m)), other quality increases \{\i\} \rightarrow\"
onl \Gamma_{AODV}
(\lambda(\sigma, \_). \forall dip. dip \in vD (\rt (\sigma \i)) \land vD (\rt (\sigma (\nhop (\rt (\sigma \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 cosn_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) ∈ orachable (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 (the (nhop (rt (σ i)) dip)))"

from this(1-2) have or': "(σ', p) ∈ orachable (opaodv i) (?S i) (other quality_increases {i})"
  by (rule orachable_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 (σ i) (σ' i)" by auto
  with "other quality_increases (σ 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 (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 ⟨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 "... = 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 "opaodv i \|= (otherwith (op=) {i} (orecvmsg msg_fresh),
     other quality_increases {i} →)
     global (λσ. ∀ dip. let nhip = the (nhop (rt (σ i)) dip)
\[ \text{in } \text{dip} \in kD(\text{rt } (\sigma \text{i})) \land \text{nhip} \neq \text{dip} \rightarrow \]
\[ \text{dip} \in kD(\text{rt } (\sigma \text{nhip})) \]
\[ \land \text{nsqn } (\text{rt } (\sigma \text{i})) \text{ dip} \leq \text{nsqn } (\text{rt } (\sigma \text{nhip})) \text{ dip} \]

by (rule oinvariant_weakenE \([\text{OF seq_compare_next_hop'}]\)) (auto dest!: onlD)

lemma seq_nhop_quality_increases:
shows "\text{opaodv i } \parallel \text{=} (\text{otherwith } (\text{op=} \{i\}) \\
(\text{orecvmsg } (\lambda \sigma. \text{msg_fresh } \sigma \text{ m} \land \text{msg_zhops } \text{m})), \text{ other quality_increases } \{i\} \rightarrow \text{global } (\lambda \sigma. \forall \text{ dip}. \text{let nhip = the } (\text{nhop } (\text{rt } (\sigma \text{i})) \text{ dip}) \]
\[ \text{in } \text{dip} \in vD (\text{rt } (\sigma \text{i})) \cap vD (\text{rt } (\sigma \text{nhip})) \land \text{nhip} \neq \text{dip} \rightarrow (\text{rt } (\sigma \text{i})) \sqsubseteq \text{dip} (\text{rt } (\sigma \text{nhip})))"\]

by (rule oinvariant_weakenE \([\text{OF seq_nhop_quality_increases'}]\)) (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 \text{rt_graph} :: "(ip \Rightarrow \text{state}) \Rightarrow ip \Rightarrow ip \text{ rel}"
where
\[ \text{rt_graph } \sigma = (\lambda \text{dip}. \]
\[ (\text{ip, ip'}) \mid \text{ip ip' dsn dsk hops pre.} \]
\[ \text{ip} \neq \text{dip} \land \text{rt } (\sigma \text{ ip}) \text{ dip} = \text{Some } (dsn, dsk, val, hops, ip', pre))"\]

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 \text{rt_graphE} \([\text{elim}]\):
fixes \(n\) \(\text{dip}\) \(\text{ip}\) \(\text{ip'}\)
assumes "\((\text{ip, ip')} \in \text{rt_graph } \sigma \text{ dip}\)"
shows "\(\text{ip} \neq \text{dip} \land (\exists \text{r}. \text{rt } (\sigma \text{ ip}) = \text{r} \]
\[ \land (\exists \text{dsn dsk hops pre. } \text{r dip} = \text{Some } (dsn, dsk, val, hops, ip', pre)))"\]

using asms unfolding \text{rt_graph_def} by auto

lemma \text{rt_graph_vD} \([\text{dest}]\):
"\(\forall \text{ip ip'} \sigma \text{ dip}. (\text{ip, ip'}) \in \text{rt_graph } \sigma \text{ dip} \Rightarrow \text{dip} \in vD(\text{rt } (\sigma \text{ ip}))\)"

unfolding \text{rt_graph_def} vD_def by auto

lemma \text{rt_graph_vD_trans} \([\text{dest}]\):
"\(\forall \text{ip ip'} \sigma \text{ dip}. (\text{ip, ip'}) \in (\text{rt_graph } \sigma \text{ dip})^+ \Rightarrow \text{dip} \in vD(\text{rt } (\sigma \text{ ip}))\)"

by (erule converse_tranclE) auto

lemma \text{rt_graph_not_dip} \([\text{dest}]\):
"\(\forall \text{ip ip'} \sigma \text{ dip}. (\text{ip, ip'}) \in \text{rt_graph } \sigma \text{ dip} \Rightarrow \text{ip} \neq \text{dip}\)"

unfolding \text{rt_graph_def} by auto

lemma \text{rt_graph_not_dip_trans} \([\text{dest}]\):
"\(\forall \text{ip ip'} \sigma \text{ dip}. (\text{ip, ip'}) \in (\text{rt_graph } \sigma \text{ dip})^+ \Rightarrow \text{ip} \neq \text{dip}\)"

by (erule converse_tranclE) auto

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

lemma \text{rt_graph_nhip_is_nhop} \([\text{dest}]\):
"\(\forall \text{ip ip'} \sigma \text{ dip}. (\text{ip, ip'}) \in \text{rt_graph } \sigma \text{ dip} \Rightarrow \text{ip} = \text{the } (\text{nhop } (\text{rt } (\sigma \text{ ip})) \text{ dip})\)"

unfolding \text{rt_graph_def} by auto

theorem \text{inv_to_loop_freedom}:
assumes "\(\forall i \text{ dip}. \text{let nhip = the } (\text{nhop } (\text{rt } (\sigma \text{ i})) \text{ dip})\)
\[\text{in dip } \in vD (rt (\sigma i)) \cap vD (rt (\sigma \text{nhip})) \land \text{nhip } \neq \text{dip} \quad \rightarrow (rt (\sigma i) \sqsubseteq_{\text{dip}} rt (\sigma \text{nhip}))\]

shows "\(\forall \text{dip}. \text{irrefl} ((rt_{\text{graph}} \sigma \text{dip})^+)\)"

using assms proof (intro allI)

fix \sigma :: "ip \Rightarrow \text{state}" and dip

assume inv: "\(\forall \text{ip dip}. \text{let nhip } = \text{the (nhop (rt (\sigma ip)) dip)} \rightarrow \text{dip } \in vD (rt (\sigma \text{ip})) \cap vD (rt (\sigma \text{nhip})) \land \text{nhip } \neq \text{dip} \quad \rightarrow \quad \text{rt (\sigma ip)} \subseteq_{\text{dip}} \text{dip} \text{rt (\sigma nhip)}\)"

\{ fix ip ip'
assume "\((ip, ip') \in (rt_{\text{graph}} \sigma \text{dip})^+\)"
and "\(\text{dip } \in vD (rt (\sigma \text{ip}'))\)"
and "\(\text{ip' } \neq \text{dip}\)"
hence "\(\text{rt (\sigma ip)} \subseteq_{\text{dip}} \text{dip} \text{rt (\sigma ip')}\)"
proof induction

fix nhip
assume "\((ip, nhip) \in rt_{\text{graph}} \sigma \text{dip}\)"
and "\(\text{dip } \in vD (rt (\sigma \text{nhip}))\)"
and "\(\text{nhip } \neq \text{dip}\)"

from \((ip, nhip) \in rt_{\text{graph}} \sigma \text{dip}\) have "\(\text{dip } \in vD (rt (\sigma \text{ip}))\)"
and "\(\text{nhip } = \text{the (nhop (rt (\sigma \text{ip})) \text{dip})}\)"
and inv

show "\(\text{rt (\sigma ip)} \subseteq_{\text{dip}} \text{dip} \text{rt (\sigma nhip)}\)"
by (clarsimp simp: Let_def)

next

fix nhip nhip'
assume "\((ip, nhip) \in rt_{\text{graph}} \sigma \text{dip}\)"
and "\((nhip, nhip') \in rt_{\text{graph}} \sigma \text{dip}\)"
and IH: "\(\quad \text{dip } \in vD (rt (\sigma \text{nhip})); \text{nhip } \neq \text{dip} \quad \rightarrow \quad \text{rt (\sigma ip)} \subseteq_{\text{dip}} \text{rt (\sigma nhip)}\)"
and "\(\text{dip } \in vD (rt (\sigma \text{nhip}'))\)"
and "\(\text{nhip'} \neq \text{dip}\)"

from \((nhip, nhip') \in rt_{\text{graph}} \sigma \text{dip}\) have 1: "\(\text{dip } \in vD (rt (\sigma \text{nhip}))\)"
and 2: "\(\text{nhip } \neq \text{dip}\)"
and "\(\text{nhip'} = \text{the (nhop (rt (\sigma \text{nhip})) \text{dip})}\)"
by auto

from 1 2 have "\(\text{rt (\sigma ip)} \subseteq_{\text{dip}} \text{rt (\sigma nhip)}\)" by (rule IH)
also have "\(\text{rt (\sigma nhip)} \subseteq_{\text{dip}} \text{rt (\sigma nhip')}\)"
proof -

from \((\text{dip } \in vD (rt (\sigma \text{nhip}))); \text{nhip } \neq \text{dip})\) have "\(\text{dip } \in vD (rt (\sigma \text{nhip})); \text{dip } \neq \text{nhip} \quad \rightarrow \quad \text{rt (\sigma nhip')}\)"
and "\(\text{nhip'} \neq \text{dip}\)"
and inv

show "\(\text{rt (\sigma nhip)} \subseteq_{\text{dip}} \text{dip} \text{rt (\sigma nhip')}\)"
by (clarsimp simp: Let_def)

qed

finally show "\(\text{rt (\sigma ip)} \subseteq_{\text{dip}} \text{dip} \text{rt (\sigma nhip')}\)" .

qed } note fresher = this

show "\(\text{irrefl} ((rt_{\text{graph}} \sigma \text{dip})^+)\)"
unfolding irrefl_def proof (intro allI notI)

fix ip
assume "\((ip, ip) \in (rt_{\text{graph}} \sigma \text{dip})^+\)"
moreover then have "\(\text{dip } \in vD (rt (\sigma \text{ip}))\)"
and "\(\text{ip } \neq \text{dip}\)"
by auto

ultimately have "\(\text{rt (\sigma ip)} \subseteq_{\text{dip}} \text{dip} \text{rt (\sigma ip)}\)" 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/OClosed_Transfer" "/.../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 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. 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 | A (otherwith (op=) {i} (orecvmsg (λσ. rreq_rrep_sn)), other (λ_ _. True) {i} →)
globala (λ(σ, _, 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', _)). ∀ 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} (orecvmsg (λ_. orecvmsg (λ_. rreq_rrep_sn) σ), other (λ_ _. True) {i} →)
globala (λ(σ, _, σ'). quality_increases (σ i) (σ' i))"
by (erule weaken_otherwith, auto)
Qed

lemma par_rqreq_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 | A (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])
(auto dest!: on1D seq1D elim!: aodv_ex_labelE)
hence "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))"
by (rule lift_step_into_qmsg_statelessassm) simp_all
thus thesis by rule auto
Qed

lemma par_rqreq_rrep_nsqf_fresh_any_step:
shows "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 -

have "opaodv i = (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_msg_fresh_any_step_invariant])
fix t
assume "onll Γ̂ AODV A (λ((σ, _, a), a, _). anycast (msg_fresh σ) a) t"
thus "globala (λ(σ, a, σ'). anycast (msg_fresh σ) a) t"
by (cases t) (clarsimp dest!: onllD, metis aodv_ex_label)
qed auto

hence "opaodv i (⟨⟨ i qmsg | = A (λσ _. (orecvmsg (λ_. rreq_rrep_sn)) σ, other (λ_. True) {i} →) globala (λ(σ, a, _). anycast (msg_fresh σ) a)⟩) t"
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} →) globala (λ(_, a, _). anycast msg_zhops a)⟩) t"
proof -
from anycast_msg_zhops initiali_aodv oaodv_trans aodv_trans
have "opaodv i (⟨⟨ i qmsg | = A (act TT, other (λ_ _. True) {i} →) seqll i (onll Γ̂ AODV (λ(_, a, _). anycast msg_zhops a)))⟩) t"
by (rule open_seq_step_invariant)

hence "opaodv i (⟨⟨ i qmsg | = A (λσ _. orecvmsg (λ_. rreq_rrep_sn)) σ, other (λ_. True) {i} →) globala (λ(_, a, _). anycast msg_zhops a)⟩) t"
proof (rule ostep_invariant_weakenE)
fix t :: "(((nat ⇒ state) × (state, msg, pseqp, pseqp label) seqp), msg seq_action) transition"
assume "seqll i (onll Γ̂ AODV (λ(_, a, _). anycast msg_zhops a)) t"
thus "globala (λ(_, a, _). anycast msg_zhops a) t"
by (cases t) (clarsimp dest!: seqllD 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} →) globala (λ(_, a, _). anycast msg_zhops a)⟩) t"
by (rule lift_step_into_qmsg_statelessassm) simp_all
thus ?thesis by rule auto
qed

4.11.2 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 (oparp_sos i (oseqp_sos Γ̂ AODV i) (seqp_sos Γ̂ QMSG i))"
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 ⟩⟩ o | = (otherwith (op=) {i})
(oarrivemsg (λm. 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:
"⟨ i : opaodv i (⟨⟨ i qmsg : R ⟩⟩ o | = (otherwith (op=) {i})
(oarrivemsg (λm. 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_step_statelessassm [OF par_rreq_rrep_sn_quality_increases]) simp

lemma node_rreq_rrep_msg_fresh_any_step:
shows "⟨ i : opaodv i (⟨⟨ i qmsg : R ⟩⟩ o | = A (λσ _. oarrivemsg (λ_. rreq_rrep_sn)) σ,
other (λ_. True) {i} →)
globala (λ(σ, a, σ'). quality_increases (σ i) (σ' i))⟩) t"
by (rule node_lift_step_statelessassm [OF par_rreq_rrep_sn_msg_fresh_any_step_invariant]) simp
lemma node_anycast_msg_zhops:
shows "\( i : \text{oapadv } i \langle \langle i \text{ qmsg : } R \rangle_0 \rangle_0 \models_A (\lambda \sigma \_. \text{arrivemsg } (\lambda \_. \text{rreq_rrep_sn}) \sigma, \text{other } (\lambda \_. \_. \text{True} ) \{ i \} \rightarrow) \) globala \( (\lambda (\sigma, a, \sigma'). \text{castmsg } (\text{msg_fresh } \sigma) a) \)"
by (rule node_lift_anycast_statelessassm [OF par_rreq_rrep_nsqn_fresh_any_step])

lemma node_silent_change_only:
shows "\( i : \text{oapadv } i \langle \langle i \text{ qmsg : } R_i \rangle_0 \rangle_0 \models_A (\lambda \sigma \_. \text{arrivemsg } (\lambda \_. \text{rreq_rrep_sn}) \sigma, \text{other } (\lambda \_. \_. \text{True} ) \{ i \} \rightarrow) \) globala \( (\lambda (\sigma, a, \_. ) \text{. castmsg } \text{zhops} a) \)"
by (rule node_lift_anycast_statelessassm [OF par_anycast_msg_zhops])

proof
shows \( \sigma \) \( \alpha \) \( \sigma' \)
assume or: "\( (\sigma, \zeta) \in \text{oreachable } (i : \text{oapadv } i \langle \langle i \text{ qmsg : } R_i \rangle_0 \rangle) \)
\( (\lambda \sigma \_. \text{arrivemsg } (\lambda \_. \text{rreq_rrep_sn}) \sigma) \)
\( \text{other } (\lambda \_. \_. \text{True} ) \{ i \})"\)
and tr: "\( ((\sigma, \zeta), a, (\sigma', \zeta')) \in \text{trans } (i : \text{oapadv } i \langle \langle i \text{ qmsg : } R_i \rangle_0 \rangle)"\)
and "\( a \neq \tau_n \)"
from or obtain \( p R \) where "\( \zeta = \text{NodeS } i \) p R"
by (drule node_net_state, metis)
with tr have "\( ((\sigma, \text{NodeS } i \) p R), a, (\sigma', \zeta')) \in \text{onode_sos } (\text{oparp_sos } i \langle \text{trans } (\text{oapadv } i) \rangle (\text{trans } \text{qmsg}))"\)
by simp
thus "\( \sigma' i = \sigma i \)" using \( (a \neq \tau_n) \)
by (cases rule: onode_sos.cases)
(auto elim: qmsg_no_change_on_send_or_receive)
qed

4.11.3 Lift to partial networks

lemma arrive_rreq_rrep_nsqn_fresh_inc_sn [simp]:
assumes "\( \text{arrivemsg } (\lambda \sigma m. \text{msg_fresh } \sigma m \land P \sigma m) \sigma m \)"
shows "\( \text{arrivemsg } (\lambda \_. \text{rreq_rrep_sn}) \sigma m \)"
using assms by (cases \( m \)) auto

lemma opnet_nhop_quality_increases:
shows "\( \text{opnet } (\lambda i. \text{oapadv } i \langle \langle i \text{ qmsg} : p \rangle \rangle) \models \)\)
(\( \text{other with } (\text{op=}) (\text{net_tree_ips } p) \) \( \)\)
\( (\text{arrivemsg } (\lambda \sigma m. \text{msg_fresh } \sigma m \land \text{zhops } m)) \), \( \)\)
\( \text{other_quality_increases } (\text{net_tree_ips } p) \rightarrow) \)\)
\( \text{globala } (\lambda \sigma. \forall i \in \text{net_tree_ips } p. \forall \text{ dip.} \)\)
\( \text{let } \text{nhip } = \text{the } (\text{nhop } (\text{rt } (\sigma i)) \text{ dip}) \)\)
\( \text{in } \text{dip } \in \text{vd } (\text{rt } (\sigma i)) \cap \text{vd } (\text{rt } (\text{nhip})) \land \text{nhip } \neq \text{dip} \)\)
\( \rightarrow (\text{rt } (\sigma i)) \subseteq \text{dip } (\text{rt } (\text{nhip}))"\)
proof (rule pnet_lift [OF node_nhop_quality_increases])
fix \( i R \) have "\( i : \text{oapadv } i \langle \langle i \text{ qmsg : } R \rangle_0 \rangle_0 \models_A (\lambda \sigma \_. \text{arrivemsg } (\lambda \_. \text{rreq_rrep_sn}) \sigma, \)\)
\( \text{other } (\lambda \_. \_. \text{True} ) \{ i \} \rightarrow) \) globala \( (\lambda (\sigma, a, \sigma'). \text{castmsg } (\text{msg_fresh } \sigma m \land \text{zhops } m) a) \)"
proof (rule ostep_invariantI, simp (no_asm))
fix \( \sigma s a \) \( \sigma' s' \)
assume or: "\( (\sigma, s) \in \text{oreachable } (i : \text{oapadv } i \langle \langle i \text{ qmsg : } R \rangle_0 \rangle) \)\)
\( (\lambda \sigma \_. \text{arrivemsg } (\lambda \_. \text{rreq_rrep_sn}) \sigma) \)
\( \text{other } (\lambda \_. \_. \text{True} ) \{ i \})"\)
and tr: "\( ((\sigma, s), a, (\sigma', s')) \in \text{trans } (i : \text{oapadv } i \langle \langle i \text{ qmsg : } R \rangle_0 \rangle)"\)
and am: "\( \text{arrivemsg } (\lambda \_. \text{rreq_rrep_sn}) \sigma a \)"
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 "\( \text{castmsg } (\text{zhops} a) \)"
by (auto dest!: ostep_invariantD [OF node_anycast_msg_zhops])
ultimately show "\( \text{castmsg } (\lambda m. \text{msg_fresh } \sigma m \land \text{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 \rangle globala (\lambda(\sigma, a, \_). castmsg (\lambda m. msg_fresh \sigma m \land msg_zhops m) a)"

by rule auto

next

fix i R

show "\langle i : opaodv i \langle\langle i qmsg : R \rangle \rangle \sigma, other quality_increases \{i\} \rightarrow \rangle \rangle globala (\lambda(\sigma, a, \_). castmsg (\lambda m. msg_fresh \sigma m \land msg_zhops m) a) \land \tau \land (\forall i 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

show "\langle i : opaodv i \langle\langle i qmsg : R \rangle \rangle \sigma, other quality_increases \{i\} \rightarrow \rangle \rangle globala (\lambda(\sigma, a, \_). castmsg (\lambda m. msg_fresh \sigma m \land msg_zhops m) a) \lor \land (\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 (opnet (\lambda i. opaodv i \langle\langle i qmsg \rangle \rangle p)|= (\lambda _ _ _ True, other quality_increases (net_tree_ips p) \rightarrow) global (\lambda_. \forall i \in net_tree_ips p. \forall dip.
let nhip = the (nhop (rt (\sigma i)) dip)
in dip \in V (rt (\sigma i)) \cap V (rt (\sigma nhip)) \land nhip \neq dip 
\rightarrow (rt (\sigma i)) \sqsubseteq dip (rt (\sigma nhip)))"

(is "\_ |= (\_, ?U \rightarrow) ?inv")

proof (rule inclosed_closed)

from onet_nhop_quality_increases

show "opnet (\lambda i. opaodv i \langle\langle i qmsg \rangle \rangle p)

|= (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 m. msg_fresh \sigma m \land msg_zhops m) \sigma \sigma' a)"

proof (rule otherwithE)

fix \sigma :: "ip \Rightarrow state" and a :: "msg node_action"

assume "inoclosed \sigma a"

thus "oarrivemsg (\lambda m. msg_fresh \sigma m \land msg_zhops m) \sigma a"

proof (cases a)

fix ii ni ms

assume "a = ii\_ni.arrive(ms)"

moreover with inoclosed \sigma a obtain dip where "ms = newpkt(d, dip)"

by (cases ms) auto

ultimately show ?thesis by simp

qed simp_all

qed

qed

4.11.5 Transfer into the standard model

interpretation adv_openproc: openproc paodv opaodv id

where "adv_openproc.initmissing = initmissing"

proof -

show "openproc paodv opaodv id"

proof unfold_locales
fix \( i : \mathbb{I} \)

have \({(\sigma, \zeta). (\sigma, \zeta) \in \sigma_{AODV} \ \land (\forall j \neq i \rightarrow \sigma j \in \text{fst} \ \prime \ \sigma_{AODV} j)} \subseteq \sigma_{AODV} \} \) unfolding \( \sigma_{AODV} \text{def} \ \sigma_{AODV} \text{'} \text{def} \)

proof (rule equalityD1)

show \({f p. \{(\sigma, \zeta). (\sigma, \zeta) \in \{(f i, p)\} \ \land (\forall j \neq i \rightarrow \sigma j \in \text{fst} \ \prime \ \{(f j, p)\})} = \{(f, p)\}} \)

by (rule set_eqI) auto

qed

thus \({ (\sigma, \zeta) \mid \sigma \zeta s. s \in \text{init} (\text{paodv} i) \ \land (\sigma i, \zeta) = \text{id} s \ \land (\forall j \neq i \rightarrow \sigma j \in \text{fst o id} \ \prime \ \text{init} (\text{paodv} j))} \subseteq \text{init} (\text{opaodv} i) \) unfolding \( \sigma_{AODV} \text{def} \ \sigma_{QMSG} \text{def} \)

by simp

next

show \("(\sigma, \zeta) \mid \sigma \zeta s. s \in \text{init} (\text{paodv} i) \) \neq \{\}" unfolding \( \sigma_{AODV} \text{def} \) by simp

next

fix \( i, s, a, s' \)
assume \"\sigma i = \text{fst} (\text{id} s)\"
and \"\sigma' i = \text{fst} (\text{id} s')\"
and \"(s, a, s') \in \text{trans} (\text{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} (\text{paodv} i)\"

by (cases \( s \), cases \( s' \)) auto

from this(3) have \"\{\sigma q, a, (\sigma', q')\} \in \text{trans} (\text{opaodv} i)\"

by simp (rule open_seqp_action [OF aodv_wf])

with \( s = (\sigma i, q) \) and \( s' = (\sigma' i, q') \)
show \"\{\sigma \text{snd} (\text{id} s), a, (\sigma', \text{snd} (\text{id} s'))\} \in \text{trans} (\text{opaodv} i)\"

by simp

qed

interpretation aodv_openproc_par_qmsg: openproc_parq paodv opaodv id qmsg

where "aodv_openproc_par_qmsg.netglobal = \text{netglobal}"
and "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: \"\bigwedge \sigma. \text{openproc.initmissing} (\lambda i. \text{paodv} i \ (\langle\langle qmsg\rangle \ (\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 \( \sigma_{AODV} \text{def} \ \sigma_{QMSG} \text{def} \) by (clarsimp cong: option.case_cong)

thus \"openproc.initmissing (\lambda i. \text{paodv} i \ (\langle\langle qmsg\rangle \ (\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 \( \sigma_{AODV} \text{def} \ \sigma_{QMSG} \text{def} \)

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 \"openproc.netglobal (\lambda i. \text{paodv} i \ (\langle\langle qmsg\rangle \ (\lambda(p, q). (\text{fst} (\text{id} p), \text{snd} (\text{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
(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 ⟨⟨ 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_sosTau1
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 \( \pi_6 \equiv \lambda(_, _, _, _, nhip). nhip \)

**Lemma** \( \text{projs [simp]}: \)

\[
\begin{align*}
\pi_2 & (dsn, dsk, flag, hops, nhip) = dsn \\
\pi_3 & (dsn, dsk, flag, hops, nhip) = dsk \\
\pi_4 & (dsn, dsk, flag, hops, nhip) = flag \\
\pi_5 & (dsn, dsk, flag, hops, nhip) = hops \\
\pi_6 & (dsn, dsk, flag, hops, nhip) = nhip
\end{align*}
\]

by \((\text{clarsimp simp: proj2_def proj3_def proj4_def proj5_def proj6_def})+\)

**Lemma** \( \text{proj3_pred [intro]}: \)

\[
[ [ P kno; P unk ] \] \Rightarrow P (\pi_3 x)
\]

by \((\text{rule k.induct})\)

**Lemma** \( \text{proj4_pred [intro]}: \)

\[
[ [ P val; P inv ] \] \Rightarrow P (\pi_4 x)
\]

by \((\text{rule f.induct})\)

**Lemma** \( \text{proj6_pair_snd [simp]}: \)

fixes \( dsn' \ r \)

shows \( \pi_6 (dsn', \text{snd (r)}) = \pi_6 (r) \)

by \((\text{cases r simp})\)

### 5.1.3 Routing Tables

Routing tables map ip addresses to route entries.

**Type Synonym** \( \text{rt} = "ip \rightarrow r" \)

**Syntax**

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

**Translations**

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

**Definition** \( \text{sqn} :: \text{"rt \Rightarrow ip \Rightarrow sqn"} \)

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

**Definition** \( \text{sqnf} :: \text{"rt \Rightarrow ip \Rightarrow k"} \)

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

**Abbreviation** \( \text{flag} :: \text{"rt \Rightarrow ip \Rightarrow f"} \)

where \( \text{flag rt dip \equiv map_option \pi_4 (\sigma_{route}(rt, dip))} \)

**Abbreviation** \( \text{dhops} :: \text{"rt \Rightarrow ip \Rightarrow nat"} \)

where \( \text{dhops rt dip \equiv map_option \pi_5 (\sigma_{route}(rt, dip))} \)

**Abbreviation** \( \text{nhop} :: \text{"rt \Rightarrow ip \Rightarrow ip"} \)

where \( \text{nhop rt dip \equiv map_option \pi_6 (\sigma_{route}(rt, dip))} \)

**Definition** \( \text{vD} :: \text{"rt \Rightarrow ip set"} \)

where \( \text{vD rt \equiv \{dip. flag rt dip = Some val\}} \)

**Definition** \( \text{iD} :: \text{"rt \Rightarrow ip set"} \)

where \( \text{iD rt \equiv \{dip. flag rt dip = Some inv\}} \)

**Definition** \( \text{kD} :: \text{"rt \Rightarrow ip set"} \)

where \( \text{kD rt \equiv \{dip. rt dip \neq None\}} \)

**Lemma** \( \text{kD_is_vD_and_iD: } \text{kD rt = vD rt \cup iD rt} \)

unfolding \( \text{kD_def vD_def iD_def by auto} \)

**Lemma** \( \text{vD_iD_gives_kD [simp]:} \)

\[
\begin{align*}
\forall ip rt. ip \in vD rt \Rightarrow ip \in kD rt \\
\forall ip rt. ip \in iD rt \Rightarrow ip \in kD rt
\end{align*}
\]

unfolding \( \text{kD_is_vD_and_iD_def by simp_all} \)

416
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 $\langle ip \notin vD(rt) \rangle$ have "$f \neq \text{val}" \\
proof (rule contrapos_nn) \\
\hspace{1em} assume "$f = \text{val}" \\
\hspace{1em} with rtip have "the (flag rt ip) = \text{val}" by simp \\
\hspace{1em} with ($ip \in kD(rt)$) show "$ip \in vD(rt)$" .. \\
qed \\
\hspace{1em} with rtip have "the (flag rt ip) = \text{inv}" by simp \\
\hspace{1em} with ($ip \in kD(rt)$) show "$ip \in iD(rt)$" .. \\
qed \\
\indent lemma vD_or_iD [elim]: \\
\hspace{1em} fixes $ip$ rt \\
\hspace{1em} assumes "$ip \in kD(rt)" \\
\hspace{1em} and "$ip \in vD(rt) \implies P\ rt\ ip$ \\
\hspace{1em} and "$ip \in iD(rt) \implies P\ rt\ ip$ \\
\hspace{1em} shows "$P\ rt\ ip$" \\
\hspace{1em} proof - \\
\hspace{1em} from ($ip \in kD(rt)$) have "$ip \in vD(rt) \cup iD(rt)$" \\
\hspace{1em} by (simp add: kD_is_vD_and_iD) \\
\hspace{1em} thus $\text{thesis}$ by (auto elim: assms(2-3)) \\
\hspace{1em} qed \\
\indent lemma proj5_eq_dhops: "$\land dip\ rt. dip \in kD(rt) = \implies \pi_5 (\text{the (rt dip)}) = \text{the (dhops rt dip)}" \\
\hspace{1em} unfolding sqn_def by (drule kD_Some) clarsimp \\
\indent lemma proj4_eq_flag: "$\land dip\ rt. dip \in kD(rt) = \implies \pi_4 (\text{the (rt dip)}) = \text{the (flag rt dip)}" \\
\hspace{1em} unfolding sqn_def by (drule kD_Some) clarsimp \\
\indent lemma proj2_eq_sqn: "$\land dip\ rt. dip \in kD(rt) = \implies \pi_2 (\text{the (rt dip)}) = \text{sqn rt dip}" \\
\hspace{1em} unfolding sqn_def by (drule kD_Some) clarsimp \\
\indent lemma kD_sqnf_is_proj3 [simp]: \\
\hspace{1em} "$\land \text{ip\ rt. ip} \in kD(rt) = \implies \text{sqnf rt ip} = \pi_3 (\text{the (rt ip)})" \\
\hspace{1em} unfolding sqnf_def by auto \\
\indent lemma vD_flag_val [simp]: \\
\hspace{1em} "$\land dip\ rt. dip \in vD (rt) = \implies \text{the (flag rt dip)} = \text{val}$" \\
\hspace{1em} unfolding vD_def by clarsimp \\
\indent lemma kD_update [simp]: \\
\hspace{1em} "$\land rt\ nip\ v. kD (rt(nip \mapsto v)) = \text{insert nip (kD rt)}$" \\
\hspace{1em} unfolding kD_def by auto \\
\indent lemma kD_empty [simp]: "$kD \text{ Map.empty} = \{\}$" \\
\hspace{1em} unfolding kD_def by simp \\
\indent lemma ip_equal_or_known [elim]: \\
\hspace{1em} fixes rt ip ip' \\
\hspace{1em} assumes "$ip = ip' \lor ip \in kD(rt)" \\
\hspace{1em} and "$ip = ip' \implies P\ rt\ ip'$" \\
\hspace{1em} and "$[ ip \neq ip'; ip \in kD(rt)] \implies P\ rt\ ip'$" \\
\hspace{1em} shows "$P\ rt\ ip'$" \\
\hspace{1em} using assms by auto \\
\subsection{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 \\
\indent lemma in_kD_case [simp]: \\
\hspace{1em} fixes dip rt \\
\hspace{1em} assumes "$dip \in kD(rt)$"
shows "(case rt dip of None ⇒ en | Some r ⇒ es r) = es (the (rt dip))"
using assms [THEN kD_None] 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 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"
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 σroute(rt, ip) of
  None ⇒ rt (ip ↦→ r)
  | Some s ⇒
    if π2(s) < π2(r) then rt (ip ↦→ r)
    else if π2(s) = π2(r) ∧ (π5(s) > π5(r) ∨ π4(s) = inv)
    then rt (ip ↦→ r)
    else if π3(r) = unk
    then rt (ip ↦→ (π3(s), snd (r)))
    else rt (ip ↦→ s)"

lemma update_simps [simp]:
fixes r s nrt nr' ns rt ip
defines "s ≡ the σroute(rt, ip)"
and "nr' ≡ (π2(s), π3(r), π4(r), π5(r), π6(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 ↦→ r)"
"[ [ ip ∈ kD(rt); sqn rt ip = π2(r);
    the (dhops rt ip) > π5(r) ] ] ⇒ update rt ip r = rt (ip ↦→ r)"
"[ [ 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 ↦→ s)"

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 < π2(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 < π2(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 = π2(r)"
and "the (dhops rt ip) > π5(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 = π2(r) ⟩ and ⟨the (dhops rt ip) > π5(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 = 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 ↦ (π₂(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) \implies P (rt (ip \mapsto r))"
and c3: "[sqn rt ip = π₂(r); the (dhops rt ip) > π₅(r)]
\implies P (rt (ip \mapsto r))"
and c4: "[sqn rt ip = π₂(r); the (flag rt ip) = inv]
\implies P (rt (ip \mapsto r))"
and c5: "π₃(r) = unk \implies P (rt (ip \mapsto (π₂(\sigma_{\text{route}}(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) \implies the (dhops rt ip) ≤ π₅(r) \land the (flag rt ip) = val]
\implies P (rt (ip \mapsto \sigma_{\text{route}}(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 \mapsto r))" by (rule c2)
next
assume "sqn rt ip = π₂(r)"
and "the (dhops rt ip) > π₅(r)"
thus "P (rt(ip \mapsto r))"
by (rule c3)
next
assume "sqn rt ip = π₂(r)"
and "the (flag rt ip) = inv"
thus "P (rt(ip \mapsto r))"
by (rule c4)
next
assume "π₃(r) = unk"
thus "P (rt (ip \mapsto (π₂(\sigma_{\text{route}}(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) \implies the (dhops rt ip) ≤ π₅(r) \land the (flag rt ip) = val"
thus "P (rt (ip \mapsto (the (rt ip))))"
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)) = 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 \not= 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: "\forall v. rt dip = Some v \implies
update rt dip (dsn, dsk, flag, hops, nhip) 
\neq rt(dip \mapsto (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 \neq 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 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 "((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 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 dsn dsk flag hops nhip. sqn (rt(ip → ν)) ip = π2(ν)"
  unfolding sqn_def by simp

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


lemma nhop_update_unk_val [simp]:
"\(\forall \text{rt dip ip dsn hops.}
\text{the (nhop (update rt dip (dsn, unk, val, hops, ip)) dip)} = \text{ip}\)"
unfolding update_def by (clarsimp)


lemma nhop_update_changed [simp]:
fixes rt dip dsn dsk flg hops sip
assumes "update rt dip (dsn, dsk, flg, hops, sip) \neq rt"
shows "\(\text{the (nhop (update rt dip (dsn, dsk, flg, hops, sip)) dip)} = \text{sip}\)"


lemma update_rt_split_asm:
"\(\forall \text{rt ip dsn dsk flag hops sip.}
\text{P (update rt ip (dsn, dsk, flag, hops, sip))} =
(\neg (rt = \text{update rt ip (dsn, dsk, flag, hops, sip)} \land \neg \text{P rt})
\lor \neg \text{update rt ip (dsn, dsk, flag, hops, sip)}
\land \neg \text{P (update rt ip (dsn, dsk, flag, hops, sip))))\)"
by auto


lemma sqn_update [simp]: "\(\forall \text{rt dip dsn flg hops sip.}
rt \neq \text{update rt dip (dsn, kno, flg, hops, sip)}
\implies \text{sqn (update rt dip (dsn, kno, flg, hops, sip)) dip = dsn}\)"
unfolding update_def sqnf_def by (clarsimp)


lemma sqnf_update [simp]: "\(\forall \text{rt dip dsn dsk flg hops sip.}
rt \neq \text{update rt dip (dsn, dsk, flg, hops, sip)}
\implies \text{sqnf (update rt dip (dsn, dsk, flg, hops, sip)) dip = dsk}\)"
unfolding update_def sqnf_def by (clarsimp)

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


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


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


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


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
  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.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), 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 \("\forall rsn. \text{dests dip = Some rsn} \rightarrow \text{sqn rt dip} \leq rsn\)"
  shows \("\text{sqn rt dip} \leq \text{sqn (invalidate rt dests)} \text{ dip}\"
proof (cases "\text{dip} \notin \text{kD(rt)}")
  assume "\neg \text{dip} \notin \text{kD(rt)}"
  hence "\text{dip} \in \text{kD(rt)}" by simp
  then obtain dsn dsk flag hops nhip where \\
  \("\text{rt dip} = \text{Some (dsn, dsk, flag, hops, nhip)}"
  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 = rsn}\"
unfolding invalidate_def sqn_def
using assms(1) assms(2) [THEN kD_Some]
by clarsimp

lemma dhops_invalidate [simp]:
  "\(\forall \text{dip. the (dhops (invalidate rt dests) dip) = the (dhops rt dip)}\"
unfolding invalidate_def
by (clarsimp split: option.split)

lemma sqnf_invalidate [simp]:
  "\(\forall \text{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]:
  "\(\forall \text{dip. the (nhop (invalidate (rt \xi) (dests \xi)) dip) = the (nhop (rt \xi) dip)}\"
unfolding invalidate_def
by (clarsimp split: option.split)

lemma invalidate_other [simp]:
  fixes rt dests dip
nenumes \("\text{dip} \notin \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
nenumes \("\text{dip} \notin \text{kD(rt)}\"
  shows \("\text{invalidate rt dests dip = None}\"
using assms unfolding invalidate_def by clarsimp

lemma vD_invalidate_vD_not_dests:
  "\(\forall \text{dip rt dests dip} \in \text{vD(invalidate rt dests)} \rightarrow \text{dip} \in \text{vD(rt)} \land \text{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 \("\text{dip} \notin \text{dom(dests)}\"
  shows \("\text{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 \("\text{invalidate rt dests dip = Some (dsn, dsk, flag, hops, nhip)}"
  shows "\ dsn = (case dests dip of None \Rightarrow \pi_2(\text{the (rt dip)}) | \text{Some rsn} \Rightarrow rsn)\"
  \land dsk = \pi_3(\text{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: "\[\[ dip rt dests. dip∈kD (rt)
       \implies \pi₃(the (invalidate rt dests dip)) = \pi₃(the (rt dip))\]
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]))"

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 ≡ λdip. if dests dip = None then store dip
        else map_option (λ(_, q). (req, q)) (store dip)"

5.1.7 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. \( \sigma_{p\text{-}flag}(\text{store}, \text{dip}) \) 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 \texttt{invalidate}, and \texttt{store}.

end

5.2 AODV protocol messages

theory \texttt{E_Aodv_Message}
imports \texttt{E_All_ABCD}
begin

datatype msg =
\begin{align*}
\text{Rreq } & \text{n} \times \text{ip} \times \text{sqn} \times k \times \text{ip} \times \text{sqn} \times \text{ip} \times \text{bool} \\
\text{Rrep } & \text{n} \times \text{ip} \times \text{sqn} \times \text{ip} \\
\text{Rerr } & \text{"ip } \Rightarrow \text{sqn" } \text{ip} \\
\text{Newpkt } & \text{data} \times \text{ip} \\
\text{Pkt } & \text{data} \times \text{ip} \times \text{ip}
\end{align*}

instantiation msg :: msg
begin
\begin{align*}
definition \text{newpkt_def [simp]}: \text{"newpkt } \equiv \lambda(dip, dip). \text{Newpkt } d \text{ dip}"
\definition \text{eq_newpkt_def}: \text{"eq_newpkt } m \equiv \text{case } m \text{ of Newpkt } d \text{ dip } \Rightarrow \text{True } | \_ \Rightarrow \text{False}"
\end{align*}

\text{instance by intro_classes (simp add: eq_newpkt_def)}
end

The \texttt{msg} type models the different messages used within AODV. The instantiation as a \texttt{msg} is a technicality due to the special treatment of \texttt{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.

\textbf{definition} \texttt{rreq} :: "\texttt{nat} \times \texttt{ip} \times \texttt{sqn} \times k \times \texttt{ip} \times \texttt{sqn} \times \texttt{ip} \times \texttt{bool} \Rightarrow \texttt{msg}"
where "\texttt{rreq } \equiv \lambda(\text{hops, dip, dsn, dsk, oip, osn, sip, handled}). \text{Rreq } \text{hops } \text{dsn } \text{dsk } \text{oip } \text{osn } \text{sip } \text{handled}"
\texttt{lemma} \texttt{rreq_simp [simp]}:
"\texttt{rreq(hops, dip, dsn, dsk, oip, osn, sip, handled) } = \texttt{Rreq } \text{hops } \text{dsn } \text{dsk } \text{oip } \text{osn } \text{sip } \text{handled}"
\texttt{unfolding} \texttt{rreq_def by simp}

\textbf{definition} \texttt{rrep} :: "\texttt{nat} \times \texttt{ip} \times \texttt{sqn} \times \texttt{ip} \times \texttt{ip} \Rightarrow \texttt{msg}"
where "\texttt{rrep } \equiv \lambda(\text{hops, dip, dsn, oip, sip}). \text{Rrep } \text{hops } \text{dsn } \text{oip } \text{sip}"
\texttt{lemma} \texttt{rrep_simp [simp]}:
"\texttt{rrep(hops, dip, dsn, oip, sip) } = \texttt{Rrep } \text{hops } \text{dsn } \text{oip } \text{sip}"
\texttt{unfolding} \texttt{rrep_def by simp}

\textbf{definition} \texttt{rerr} :: "(\texttt{ip } \Rightarrow \texttt{sqn}) \times \texttt{ip} \Rightarrow \texttt{msg}"
where "\texttt{rerr } \equiv \lambda(\text{dests, sip}). \text{Rerr } \text{dests } \text{sip}"
\texttt{lemma} \texttt{rerr_simp [simp]}:
"\texttt{rerr(dests, sip) } = \texttt{Rerr } \text{dests } \text{sip}"
\texttt{unfolding} \texttt{rerr_def by simp}

\textbf{lemma} \texttt{not_eq_newpkt_rreq [simp]}: "\texttt{¬eq_newpkt } (\texttt{Rreq } \text{hops } \text{dsn } \text{dsk } \text{oip } \text{osn } \text{sip } \text{handled})"
\texttt{unfolding} \texttt{eq_newpkt_def by simp}

\textbf{lemma} \texttt{not_eq_newpkt_rrep [simp]}: "\texttt{¬eq_newpkt } (\texttt{Rrep } \text{hops } \text{dsn } \text{oip } \text{sip})"
\texttt{unfolding} \texttt{eq_newpkt_def by simp}

428
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]: "¬((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),
  dip := (SOME x. True),
  oip := (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_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

5.3.2 Auxiliary 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' dip' dsn' dsk' oip' osn' sip' handled' ⇒
  {ξ | hops := hops', dip := dip', dsn := dsn',
  dsk := dsk', oip := oip', osn := osn', sip := sip',
  handled := handled'}
| _ ⇒ {}"}

lemma is_rreq_asm [dest!]:
assumes "ξ ∈ is_rreq ξ"
shows "(∃ hops' dip' dsn' dsk' oip' osn' sip' handled'.
  msg ξ = Rreq hops' dip' dsn' dsk' oip' osn' sip' handled' ∧
  ξ' = ξ | hops := hops', dip := dip', dsn := dsn',
  dsk := dsk', oip := oip', osn := osn', sip := sip',
  handled := handled')"
using assms unfolding is_rreq_def
by (cases "msg ξ") simp_all

definition is_rrep
where "is_rrep ξ ≡ case msg ξ of
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(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(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 ⇒ 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]:
"\( \xi' \in is_rerr \, \xi \Rightarrow store \, \xi' = store \, \xi \)"
"\( \xi' \in is_rrep \, \xi \Rightarrow store \, \xi' = store \, \xi \)"
"\( \xi' \in is_rreq \, \xi \Rightarrow store \, \xi' = store \, \xi \)"
"\( \xi' \in is_pkt \, \xi \Rightarrow store \, \xi' = store \, \xi \)"
"\( \xi' \in is_newpkt \, \xi \Rightarrow store \, \xi' = store \, \xi \)"
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+) +

5.3.3 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]: "\( l1 \leq l2 = (nat_of_seqp l1 \leq nat_of_seqp l2) \)"
  definition less_seqp [iff]: "\( l1 < l2 = (nat_of_seqp l1 < nat_of_seqp l2) \)"
instance ..
end

abbreviation AODV
where
  "AODV \equiv \lambda _. [clear_locals] call(PAodv)"

abbreviation PKT
where
  "PKT args \equiv
   [\xi. let (data, dip, oip) = args \, \xi in
    (clear_locals \, \xi) () data := data, dip := dip, oip := oip ]
    call(PPkt)"

abbreviation NEWPKT
where
  "NEWPKT args \equiv
   [\xi. let (data, dip) = args \, \xi in
    (clear_locals \, \xi) () data := data, dip := dip ]
    call(PNewPkt)"

abbreviation RREQ
where
  "RREQ args \equiv
   [\xi. let (hops, dip, dsn, dsk, oip, osn, sip, handled) = args \, \xi in
    (clear_locals \, \xi) () hops := hops, dip := dip,
abbreviation RREP
where
"RREP args ≡
[ξ. let (hops, dip, ds, oi, si) = args ξ in
  (clear_locals ξ) () hops := hops, dip := dip, ds := ds,
  oi := oi, si := si ()]
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 (msg := msg')).
  (is_newpkt) NEWPKT(λξ. (data ξ, ip ξ))
  Ω (is_pkt) PKT(λξ. (data ξ, dip ξ, oip ξ))
  Ω (is_rreq)
    [ξ. ξ (rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ) ()]
     RREQ(λξ. (hops ξ, dip ξ, ds ξ, dk ξ, oip ξ, osn ξ, sip ξ, handled ξ))
  Ω (is_rerr)
    [ξ. ξ (rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ) ()]
     RERR(λξ. (hops ξ, dip ξ, ds ξ, oip ξ, sip ξ))
  Ω (is_rerr)
    [ξ. ξ (rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ) ()]
     RERR(λξ. (dests ξ, sip ξ))
  Ω (λξ. {ξ (dip := dip {}) / dip. dip ∈ qD(store ξ) \ vD(rt ξ) {ξ})
    [ξ. ξ () data := hd(queue(store ξ, dip ξ) {})]
    unicast(λξ. the (nhop (rt ξ) (dip ξ), λξ. pkt(data ξ, dip ξ, ip ξ))).
    [ξ. ξ () store := (drop (dip ξ) (store ξ) {})]
    AODV()
  > [ξ. ξ () dests := (∀rips. 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. err(dests ξ, ip ξ)). AODV()
  Ω (λξ. {ξ (dip := dip {}) / dip. dip ∈ qD(store ξ) \ vD(rt ξ) \ the (σp-flag(store ξ, dip)) = req {})
    [ξ. ξ () store := unsetRRF (store ξ) (dip ξ) {}]
    [ξ. ξ () sn := inc (sn ξ) {}]
    [ξ. ξ () rreq := rreqs ∪ {(ip ξ, sn ξ) {ξ}}]
    broadcast(λξ. rreq(0, dip ξ, sqn (rt ξ) (dip ξ), sqn (rt ξ) (dip ξ), ip ξ, sn ξ, ip ξ, False)). AODV())"

/ "ΓAoDV PNewPkt = labelled PNewPkt {
  (ξ. dip ξ = ip ξ)
  deliver(λξ. data ξ).AODV()
  Ω (ξ. dip ξ ≠ ip ξ)
    [ξ. ξ () store := add (data ξ) (dip ξ) (store ξ) {}]
    AODV()"

/ "ΓAoDV PPkt = labelled PPkt {
  (ξ. dip ξ = ip ξ)
  deliver(λξ. data ξ).AODV()
  Ω (ξ. dip ξ ≠ ip ξ)
\[
\langle \xi. \text{dip } \xi \in \nuD (rt \xi) \rangle
\]
\[
\quad \text{unicast}(\lambda \xi. \text{the } (\text{nhop } (rt \xi) (\text{dip } \xi)), \lambda \xi. \text{pkt(data } \xi, \text{ dip } \xi, \text{ oip } \xi)).AODV()
\]
\[
\triangleright \quad \langle \xi. \xi \not\in \nuD (rt \xi) \rangle
\]
\[
\quad \text{dests } := (\lambda \text{rip}. \text{if } (\text{rip } \in \nuD (rt \xi) \land \text{nhop } (rt \xi) \text{ rip } = \text{nhop } (rt \xi) (\text{dip } \xi))
\quad \text{then } \text{Some (inc } (\text{sqn } (rt \xi) \text{ rip} )) \text{ else None } ) \rangle
\]
\[
\langle \xi. \xi \not\in \nuD (rt \xi) \rangle
\]
\[
\quad \text{rt } := \text{invalidate } (rt \xi) (\text{dests } \xi) \rangle
\]
\[
\langle \xi. \xi \not\in \text{store } := \text{setRRF } (\text{store } \xi) (\text{dests } \xi) \rangle
\]
\[
\quad \text{broadcast}(\lambda \xi. \text{rerr(dests } \xi, \text{ ip } \xi)).AODV()
\]
\[
\triangleright \langle \xi. \text{dip } \xi \not\in \nuD (rt \xi) \rangle
\]
\[
\quad \text{broadcast}(\lambda \xi. \text{rerr}((\text{dip } \xi \rightarrow \text{sqn } (rt \xi) (\text{dip } \xi)), \text{ip } \xi)). AODV()
\]
\[
\triangleright \langle \xi. \text{handled } \xi = \text{False} \rangle
\]
\[
\quad \langle \xi. \text{dip } \xi = \text{ip } \xi \rangle
\]
\[
\quad \langle \xi. \xi \not\in \text{sn } := \text{max } (\text{sn } \xi) (\text{dsn } \xi) \rangle
\]
\[
\quad \text{unicast}(\lambda \xi. \text{the } (\text{nhop } (rt \xi) (\text{oip } \xi)), \lambda \xi. \text{rrep(0, dip } \xi, \text{ sn } \xi, \text{ oip } \xi, \text{ ip } \xi)).
\quad \text{broadcast}(\lambda \xi. \text{rreq(hops } \xi + 1, \text{ dip } \xi, \text{ dsn } \xi, \text{ dsk } \xi, \text{ oip } \xi, \text{ osn } \xi, \text{ ip } \xi, \text{ True})). AODV()
\]
\[
\triangleright \langle \xi. \xi \not\in \text{ip } \xi \rangle
\]
\[
\quad \langle \xi. \text{dip } \xi \in \nuD (rt \xi) \land \text{dsn } \xi \leq \text{sqn } (rt \xi) (\text{dip } \xi) \land \text{sqnf } (rt \xi) (\text{dip } \xi) = \text{kno}
\quad \text{unicast}(\lambda \xi. \text{the } (\text{nhop } (rt \xi) (\text{oip } \xi)), \lambda \xi. \text{rrep(the } (\text{dhops } (rt \xi) (\text{dip } \xi)), \text{ dip } \xi,
\quad \text{sqn } (rt \xi) (\text{dip } \xi), \text{ oip } \xi, \text{ ip } \xi)).
\quad \text{broadcast}(\lambda \xi. \text{rreq(hops } \xi + 1, \text{ dip } \xi, \text{ dsn } \xi, \text{ dsk } \xi, \text{ oip } \xi, \text{ osn } \xi, \text{ ip } \xi, \text{ True})). AODV()
\]
\[
\triangleright \langle \xi. \xi \not\in \nuD (rt \xi) \lor \text{sqn } (rt \xi) (\text{dip } \xi) < \text{dsn } \xi \lor \text{sqnf } (rt \xi) (\text{dip } \xi) = \text{unk}
\quad \text{broadcast}(\lambda \xi. \text{rreq(hops } \xi + 1, \text{ dip } \xi, \text{ max } (\text{sqn } (rt \xi) (\text{dip } \xi)) (\text{dsn } \xi),
\quad \text{dsk } \xi, \text{ oip } \xi, \text{ osn } \xi, \text{ ip } \xi, \text{ False})). AODV()
\]
\[
\quad AODV()
\]
(ξ. oip ξ = ip ξ)
AODV()
⊕ (ξ. oip ξ ≠ ip ξ)
(ξ. oip ξ ∈ vD (rt ξ) ∧ dip ξ ∈ vD (rt ξ))
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)).

AODV()
⊕ [ξ. dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (oip ξ)) then Some (inc (sqn (rt ξ) rip)) else None) ]
[ξ. rt := invalidate (rt ξ) (dests ξ) ]
[ξ. store := setRRF (store ξ) (dests ξ) ]
broadcast(λξ. rerr(dests ξ, ip ξ)).AODV()
⊕ (ξ. dests ξ ≠ Map.empty)
AODV()
\[ p \in \text{ctermsl}(\Gamma_{AODV} \text{ PPkt}) \lor \]
\[ p \in \text{ctermsl}(\Gamma_{AODV} \text{ PRreq}) \lor \]
\[ p \in \text{ctermsl}(\Gamma_{AODV} \text{ PRrep}) \lor \]
\[ p \in \text{ctermsl}(\Gamma_{AODV} \text{ PRerr}) \]

using assms
by (cases \( pn \)) simp_all

definition \( \sigma_{AODV} \) :: \( \text{ip} \Rightarrow (\text{state} \times (\text{state}, \text{msg}, \text{pseq}, \text{pseq label}) \text{ seqp}) \text{ set} \)
where "\( \sigma_{AODV} i \equiv \{(aodv\_init i, \Gamma_{AODV} \text{ PAodv})\} \)

abbreviation \( \text{paodv} \) :: \( \text{ip} \Rightarrow (\text{state} \times (\text{state}, \text{msg}, \text{pseq}, \text{pseq label}) \text{ seqp}, \text{msg seq_action}) \text{ automaton} \)
where "\( \text{paodv i} \equiv (| \text{init} = \sigma_{AODV} i, \text{trans} = \text{seqp sos} \Gamma_{AODV} |) \)

lemma aodv_trans: \( \text{trans (paodv i) = seqp sos} \Gamma_{AODV} \)
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: \( \Gamma_{AODV} \_\text{simps} \))

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} \text{ p}"
by (metis aodv_labels_not_empty all_not_in_conv)

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

lemma aodv_simple_labels [simp]: "\text{simple_labels} \Gamma_{AODV}"
proof
fix \( pn \ p \)
assume "p \in \text{subterms}(\Gamma_{AODV} \text{ pn})"
thus "\exists \ l. \ \text{labels} \Gamma_{AODV} \text{ 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 \Longrightarrow \text{labels} \Gamma_{AODV} \ p = \{PAodv:-0\}"
unfolding \( \sigma_{AODV} \_\text{def} \) by simp

lemma aodv_init_kD_empty [simp]:
"(\xi, p) \in \sigma_{AODV} \ i \Longrightarrow \text{kD (rt } \xi) = \{\}"
unfolding \( \sigma_{AODV} \_\text{def} \) kD_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 "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 seq_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

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 _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
lemma rreq_rrep_sn_simps [simp]:
  "\hops dip dsn dsk oip osn sip handled.
    rreq_rrep_sn (Rreq hops dip dsn dsk oip osn sip handled) = (osn \geq 1)"
  "\hops dip dsn oip sip. rreq_rrep_sn (Rreq hops dip dsn oip sip) = (dsn \geq 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 \Rightarrow msg \Rightarrow bool"
where "rreq_rrep_fresh crt m \equiv \case m of Rreq hopsc _ _ oipc osnc ipcc _ 
  \Rightarrow (ipcc \neq oipc \rightarrow
    ipcc \in kD(crt) \land (sqn crt oipc > osnc
    \lor (sqn crt oipc = osnc
    \land (dhops crt oipc) \leq hopsc
    \land (flag crt oipc) = val)))
  \land (dhops crt oipc) \leq hopsc
  \land (flag crt oipc) = val))
  \land (dhops crt oipc) \leq hopsc
  \land (flag crt oipc) = val)
  \land (dhops crt oipc) \leq hopsc
  \land (flag crt oipc) = val)"

lemma rreq_rrep_fresh [simp]:
  "\hops dip dsn dsk oip osn sip handled.
    rreq_rrep_fresh crt (Rreq hops dip dsn dsk oip osn sip handled) = (sip \neq oip \rightarrow
      dip \in kD(crt)
      \land (sqn crt dip = dsn
      \land (dhops crt dip) = hops
      \land (flag crt dip) = val)"
  "\hops dip dsn oip sip. rreq_rrep_fresh crt (Rreq hops dip dsn oip sip) = (sip \neq dip \rightarrow
      dip \in kD(crt)
      \land (sqn crt dip = dsn
      \land (dhops crt dip) = hops
      \land (flag crt dip) = val)
      \land (dhops crt dip) = hops
      \land (flag crt dip) = val)
      \land (dhops crt dip) = hops
      \land (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 \Rightarrow msg \Rightarrow bool"
where "rerr_invalid crt m \equiv \case m of Rerr destsc _ 
  \Rightarrow (\forall ripc \in dom(destsc).
    (ripc \in iD(crt) \land the (destsc ripc) = sqn crt ripc))
  \land (dhops crt ripc) \leq hopsc
  \land (flag crt ripc) = val)"

lemma rerr_invalid [simp]:
  "\hops dip dsn dsk oip osn sip handled.
    rerr_invalid crt (Rreq hops dip dsn dsk oip osn sip handled) = True"
  "\hops dip dsn oip sip. rerr_invalid crt (Rreq hops dip dsn oip sip) = True"
  "\dests sip. rerr_invalid crt (Rerr dests sip) = (\forall ripc \in dom(dests).
    rip \in iD(crt) \land the (dests rip) = sqn crt rip)
    \land (dhops crt ripc) \leq hopsc
    \land (flag crt ripc) = val)
  \land (dhops crt ripc) \leq hopsc
  \land (flag crt ripc) = val)
  \land (dhops crt ripc) \leq hopsc
  \land (flag crt ripc) = val)
  \land (dhops crt ripc) \leq hopsc
  \land (flag crt ripc) = val)
  \land (dhops crt ripc) \leq hopsc
  \land (flag crt ripc) = val)
unfolding rerr_invalid_def by simp_all

definition initmissing :: "nat \Rightarrow state option \times 'a \Rightarrow (nat \Rightarrow state) \times 'a"
where
  "initmissing = (\lambda i. \case (fst \sigma) i of None \Rightarrow aodv_init i \mid Some s \Rightarrow s, snd \sigma)"

lemma not_in_net_ips_fst_init_missing [simp]:

assumes "i \notin \text{net_ips} \sigma"
shows "\text{fst (initmissing (netgmap \text{fst} \sigma))} 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} :: "((\text{nat} ⇒ \text{state}) ⇒ \text{bool}) ⇒ ((\text{state} × \text{'b}) × \text{'c}) \text{net_state} ⇒ \text{bool}"
where "\text{netglobal P} ≡ (\lambda s. P (\text{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 } \pi_4(r) = \text{val} ∨ \pi_2(r) = 0 \text{ then } \pi_2(r) \text{ else } (\pi_2(r) - 1)"

lemma \text{nsqn\_r_def'}:
"\text{nsqn \_r} r = (\text{if } \pi_4(r) = \text{inv then } \pi_2(r) - 1 \text{ else } \pi_2(r))"
unfolding \text{nsqn\_r_def} by simp

lemma \text{nsqn\_r_zero [simp]}:
"\text{\&\& dsn dsk flag hops nhip. nsqn\_r} (dsn, dsk, flag, hops, nhip) = 0"
unfolding \text{nsqn\_r_def} by clarsimp

lemma \text{nsqn\_r_val [simp]}:
"\text{\&\& dsn dsk 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 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 flag hops nhip. nsqn\_r} (dsn, dsk, flag, hops, nhip) \leq dsn"
unfolding \text{nsqn\_r_def} by clarsimp

On routes in routing tables

definition \text{nsqn :: "rt ⇒ ip ⇒ sqn"}
where "\text{nsqn} ≡ \lambda rt dip. \text{case } \sigma_{\text{route}}(rt, dip) \text{ of } \text{None} ⇒ 0 | \text{Some } r ⇒ \text{nsqn\_r}(r)"

439
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 nsqn_not_kD_sqn [simp]:
  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"
  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)
  (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 unfolding invalidate_def
  by (clarsimp)

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" ("_ ⊑ _") [51, 51] 50)
where
  "fresher r r' ≡ ((nsqn r r < nsqn r r') ∨ (nsqn r r = nsqn r r' ∧ π_5(r) ≥ π_5(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 "π_5(r) ≥ π_5(r')"
  shows "r ⊑ r'"
  unfolding fresher_def using assms by simp

lemma fresherI [intro]:
  assumes "(nsqn r r < nsqn r r') ∨ (nsqn r r = nsqn r r' ∧ π_5(r) ≥ π_5(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' ∧ π_5(r) ≥ π_5(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'
shows "dsn (dsn, dsk, flag, hops, nhip) ⊑ (dsn, dsk', flag, hops, nhip')"
unfolding fresher_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 ⊑_ip rt2 ≡ rt_fresher i rt1 rt2"

lemma rt_fresher_def':
  "(rt1 ⊑_ip 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 ⊑_ip 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 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_fresherII2 [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 "\[ \text{nsqn } rt1 \text{ dip } < \text{nsqn } rt2 \text{ dip } \] \implies P rt1 rt2 dip"
and "\[ \text{nsqn } rt1 \text{ dip } = \text{nsqn } rt2 \text{ dip}; \\
\text{the (dhops rt1 dip)} \geq \text{the (dhops rt2 dip)} \] \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" ("(_/≈/ _)"
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 "\[ \text{nsqn } rt \text{ dip } = \text{nsqn } rt' \text{ dip } \]"
and "\( π_5(\text{the (rt dip)}) = π_5(\text{the (rt' dip)}) \)"
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 "¬(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_nsqr:
assumes "dip ∈ kD(rt1)"
and "dip ∈ kD(rt2)"
and "rt1 ≈_{dip} rt2"
shows "nsqr (the (rt2 dip)) = nsqr (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_nsqr [OF ⟨dip ∈ kD(rt1)⟩]
kD_nsqr [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 rtn [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)⟩ 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 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 nhp
  from ⟨hops_n ≥ 1⟩ have "(dsn_n, dsk_n, f_n, hops_n, nhip_n) ⊆ (dsn_n, unk, val, Suc 0, nhp)"

445
unfolding fresher_def sqn_def by (cases $f_n$) auto
thus "$(dsn_n, dsnk_n, f_n, hops_n, nhip_n)$
\[)$ \leq$ the (update rt dip (0, unk, val, Suc 0, nhip) dip)"
using (dip $\in$ kD(rt)) by - (rule update_cases_kD, simp_all)
next
fix dsn :: sqn and hops nhip
assume "0 < dsn"
show "$(dsn_n, dsnk_n, f_n, hops_n, nhip_n)$
\[)$ \leq$ (dsn, kno, val, hops, nhip)"
proof
unfolding fresher_def by auto
next
assume "dsn = dsn"
and "hops < hops_n"
thus "$(dsn, dsnk_n, f_n, hops_n, nhip_n)$
\[)$ \leq$ (dsn, kno, val, hops, nhip)"
next
assume "dsn = dsn"
with (0 < dsn)
show "$(dsn, dsnk_n, inv, hops_n, nhip_n)$
\[)$ \leq$ (dsn, kno, val, hops, nhip)"
qed

hence "rt $\subseteq$ dip update rt dip r"
by - (rule single_rt_fresher, simp)

theorem rt_fresher_invalidate [simp]:
assumes "dip $\in$ kD(rt)"
and indests: "\(\forall rip \in \text{dom}(dests) \cdot rip \in vD(rt) \land sqn rt rip < \text{the (dests rip)}\)"
shows "rt $\nsubseteq$ dip invalidate rt dests"
proof (cases "dip $\in$ dom(dests)"
assume "dip $\in$ dom(dests)"
thus ?thesis using (dip $\in$ kD(rt))
by - (rule single_rt_fresher, simp)
next
assume "dip $\in$ dom(dests)"
moredwith 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 $\in$ 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 $\in$ kD(rt)"
and "\(\forall rip \in \text{dom}(dests) \cdot rip \in vD(rt) \land \text{the (dests rip)} = inc \ (sqn rt rip)\)"
shows "rt $\approx$ dip invalidate rt dests"
proof (cases "dip $\in$ dom(dests)"
assume "dip $\in$ dom(dests)"
with (dip $\in$ kD(rt)) have "dip $\in$ 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 ?thesis
    proof (rule nsqn_rt_fresh_asI)
    from ⟨dip ∈ vD(rt)⟩ have "nsqn rt dip = sqn rt dip" by simp
    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))" by simp
    qed
    also from ⟨dip ∈ kD(invalidate rt dests)⟩
    have "nsqn r (the (invalidate rt dests dip)) = nsqn (invalidate rt dests) dip"
      by (simp add: kD_nsqn)
    finally have "nsqn rt dip = nsqn (invalidate rt dests) dip".
    qed simp
  qed

lemmas rt_fresher_inc_invalidate [simp] = rt_fresh_as_inc_invalidate [THEN rt_fresh_asD1]

5.5.4 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 ⊑ i rt2 = ((rt1 ⊑ dip rt2) ∧ ¬(rt2 ⊑ dip 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 ≈ dip 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 \sqsubseteq_1 rt2" and "[ rt1 \sqsubseteq_1 rt2; \neg(rt1 \approx_1 rt2 ) ] \implies 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_1 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 (rt1 i)) > \pi_5 (the (rt2 i))))"

unfolding rt_strictly_fresher_def'' rt_fresher_def fresher_def by auto

lemma rt_strictly_fresher_fresherD [dest]:
assumes "rt1 \sqsubseteq_1 dip rt2"
shows "the (rt1 dip) \sqsubseteq 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 \sqsubseteq_1 dip rt2"
shows "\neg rt1 \approx dip rt2"
using assms unfolding rt_strictly_fresher_def by auto

lemma rt_strictly_fresher_trans [elim, trans]:
assumes "rt1 \sqsubseteq_1 dip rt2"
and "rt2 \sqsubseteq_1 dip rt3"
shows "rt1 \sqsubseteq_1 dip rt3"
using assms proof -
from rt1 \sqsubseteq_1 dip rt2 obtain "the (rt1 dip) \sqsubseteq dip the (rt2 dip)" by auto
also from rt2 \sqsubseteq_1 dip rt3 obtain "the (rt2 dip) \sqsubseteq dip the (rt3 dip)" by auto
finally have "the (rt1 dip) \sqsubseteq dip the (rt3 dip)" .

moreover have "\neg (rt1 \approx dip rt3)"
proof -
from rt1 \sqsubseteq_1 dip rt2 obtain "\neg (the (rt2 dip) \sqsubseteq dip the (rt1 dip))" by auto
also from rt2 \sqsubseteq_1 dip rt3 obtain "\neg (the (rt3 dip) \sqsubseteq dip the (rt2 dip))" by auto
finally have "\neg (the (rt3 dip) \sqsubseteq dip the (rt1 dip))" .
thus ?thesis ..
qed
ultimately show "rt1 \sqsubseteq dip rt3" ..
qed

lemma rt_strictly_fresher_irefl [simp]: "\neg (rt \sqsubseteq dip rt)"
using assms unfolding rt_strictly_fresher_def
by clarsimp

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 rt1 \sqsubseteq dip rt2 have "rt1 \sqsubseteq dip rt2"
and "\neg(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_nh)
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

448
lemma \texttt{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 \texttt{rt\_strictly\_fresher\_def''} by auto
  from rt1 \sqsubseteq dip rt2: and this(1) have "rt1 \sqsubseteq dip rt3" ..
moreover from "¬(rt3 \sqsubseteq dip rt2)" have "¬(rt3 \sqsubseteq dip rt1)"
proof (rule contrapos_nn)
  assume "rt3 \sqsubseteq dip rt1"
  hence "rt2 \sqsubseteq dip rt1" ..
  hence "nsqn rt2 dip \leq nsqn rt1 dip"
  using ⟨dip \in kD(rt2)⟩ ⟨dip \in kD(rt1)⟩ by (rule \texttt{rt_fresher\_imp\_nsqn\_le})
  with ⟨nsqn rt1 dip < nsqn rt2 dip⟩ show "False"
  by simp
qed

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

lemma \texttt{rt\_fresher\_imp\_nsqn\_le}:
assumes "rt1 \sqsubseteq dip 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: \texttt{rt\_fresher\_def2} [OF assms(2-3)])

lemma \texttt{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 \texttt{rt\_fresher\_imp\_nsqn\_le})
    with ⟨nsqn rt1 dip < nsqn rt2 dip⟩ show "False"
    by simp
  qed
qed

lemma \texttt{rt\_strictly\_fresher\_eqI} [intro]:
assumes "i \in kD(rt1)"
and "i \in kD(rt2)"
and "nsqn rt1 i = nsqn rt2 i"
and "\pi_5\text{(the \texttt{(rt1 i)})} < \pi_5\text{(the \texttt{(rt2 i)})}"
shows "rt1 \sqsubseteq dip rt2"
using assms unfolding \texttt{rt\_strictly\_fresher\_def'} by (auto simp add: kD_nsqn)

lemma \texttt{invalidate\_rtsf\_left} [simp]:
"\\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)
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 s: "osn < sqn (rt2 nhip) dip"
  and h: "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 h 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 s)
also have "sqn (rt2 nhip) dip = osn, (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 s: "osn < sqn (rt2 nhip) dip"
  and h: "the (dhops (rt2 nhip) dip) ≤ hops"
  and h': "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 h' 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, (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 s)
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 "the (dhops (rt2 nhip) dip) ≤ hops" by (rule h)
also have "hops < hops + 1" by simp
also have "hops + 1 = the (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 "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 -
  { fix ip and ξ ξ' :: state
    assume "∀ dip∈kD(rt). 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)."
}
  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 |= 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 |= (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, 1). l ∈ {{PAodv=-7} ∪ {PAodv=-5} ∪ {PRreq=0..PRreq=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)"
} note one_hop = this

{ fix ip sn hops and ξ ξ' :: state 
  assume "∀ dip ∈ kD (rt ξ). the (nhop (update (rt ξ) dip) ip) = ip \\
 ∨ the (nhop (update (rt ξ) dip) ip) ∈ kD (rt ξ)"
  hence "(the (nhop (update (rt ξ) ip (sn, kno, val, Suc hops, sip)) dip) = ip \\
 ∨ the (nhop (update (rt ξ) dip) ip) ∈ kD (rt ξ))"
} 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 ΓAODV (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 ΓAODV (λ((ξ, _), _, (ξ', _)). rreqs ξ ⊆ rreqs ξ')"
by (inv_cterms simp add: subset_insertI)

lemma dests_bigger_than_sqn:
"paodv i |= onl ΓAODV (λ(ξ, 1). 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 |= onl ΓAODV (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: 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_1_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) auto

lemma rreq_dip_in_vD_dip_eq_ip:
"paodv i |= onl Γ_AODV (λ(ξ, l). (l ∈ {PRreq-:16..PRreq-:17} −→ dip ξ ∈ vD(rt ξ))
∧ (l ∈ {PRreq-:6..PRreq-:7} −→ dip ξ = ip ξ)
∧ (l ∈ {PRreq-:15..PRreq-:17} −→ dip ξ ≠ ip ξ))"
by inv_cterms

lemma rrep_dip_in_vD:
"paodv i |= onl Γ_AODV (λ(ξ, l). (l ∈ {PRrep-:4} −→ dip ξ ∈ vD(rt ξ)))"
by inv_cterms

Proposition 7.11

lemma anycast_msg_zhops:
"∀rreqid dip dsn dsk oip osn sip.
 paodv i |=A onll Γ_AODV (λ(_, a, _). anycast msg_zhops a)"

453
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 :: pp p' pp'
assume "l ∈ reachable (paodv i) \ TT"
  and "∀ l ∈ reachable (paodv i) \ TT. Suc 0 ≤ (dips (rt l) (oip l))" (simp_all)
  thus "0 < (dips (rt l) (ip l))" by simp
next
fix l :: pp p' pp'
assume "l ∈ reachable (paodv i) \ TT"
  and "∀ l ∈ reachable (paodv i) \ TT. Suc 0 ≤ (dips (rt l) (oip l))" (simp_all)
  thus "0 < (dips (rt l) (ip l))" by simp
qed

lemma hop_count_zero_oip_dip_sip:
  "paodv i \= (recvmsg msg_zhops \→) onl Γ_AODV (λ(ξ, 1). (Suc 0 \→ oip ξ = sip ξ))" (auto)

lemma osn_rreq:
  "paodv i \= (recvmsg rreq_rrep_sn \→) onl Γ_AODV (λ(ξ, 1). 1 ∈ {PAodv-:6, PAodv-:7} \union \ PRreq-nn True \→ 1 ≤ osn ξ)" (clarsimp)

lemma osn_rreq':
  "paodv i \= (recvmsg (λm. rreq_rrep_sn m \and msg_zhops m) \→) onl Γ_AODV (λ(ξ, 1). 1 ∈ {PAodv-:6, PAodv-:7} \union \ PRreq-nn True \→ 1 ≤ osn ξ)" (clarsimp)
proof (rule invariant_weakenE [of osn_rreq])
fix a
  assume "recvmsg (λm. rreq_rrep_sn m \and msg_zhops m) a" (clarsimp)
  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} \union \ PRreq-nn 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 (\lambda m. rreq_rrep_sn m \land msg_zhops m) \rightarrow) onl_{AODV} (\lambda(\xi, l).
  \begin{cases}
    \{PAodv-:6, PAodv-:7\} \cup \{PRrep-:n|n. True\} \rightarrow 1 \leq dsn (\xi)\
  \end{cases})"

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_{AODV} (\lambda(\xi, l).
  \begin{cases}
    \{PAodv-:4..PAodv-:5\} \cup \{PRreq-:n|n. True\} \rightarrow (hops (\xi) = 0 \rightarrow oip (\xi) = sip (\xi))\
    \{PAodv-:6..PAodv-:7\} \cup \{PRrep-:n|n. True\} \rightarrow (hops (\xi) = 0 \rightarrow dip (\xi) = sip (\xi)))\
  \end{cases})"

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_{AODV} (\lambda(\xi, l).
  \begin{cases}
    \forall dip \in kD(rt (\xi)). (sqn (rt (\xi)) dip = 0 \rightarrow sqnf (rt (\xi)) dip = unk) \\
    \land (sqnf (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))\
  \end{cases})"

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 \rightarrow 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 \in kD(rt).
    \begin{cases}
      (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)\
    \end{cases}"
    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).
    \begin{cases}
      (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)\
    \end{cases}"
    and "Suc 0 \leq dsn"
  }
and "ip \neq dip \rightarrow ip \in kD(rt)"

hence "\pi_3 \left( \text{the (update rt dip (dsn, kno, Suc hops, sip)} ip) \right) = unk \rightarrow
the \left( \text{dhops (update rt dip (dsn, kno, Suc hops, sip)} ip) \right) = \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{(dhops rt dip) = Suc 0}) \land
(\text{(dhops rt dip)} = Suc 0 \rightarrow \text{(nhop rt dip)} = dip)"
and "Suc 0 \leq dsn"
and "ip \neq dip \rightarrow ip \in kD(rt)"

hence "sqn \left( \text{update rt dip (dsn, kno, val, Suc hops, sip)} ip \right) = 0 \rightarrow
\pi_3 \left( \text{the (update rt dip (dsn, kno, val, Suc hops, sip)} ip \right) = unk"
by - (rule update_cases, auto simp add: sqn_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{(dhops rt dip)} = Suc 0) \land
(\text{(dhops rt dip)} = Suc 0 \rightarrow \text{(nhop rt dip)} = dip)"

hence "\forall dip \in kD rt.
(sqn \left( \text{update rt sip (0, unk, val, Suc 0, sip)} dip \right) = 0 \rightarrow
\pi_3(\text{the (update rt sip (0, unk, val, Suc 0, sip)} dip)) = unk \land
\text{(dhops (update rt sip (0, unk, val, Suc 0, sip)) dip) = Suc 0) \land
(\text{(dhops (update rt sip (0, unk, val, Suc 0, sip)) dip)} = Suc 0 \rightarrow
\text{(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]: "\forall sip rt.
\pi_3(\text{the (update rt sip (0, unk, val, Suc 0, sip)} sip)) = unk \rightarrow
\text{(dhops (update rt sip (0, unk, val, Suc 0, sip)) sip) = Suc 0)"
by (rule update_cases) simp_all

have prreq_ok6 [simp]: "\forall sip rt.
sqn \left( \text{update rt sip (0, unk, val, Suc 0, sip)} sip \right) = 0 \rightarrow
\pi_3(\text{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’t]
seq_step_invariant_sterms_TT [OF sqns_increase aodv_wf aodv_trans]
onl_invariant_sterms [OF aodv_wf osn_rreq’t]
onl_invariant_sterms [OF aodv_wf dsn_rrep’t]) clarsimp+

qed

lemma zero_seq_unk_hops_one:
"paodv i \models \text{(recvmsg (\lambda m. rreq_rrep_sn m \land msg_zhops m) \rightarrow onl_\text{AODV} (\lambda (\xi, _).}
\forall dip \in kD(rt \xi). (sqn (rt \xi) dip = 0 \rightarrow \text{(sqnf (rt \xi dip)} = unk \land
\text{the (dhops (rt \xi dip)} = 1 \land
\text{the (nhop (rt \xi dip)} = dip)))""
by (rule invariant_weakenE [OF zero_seq_unk_hops_one’t]) auto

lemma kD_unk_or_atleast_one:
"paodv i \models \text{(recvmsg rreq_rrep_sn \rightarrow) onl_\text{AODV} (\lambda (\xi, 1).}
\forall dip \in kD(rt \xi). \pi_3(\text{the (rt \xi dip)} = unk \lor 1 \leq \pi_2(\text{the (rt \xi dip)}))"
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(\text{the (update rt sip (dsn1, dsk1, flag1, hops1, nhip1, nhip2) sip}) = unk

 Suc 0 ≤ sqn (update rt sip (dsn1, dsk1, flag1, hops1, nhip1) dip)
 unfolding update_def by (cases "dsk1 = unk") (clarsimp split: option.split)+
 note fromsip [simp] = this

{ fix dip sip rt dsn1 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
   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_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)
 qed
Proposition 7.13

**Lemma rreq_rrep_sn_any_step_invariant:**

"paodv i ⊨_A (recvmsg rreq_rrep_sn → onl Γ_{AODV} (λ(\_, a, \_). anycast rreq_rrep_sn a))"

**Proof -**

have sqnf_kno: "paodv i \(\vdash\) onl Γ_{AODV} (λ(ξ, 1).

\((l ∈ \{PRreq-:16\} → dip ξ ∈ kD (rt ξ) ∧ sqnf (rt ξ) (dip ξ) = kno))"

by (inv_cterms)

have rep_sqn_greater_dsn: "paodv i \(\vdash\) (recvmsg rreq_rrep_sn → onl Γ_{AODV} (λ(ξ, 1).

\((l ∈ \{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])

(auto simp: update_kno_dsn_greater_zero [simplified])

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 [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 \(\vdash\) onl Γ_{AODV} (λ(ξ, 1).

\((l ∈ \{PRreq-:3..PRreq-:9\} ∪ \{PRreq-:15, PRreq-:24, PRreq-:26\} → oip ξ ∈ kD (rt ξ)

∧ (sqn (rt ξ) (oip ξ) > (osn 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

fix \(l \ xi \ l' pp p'\)

assume "\((\xi, pp) ∈ reachable (paodv i) TT"

and "\{PRreq-:2\}[\xi, 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)

(auto simp linorder_neqE_nat not_less)

qed

have rrep_prrep: "paodv i \(\vdash\) onl Γ_{AODV} (λ(ξ, 1).

\((l ∈ \{PRreq-:4\} → (dip ξ ∈ kD (rt ξ)

∧ 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 \(\vdash\) onl Γ_{AODV} (λ(ξ, 1).

\((l ∈ \{PRreq-:15, PRreq-:24, PRreq-:26\} → oip ξ ∈ kD (rt ξ)))"

by (inv_cterms)

have rreq_dip_kD_oip_sqn: "paodv i \(\vdash\) onl Γ_{AODV} (λ(ξ, 1).

\((l ∈ \{PRreq-:16..PRreq-:17\} → (dip ξ ∈ kD (rt ξ)

∧ (sqn (rt ξ) (oip ξ)) > (osn ξ)"
Proposition 7.15

```text
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 ||=\ A onll Γ AODV (λ(ξ, 1). (l ∈ {PAodv-:15, PPkt-:7, PRreq-:11, PRreq-:20, PRrep-:7, PRerr-:1} \\
                                       ∪ {PPkt-:8..PPkt-:9} \\
                                       ∪ {PRreq-:12..PRreq-:13} \\
                                       ∪ {PRreq-:21..PRreq-:22} \\
                                       ∪ {PRrep-:8..PRrep-:9} \\
                                       ∪ {PRerr-:2..PRerr-:4} → (∀ip∈dom(dests ξ). ip∈vD(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 simp: domIff)+
  show ?thesis
  by (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]
  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 ||=\ A onll Γ AODV (λ(ξ, 1). (l ∈ {PAodv-:15, PPkt-:7, PRreq-:11, PRreq-:20, PRrep-:7} \\
                                           ∪ {PPkt-:8..PPkt-:9} \\
                                           ∪ {PRreq-:12..PRreq-:13} \\
                                           ∪ {PRreq-:21..PRreq-:22} \\
                                           ∪ {PRrep-:8..PRrep-:9} \\
                                           ∪ {PRerr-:2..PRerr-:4} → (∀ip∈dom(dests ξ). ip∈vD(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 simp: domIff)+
```

Proposition 7.27

```
lemma route_tables_fresher:
  "paodv i ||=\ A (recvmsg rreq_rrep_sn → 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_rrep]
```

459
5.7 The quality increases predicate
fixes \( ip \)
assumes "quality_increases \( \xi \xi \)"
and "\( ip \in kD(\xi) \)"
shows "\( \xi \subseteq ip \rt \xi \)"
using assms by auto

lemma quality_increases_sqnE [elim]:
fixes dip
assumes "quality_increases \( \xi \xi \)"
and "sqn (\rt \xi \) dip \leq sqn (\rt \xi \) dip \Rightarrow R \ rt \xi \)
shows "R \rt \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 \( \sigma \sigma \)' dip
assumes "\( \rt (\sigma i) \subseteq dip \rt (\sigma nhip) \)"
and qinc: "quality_increases (\sigma nhip) (\sigma \)' nhip"
and "\( \text{dip} \in kD(\rt (\sigma nhip)) \)"
shows "\( \rt (\sigma i) \subseteq dip \rt (\sigma \)' nhip) \)
proof -
from qinc have "\( \rt (\sigma nhip) \subseteq dip \rt (\sigma \)' nhip) \)" using \( \langle \text{dip} \in kD(\rt (\sigma nhip)) \rangle \)
by auto
with \( \langle \rt (\sigma i) \subseteq dip \rt (\sigma nhip) \rangle \) show ?thesis ..
qed

lemma kD_quality_increases [elim]:
assumes "\( i \in kD(\xi) \)"
and "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 "quality_increases \( \xi \xi \)"
shows "\( i \in kD(\rt \xi \) ∧ nsqn (\rt \xi \) i \leq nsqn (\rt \xi \) i "
proof -
from assms have "\( i \in kD(\rt \xi \)" ..
morover with assms have "\( \rt \xi \subseteq i \rt \xi \)" by auto
ultimately have "nsqn (\rt \xi \) i \leq nsqn (\rt \xi \) i "
using \( \langle i \in kD(\rt \xi \) \rangle \) by (erule(2) rt_fresher_imp_nsqn_le)
with \( \langle i \in kD(\rt \xi \) \rangle \) show ?thesis ..
qed

lemma nsqn_quality_increases [elim]:
assumes "\( i \in kD(\rt \xi \)"
and "quality_increases \( \xi \xi \)"
shows "nsqn (\rt \xi \) i \leq 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 "\( s \leq nsqn (\rt \xi \) i "
and "quality_increases \( \xi \xi \)"
shows "\( i \in kD(\rt \xi \) ∧ s \leq nsqn (\rt \xi \) i "
proof
from \( \langle i \in kD(\rt \xi \) \rangle \) and \( \langle quality_increases \( \xi \xi \) \rangle \) show "\( i \in kD(\rt \xi \)" ..
next
from \( \langle i \in kD(\rt \xi \) \rangle \) and \( \langle quality_increases \( \xi \xi \) \rangle \) have "nsqn (\rt \xi \) i \leq nsqn (\rt \xi \) i " ..
with \( s \leq nsqn (\rt \xi \) i \) show "s \leq nsqn (\rt \xi \) 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

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" ...
   with ⟨sqn (rt (σ sip)) ip > 1⟩
   and ⟨sn = nsqn (rt (σ sip)) ip⟩
   have "nsqn (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)) ip⟩ show ?thesis by simp
   next
      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" by simp
      with ⟨nsqn (rt (σ' sip)) ip > nsqn (rt (σ sip)) ip⟩
      have "nsqn (rt (σ' sip)) ip > sn" by simp
      thus ?thesis ...
   qed
   qed
   qed
   thus ?thesis by (metis (mono_tags) le_cases not_le)
   qed
   with ⟨ip ∈ kD (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
      ∧ (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 \textit{thesis} using \textit{assms}(2-3) by (rule \textit{quality\_increases\_rreq\_rrep\_props})

\textbf{qed}

\textbf{lemma \textit{rteq\_quality\_increases}}:
assumes \"\forall j. \ j \neq i \implies \textit{quality\_increases} (\sigma j) (\sigma' j)\"
and \"rt (\sigma' i) = rt (\sigma i)\"
shows \"\forall j. \textit{quality\_increases} (\sigma j) (\sigma' j)\"
using \textit{assms} by clarisimp (metis \textit{order\_refl} \textit{quality\_increasesI} \textit{rt\_fresher\_refl})

\textbf{definition \textit{msg\_fresh} :: "}(ip ⇒ state) ⇒ msg ⇒ bool"
where \"\textit{msg\_fresh} \sigma m ≡ \\begin{align*}
\textit{case} m \ \textit{of} \ Rreq \ hopsc _ _ oipc osnc sipc _ ⇒ \ & oipc ∈ kD(rt (\sigma sipc)) \land nsqn (rt (\sigma sipc)) osnc ≥ oipc
\land \ (nsqn (rt (\sigma sipc)) oipc = osnc \\
& \implies \ (hopsc ≥ the (dhops (rt (\sigma sipc)) oipc) \lor \ (flag (rt (\sigma sipc)) oipc) = inv)) \\
\mid Rrep \ dipsc dip dipc dsnc _ sipc ⇒ \ & dsnc ≥ 1 \land (sipc ≠ dipc ⇒ dipc ∈ kD(rt (\sigma sipc)) \land nsqn (rt (\sigma sipc)) dipc ≥ dsnc
\land \ (nsqn (rt (\sigma sipc)) dipc = dsnc \\
& \implies \ (hopsc ≥ the (dhops (rt (\sigma sipc)) dipc) \lor \ (flag (rt (\sigma sipc)) dipc) = inv))) \\
\mid Rerr \ destsc sipc ⇒ \ & (\forall ripc ∈ dom(destsc). (ripc ∈ kD(rt (\sigma sipc))) \land nsqn (rt (\sigma sipc)) ripc - 1 ≤ nsqn (rt (\sigma sipc)) ripc)
\mid _ ⇒ True
\end{align*}"
\textbf{lemma \textit{msg\_fresh} [simp]}:
\"\\begin{align*}
\land \textit{hops dip dsn dsk oip osn sip handled.} \\
\textit{msg\_fresh} \sigma (Rreq \ hops \ dip \ dsn \ dsk \ oip \ osn \ sip \ handled) & = \ (osn ≥ 1 \land (sip ≠ oip ⇒ oipc ∈ kD(rt (\sigma sipc))) \land nsqn (rt (\sigma sipc)) oipc ≥ osn
\land \ (nsqn (rt (\sigma sipc)) oipc = osnc \\
& \implies \ (hopsc ≥ the (dhops (rt (\sigma sipc)) oipc) \lor \ (flag (rt (\sigma sipc)) oipc) = inv)))) \\
\land \textit{hops dip dsn oip sip.} \textit{msg\_fresh} \sigma (Rreq \ hops \ dip \ dsn \ oip \ sip) & = \ (dsn ≥ 1 \land (sip ≠ dip ⇒ dipsc ∈ kD(rt (\sigma sipc))) \land nsqn (rt (\sigma sipc)) dipc ≥ dsnc
\land \ (nsqn (rt (\sigma sipc)) dipc = dsnc \\
& \implies \ (hopsc ≥ the (dhops (rt (\sigma sipc)) dipc) \lor \ (flag (rt (\sigma sipc)) dipc) = inv)))) \\
\land \textit{dests sip.} \textit{msg\_fresh} \sigma (Rreq \ dests \ sip) & = \ (\forall ripc ∈ dom(dests). (ripc ∈ kD(rt (\sigma sipc)) \\
& \land \ (dests ripc - 1 ≤ nsqn (rt (\sigma sipc)) ripc)) \\
\land \textit{d dip.} \textit{msg\_fresh} \sigma (Newpkt d dip) & = \ True \\
\land \textit{d dip.} \textit{msg\_fresh} \sigma (Pkt d dip sip) & = \ True
\end{align*}"
unfolding \textit{msg\_fresh\_def} by \textit{simp\_all}

\textbf{lemma \textit{msg\_fresh\_inc\_sn} [simp, elim]}:
\"\textit{msg\_fresh} \sigma m \implies \textit{rreq\_rrep\_sn} \ m\"
by (cases m) \textit{simp\_all}

\textbf{lemma \textit{recv\_msg\_fresh\_inc\_sn} [simp, elim]}:
\"\textit{orecvmsg} (\textit{msg\_fresh}) \sigma m \implies \textit{recvmsg} \textit{rreq\_rrep\_sn} \ m\"
by (cases m) \textit{simp\_all}

\textbf{lemma \textit{rreq\_nsqn\_is\_fresh} [simp]}:
fixes \sigma \ msg \ hops \ dip \ dsn \ dsk \ oip \ osn \ sip \ handled
assumes \"\textit{rreq\_rrep\_fresh} (rt (\sigma sipc)) (\textit{Rreq} \ hops \ dip \ dsn \ dsk \ oip \ osn \ sip \ handled)\"
and \"\textit{rreq\_rrep\_sn} (\textit{Rreq} \ hops \ dip \ dsn \ dsk \ oip \ osn \ sip \ handled)\"
shows \"\textit{msg\_fresh} \sigma (\textit{Rreq} \ hops \ dip \ dsn \ dsk \ oip \ osn \ sip \ handled)\"
(is \"\textit{msg\_fresh} \sigma ?msg\")
proof -
let \textit{rt} = \"rt (\sigma sipc)\"
from \textit{assms}(2) have \"1 ≤ osn\" by \textit{simp}
thus \textit{?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) 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
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 \(?rt = \text{"rt (}\sigma \text{ sip})\)
from \(\text{assms have }\): "(\(\forall \text{ rip}\in \text{dom(\text{dests})}. (\text{rip}\in \text{D(}\text{rt (}\sigma \text{ sip}))
\text{ ∧ the (\text{dests rip}) = sqn (\text{rt (}\sigma \text{ sip)} \text{ rip)})\)"
by \text{clarsimp}
have "(\(\forall \text{ rip}\in \text{dom(\text{dests})}. (\text{rip}\in \text{D(}\text{rt (}\sigma \text{ sip)})
\text{ ∧ the (\text{dests rip}) - 1} \leq \text{sqn (rt (}\sigma \text{ sip)) \text{ rip)})\)"

proof
fix \text{rip}
assume "\text{rip} \in \text{dom \text{dests}}"
with * have "\text{rip}\in \text{D(}\text{rt (}\sigma \text{ sip)})" and "the (\text{dests rip}) = sqn (\text{rt (}\sigma \text{ sip)) \text{ rip})"
by auto
from this(2) have "\text{rip}\in \text{kD(}\text{rt (}\sigma \text{ sip)})" and "the (\text{dests rip}) - 1 \leq \text{sqn (rt (}\sigma \text{ sip)) \text{ rip})"
by \text{clarsimp}
qed
thus "msg\text{\_fresh }\sigma \text{ ?msg}"
by \text{simp}
qed

lemma \text{quality\_increases\_msg\_fresh [elim]:}
assumes \(\text{qinc: }\forall j. \text{quality\_increases (}\sigma j) (\sigma' j)\)"
and "\text{msg\_fresh }\sigma \text{ m}"
shows "\text{msg\_fresh }\sigma' \text{ m}"
using \(\text{assms(2)}\)

proof (\text{cases m})
fix \text{hops dip dsn dsk oip osn sip handled}
assume [simp]: "\text{m} = \text{Rreq hops dip dsn dsk oip osn sip handled}"
and "\text{msg\_fresh }\sigma \text{ m}"
then have "\text{osn} \geq 1" and "\text{sip} = \text{oip} \lor (\text{oip}\in \text{D(}\text{rt (}\sigma \text{ sip)}) \land \text{osn} \leq \text{sqn (rt (}\sigma \text{ sip)) oip}
\land (\text{sqn (rt (}\sigma \text{ sip)) oip = osn}
\rightarrow (\text{the (dhops (rt (}\sigma \text{ sip)) oip) \leq hops}
\lor \text{the (flag (rt (}\sigma \text{ sip)) oip) = inv}))"
by \text{auto}
from this(2) show \text{?thesis}
proof
assume "\text{sip} = \text{oip}" with (\text{osn} \geq 1) show \text{?thesis} by \text{simp}
next
assume "\text{oip}\in \text{D(}\text{rt (}\sigma \text{ sip)}) \land \text{osn} \leq \text{sqn (rt (}\sigma \text{ sip)) oip}
\land (\text{sqn (rt (}\sigma \text{ sip)) oip = osn}
\rightarrow (\text{the (dhops (rt (}\sigma \text{ sip)) oip) \leq hops}
\lor \text{the (flag (rt (}\sigma \text{ sip)) oip) = inv}))"
moreover from \text{qinc have }"\text{quality\_increases (}\sigma \text{ sip) (}\sigma' \text{ sip)}\
ultimately have "\text{oip}\in \text{D(}\text{rt (}\sigma' \text{ sip)}) \land \text{osn} \leq \text{sqn (rt (}\sigma' \text{ sip)) oip}
\land (\text{sqn (rt (}\sigma' \text{ sip)) oip = osn}
\rightarrow (\text{the (dhops (rt (}\sigma' \text{ sip)) oip) \leq hops}
\lor \text{the (flag (rt (}\sigma' \text{ sip)) oip) = inv}))"
using (\text{osn} \geq 1) by (\text{rule quality\_increases\_rreq\_rrep\_props [rotated 2]})
with (\text{osn} \geq 1) show "\text{msg\_fresh }\sigma' \text{ m}"
by (\text{clarsimp})
qed
next
fix \text{hops dip dsn oip sip}
assume [simp]: "\text{m} = \text{Rrep hops dip dsn oip osn sip}
and "\text{msg\_fresh }\sigma \text{ m}"
then have "\text{dsn} \geq 1" and "\text{sip} = \text{dip} \lor (\text{dip}\in \text{D(}\text{rt (}\sigma \text{ sip)}) \land \text{dsn} \leq \text{sqn (rt (}\sigma \text{ sip)} \land \text{dsn}
\rightarrow (\text{the (dhops (rt (}\sigma \text{ sip)) dip) \leq hops}
\lor \text{the (flag (rt (}\sigma \text{ sip)) dip) = inv}))"

qed
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))
    ∧ (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))
    ∧ (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

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 σ_AODV' :: "((ip ⇒ state) × ((state, msg, pseqp, pseqp label) seqp)) set"
where "σ_AODV' ≡ {{λi. aodv_init i, Γ_AODV PAdvv}}"
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 oaadv_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' → labels Γ_AODV p = {PAodv::0})"
unfolding σ_AODV'_def by simp
lemma oaodv_init_kD_empty [simp]:
"(σ, p) ∈ σAODV' =⇒ kD (rt (σ i)) = {}"
unfolding σAODV'_def kD_def by simp

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

lemma oaodv_trans: "trans (opaodv i) = oseqp_sos ΓAODV 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 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 =⇒ (σ i := σ i (msg := m)) m"
shows "opaodv i = (otherwith Q {i} (orecvmsg P), other Q {i} →)
≡ (λ l. P (msg (σ i)) m) ΓAODV (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 "σ 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 \( P \sigma' (msg (\sigma' i)) \) by simp

next

fix \( \sigma \sigma' \) msg

assume "otherwith Q \{i\} (orecvmsg P) \sigma \sigma' (receive msg)"

and "\( \sigma' i = \sigma i[[msg := msg]] \)"

from this(1) have "P \( \sigma \) \( \sigma' \) (msg (\sigma' i))"

and "\( \forall j. j \neq i \rightarrow Q (\sigma j) (\sigma' j) \)" by auto

from this(1) have "P (\sigma (i := \sigma i[[msg := msg]]) \) (msg (\sigma' i))"

thus "P \( \sigma' \) msg"

proof (rule other)

from \( \langle \sigma' i = \sigma i[[msg := msg]] \rangle \) and \( \langle \forall j. j \neq i \rightarrow Q (\sigma j) (\sigma' j) \rangle \)

show "other Q \{i\} \( \sigma (i := \sigma i[[msg := msg]]) \) \sigma'"

by (rule otherI, auto)

qed

(Equivalent to) Proposition 7.27

lemma local_quality_increases:

\[ \text{opaodv i} \models_A (\text{recvmsg rreq\_rrep\_sn} \rightarrow \text{onll} \ \Gamma_{AODV} (\lambda((\xi, _), _, (\xi', _))). \text{quality\_increases} \ (\xi') \] 

proof (rule step_invariantI)

fix \( s \ a \ s' \)

assume sr: "s \in \text{reachable} (\text{opaodv i}) (\text{recvmsg rreq\_rrep\_sn})"

and tr: "(s, a, s') \in \text{trans} (\text{opaodv i})"

and rm: "\text{recvmsg rreq\_rrep\_sn} a"

from sr have srTT: "s \in \text{reachable} (\text{opaodv i}) TT" ..

from route_tables_fresher sr tr rm

have "\text{onll} \ \Gamma_{AODV} (\lambda((\xi, _), _, (\xi', _)). \ \forall \text{dip} \in \text{kD} (\text{rt}\ \xi) \ \text{rt}\ \xi \leq \text{dip} \text{rt}\ \xi') (s, a, s')"

by (rule step_invariantD)

moreover from known_destinations_increase srTT tr TT_True

have "\text{onll} \ \Gamma_{AODV} (\lambda((\xi, _), _, (\xi', _)). \ \forall \text{ip} \in \text{sqn} (\text{rt}\ \xi) \ \text{ip} \leq \text{sqn} (\text{rt}\ \xi') \text{ip}) (s, a, s')"

by (rule step_invariantD)

moreover from sqns_increase srTT tr TT_True

have "\text{onll} \ \Gamma_{AODV} (\lambda((\xi, _), _, (\xi', _)). \ \forall \text{ip} \in \text{sln} (\text{rt}\ \xi) \ \text{ip} \leq \text{sln} (\text{rt}\ \xi') \text{ip}) (s, a, s')"

by (rule step_invariantD)

ultimately show "\text{onll} \ \Gamma_{AODV} (\lambda((\xi, _), _, (\xi', _)). \ \text{quality\_increases} \ (\xi') (s, a, s')"

unfolding \text{onll} \text{def} by auto

qed

lemmas olocal_quality_increases =

open_seq_step_invariant \[ \text{OF local_quality\_increases} \ \text{initiali\_aodv} \ \text{oao\_trans} \ \text{aodv\_trans, simplified seqll\_onll\_swap} \]

lemma quality_increases:

\[ \text{opaodv i} \models_A (\text{otherwith quality\_increases} \ \{i\} (\text{orecvmsg} (\lambda_. \text{rreq\_rrep\_sn})), \text{other quality\_increases} (i) \rightarrow) \] 

\[ \text{onll} \ \Gamma_{AODV} (\lambda((\sigma, _), _, (\sigma', _)). \ \forall j. \ \text{quality\_increases} (\sigma j) (\sigma' j)) \] 

(is "\_ \models_A (?S, _ \rightarrow) \_")

proof (rule onll\_ostep\_invariantI, simp)

fix \( \sigma \ p \ p' \ 1' \)

assume or: "(\sigma, p) \in \text{reachable} (\text{opaodv i}) ?S (\text{other quality\_increases} (i))"

and 11: "1 \in \text{labels} \ \Gamma_{AODV} p"

and "?S \ \sigma \ \sigma' \ a"

and tr: "((\sigma, p), a, (\sigma', p')) \in \text{oseqp\_sos} \ \Gamma_{AODV} i"

and 11': "1' \in \text{labels} \ \Gamma_{AODV} p'"

from this(1-3) have "\text{orecvmsg} (\lambda_. \text{rreq\_rrep\_sn}) \sigma a"

by (auto dest!: reachable\_weakenE [where \text{QS}=\"act (\text{recvmsg rreq\_rrep\_sn})\"

and \text{QU}=\"other quality\_increases (i)\"])

otherwith\_actionD

with or have crw: "(\sigma, p) \in \text{reachable} (\text{opaodv i}) (\text{act (recvmsg rreq\_rrep\_sn)}) (\text{other quality\_increases (i)})"
by - (erule reachable weakenE, auto)
with tr l l' and Orecvmsg (λ _. rreq rrep sn) σ a) have "quality increases (σ i) (σ' i)"
by - (erule 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(σ, _, a, _). anycast (msg fresh σ) a)"
proof (rule ostep invariantI, simp del: act simp)
fix σ p a σ' p'
assume or: "((σ, p), a, (σ', p')) ∈ seqp 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')) ∈ seqp sos Γ AODV i " have tr: "((σ, p), a, (σ', p')) ∈ trans (opaodv i)" by simp
have "anycast (rreq rrep fresh (rt σ)) a"
proof -
from or tr recv have "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)
thus "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)) a"
proof -
from or tr recv have "onll Γ AODV (seqll i (λ(ξ, a, _). anycast (rerr invalid (rt ξ)) a))" by (rule ostep invariant weakenE [OF open seq step invariant [OF rerr invalid any step invariant initiali aodv, simplified seqll onll swap]] auto)
thus "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(ξ, a, _). anycast rreq rrep sn a)) ((σ, p), a, (σ', p'))" by (rule ostep invariant weakenE [OF open seq step invariant [OF rreq rrep sn any step invariant initiali aodv, oadv trans aodv trans, 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 -
from or tr recv have "onll Γ AODV (seqll i (λ(ξ, a, _). anycast rreq rrep sn a) a)) ((σ, p), a, (σ', p'))" by (rule ostep invariant weakenE [OF open seq step invariant [OF rreq rrep sn any step invariant initiali aodv, oadv trans aodv trans, 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 -
from or tr recv have "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 rreq rrep sn any step invariant initiali aodv, oadv trans aodv trans, simplified seqll onll swap]] auto)
thus ?thesis using ⟨l ∈ labels Γ AODV p ⟩ and ⟨l' ∈ labels Γ AODV p' ⟩ by auto
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 ⟨l∈labels Γ_AODV p⟩ and ⟨l′∈labels Γ_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_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(l. l∈{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

qed

lemma oquality_increases_nsqn_fresh:
"opaodv i |= A (otherwith quality_increases {i} (orecvmsg msg_fresh), other quality_increases {i} →)
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} →)
onll 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} →)
onll Γ_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 oseq_wf oadvaodv_trans]
onll_oinvariant_sterms [OF aodv_wf oreceived_rreq_rrep_nsqn_fresh_inv]
onll_oinvariant_sterms [OF aodv_wf oosn_rreq]
\[ \text{simp add: seqlsimp} \]
\[ \text{simp del: One_nat_def, rule implI} \]
\[ \text{fix } \sigma, \sigma' \text{ p l} \]
\[ \text{assume } "(\sigma, p) \text{ o reachable (opaodv i) ?S ?U}" \]
\[ \text{and } "1 \in \text{labels } \Gamma_{AODV} p" \]
\[ \text{and pre:} \]
\[ (1 = PAdv-:4 \lor 1 = PAdv-:5 \lor 1 = PRreq-:0 \lor 1 = PRreq-:2) \land \text{sip } (\sigma i) \neq \text{oop } (\sigma i) \]
\[ \rightarrow \text{oop } (\sigma i) \in kD (rt (\sigma (\text{sip } (\sigma i)))) \]
\[ \land \text{osn } (\sigma i) \leq \text{nsqn } (rt (\sigma (\text{sip } (\sigma i)))) (\text{oop } (\sigma i)) \]
\[ \land (\text{nsqn } (rt (\sigma (\text{sip } (\sigma i)))) (\text{oop } (\sigma i)) = \text{osn } (\sigma i) \]
\[ \rightarrow \text{the } (\text{dhops } (rt (\sigma (\text{sip } (\sigma i)))) (\text{oop } (\sigma i)) \leq \text{hops } (\sigma i) \]
\[ \land \text{the } (\text{flag } (rt (\sigma (\text{sip } (\sigma i)))) (\text{oop } (\sigma i))) = \text{inv}" \]
\[ \text{and } "\text{other quality}\_\text{increases } \{i\} } \sigma' \text" \]
\[ \text{and hyp: } "(1 = PAdv-:4 \lor 1 = PAdv-:5 \lor 1 = PRreq-:0 \lor 1 = PRreq-:2) \land \text{sip } (\sigma' i) \neq \text{oop } (\sigma' i)" \]
\[ \text{(is } "?\text{labels } \land \text{sip } (\sigma' i) \neq \text{oop } (\sigma' i)" \]
\[ \text{from this(4) have } "\sigma' i = \sigma i" \ldots \]
\[ \text{with hyp have hyp': } "?\text{labels } \land \text{sip } (\sigma i) \neq \text{oop } (\sigma i)" \text{ by simp} \]
\[ \text{show } "\text{oop } (\sigma i) = kD (rt (\sigma (\text{sip } (\sigma i)))) \]
\[ \land \text{osn } (\sigma i) \leq \text{nsqn } (rt (\sigma (\text{sip } (\sigma i)))) (\text{oop } (\sigma i)) \]
\[ \land (\text{nsqn } (rt (\sigma (\text{sip } (\sigma i)))) (\text{oop } (\sigma i)) = \text{osn } (\sigma i) \]
\[ \rightarrow \text{the } (\text{dhops } (rt (\sigma (\text{sip } (\sigma i)))) (\text{oop } (\sigma i)) \leq \text{hops } (\sigma i) \]
\[ \land \text{the } (\text{flag } (rt (\sigma (\text{sip } (\sigma i)))) (\text{oop } (\sigma i))) = \text{inv}" \]
\[ \text{proof (cases } "\text{sip } (\sigma i) = i" \) \]
\[ \text{assume } "\text{sip } (\sigma i) \neq i" \]
\[ \text{from } "\text{other quality}\_\text{increases } \{i\} } \sigma' \text" \]
\[ \text{have } "\text{quality}\_\text{increases } (\sigma (\text{sip } (\sigma i))) (\sigma' (\text{sip } (\sigma i)))" \]
\[ \text{by (rule otherE) (clarsimp simp: seqlsimp) \text{ simp add: seqlsimp } \sigma' i = \sigma i) \]
\[ \text{moreover from } "(\sigma, p) \text{ o reachable (opaodv i) ?S ?U}" \text{ and hyp} \]
\[ \text{have } "1 \leq \text{osn } (\sigma i)" \]
\[ \text{by (auto dest!: onl_oivariable_weakenD [OF oosn_rreq]} \]
\[ \text{simp add: seqlsimp } \sigma' i = \sigma i) \]
\[ \text{moreover from } "\text{sip } (\sigma i) \neq i" \text{ hyp' and pre} \text{ by auto} \]
\[ \text{ultimately show } ?\text{thesis} \]
\[ \text{by (rule quality\_increases\_rreq\_rrep\_props) \text{ simp add: seqlsimp } \sigma' i = \sigma i) \]
\[ \text{next} \]
\[ \text{assume } "\text{sip } (\sigma i) = i" \text{ thus } ?\text{thesis} \]
\[ \text{using } \sigma' i = \sigma i \text{ hyp and pre by auto} \]
\[ \text{qed} \]
\[ \text{(auto elim!: quality\_increases\_rreq\_rrep\_props')} \]

**Lemma odsn\_rrep:**
\[ \text{oapdv i } \models (\text{otherwith quality}\_\text{increases } \{i\} (\text{orecovmsg msg\_fresh}), \]
\[ \text{other quality}\_\text{increases } \{i\} \rightarrow \]
\[ \text{onl } \Gamma_{AODV} (\text{seql i } (\lambda(x, i). \text{ l } \in \{PAdv-:6, PAdv-:7\} \cup \{PRrep-:n/n. \text{ True} \rightarrow 1 \leq dsn x)))" \]
\[ \text{by (rule oivariable\_weakenE [OF open\_seq\_invariant [OF dsn\_rrep initial\_aodv]]} \]
\[ \text{(auto simp: seql\_onl\_swap)} \]

**Lemma rrep\_sip:**
\[ \text{oapdv i } \models (\text{otherwith quality}\_\text{increases } \{i\} (\text{orecovmsg msg\_fresh}), \]
\[ \text{other quality}\_\text{increases } \{i\} \rightarrow \]
\[ \text{onl } \Gamma_{AODV} (\lambda(x, i). \]
\[ (1 \in \{PAdv-:6, PAdv-:7, PRrep-:0, PRrep-:1\} \land \text{sip } (\sigma i) \neq \text{dip } (\sigma i)) \]
\[ \rightarrow \text{dip } (\sigma i) \in kD(rt (\sigma (\text{sip } (\sigma i)))) \]
\[ \land \text{nsqn } (rt (\sigma (\text{sip } (\sigma i)))) (\text{dip } (\sigma i)) \geq \text{dsn } (\sigma i) \]
\[ \land (\text{nsqn } (rt (\sigma (\text{sip } (\sigma i)))) (\text{dip } (\sigma i)) = \text{dsn } (\sigma i) \]
\[ \rightarrow (\text{hops } (\sigma i) \geq \text{the } (\text{dhops } (rt (\sigma (\text{sip } (\sigma i)))) (\text{dip } (\sigma i))) \]
\[ \lor \text{the } (\text{flag } (rt (\sigma (\text{sip } (\sigma i)))) (\text{dip } (\sigma i))) = \text{inv}))" \]

(is "_ } (\text{dip } ?S, ?U \rightarrow _) "}
fix σ σ' p l
assume "((σ, p) ∈ reachable (opaodv i) ?S ?U"
and "l ∈ labels Γ_AODV p"
and pre:

"(1 = PAodv−:6 ∨ 1 = PAodv−:7 ∨ 1 = PRrep−:0 ∨ 1 = PRrep−:1) ∧ \sigma (i) ≠ dip (σ i)
\rightarrow 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)
\rightarrow the (dhops (rt (σ (sip (σ i)))) (dip (σ i))) ≤ hops (σ i)
\rightarrow \{\text{flag (rt (σ (sip (σ i)))) (dip (σ i))} = \text{inv}""
and "other quality_increases {i} σ σ'"
and hyp: "(l = PAodv−:6 \lor l = PAodv−:7 \lor l = PRrep−:0 \lor l = PRrep−:1) ∧ \sigma (i) ≠ dip (σ' i)"
(is "labels \land sip (σ' i) \neq dip (σ' i)"
from (this(4)) have "σ' i = σ i"
with hyp have hyp': "labels \land sip (σ i) \neq 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)
\rightarrow the (dhops (rt (σ' (sip (σ' i)))) (dip (σ' i))) ≤ hops (σ' i)
\rightarrow \{\text{flag (rt (σ' (sip (σ' i)))) (dip (σ' i))} = \text{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) ∈ reachable (opaodv i) ?S ?U l ∈ labels Γ_AODV p and hyp have "l ≤ dsn (σ' i)"
by (auto dest!: onl_oivarint_weakenD [OF odsn_rrep]
simp add: seqlsimp '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)
\rightarrow the (dhops (rt (σ (sip (σ i)))) (dip (σ i))) ≤ hops (σ i)
\rightarrow \{\text{flag (rt (σ (sip (σ i)))) (dip (σ i))} = \text{inv}""
by (auto simp: '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 simp add: seqlsimp elim!: quality_increases_rreq_rrep_props')

lemma rerr_sip:
"opaodv i |= (other with quality_increases {i} (orecvmsg msg_fresh),
other quality_increases {i} →)
\land Γ_AODV (\lambda (σ, i).
\land 1 ∈ \{PAodv−:8, PAodv−:9, PRerr−:0, PRerr−:1\}
\rightarrow (∀vrip∈dom(dests (σ i)). vrip∈kD(rt (σ (sip (σ i)))) ∧ the (dests (σ i)) vrip) - 1 ≤ nsqn (rt (σ (sip (σ i)))) vrip))""
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
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 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 (λξ. 1 = PErr →
  (∀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 odests_vD_inc_sqn =
open_seq_invariant [OF odests_vD_inc_sqn initialise_aodv oaodv_trans aodv_trans,
simplified seql_onl_swap, THEN oinvariant_anyact]
lemmas oprerr_guard =
open_seq_invariant [OF prerr_guard initialise_aodv oaodv_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 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"
  qed
  qed
by auto

from qinc have qinc_nhlp: "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_nhlp 
  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: "∀σ∈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: "∀σ∈kd(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: "∀σ∈kd(dests (σ i)). nhop ip = sip (σ i)"
    and qinc: "∀σ∈kd(dests (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∈kd (dests (σ i))")
    assume "dip∈kd (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∈kd (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∈kd (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_qualityIncreases_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

\{ fix \sigma \sigma' :: "ip \Rightarrow state"
 assume a1: "\forall dip : kD(rt (\sigma i)). the (nhop (rt (\sigma i)) dip) \neq dip 
 \rightarrow dip : 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 : kD(rt (\sigma i)). the (nhop (rt (\sigma i)) dip) \neq dip 
 \rightarrow dip : 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 \sigma oip osn hops
 assume pre: "\forall dip : kD (rt (\sigma i)). the (nhop (rt (\sigma i)) dip) \neq dip 
 \rightarrow dip : 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 qinc: "\forall j. quality_increases (\sigma j) (\sigma' j)"
 and *: "sip \neq oip \rightarrow oip : kD(rt (\sigma sip)) 
 \land osn \leq nsqn (rt (\sigma sip)) oip 
 \land (nsqn (rt (\sigma sip)) oip = osn 
 \rightarrow the (dhops (rt (\sigma sip)) oip) \leq hops 
 \lor the (flag (rt (\sigma sip)) oip) = inv)"
 from pre and qinc
 have pre': "\forall dip : kD (rt (\sigma i)). the (nhop (rt (\sigma i)) dip) \neq dip 
 \rightarrow dip : 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 : kD(rt (\sigma' (the (nhop (update (rt (\sigma i)) oip 
 (osn, 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 
 (osn, kno, val, Suc hops, sip)) oip)))) oip)" 
 (is "?nhop_not_oip \rightarrow \?oip_in_kD \land \?nsqn_le_nsqn")
 proof (rule, split update_rt_split_asm)
 assume "rt (\sigma i) = 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 \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 #: "σ' ≠ oip → oip ∈ kD (rt (σ oip))
     ∧ 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 →) oln 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 quality_increases_nsqn_fresh aodv_wf
                                                 oadv_trans]
       onl_oinvariant_sterms [OF aodv_wf onl_ests_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]

  477
thus \(?\)thesis unfolding Let_def by auto

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 "(\(\sigma, p\) \(\in\) ororeachable (opaodv i))
            (otherwith (op=) \{i\} (orecvmmsg (\(\lambda\sigma m.\ msg\_fresh\ \sigma\ m\ \\
                                          \land \ msg\_zhops\ m)))))
            (other quality_increases \{i\})"
  and "dip\(\in\)kD(rt (\(\sigma\ i\)))"
  shows "\(\pi_3\)\(\left(\text{rt (\(\sigma\ i\)) dip}\right) = unk \lor 1 \leq \pi_2\(\left(\text{rt (\(\sigma\ i\)) dip}\right)\)"
(is "?P dip")
proof -
  have "\(\exists l. l\in\text{labels }\Gamma_{AODV} p\)" by (metis aodv_ex_label)
  with assms(1) have "\(\forall dip\in\text{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 \(\langle dip\in\text{kD (rt (\(\sigma\ i\)))))\rangle\) show ?thesis by simp
qed

lemma ororeachable_fresh_ozero_seq_unk_hops_one:
  fixes dip
  assumes "(\(\sigma, p\) \(\in\) ororeachable (opaodv i))
            (otherwith (op=) \{i\} (orecvmmsg (\(\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 "\(\exists l. l\in\text{labels }\Gamma_{AODV} p\)" by (metis aodv_ex_label)
  with assms(1) have "\(\forall dip\in\text{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 \(\langle dip\in\text{kD (rt (\(\sigma\ i\)))))\rangle\) show ?thesis by simp
qed

lemma seq_nhop_quality_increases':
  shows "\(\exists dip\in\text{vD (rt (\(\sigma\ i\))) \cap vD (rt (\(\sigma\ nhip\)) \land nhip \neq dip\)\)"

proof -

have weaken:

"(\exists I \ Q \ R \ P. \ p \models (otherwith quality_increases I \ orevmsg Q, other quality_increases I \→ P) \n ⇒ \ models (otherwith (op='=) I \ orevmsg (\lambda \sigma. Q \ σ \ R \ σ) m), other quality_increases I \→ P)"

by auto

\{

fix a and \( a' :: \"ip \Rightarrow state\"

assume a1: "\forall a' dip. dip \in D (rt \ (σ \ i))
∧ dip \in D (rt \ (σ (\σ (rt \ (σ \ i)) dip)))
∧ (the (\σ (rt \ (σ \ i)) dip)) \neq dip
→ rt \ (σ \ i) \ ⊑ dip \ rt \ (σ (the (\σ (rt \ (σ \ i)) dip))"

and ov: "∀ i a σ a' a"

have "\forall dip. dip \in D (rt \ (σ \ i))
∧ dip \in D (rt \ (σ' (the (\σ (rt \ (σ \ i)) dip)))
∧ (the (\σ (rt \ (σ \ i)) dip)) \neq dip
→ rt \ (σ \ i) \ ⊑ dip \ rt \ (σ' (the (\σ (rt \ (σ \ i)) dip)))"

prooof clarify

fix dip

assume a2: "dip \in D (rt \ (σ \ i))"

and a3: "dip \in D (rt (σ' (the (\σ (rt \ (σ \ i)) dip))))"

and a4: "(the (\σ (rt \ (σ \ i)) dip)) \neq dip"

from ov have "∀ j. j \neq i \→ j \neq j'" by auto

show "rt (σ \ i) \ ▢ dip rt (σ' (the (\σ (rt \ (σ \ i)) dip)))"

proof (cases "\the (\σ (rt \ (σ \ i)) dip) = i"
assume "\the (\σ (rt \ (σ \ i)) dip)) \neq i"

with \( dip \in D (rt \ (σ \ i)) \) have "dip \in D (rt (σ (the (\σ (rt \ (σ \ i)) dip))))" by simp

with a1 a2 a4 have "rt (σ \ i) \ ▢ dip rt (σ (the (\σ (rt \ (σ \ i)) dip)))" by simp

with (the (\σ (rt \ (σ \ i)) dip) = i) have "rt (σ \ i) \ ▢ dip rt (σ (the (\σ (rt \ (σ \ i)) dip)))" by simp

hence False by simp

thus \$\$thesis ..

next

assume "\the (\σ (rt \ (σ \ i)) dip)) \neq i"

with ∀ j. j \neq i \→ j \neq j' have *: "σ (the (\σ (rt \ (σ \ i)) dip))) = σ' (the (\σ (rt \ (σ \ i)) dip)))" by simp

with dip \in D (rt (σ' (the (\σ (rt \ (σ \ i)) dip)))) have "dip \in D (rt (σ (the (\σ (rt \ (σ \ i)) dip))))" by simp

with (the (\σ (rt \ (σ \ i)) dip) = i) have "rt (σ \ i) \ ▢ dip rt (σ (the (\σ (rt \ (σ \ i)) dip)))" by simp

with * show \$\$thesis by simp

qed

qed

\}

\} note basic = this

\{ fix σ a' dip i

assume a1: "∀ dip. dip \in D (rt \ (σ \ i))
∧ dip \in D (rt (σ (the (\σ (rt \ (σ \ i)) dip))))
∧ (the (\σ (rt \ (σ \ i)) dip)) \neq dip
→ rt (σ \ i) ⊑ dip rt (σ (the (\σ (rt \ (σ \ i)) dip)))"

and ov: "∀ i a σ a' a"

have "∀ dip. dip \in D (\σ (\σ (rt \ (σ \ i)) dip)))
∧ dip \in D (rt (σ' (the (\σ (rt \ (σ \ i)) dip))))
∧ (the (\σ (rt \ (σ \ i)) dip)) \neq dip
→ rt (σ \ i) \ ▢ dip rt (σ' (the (\σ (rt \ (σ \ i)) dip)))"

prooof clarify

fix dip

assume a2: "dip \in D (\σ (\σ (rt \ (σ \ i)) dip)))
∧ dip \in D (rt (σ' (the (\σ (rt \ (σ \ i)) dip))))
∧ (the (\σ (rt \ (σ \ i)) dip)) \neq dip"

and a3: "dip \in D (\σ (\σ (rt \ (σ \ i)) dip)))
∧ dip \in D (rt (σ' (the (\σ (rt \ (σ \ i)) dip))))
∧ (the (\σ (rt \ (σ \ i)) dip)) \neq dip"

and a4: "dip \in D (\σ (\σ (rt \ (σ \ i)) dip)))
∧ dip \in D (rt (σ' (the (\σ (rt \ (σ \ i)) dip))))
∧ (the (\σ (rt \ (σ \ i)) dip)) \neq dip"

show "update (\σ (\σ (rt \ (σ \ i)) dip))) " by simp

□ dip rt (σ' (the (\σ (rt \ (σ \ i)) dip)))" by simp

proof (cases "dip = sip")
\begin{verbatim}
assume "dip = sip"
with \langle the (nhop (update (rt (σ i)) sip (0, unk, val, Suc 0, sip)) dip) \neq dip \rangle
have False by simp
thus ?thesis ..
next
assume [simp]: "dip \neq sip"
from a2 have "\langle dip \in vD(rt (σ i)) \lor dip = sip \rangle"
  by (rule vD_update_val)
with \langle dip \neq sip \rangle have "\langle dip \in vD(rt (σ i)) \rangle" by simp
moreover from a3 have "\langle dip \in vD((\σ (\nhop (rt (σ i)) dip))) \rangle" by simp
moreover from a4 have "\langle the (nhop (rt (σ i)) dip) \neq dip \rangle" by simp
ultimately have "\langle rt (σ i) \sqsubseteq dip rt (σ' (the (nhop (rt (σ i)) dip))) \rangle"
  using a1 ow by (drule(1) basic, simp)
with \langle dip \neq sip \rangle show ?thesis
  by -(erule rt_strictly_fresher_update_other, simp)
qed

\end{verbatim}
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 \implies σ j = σ' j" by auto
show "update (rt (σ i)) oip (osn, kno, val, Suc hops, sip)
  \sqsubseteq dip
   rt (σ' (the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip)) dip))))"
(is "?rt1 \sqsubseteq 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 (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) \sqsubseteq dip rt (σ (the (nhop (rt (σ i)) dip)))"
  using pre by simp
hence "rt (σ i) \sqsubseteq 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 \(\text{dip} \neq oip\)
  have "dip∈vD(rt (σ (the (nhop (rt (σ i)) dip))))"
    by simp metis
moreover from a4 and \(\text{dip} \neq oip\) have "the (nhop (rt (σ i)) dip) \̸= dip" by simp
ultimately have "rt (σ i) \sqsubseteq dip rt (σ (the (nhop (rt (σ i)) 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∈kD (rt (σ sip))"
  and "osn \leq nsqn (rt (σ sip)) oip" by auto
from a3 change \(\text{dip} = oip\) have "oip∈vD(rt (σ' sip))" by simp
hence "the (flag (rt (σ' sip)) oip) = val" by simp
from \(\text{oip} \in kD(rt (σ sip))\)
  have "osn < nsqn (rt (σ' sip)) oip \lor (osn = nsqn (rt (σ' sip)) oip
    \land the (dhops (rt (σ' sip)) oip) \leq hops)"
proof
  assume "oip∈vD(rt (σ sip))"
  hence "the (flag (rt (σ sip)) oip) = val" by simp
  with a6 \(\text{sip} \neq oip\) have "nsqn (rt (σ sip)) oip = osn \implies
    the (dhops (rt (σ sip)) oip) \leq 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))⟩
      have "the (flag (rt (σ i)) oip) = inv" 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 "sqn (rt (σ i)) oip < osn"
    also 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)
        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)
        else rt (σ i) a) = rt (σ i)" ..
  next
  from ⟨oip ∈ kD(rt (σ sip))⟩
  show "(?λa. if a = oip then Some (osn, kno, val, Suc hops, i)
      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 ⟨osn ≤ nsqn (rt (σ sip)) oip⟩
  have "nsqn (rt (σ sip)) oip < nsqn (rt (σ’ sip)) oip"
    unfolding update_def
    by (clarsimp split: option.split_asm split_if_asm)
  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 ∧ dhops (rt (σ' i)) oip ≤ hops"
  have "oip∈kD(?rt1)" by simp
  moreover have "nsqn ?rt1 oip = nsqn (?rt2 oip) oip"
  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
  ultimately 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_eql)
  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 Γ_AODV
(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 onl_oinvariant_sterms [OF aodv_wf rrep_sip [THEN weaken]]
onl onl_oinvariant_sterms [OF aodv_wf rerr_sip [THEN weaken]]
483
\texttt{onl\_oinvariant\_sterms [OF aodv\_wf oosen\_rreq [THEN weaken]]}
\texttt{onl\_oinvariant\_sterms [OF aodv\_wf odsn\_rrep [THEN weaken]]}
\texttt{solve: basic update\_0\_unk invalidate rreq\_rrep\_update}
\texttt{simp add: seqsimp)}

\texttt{fix \sigma \sigma' \_ l}
\texttt{assume or: "(\sigma, p) \in orreachable (opaodv i) (?S i) (other quality\_increases \{i\})"}
\texttt{and "other quality\_increases \{i\} \sigma \sigma'"}
\texttt{and ll: "l \in labels \Gamma_{AODV} AODV p"}
\texttt{and pre: \"\forall dip. dip\in\mathbb{D} (rt (\sigma i))}
\texttt{\wedge dip\in\mathbb{D} (rt (\sigma (\text{the (nhop (rt (\sigma i)) dip))))}
\texttt{\wedge the (nhop (rt (\sigma i)) dip) \neq dip}
\texttt{\rightarrow rt (\sigma i) \sqsubseteq dip rt (\sigma (\text{the (nhop (rt (\sigma i)) dip))))"}

\texttt{from this(1-2)}
\texttt{have or': "(\sigma', p) \in orreachable (opaodv i) (?S i) (other quality\_increases \{i\})"}
\texttt{by (rule orreachable\_other')}

\texttt{from or and ll have next\_hop: \"\forall dip. let nhip = the (nhop (rt (\sigma i)) dip)\}
\texttt{\in dip \in kD (rt (\sigma i)) \wedge nhip \neq dip}
\texttt{\rightarrow dip \in kD (rt (\sigma nhip))}
\texttt{\wedge sqn (rt (\sigma i)) dip \leq sqn (rt (\sigma nhip)) dip"}
\texttt{by (auto dest!: onl\_oinvariant\_weakenD \[OF seq\_compare\_next\_hop']}

\texttt{from or and ll have unk\_hops\_one: \"\forall dip. dip \in kD (rt (\sigma i)). sqn (rt (\sigma i)) dip = 0}
\texttt{\rightarrow sqnf (rt (\sigma i)) dip = unk}
\texttt{\wedge the (dhop\_s (rt (\sigma i)) dip) = 1}
\texttt{\wedge the (nhop (rt (\sigma i)) dip) = dip"}
\texttt{by (auto dest!: onl\_oinvariant\_weakenD \[OF ozero\_seq\_unk\_hops\_one}
\texttt{\[OF oaodv\_trans aodv\_trans\]} for otherwith\_actionD simp: seqsimp)

\texttt{from \langle other quality\_increases \{i\} \sigma \sigma' \rangle have \"\sigma' i = \sigma i\" by auto}
\texttt{hence \"quality\_increases (\sigma i) (\sigma' i)\" by auto}

\texttt{with \langle other quality\_increases \{i\} \sigma \sigma' \rangle have \"\forall j. quality\_increases (\sigma j) (\sigma' j)\"}
\texttt{by \{-erule otherE, metis singleton\_iff\}

\texttt{show \"\forall dip. dip \in \mathbb{D} (rt (\sigma\'_i))}
\texttt{\wedge dip \in \mathbb{D} (rt (\sigma' (\text{the (nhop (rt (\sigma' i)) dip))}))}
\texttt{\wedge the (nhop (rt (\sigma' i)) dip) \neq dip}
\texttt{\rightarrow rt (\sigma' i) \sqsubseteq dip rt (\sigma' (\text{the (nhop (rt (\sigma' i)) dip))))"}

\texttt{proof clarify}
\texttt{fix dip}
\texttt{assume \"dip\in\mathbb{D}(rt (\sigma' i))\"
\texttt{and \"\text{dip}\in\mathbb{D}(rt (\sigma' (\text{the (nhop (rt (\sigma' i)) dip)))))\"
\texttt{and \"the (nhop (rt (\sigma' i)) dip) \neq dip\"
\texttt{from this(1) and \langle \sigma' i = \sigma i \rangle have \"dip\in\mathbb{D}(rt (\sigma i))\"
\texttt{and \"dip\in\mathbb{D}(rt (\sigma i))\"}
\texttt{by auto}

\texttt{from \langle the (nhop (rt (\sigma' i)) dip) \neq dip \rangle and \langle \sigma' i = \sigma i \rangle}
\texttt{have \"the (nhop (rt (\sigma i)) dip) \neq dip\" (is "?nhip \neq _") by simp
\texttt{with \langle dip\in\mathbb{D}(rt (\sigma i)) \rangle and next\_hop
\texttt{have \"dip\in\mathbb{D}(rt (\sigma ?nhip))\"
\texttt{and nsqns: \"nsqn (rt (\sigma i)) dip \leq nsqn (rt (\sigma ?nhip)) dip\"}
\texttt{by (auto simp: Let_def)}

\texttt{have \"0 < sqn (rt (\sigma i)) dip\"
\texttt{proof \{rule neq0\_conv [THEN iff1, OF notI]\}
\texttt{assume \"sqn (rt (\sigma i)) dip = 0"}
\texttt{with \langle dip\in\mathbb{D}(rt (\sigma i)) \rangle and unk\_hops\_one
\texttt{have \"?nhip = dip\" by simp
\texttt{with \langle ?nhip \neq dip \rangle show False ..
\texttt{qed
\texttt{also have \"... = nsqn (rt (\sigma i)) dip\"}

484
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 ∀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 "... = 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)
  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 "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 \[ \text{"opaodv i \models (otherwith (op=) \{i\})} \]
\[ (orecmsg (\lambda \sigma. \text{msg\_fresh } \sigma \land \text{msg\_zhops } \sigma), \]
\[ \text{other quality\_increases (i) \rightarrow} \]
\[ \text{global (} \lambda \sigma. \forall \text{dip. let nhip = the (nhop (rt (\sigma i)) dip)} \]
\[ \text{in dip \in vD (rt (\sigma i)) \land vD (rt (\sigma nhip)) \land nhip \neq dip} \]
\[ \rightarrow (rt (\sigma i)) \sqsubseteq \text{dip (rt (\sigma nhip))")} \]
\[ \text{by (rule oinvariant\_weakenE [OF seq\_nhop\_quality\_increases']) (auto dest!: onlD)} \]
end

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 \[ \text{rt\_graph :: } (\text{ip } \Rightarrow \text{state}) \Rightarrow \text{ip } \Rightarrow \text{ip rel} \]
where \[ \text{"rt\_graph } \sigma = (\lambda \text{dip. } \{ (\text{ip, ip'}) | \text{ip} \text{ ip'} \text{ dsn dsk hops. } \]
\[ \text{ip} \neq \text{dip} \land \text{rt (\sigma ip) dip = Some (dsn, dsk, val, hops, ip')}\} )" \]

Given the state of a network \( \sigma \), 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') \in rt\_graph \sigma dip"
shows "ip \neq dip \land (\exists r. rt (\sigma ip) = r \]
\[ \land (\exists dsn dsk hops. r dip = Some (dsn, dsk, val, hops, ip'))\)"
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 \Rightarrow 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)^+ \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 rt\_graph \sigma dip \Rightarrow 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)^+ \Rightarrow ip \neq dip"
by (erule converse_tranclE) auto

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

lemma rt_graph_nhlp_is_nhlp [dest]:
"\forall ip ip' \sigma dip. (ip, ip') \in rt\_graph \sigma dip \Rightarrow ip' = the (nhop (rt (\sigma ip)) dip)"
unfolding rt_graph_def by auto

theorem inv_to_loop_freedom:
assumes "\forall dip. \text{let nhlp = the (nhop (rt (\sigma i)) dip)} \]
\[ \text{in dip \in vD (rt (\sigma i)) \land vD (rt (\sigma nhlp)) \land nhlp \neq dip} \]
\[ \rightarrow (rt (\sigma i)) \sqsubseteq \text{dip (rt (\sigma nhlp))")} \]
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 dip.

486
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
    fix nhip
    assume "(ip, nhip) ∈ rt_graph σ dip"
    and "dip ∈ vD(rt (σ nhip))"
    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'))"
    then 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'))"
      then 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

qed
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 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} →)
      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 ||= A (otherwith (op=) {i} (orecvmsg (λ_. orecvmsg (λ_. rreq_rrep_sn)
      σ i, other (λ_ _. True) {i} →))
      global (λ (σ, a, σ'). anycast (msg_fresh σ a)) a)
    by (auto)
    qed

next
  fix t :: "(((nat ⇒ state) × (state, msg, pseqp, pseqp label) seqp), msg seq_action) transition"
  assume "onll Γ AODV (λ((σ, s), a, (σ', s'))).
    other 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 (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:
  shows "opaodv i ⟨⟨ i qmsg ||= (otherwith (op=) {i} (orecvmsg (λσ m.
    msg_fresh σ m ∧ msg_zhops m)),
    other quality_increases {i} →)
    global (λ (σ, a, σ'). anycast (msg_fresh σ a)) a)
  by (auto)

proof -
  have "opaodv i ||= A (λσ _.. orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
    globala (λ (σ, a, σ'). quality_increases (σ i) (σ' i))"
  by (rule ostep_invariant_weakenE [OF rreq_rrep_nsqn_fresh_any_step_invariant])
  (auto)

  hence "opaodv i ⟨⟨ i qmsg ||= A (λσ _.. orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
    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_nsqn_fresh_any_step:
  shows "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)) a)"

proof -
  have "opaodv i ||= A (λσ _.. orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
    globala (λ (σ, a, σ'). anycast (msg_fresh σ a)) a)"
  by (auto)

  thus ?thesis by rule auto
qed

488
assume "onll \( \Gamma_{AODV} (\lambda(\sigma, a, \_). \text{anycast} (\text{msg}_{\text{fresh}} \sigma) a) t)"
thus "globala (\lambda(\sigma, a, \_). \text{anycast} (\text{msg}_{\text{fresh}} \sigma) a) t"

by (cases t) (clarsimp dest!: onllD, metis aodv_ex_label)

qed

hence "opaodv i \langle\langle i \text{qmsg} \mid= A (\lambda \sigma \_. \text{orecvmsg} (\lambda_. \text{rreq}_{\text{rrep}} sn) \sigma, \text{other} (\lambda_. \_ True) \{i\} \rightarrow) \text{globala} (\lambda(\_, a, \_). \text{anycast} (\text{msg}_{\text{fresh}} \sigma) a)"

by (rule lift_step_into_qmsg_statelessassm) simp_all

thus ?thesis by rule auto

qed


5.11.2 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) \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)

lemma node_nhop_quality_increases:
shows "\langle i : \text{opaodv} i \langle\langle i \text{qmsg} \mid= A (\lambda \sigma \_. \text{orecvmsg} (\lambda_. \text{rreq}_{\text{rrep}} sn) \sigma, \text{other} (\lambda_. \_ True) \{i\} \rightarrow) \text{globala} (\lambda(\_, a, \_). \text{anycast} \sigma_{\text{zhops}} a)"
by (rule node_lift \[OF par_nhop_quality_increases\]) auto

lemma node_quality_increases:
shows "\langle i : \text{opaodv} i \langle\langle i \text{qmsg} \mid= A (\lambda \sigma \_. \text{orecvmsg} (\lambda_. \text{rreq}_{\text{rrep}} sn) \sigma, \text{other} (\lambda_. \_ True) \{i\} \rightarrow) \text{globala} (\lambda(\_, a, \_). \text{anycast} \sigma_{\text{zhops}} a)"
by (rule node_lift_step_statelessassm \[OF par_rreq_rrep_sn_quality_increases\]) simp

lemma node_rreq_rrep_nsqn_fresh_any_step:
shows "\langle i : \text{opaodv} i \langle\langle i \text{qmsg} \mid= A (\lambda \sigma \_. \text{orecvmsg} (\lambda_. \text{rreq}_{\text{rrep}} sn) \sigma, \text{other} (\lambda_. \_ True) \{i\} \rightarrow) \text{globala} (\lambda(\_, a, \_). \text{anycast} \sigma_{\text{zhops}} 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 \(\lambda \sigma \_ . \text{arrivemsg} (\lambda \_ . \text{rreq_rrep_sn}) \sigma, \text{other} (\lambda \_ . \text{True}) \{i\} \rightarrow\))
globala ((\lambda (\_ \_ a \_ \_). \text{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 \(\lambda \sigma \_ . \text{arrivemsg} (\lambda \_ . \text{True}) \sigma, \text{other} (\lambda \_ . \text{True}) \{i\} \rightarrow\))
globala ((\lambda (\_ a \_ a \_). \text{castmsg msg_zhops a}))"
by (rule node_lift_anycast_statelessassm [OF par_anycast_msg_zhops])

proof
(rule ostep_invariantI, simp (no_asm), rule impI)
fix σ a
have "((σ, a) ∈ reachable ((i : opaodv i @(i qmsg : R)@o))
(\lambda \sigma \_ . \text{arrivemsg} (\lambda \_ . \text{True}) \sigma)
(\text{other} (\lambda \_ . \text{True}) \{i\}))"
and tr: "((σ, a), (σ', a')) ∈ trans ((i : opaodv i @(i qmsg : R)@o))
and "a ≠ 𝜋n" from or obtain p R where "(ζ = NodeS i p R"
bys (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))"
bysimp thus "σ' i = σ i" using (a ≠ 𝜋n)
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_rreq_rrep_nsqn_fresh_inc_sn [simp]:
assumes "\text{arrivemsg} (\lambda \sigma m. \text{msg_fresh} σ m ∧ P σ m) σ m"
shows "\text{arrivemsg} (\lambda \_ . \text{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)
(\text{arrivemsg} (λσ m. \text{msg_fresh} σ m ∧ msg_zhops m)),
other_quality_increases (net_tree_ips p) \rightarrow\))
global (λσ. \forall i\in\text{net_tree_ips p}. \forall dip.
let nhip = the (nhop (rt (i qmsg : R)@p)
in dip ∈ vD (rt (σ i)) \text{other} (\text{arrivemsg} (\lambda \_ . \text{rreq_rrep_sn}) σ)
(\text{other} (\lambda \_ . \text{True}) \{i\})
\rightarrow (rt (σ i)) ⊑ \text{dip} (rt (\sigma nhip)))"
by (rule pnet_lift [OF node_nhop_quality_increases])

proof
(rule ostep_invariantI, simp (no_asm))
fix s a
have "(i : opaodv i @(i qmsg : R)@o \(\lambda \sigma \_ . \text{arrivemsg} (\lambda \_ . \text{rreq_rrep_sn}) \sigma, \text{other} (\lambda \_ . \text{True}) \{i\} \rightarrow\))
globala ((\lambda (\sigma a \_ \_ a \_ . \text{castmsg msg_zhops a}))"
by (auto dest!: ostep_invariantD [OF node_rreq_rrep_nsqn_fresh_any_step])
moreover from or am have "\text{castmsg} (msg_zhops a) a"
by (auto dest!: ostep_invariantD [OF node_anycast_Msg_zhops])
ultimately show "\text{castmsg} (λm. \text{msg_fresh} σ m ∧ msg_zhops m) a"
by (case_tac a) auto

qed

thus "(i : opaodv i @(i qmsg : R)@o \(\lambda \sigma \_ . \text{arrivemsg} (\lambda \sigma m. \text{msg_fresh} σ m ∧ msg_zhops m) \sigma,"
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, σ').
  a ≠ τ ∧ (∀i d. a ≠ i:deliver(d)) → σ i = σ' i)"

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, σ').
  a ≠ τ ∨ (∃d. a = i:deliver(d)) → quality_increases (σ i) (σ' 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 (λi. opaodv i ⟨⟨i qmsg⟩⟩ p) |
    (otherwith (op=) (net_tree_ips p) inoclosed, ?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 "oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m) σ 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 simp_all

  qed

  qed

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 "{(σ, ζ). (σ i, ζ) ∈ S_{AODV} i ∧ (∀j. j ≠ i → σ j ∈ fst ' S_{AODV} j)} ⊆ S_{AODV}'"

    unfolding S_{AODV}_def S_{AODV}'_def

    proof (rule equalityD1)

    qed

    qed

    qed
show "\( \forall j. (\sigma, \zeta) \in \{(f j, p)\} \Rightarrow f j \in \text{fst } \{(f j, p)\} = \{(f, p)\} \)"
by (rule set_eqI) auto
qed
thus "\{ (\sigma, \zeta) \mid \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 \( \sigma_{AODV} \_\text{def} \) 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 "\( (\sigma, a, \sigma') \in \text{trans (paodv i)} \)"
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 \( s = (\sigma i, q) \) and \( s' = (\sigma' i, q') \)
show "\( ((\sigma, \text{snd (id s)}), a, (\sigma', \text{snd (id s')}) \in \text{trans (opaodv i)} \)"
by simp
qed
then interpret 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 \( \sigma_{AODV} \_\text{def} \) by simp
hence "\( \forall i. \text{openproc.initmissing paodv id i} = \text{initmissing i} \)"
unfolding op.initmissing_def op.someinit_def initmissing_def
by (auto split: option.split)
thus "\( \text{openproc.initmissing paodv id} = \text{initmissing} \) ..
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 "\( \text{openproc_parq paodv opaodv id qmsg} \)"
by (unfold_locales) simp
then interpret opq: openproc_parq paodv opaodv id qmsg
have im: "\( \forall \sigma. \text{openproc.initmissing } (\lambda i. \text{paodv i } \text{<< qmsg}) (\lambda (p, q). (\text{fst (id p)}, \text{snd (id p)}, q)) \sigma = \text{initmissing } \sigma \)"
unfolding opq.initmissing_def opq.someinit_def initmissing_def
unfolding \( \sigma_{AODV} \_\text{def} \) by (clarsimp cong: option.case_cong)
thus "\( \text{openproc.initmissing } (\lambda i. \text{paodv i } \text{<< qmsg}) (\lambda (p, q). (\text{fst (id p)}, \text{snd (id p)}, q)) \Rightarrow \text{netglobal } P \sigma \)"
unfolding opq.netglobal_def netglobal_def opq.initmissing_def initmissing_def opq.someinit_def
unfolding \( \sigma_{AODV} \_\text{def} \) by (clarsimp cong: option.case_cong)
thus "\( \text{openproc.netglobal } (\lambda i. \text{paodv i } \text{<< qmsg}) (\lambda (p, q). (\text{fst (id p)}, \text{snd (id p)}, q)) = \text{netglobal} \)"
by auto
qed

lemma net_nhop_quality_increases:
assumes "\( \text{wf_net_tree n} \)"
shows "closed (pnet (λi. paodv i ⟨⟨ 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 ⟨⟨ 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

5.11.6 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
**Bibliography**

http://afp.sf.net/entries/AWN.shtml.


