Mechanization of the Algebra for Wireless Networks (AWN)

Timothy Bourke

To cite this version:
Timothy Bourke. Mechanization of the Algebra for Wireless Networks (AWN). 2014, pp.186. hal-01104031

HAL Id: hal-01104031
https://inria.hal.science/hal-01104031
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.
Mechanization of the Algebra for Wireless Networks (AWN)

Timothy Bourke*
August 28, 2014

Abstract

AWN is a process algebra developed for modelling and analysing protocols for Mobile Ad hoc Networks (MANETs) and Wireless Mesh Networks (WMNs) [2, §4]. AWN models comprise five distinct layers: sequential processes, local parallel compositions, nodes, partial networks, and complete networks.

This development mechanises the original operational semantics of AWN and introduces a variant ‘open’ operational semantics that enables the compositional statement and proof of invariants across distinct network nodes. It supports labels (for weakening invariants) and (abstract) data state manipulations. A framework for compositional invariant proofs is developed, including a tactic (inv_cterms) for inductive invariant proofs of sequential processes, lifting rules for the open versions of the higher layers, and a rule for transferring lifted properties back to the standard semantics. A notion of ‘control terms’ reduces proof obligations to the subset of subterms that act directly (in contrast to operators for combining terms and joining processes).

Further documentation is available in [1].
## Contents

1. **Generic functions and lemmas** 3
2. **Transition systems (automata)** 4
3. **Reachability and Invariance** 4
   - 3.1 Reachability ........................................ 4
   - 3.2 Invariance .......................................... 5
4. **Open reachability and invariance** 9
   - 4.1 Open reachability .................................. 9
   - 4.2 Open Invariance ................................... 11
   - 4.3 Standard assumption predicates .................. 18
5. **Terms of the Algebra for Wireless Networks** 19
   - 5.1 Sequential Processes ............................... 19
   - 5.2 Actions ............................................ 23
     - 5.2.1 Sequential Actions (and related predicates) .... 23
     - 5.2.2 Node Actions (and related predicates) ......... 24
   - 5.3 Networks ........................................... 25
6. **Semantics of the Algebra of Wireless Networks** 31
   - 6.1 Table 1: Structural operational semantics for sequential process expressions ............... 31
   - 6.2 Table 2: Structural operational semantics for parallel process expressions .............. 33
   - 6.3 Table 3: Structural operational semantics for node expressions ........................... 34
   - 6.4 Table 4: Structural operational semantics for partial network expressions ............. 36
   - 6.5 Table 5: Structural operational semantics for complete network expressions .......... 38
7. **Control terms and well-definedness of sequential processes** 38
   - 7.1 Microsteps ........................................ 38
   - 7.2 Wellformed process specifications ................... 40
   - 7.3 Start terms (sterns) ................................ 41
   - 7.4 Start terms ........................................ 42
   - 7.5 Derivative terms ................................... 47
   - 7.6 Control terms ..................................... 49
   - 7.7 Local control terms ............................... 50
   - 7.8 Local derivative terms ............................ 53
   - 7.9 More properties of control terms .................. 54
8. **Labelling sequential processes** 56
   - 8.1 Labels ............................................. 56
9. **A custom tactic for showing invariants via control terms** 60
10. **Configure the inv-cterms tactic for sequential processes** 63
11. **Lemmas for partial networks** 64
12. **Lemmas for closed networks** 73
13. **Open semantics of the Algebra of Wireless Networks** 76
    - 13.1 Open structural operational semantics for sequential process expressions ................ 76
    - 13.2 Open structural operational semantics for parallel process expressions ................... 77
    - 13.3 Open structural operational semantics for node expressions .......................... 79
    - 13.4 Open structural operational semantics for partial network expressions ............. 82
    - 13.5 Open structural operational semantics for complete network expressions ........... 83
14. **Configure the inv-cterms tactic for open sequential processes** 84
theory Lib
imports Main
begin

definition
  "TT :: 'a ⇒ bool"
where
  "TT = (λ_. True)"

lemma TT_True [intro, simp]: "TT a"
  unfolding TT_def by simp

lemma in_set_tl: "x ∈ set (tl xs) ⟹ x ∈ set xs"
  by (metis Nil_tl insert_iff list.collapse set_simps(2))

lemma nat_le_eq_or_lt [elim]:
  fixes x :: nat
  assumes "x ≤ y"
  and eq: "x = y ⟹ P x y"
  and lt: "x < y ⟹ P x y"

21 Transfer standard invariants into open invariants
22 Model the standard queuing model
23 Lifting rules for parallel compositions with QMSG
24 Transfer open results onto closed models
25 Import all AWN-related theories
26 Simple toy example
  26.1 Messages used in the protocol
  26.2 Protocol model
  26.3 Define an open version of the protocol
  26.4 Predicates
  26.5 Sequential Invariants
  26.6 Global Invariants
  26.7 Lifting
  26.8 Transfer
  26.9 Final result
27 Acknowledgements
shows "P x y"
using `assms` unfolding `nat_less_le` by auto

lemma `disjoint_commute`:
"(A ∩ B = {}) ⇒ (B ∩ A = {})"
by auto

definition `default` :: 

<table>
<thead>
<tr>
<th>&quot;(i ⇒ 's) ⇒ ('i ⇒ 's option) ⇒ ('i ⇒ 's)&quot;</th>
</tr>
</thead>
</table>

where
"default df f = (λi. case f i of None ⇒ df i | Some s ⇒ s)"

eend

2 Transition systems (automata)

theory `TransitionSystems`
imports Main
begin

type_synonym ('s, 'a) transition = "'s × 'a × 's"

record ('s, 'a) automaton =
iinit :: "'s set"
itrans :: "('s, 'a) transition set"

eend

3 Reachability and Invariance

theory `Invariants`
imports Lib `TransitionSystems`
begin

3.1 Reachability

A state is 'reachable' under I if either it is the initial state, or it is the destination of a transition whose action satisfies I from a reachable state. The 'standard' definition of reachability is recovered by setting I to `TT`.

inductive_set reachable
for A :: "('s, 'a) automaton"
and I :: "'a ⇒ bool"
where
reachable_init: "s ∈ iinit A ⇒ s ∈ reachable A I"
| reachable_step: "[ s ∈ reachable A I; (s, a, s') ∈ itrans A; I a ] ⇒ s' ∈ reachable A I"

inductive_cases `reachable_icases`: "s ∈ reachable A I"

lemma `reachable_pair_induct` [consumes, case_names init step]:
assumes "("(ξ, p) ∈ reachable A I"
and "A ξ p. (ξ, p) ∈ iinit A ⇒ P ξ p"
and "(Λξ p ξ' p' a. [ (ξ, p) ∈ reachable A I; P ξ p; ((ξ, p), a, (ξ', p')) ∈ itrans A; I a ] ⇒ P ξ' p')"
shows "P ξ p"
using `assms(1)` proof (induction "("(ξ, p)" arbitrary: ξ p)
fix ξ p
assume "("(ξ, p) ∈ iinit A"
with `assms(2)` show "P ξ p" .
next
fix s a ξ' p'
assume "s ∈ reachable A I"
and tr: "(s, a, (ξ', p')) ∈ itrans A"
and "I a"

and IH: "\(\forall \xi. p. s = (\xi, p) \implies P \xi p\)"
from this(1) obtain \(\xi\) \(p\) where "\(s = (\xi, p)\)"
and "\((\xi, p) \in \text{reachable A I}\)"

by (metis pair_collapse)

note this(2)

moreover from IH and 's = (\(\xi, p\))' have "\(P \xi p\)".
moreover from tr and 's = (\(\xi, p\))' have "\(((\xi, p), a, (\(\xi', p')\)) \in \text{trans A}\)" by simp

ultimately show "\(P \xi' p'\)"

using 'I a' by (rule assms(3))

qed

lemma reachable_weakenE [elim]:
assumes "\(s \in \text{reachable A P}\)"
and PQ: "\(\forall a. P a \implies Q a\)"
shows "\(s \in \text{reachable A Q}\)"

using assms(1)

proof (induction)
  fix \(s\) assume "\(s \in \text{init A}\)"
  thus "\(s \in \text{reachable A Q}\)" ..

next
  fix \(s\) a \(s'\)
  assume "\(s \in \text{reachable A P}\)"
  and "\(s \in \text{reachable A Q}\)"
  and "\((s, a, s') \in \text{trans A}\)"
  and "\(P a\)"

from 'P a' have "\(Q a\)" by (rule PQ)

with 's \in \text{reachable A Q}' and '(s, a, s') \in \text{trans A}' show "\(s' \in \text{reachable A Q}\)" ..

qed

lemma reachable_weaken_TT [elim]:
assumes "\(s \in \text{reachable A I}\)"
shows "\(s \in \text{reachable A TT}\)"

using assms by rule simp

lemma init_empty_reachable_empty:
assumes "\(\text{init A} = {}\)"
shows "\(\text{reachable A I} = {}\)"

proof (rule ccontr)
  assume "\(\text{reachable A I} \neq {}\)"
  then obtain \(s\) where "\(s \in \text{reachable A I}\)" by auto
  thus False

proof (induction rule: reachable.induct)
  fix \(s\)
  assume "\(s \in \text{init A}\)"
  with 'init A = {}' show False by simp

qed

3.2 Invariance

definition invariant :: "('s, 'a) automaton \Rightarrow ('a \Rightarrow bool) \Rightarrow ('s \Rightarrow bool) \Rightarrow bool"
("_ \models (I \rightarrow) P\) = (\(\forall s \in \text{reachable A I}. P s\)"

abbreviation any_invariant :: "('s, 'a) automaton \Rightarrow ('s \Rightarrow bool) \Rightarrow bool"
("_ \models _\) = (\([100, 0, 9] 8\))

where
"(A \models (I \rightarrow) P) = (\forall s \in \text{reachable A I}. P s)"

abbreviation
\(\text{any\_invariant}\) :: "('s, 'a) automaton \Rightarrow ('s \Rightarrow bool) \Rightarrow bool"
("_ \models _\) = (\([100, 9] 8\))

where
"(A \models P) \equiv (A \models (TT \rightarrow) P)"

lemma invariantI [intro]:
assumes init: \( \forall s. s \in \text{init } A \implies P s \)
and step: \( \forall s, a, s'. [ s \in \text{reachable } A I; P s; (s, a, s') \in \text{trans } A; I a ] \implies P s' \)
shows \( A \models (I \to) P \)

unfolding invariant_def

proof

fix s
assume "s \in \text{reachable } A I"
thus "P s" by (rule init)

next
fix s a s'
assume "s \in \text{reachable } A I"
and "P s" and "(s, a, s') \in \text{trans } A" and "I a"
thus "P s'" by (rule step)

qed

lemma invariant_pairI [intro]:
assumes init: \( \forall \xi. (\xi, p) \in \text{init } A \implies P (\xi, p) \)
and step: \( \forall \xi, p, \xi', p', a. [ (\xi, p) \in \text{reachable } A I; P (\xi, p); ((\xi, p), a, (\xi', p')) \in \text{trans } A; I a ] \implies P (\xi', p') \)
shows \( A \models (I \to) P \)

using assms by auto

lemma invariant_arbitraryI:
assumes "\( \forall s. s \in \text{reachable } A I \implies P s \)"
shows "A \models (I \to) P"

using assms unfolding invariant_def by simp

lemma invariantD [dest]:
assumes "A \models (I \to) P"
and "s \in \text{reachable } A I"
shows "P s"

using assms unfolding invariant_def by blast

lemma invariant_initE [elim]:
assumes invP: "A \models (I \to) P"
and init: "s \in \text{init } A"
shows "P s"

proof

- from init have "s \in \text{reachable } A I" ..
with invP show ?thesis ..

qed

lemma invariant_weakenE [elim]:
fixes T \( T \in Q \)
assumes invP: "A \models (PI \to) P"
and PQ: "\( \forall s. P s \implies Q s \)"
and QIPI: "\( \forall a. QI a \implies PI a \)"
shows "A \models (QI \to) Q"

proof

fix s
assume "s \in \text{init } A"
with invP have "P s" ..
thus "Q s" by (rule PQ)

next

fix s a s'
assume "s \in \text{reachable } A QI"
and "(s, a, s') \in \text{trans } A"
and "QI a"
from '"QI a' have "PI a" by (rule QIPI)
from 's ∈ reachable A QI' and QIPI have "s ∈ reachable A PI"

hence "s' ∈ reachable A PI" using '(s, a, s') ∈ trans A' and 'PI a' ..
with invP have "P s'" ..
thus "Q s'" by (rule PQ)

qed

definition
  step_invariant :: "('s, 'a) automaton ⇒ ('a ⇒ bool) ⇒ ('s, 'a) transition ⇒ bool"
  where
    "( (A |=_A (I → P)) = (∀a. I a → (∀s∈reachable A I. (∀s′. (s, a, s′) ∈ trans A → P (s, a, s′)))) )"

lemma invariant_restrict_inD [dest]:
  assumes "A |=_A (TT → P)"
  shows "A |=_A (QI → P)"
using assms by auto

abbreviation
  any_step_invariant :: "('s, 'a) automaton ⇒ ('s, 'a) transition ⇒ bool"
  where
    "(A |=_A P) ≡ (A |=_A (TT → P))"

lemma step_invariant_true:
  "p |=_A (λ(s, a, s′). True)"
unfolding step_invariant_def by simp

lemma step_invariantI [intro]:
  assumes *: "∀ s a s′. [ s∈reachable A I; (s, a, s′)∈trans A; I a ] → P (s, a, s′)"
  shows "A |=_A (I → P)"
unfolding step_invariant_def using assms by auto

lemma step_invariantD [dest]:
  assumes "A |=_A (I → P)"
  and "s ∈ reachable A I"
  and "(s, a, s′) ∈ trans A"
  and "I a"
  shows "P (s, a, s′)"
using assms unfolding step_invariant_def by blast

lemma step_invariantE [elim]:
  fixes T σ P I s a s'
  assumes "A |=_A (I → P)"
  and "s∈reachable A I" 
  and "(s, a, s′) ∈ trans A"
  and "I a" 
  and "P (s, a, s′) → Q" 
  shows "Q"
using assms by auto

lemma step_invariant_pairI [intro]:
  assumes *: "∀ξ p p′ ξ′. [ (ξ, p) ∈ reachable A I; ((ξ, p), a, (ξ′, p′)) ∈ trans A; I a ] → P ((ξ, p), a, (ξ′, p′))"
  shows "A |=_A (I → P)"
using assms by auto

lemma step_invariant_arbitraryI:
  assumes "∀ξ p a ξ′ p′. [ (ξ, p) ∈ reachable A I; ((ξ, p), a, (ξ′, p′)) ∈ trans A; I a ]
  . . . . 

7
\[
\Rightarrow P ((\xi, p), a, (\xi', p'))
\]
shows "\(A \models_{A} (I \rightarrow) P\)"
using \texttt{assms} by \texttt{auto}

**Lemma step_invariant_weakenE [elim!]:**

fixes \(T \circ P \ Q\)
assumes \(\text{invP}: "A \models_{A} (PI \rightarrow) P"\)
and \(\text{PQ}: "\forall t. P \ t \Rightarrow Q \ t"\)
and \(\text{QIPI}: "\forall a. QI \ a \Rightarrow PI \ a"\)
shows "\(A \models_{A} (QI \rightarrow) Q"\)
proof
fix \(s \ a \ s'\)
assume "\(s \in \text{reachable} A QI""
and "\((s, a, s') \in \text{trans} A""
and "\(QI \ a"\)
from \('QI \ a' have "PI a" by (rule QIPI)"
from \('s \in \text{reachable} A QI' have "s \in \text{reachable} A PI" using QIPI ..
with invP have "P (s, a, s')" using \'(s, a, s') \in \text{trans} A' 'PI a' ..
thus "Q (s, a, s')" by (rule PQ)
qed

**Lemma step_invariant_weaken_with_invariantE [elim]:**

assumes \(\text{pinv}: "A \models_{A} (I \rightarrow) P"\)
and \(\text{qinv}: "A \models_{A} (I \rightarrow) Q"\)
and \(\text{wr}: "\forall s a s'. [ \[ P \ s; P \ s'; Q (s, a, s'); I a \] \] \Rightarrow R (s, a, s')"\)
shows "\(A \models_{A} (I \rightarrow) R"\)
proof
fix \(s \ a \ s'\)
assume \(\text{sr}: "s \in \text{reachable} A I""
and \(\text{tr}: "(s, a, s') \in \text{trans} A""
and "\(I a"\)
hence "\(s' \in \text{reachable} A I"\) ..
with pinv have "P s'" ..
from pinv and \(\text{sr} have "P s" ..
from qinv \(\text{sr} \ \text{tr} 'I a' have "Q (s, a, s')" ..
with \(P s'\) and \(P s'\) show "R (s, a, s')" using \(I a'\) by (rule wr)
qed

**Lemma step_to_invariantI:**

assumes \(\text{sinv}: "A \models_{A} (I \rightarrow) Q"\)
and \(\text{init}: "\forall s. s \in \text{init} A \Rightarrow P \ s"\)
and \(\text{step}: "\forall s s' a.\]
\[
\begin{align*}
& s \in \text{reachable} A I; \\
& P \ s; \\
& Q (s, a, s'); \\
& I a \] \Rightarrow P s'\]
shows "\(A \models (I \rightarrow) P"\)
proof
fix \(s \ a\)
assume "\(s \in \text{init} A\)" thus "\(P \ s\)" by (rule init)
next
fix \(s \ s' \ a\)
assume "\(s \in \text{reachable} A I"\)
and "\(P \ s"\)
and "\((s, a, s') \in \text{trans} A""
and "\(I a"\)
show "\(P s'\)"
proof -
from sinv and \('s\in\text{reachable} A I' and \'(s, a, s')\in\text{trans} A' and \'I a' have "Q (s, a, s')" ..
with \('s\in\text{reachable} A I' and \'P s' show "P s'" using \'I a' by (rule step)
qed
qed

end
4 Open reachability and invariance

theory OInvariants
imports Invariants
begin

4.1 Open reachability

By convention, the states of an open automaton are pairs. The first component is considered to be the global state and the second is the local state.

A state is 'open reachable' under $S$ and $U$ if it is the initial state, or it is the destination of a transition—where the global components satisfy $S$—from an open reachable state, or it is the destination of an interleaved environment step where the global components satisfy $U$.

\[
\text{inductive set ooreachable} :: \text{('('g × 'l, 'a) automaton} \\
\to ('g ⇒ 'g ⇒ 'a ⇒ bool) \to ('g × 'l) set
\]

for $A :: ('g × 'l, 'a) automaton$ and $S :: 'g ⇒ 'g ⇒ 'a ⇒ bool$ and $U :: 'g ⇒ 'g ⇒ bool$

where

-$\text{oreachable_init: } s ∈ \text{init } A ⇒ s ∈ \text{oreachable } A S U$

- $\text{oreachable_local: } [s ∈ \text{oreachable } A S U; (s, a, s') ∈ \text{trans } A; S (\text{fst } s) (\text{fst } s') a] ⇒ s' ∈ \text{oreachable } A S U$

- $\text{oreachable_other: } [s ∈ \text{oreachable } A S U; U (\text{fst } s) σ'] ⇒ (σ', \text{snd } s) ∈ \text{oreachable } A S U$

lemma ooreachable_local' [elim]:

assumes "$((σ, p) ∈ \text{oreachable } A S U$

and "$((σ, p), a, (σ', p')) ∈ \text{trans } A$

and "$S (σ σ') a$

shows "$(σ', p') ∈ \text{oreachable } A S U$

using assms by (metis fst_conv ooreachable.oreachable_local)

lemma ooreachable_other' [elim]:

assumes "$(σ, p) ∈ \text{oreachable } A S U$

and "$U (σ σ')$

shows "$(σ', p) ∈ \text{oreachable } A S U$

proof -

from "$U (σ σ')$" have "$U (\text{fst } (σ, p)) σ'$" by simp

with "$((σ, p) ∈ \text{oreachable } A S U' have "(σ', \text{snd } (σ, p)) ∈ \text{oreachable } A S U'$" by (rule ooreachable_other)

thus "$(σ', p) ∈ \text{oreachable } A S U'$ by simp

qed

lemma ooreachable_pair_induct [consumes, case_names init other local]:

assumes "$(σ, p) ∈ \text{oreachable } A S U$

and "$\text{\llcorner }σ. p. (σ, p) ∈ \text{init } A \Rightarrow P \sigma p\text{\lrcorner}$

and "$(\\text{\llcorner }σ p. σ'. [\text{\llcorner }σ p. (σ, p) ∈ \text{oreachable } A S U; P σ p; U σ σ' ] \Rightarrow P σ' p\text{\lrcorner})$"

and "$(\\text{\llcorner }σ p. σ' p'. \text{\lrcorner} a. [\text{\llcorner }σ p. (σ, p) ∈ \text{oreachable } A S U; P σ p; (\text{\llcorner }σ p, a, (σ', p')) ∈ \text{trans } A; S σ σ' a ] \Rightarrow P σ' p'\text{\lrcorner})$""

shows "$P σ p$

using assms (1) proof (induction "$\text{\llcorner }σ, p\text{\lrcorner}$" arbitrary: σ p)

fix σ p

assume "$((σ, p) ∈ \text{init } A$

with assms(2) show "$P σ p$".

next

fix s σ'

assume "$s ∈ \text{oreachable } A S U$

and "$U (\text{fst } s) σ'$"

and IH: "$\\text{\llcorner }σ. p. s = (σ, p) \Rightarrow P σ p$"

from this(1) obtain σ p where "$s = (σ, p)$"
and \((\sigma, p) \in \text{reachable A S U}\)

by \((\text{metis surjective_pairing})\)

note this(2)

moreover from IH and \(s = (\sigma, p)\) have \(" P \sigma p"\).

moreover from \(U (\text{fst} s) \sigma'\) and \(s = (\sigma, p)\) have \(" U \sigma \sigma'"\) by simp

ultimately have \(" P \sigma' p"\) by \((\text{rule assms(3)})\)

with \(s = (\sigma, p)\) show \(" P \sigma' (\text{snd} s)"\) by simp

next

fix \(s a \sigma' p'\)

assume \(s \in \text{reachable A S U}\)

and tr: \(" (s, a, (\sigma', p')) \in \text{trans A}"\)

and IH: \"\forall \sigma p. s = (\sigma, p) \implies P \sigma p"\)

from this(1) obtain \(\sigma p\) where \"s = (\sigma, p)"

and \(" (\sigma, p) \in \text{reachable A S U}\"

by \((\text{metis surjective_pairing})\)

note this(2)

moreover from IH \(s = (\sigma, p)\) have \(" P \sigma p"\).

moreover from tr and \(s = (\sigma, p)\) have \(" ((\sigma, p), a, (\sigma', p')) \in \text{trans A}"\) by simp

ultimately show \(" P \sigma' p'"\) by \((\text{rule assms(4)})\)

qed

lemma reachable_weakenE [elim]:

assumes \(" s \in \text{reachable A PS PU}"\)

and PSQS: \"\forall s s' a. PS s s' a = \implies QS s s' a"\)

and PUQU: \"\forall s s'. PU s s' = \implies QU s s'"\)

shows \"s \in \text{reachable A QS QU}\"

using assms(1)

proof (induction)

fix \(s\) assume \"s \in init A\"

thus \"s \in \text{reachable A QS QU}" ..

next

fix \(s a \sigma s'\)

assume \"s \in \text{reachable A QS QU}\"

and \"(s, a, s') \in \text{trans A}"\)

and \"PS (\text{fst} s) (\text{fst} s') a"\)

from \(\text{PS (fst s) (fst s') a'}\) have \"QS (\text{fst} s) (\text{fst} s') a" by \((\text{rule PSQS})\)

with \(s' \in \text{reachable A QS QU}\) and \'(s, a, s') \in \text{trans A}' show \"s' \in \text{reachable A QS QU}" ..

next

fix \(s g'\)

assume \"s \in \text{reachable A QS QU}\"

and \"PU (\text{fst} s) g'"\)

from \(\text{PU (fst s) g'}\) have \"QU (\text{fst} s) g'" by \((\text{rule PUQU})\)

with \(s' \in \text{reachable A QS QU}'\) show \"(g', \text{snd} s) \in \text{reachable A QS QU}" ..

qed

definition

act \::\:"(\text{'}a \Rightarrow \text{bool}\text{'} \Rightarrow \text{'}s \Rightarrow \text{'}s \Rightarrow \text{'}a \Rightarrow \text{bool}\text{')"}

where

"act I = (\text{\lambda a. I})"\)

lemma act_simp [iff]: \"act I s s' a = I a"\)

unfolding act_def ..

lemma reachable_in_oreachable [elim]:

fixes \(s\)

assumes \"s \in \text{reachable A I}"\)

shows \"s \in \text{oreachable A (act I) U}"\)

unfolding act_def using assms proof induction

fix \(s\)

assume \"s \in init A\"

thus \"s \in \text{oreachable A (act I) U}" ..

next
fix $s$ a $s'$
assume "$s \in \text{oreachable } A (\lambda \_ \_ . I) U$"
and "$(s, a, s') \in \text{trans } A$
and "$I a$
thus "$s' \in \text{oreachable } A (\lambda \_ \_ . I) U$"
by (rule $\text{oreachable\_local}$)
qed

4.2 Open Invariance

definition $\text{oinvariant}$
:: "'(g \times 'l, 'a) automaton
 \Rightarrow (('g \Rightarrow 'g \Rightarrow 'a \Rightarrow \text{bool}) \Rightarrow ('g \Rightarrow 'g \Rightarrow \text{bool})
 \Rightarrow (('g \times 'l) \Rightarrow \text{bool}) \Rightarrow \text{bool}"
("s = (1', ((1\_),/ (1\_) \rightarrow ')/) \_") [100, 0, 0, 9] 8

where
"(A = (S, U \rightarrow P) = (\forall s\in\text{oreachable } A S U. P s)"

lemma $\text{oinvariant\_I}$ [intro]:
fixes $T$ $I$ $S$ $U$ $P$
assumes init: "\[ \forall s. s \in \text{init } A \Rightarrow P s\]
and other: "\[ \forall g g' l. [ (g, 1) \in \text{oreachable } A S U; P (g, 1); U g g' ] \Rightarrow P (g', 1)\]
and local: "\[ \forall s a s'. [ s \in \text{oreachable } A S U; P s; (s, a, s') \in \text{trans } A; S (\text{fst } s) (\text{fst } s') a ] \Rightarrow P s'\]
shows "A = (S, U \rightarrow P)"

unfolding $\text{oinvariant\_def}$
proof
fix $s$
assume "$s \in \text{oreachable } A S U$"
thus "$P s$" by (rule init)
next
fix $s$ a $s'$
assume "$s \in \text{oreachable } A S U$"
and "$P s$"
and "$(s, a, s') \in \text{trans } A$
and "$S (\text{fst } s) (\text{fst } s') a$"
thus "$P s'$" by (rule local)
next
fix $s$ $g'$
assume "$s \in \text{oreachable } A S U$"
and "$P s$"
and "$U (\text{fst } s) g'$"
thus "$P (g', \text{snd } s)$"
by - (rule other [where $g$="\text{fst } s"], simp_all)
qed

lemma $\text{oinvariant\_oreachable\_I}$:
assumes "$\forall \sigma s. (\sigma, s) \in \text{oreachable } A S U \Rightarrow P (\sigma, s)"
shows "A = (S, U \rightarrow P)"
using assms unfolding $\text{oinvariant\_def}$ by auto

lemma $\text{oinvariant\_pair\_I}$ [intro]:
assumes init: "\[ \forall \sigma p. (\sigma, p) \in \text{init } A \Rightarrow P (\sigma, p)\]
and local: "\[ \forall \sigma p \sigma' p' a. [ (\sigma, p) \in \text{oreachable } A S U; P (\sigma, p); ((\sigma, p), a, (\sigma', p')) \in \text{trans } A; S \sigma \sigma' a ] \Rightarrow P (\sigma', p')\]
and other: "\[ \forall \sigma \sigma'. [ (\sigma, p) \in \text{oreachable } A S U; P (\sigma, p); U \sigma \sigma' ] \Rightarrow P (\sigma', p)\]
shows "A = (S, U \rightarrow P)"
by (rule oinvariantI)
   (clarsimp | erule init | erule(3) local | erule(2) other)+

lemma oinvariantD [dest]:
  assumes "A |= (S, U →) P"
    and "s ∈ oreachable A S U"
  shows "P s"
using assms unfolding oinvariant_def
by clarsimp

lemma oinvariant_initD [dest, elim]:
  assumes invP: "A |= (S, U →) P"
    and init: "s ∈ init A"
  shows "P s"
proof -
  from init have "s ∈ oreachable A S U" ..
  with invP show ⟨thesis ..
qed

lemma oinvariant_weakenE [elim!]:
  assumes invP: "A |= (S', U' →) P"
    and PQ: "⋀s. P s =⇒ Q s"
    and QSPS: "⋀s s' a. QS s s' a =⇒ PS s s' a"
    and QUPU: "⋀s s'. QU s s' =⇒ PU s s'"
  shows "A |= (QS, QU →) Q"
proof -
  fix s
  assume "s ∈ init A"
  with invP have "P s" ..
  thus "Q s" by (rule PQ)
  next
  fix σ p σ' p' a
  assume "((σ, p), a, (σ', p')) ∈ trans A"
  and "QS σ σ' a"
  from this(3) have "PS σ σ' a" by (rule QSPS)
  from "((σ, p), a, (σ', p')) ∈ trans A" and "PS σ σ' a" ..
  with invP have "P (σ', p')" ..
  thus "Q (σ', p')" by (rule PQ)
next
  fix σ σ' p
  assume "((σ, p), a, (σ', p')) ∈ trans A" 
  and "QU σ σ' a"
  from this(3) have "PU σ σ' a" by (rule QUPU)
  from "((σ, p), a, (σ', p')) ∈ trans A" and "PU σ σ' a" ..
  with invP have "P (σ', p')" ..
  thus "Q (σ', p')" by (rule PQ)
qed

lemma oinvariant_weakenD [dest]:
  assumes "A |= (S', U' →) P"
    and "(σ, p) ∈ oreachable A S U"
    and weakenS: "⋀s s' a. S s s' a =⇒ S' s s' a"
    and weakenU: "⋀s s'. U s s' =⇒ U' s s'"
  shows "P (σ, p)"
proof -
  from "(σ, p) ∈ oreachable A S U" have "(σ, p) ∈ oreachable A S' U'"
  by (rule oreachable_weakenE)
  (erule weakenS, erule weakenU)
  with "A |= (S', U' →) P'" show "P (σ, p)" ..
qed
lemma close_open_invariant:
  assumes oinv: "A = (act I, U →) P"
  shows "A ⊨ (I →) P"
proof
  fix s
  assume "s ∈ init A"
  with oinv show "P s" ..
next
  fix ξ p ξ' p' a
  assume sr: "((ξ, p), a, (ξ', p')) ∈ reachable A I"
  and step: "((ξ, p), a, (ξ', p')) ∈ trans A" 
  and "I a"
  hence "(ξ', p') ∈ reachable A I" ..
  hence "(ξ', p') ∈ reachable A (act I) U" ..
  with oinv show "P (ξ', p')" ..
  qed

definition local_steps :: "((('i ⇒ 's1) × 'l1) × 'a × ('i ⇒ 's2) × 'l2) set ⇒ 'i set ⇒ bool"
where
local_steps T J ≡ (∀ σ ζ s a σ' ζ' s'. ((σ, s), a, (σ', s')) ∈ T ∧ (∀ j∈J. ζ j = σ j) "→ (∃ ζ'. (∀ j∈J. ζ' j = σ' j) ∧ ((ζ, s), a, (ζ', s')) ∈ T))"

lemma local_stepsI [intro!]:
  assumes "∀ σ ζ s a σ' ζ' s'. ((σ, s), a, (σ', s')) ∈ T; (∀ j∈J. ζ j = σ j) "=⇒ (∃ ζ'. (∀ j∈J. ζ' j = σ' j) ∧ ((ζ, s), a, (ζ', s')) ∈ T)"
  shows "local_steps T J"
unfolding local_steps_def using assms by clarsimp

lemma local_stepsE [elim, dest]:
  assumes "local_steps T J" 
  and "((σ, s), a, (σ', s')) ∈ T" 
  and "∀ j∈J. ζ j = σ j" 
  shows "∃ ζ'. (∀ j∈J. ζ' j = σ' j) ∧ ((ζ, s), a, (ζ', s')) ∈ T" 
  using assms unfolding local_steps_def by blast

definition other_steps :: "((('i ⇒ 's) ⇒ ('i ⇒ 's)) ⇒ 'i set ⇒ bool) ⇒ 'i set ⇒ bool"
where
other_steps U J ≡ (∀ σ σ'. U σ σ' ; j ∈ J) =⇒ σ' j = σ j"

lemma other_stepsI [intro!]:
  assumes "∀ σ σ' j. [ U σ σ' ; j ∈ J ] =⇒ σ' j = σ j"
  shows "other_steps U J"
  using assms unfolding other_steps_def by simp

lemma other_stepsE [elim]:
  assumes "other_steps U J" 
  and "U σ σ'" 
  shows "∀ j∈J. σ' j = σ j" 
  using assms unfolding other_steps_def by simp

definition subreachable
where
subreachable A U J ≡ (∀ I. (∀ s ∈ reachable A (λs s'. I) U. 
  (∃ σ. (∀ j∈J. σ j = (fst s) j) ∧ (σ, snd s) ∈ reachable A I))"

lemma subreachableI [intro]:
  assumes "local_steps (trans A) J" 
  and "other_steps U J" 
  shows "subreachable A U J"
unfolding subreachable_def
proof (rule, rule)
  fix I s
  assume "s ∈ reachable A (λs s'. I) U" 
  thus "("∃ σ. (∀ j∈J. σ j = (fst s) j) ∧ (σ, snd s) ∈ reachable A I))" 
  proof induction

fix s
assume "s ∈ init A"
hence "(fst s, snd s) ∈ reachable A I"
  by simp (rule reachable_init)
moreover have "∀ j∈J. (fst s) j = (fst s) j"
  by simp
ultimately show "∃ σ. (∀ j∈J. σ j = (fst s) j) ∧ (σ, snd s) ∈ reachable A I"
  by auto
next
fix s a s'
assume "∃ σ. (∀ j∈J. σ j = (fst s) j) ∧ (σ, snd s) ∈ reachable A I"
and "(s, a, s') ∈ trans A"
and "I a" then obtain ζ where "∀ j∈J. ζ j = (fst s) j"
and "((ζ, snd s), a, (ζ', snd s)) ∈ trans A" by auto
from '((s, a, s') ∈ trans A' have "((fst s, snd s), a, (fst s', snd s')) ∈ trans A" by simp
with 'local_steps (trans A) J' obtain ζ' where "∀ j∈J. ζ' j = (fst s) j'" by -(drule(2) local_stepsE, clarsimp)
from '∀ j∈J. ζ j = (fst s) j' by -(erule(1) other_stepsE)
with 'other_steps U J' have "∀ j∈J. ζ j = (fst s) j'" by clarsimp
with '∀ j∈J. ζ j = (fst s) j' have "∀ j∈J. σ j = ζ j" by clarsimp
with "(ζ, snd s) ∈ reachable A I" show "∃ σ. (∀ j∈J. σ j = (fst s') j) ∧ (σ, snd s') ∈ reachable A I" by auto
qed

lemma subreachableE [elim]:
  assumes "subreachable A U J"
  and "s ∈ reachable A (λs s'. I) U"
  shows "∃ σ. (∀ j∈J. σ j = (fst s) j) ∧ (σ, snd s) ∈ reachable A I"
using assms unfolding subreachable_def by simp

lemma subreachableE_pair [elim]:
  assumes "subreachable A U J"
  and "(σ, s) ∈ reachable A (λs s'. I) U"
  shows "∃ ζ. (∀ j∈J. ζ j = σ j) ∧ (ζ, s) ∈ reachable A I"
using assms unfolding subreachable_def by (metis fst_conv snd_conv)

lemma subreachable_otherE [elim]:
  assumes "subreachable A U J"
  and "(σ, I) ∈ reachable A (λs s'. I) U"
  and "U σ σ'"
  shows "∃ ζ'. (∀ j∈J. ζ' j = σ' j) ∧ (ζ', I) ∈ reachable A I"
proof -
  from '((σ, I) ∈ reachable A (λs s'. I) U' and 'U σ σ' have "(σ', I) ∈ reachable A (λs s'. I) U"
lemma open_closed_invariant:
  fixes J
  assumes "A \models (I \rightarrow) P"
  and "subreachable A U J"
  and localp: "\forall \sigma \in J. \sigma j = \sigma j'"
  shows "A \models (act I, U \rightarrow) P"
proof (rule, simp_all only: act_def)
  fix s
  assume "s \in \init A"
  with 'A \models (I \rightarrow) P' show "P s" ..
next
  fix s a s'
  assume "s \in \oreachable A (\lambda_ _. I) U"
  and "P s"
  and "(s, a, s') \in \trans A"
  and "I a"
  hence "s' \in \oreachable A (\lambda_ _. I) U"
    by (metis \oreachable_local)
  with 'subreachable A U J' obtain \sigma'
    where "\forall j \in J. \sigma' j = (fst s') j"
    and "(\sigma', snd s') \in \reachable A I"
    by (metis \subreachableE)
  from 'A \models (I \rightarrow) P' and '(\sigma', snd s') \in \reachable A I' have "P (\sigma', snd s')" ..
  with "\forall j \in J. \sigma' j = (fst s') j'" show "P s'"
    by (metis \localp \pair_collapse)
next
  fix g g' l
  assume or: "(g, l) \in \oreachable A (\lambda s s'. I) U"
  and "U g g'"
  and "P (g, l)"
  from 'subreachable A U J' and or and 'U g g''
    obtain gg' where "\forall j \in J. gg' j = g' j"
    and "(gg', l) \in \reachable A I"
    by (auto dest!: \subreachable_otherE)
  from 'A \models (I \rightarrow) P' and '(gg', l) \in \reachable A I'
    have "P (gg', l)" ..
  with "\forall j \in J. gg' j = g' j'" show "P (g', l)"
    by (rule \localp)
qed

lemma invariant_anyact:
  assumes "A \models (act TT, U \rightarrow) P"
  shows "A \models (S, U \rightarrow) P"
using assms by (rule auto)

definition ostep_invariant
  :: "('g \times 'l, 'a) automaton
  ⇒ ('g \Rightarrow 'g, 'a) automaton
  ⇒ (\forall s \in \oreachable A S U. (\forall a s', (s, a, s') \in \trans A \land S (fst s) (fst s') a \rightarrow P (s, a, s')))"
where
  "(A \models_A (S, U \rightarrow) P) =
  (\forall s \in \oreachable A S U. (\forall a s', (s, a, s') \in \trans A \land S (fst s) (fst s') a \rightarrow P (s, a, s')))"

lemma ostep_invariant_def':
  "(A \models_A (S, U \rightarrow) P) = (\forall s \in \oreachable A S U.
    (\forall a s', (s, a, s') \in \trans A \land S (fst s) (fst s') a \rightarrow P (s, a, s')))"
unfolding ostep_invariant_def by auto
lemma ostep_invariantI [intro]:
assumes \( *: \forall \sigma s a \sigma' s'. \left[ (\sigma, s) \in \text{oreachable } A \ S \ U; ((\sigma, s), a, (\sigma', s')) \in \text{trans } A; S \ \sigma \ \sigma' \ a \right] \Rightarrow P ((\sigma, s), a, (\sigma', s')) \)
shows \( A \models_A (S, U \to) P \)
unfolding ostep_invariant_def using assms by auto

lemma ostep_invariantD [dest]:
assumes \( A \models_A (S, U \to) P \)
and \( ((\sigma, s), a, (\sigma', s')) \in \text{trans } A \)
and \( S \ \sigma \ \sigma' \ a \)
shows \( P ((\sigma, s), a, (\sigma', s')) \)
using assms unfolding ostep_invariant_def' by clarsimp

lemma ostep_invariantE [elim]:
assumes \( A \models_A (S, U \to) P \)
and \( ((\sigma, s), a, (\sigma', s')) \in \text{trans } A \)
and \( S \ \sigma \ \sigma' \ a \)
and \( P ((\sigma, s), a, (\sigma', s')) = \Rightarrow Q \)
shows \( Q \)
using assms by auto

lemma ostep_invariant_weakenE [elim!]:
assumes invP: \( A \models_A (PS, PU \to) P \)
and PQ: \( \forall t. P t = \Rightarrow Q t \)
and QSPS: \( \forall \sigma \ \sigma' a. QS \ \sigma \ \sigma' \ a = \Rightarrow PS \ \sigma \ \sigma' \ a \)
and QUPU: \( \forall \sigma \ \sigma'. QU \ \sigma \ \sigma' = \Rightarrow PU \ \sigma \ \sigma' \)
shows \( A \models_A (QS, QU \to) Q \)
proof
fix \( \sigma s a \sigma' s' \)
assume \( (\sigma, s) \in \text{oreachable } A \ QS \ QU \)
and \( ((\sigma, s), a, (\sigma', s')) \in \text{trans } A \)
and \( QS \ \sigma \ \sigma' \ a \)
from \( QS \ \sigma \ \sigma' \ a \) have \( "PS \ \sigma \ \sigma' \ a" \) by (rule QSPS)
from \( (\sigma, s) \in \text{oreachable } A \ QS \ QU \) have \( "(\sigma, s) \in \text{oreachable } A \ PS \ PU" \) using QSPS QUPU..
with invP have \( "P ((\sigma, s), a, (\sigma', s'))" \) using \( "((\sigma, s), a, (\sigma', s')) \in \text{trans } A" \) \( PS \ \sigma \ \sigma' a' \) ..
thus \( "Q ((\sigma, s), a, (\sigma', s'))" \) by (rule PQ)
qed

lemma ostep_invariant_weaken_with_invariantE [elim!]:
assumes pinv: \( A \models_A (S, U \to) P \)
and qinv: \( A \models_A (S, U \to) Q \)
and wr: \( \forall \sigma s a \sigma' s'. \left[ P (\sigma, s); P (\sigma', s'); Q ((\sigma, s), a, (\sigma', s')); S \ \sigma \ \sigma' \ a \right] \Rightarrow R ((\sigma, s), a, (\sigma', s')) \)
shows \( A \models_A (S, U \to) R \)
proof
fix \( \sigma s a \sigma' s' \)
assume sr: \( "(\sigma, s) \in \text{oreachable } A \ QS \ QU" \)
and tr: \( "((\sigma, s), a, (\sigma', s')) \in \text{trans } A" \)
and \( S \ \sigma \ \sigma' \ a \)
from \( "PS \ \sigma \ \sigma' \ a'" \) have \( "PS \ \sigma \ \sigma' \ a" \) by (rule QSPS)
from \( "QS \ \sigma \ \sigma' \ a" \) have \( "QS \ \sigma \ \sigma' \ a" \) by (rule QSPS)
from invP and sr have \( "P \ (\sigma, s)" \) ..
from qinv and sr have \( "Q ((\sigma, s), a, (\sigma', s'))" \) ..
with \( "P \ (\sigma, s)'" \) and \( "P \ (\sigma', s)'" \) show \( R ((\sigma, s), a, (\sigma', s'))" \) using \( S \ \sigma \ \sigma' \ a' \) by (rule wr)
qed

lemma ostep_to_invariantI:
assumes sinv: \( A \models_A (S, U \to) Q \)
and init: \( \forall \sigma s. (\sigma, s) \in \text{init } A \Rightarrow P (\sigma, s) \)
and local: \( \forall \sigma s a \sigma' s'. \left[ P (\sigma, s); P (\sigma', s'); Q ((\sigma, s), a, (\sigma', s')); S \ \sigma \ \sigma' \ a \right] \Rightarrow R ((\sigma, s), a, (\sigma', s')) \)
shows \( A \models_A (S, U \to) Q \)
proof
\[ (\sigma, s) \in \text{reachable A S U}; \]
\[ P(\sigma, s); \]
\[ Q((\sigma, s), a, (\sigma', s')); \]
\[ S \sigma \sigma' a \implies P(\sigma', s') \]

and other: "\( \forall \sigma \sigma' s. \ (\sigma, s) \in \text{reachable A S U}; \ U \sigma \sigma' P(\sigma, s) \implies P(\sigma', s) \)"

shows "\( A \models (S, U \rightarrow) P \)"

proof

fix \( \sigma s \) assume "\((\sigma, s) \in \text{init A}\)" thus "\( P(\sigma, s) \)" by (rule init)

next

fix \( \sigma s \sigma' s' a \)
assume "\((\sigma, s) \in \text{reachable A S U}\)

and "\( P(\sigma, s) \)"

and "\((\sigma, s), a, (\sigma', s') \) \in \text{trans A}\)

and "\( S \sigma \sigma' a \)"

show "\( P(\sigma', s') \)"

proof

- from sinv and '\((\sigma, s)\in\text{reachable A S U}' and '\((\sigma, s), a, (\sigma', s') \) \in \text{trans A}' and '\( S \sigma \sigma' a \)'

  with '\((\sigma, s)\in\text{reachable A S U}' and 'P(\sigma, s)' show "\( P(\sigma', s') \)"

  using '\( S \sigma \sigma' a \)' by (rule local)

qed

next

fix \( \sigma \sigma' l \)
assume "\((\sigma, l) \in \text{reachable A S U}\)

and "\( U \sigma \sigma' \)"

and "\( P(\sigma, l) \)"

thus "\( P(\sigma', l) \)" by (rule other)

qed

lemma open_closed_step_invariant:
assumes "\( A \models (I \rightarrow \rightarrow P) \)"

and "\( \text{local_steps (trans A) J} \)"

and "\( \text{other_steps U J} \)"

and localp: "\( \forall \sigma \zeta a \sigma' \zeta' s s'. \ [\forall j \in J. \ \sigma j = \zeta j; \forall j \in J. \ \sigma' j = \zeta' j; P((\sigma, s), a, (\sigma', s'))] \implies P((\zeta, s), a, (\zeta', s')) \)"

shows "\( A \models (\text{act I, U} \rightarrow \rightarrow P) \)"

proof

fix \( \sigma s a \sigma' s' \)
assume or: "\((\sigma, s) \in \text{reachable A (act I) U}\)

and tr: "\((\sigma, s), a, (\sigma', s') \) \in \text{trans A}\)

and "\( \text{act I} \sigma \sigma' a \)"

from 'act I \sigma \sigma' a' have "\( I a \)" ..

from 'local_steps (trans A) J' and 'other_steps U J' have "\( \text{subreachable A U J} \)" ..

then obtain \( \zeta \) where "\( \forall j \in J. \ \zeta j = \sigma j \)"

and "\( ((\zeta, s), a, (\zeta', s')) \in \text{trans A} \)"

using or unfolding act_def

by (auto dest!: subreachableE_pair)

from 'local_steps (trans A) J' and tr and '\( \forall j \in J. \ \zeta j = \sigma j \)'

obtain \( \zeta' \) where "\( \forall j \in J. \ \zeta' j = \sigma' j \)"

and "\( ((\zeta', s), a, (\zeta', s')) \in \text{trans A} \)"

by auto

from '\( A \models (I \rightarrow \rightarrow P)' and '\( ((\zeta, s), a, (\zeta', s')) \in \text{trans A}'

and '\( I a \)'

have "\( P((\zeta, s), a, (\zeta', s')) \)" ..

with '\( \forall j \in J. \ \zeta j = \sigma j \)' and '\( \forall j \in J. \ \zeta' j = \sigma' j \)'

show "\( P((\sigma, s), a, (\sigma', s')) \)"

by (rule localp)

qed

lemma oinvariant_step_anyact:
assumes "\( P \models (\text{act TT, U} \rightarrow) P \)"
shows \( p \models_A (S, U \rightarrow) p \)
using `assms` by rule auto

### 4.3 Standard assumption predicates

otherwith

**Definition**
otherwith :: 
\( (('s \Rightarrow 's) \Rightarrow 'a \Rightarrow bool) \Rightarrow (('i \Rightarrow 's) \Rightarrow ('i \Rightarrow 's) \Rightarrow 'a \Rightarrow bool) \) 
where \( \text{otherwith} \ Q \ I \ P \ \sigma \ \sigma' \ \alpha \equiv (\forall i. i \notin I \rightarrow Q (\sigma i) (\sigma' i)) \land P \ \sigma \ \alpha \)

**Lemma**
otherwithI [intro]:
assumes other: 
\( \forall j. j \notin I \Rightarrow Q (\sigma j) (\sigma' j) \)
and sync: 
\( P \ \sigma \ \alpha \)
shows 
\( \text{otherwith} \ Q \ I \ P \ \sigma \ \sigma' \ \alpha \)
unfolding otherwith_def using `assms` by simp

**Lemma**
otherwithE [elim]:
assumes "otherwith \ Q \ I \ P \ \sigma \ \sigma' \ \alpha"
and "\[ P \ \sigma \ \alpha ; \forall j. j \notin I \rightarrow Q (\sigma j) (\sigma' j) \] \Rightarrow R \ \sigma \ \sigma' \ \alpha"
shows "R \ \sigma \ \sigma' \ \alpha"
using `assms` unfolding otherwith_def by simp

**Lemma**
otherwith_actionD [dest]:
assumes "otherwith \ Q \ I \ P \ \sigma \ \sigma' \ \alpha"
shows "P \ \sigma \ \alpha"
using `assms` by auto

**Lemma**
otherwith_syncD [dest]:
assumes "otherwith \ Q \ I \ P \ \sigma \ \sigma' \ \alpha"
shows "\( \forall j. j \notin I \rightarrow Q (\sigma j) (\sigma' j) \)"
using auto

**Lemma**
otherwithEI [elim]:
assumes "otherwith \ P \ I \ P0 \ \sigma \ \sigma' \ \alpha"
and "\[ \sigma \ \alpha ; \forall j. j \notin I \rightarrow Q (\sigma j) (\sigma' j) \] \Rightarrow Q0 \ \sigma \ \sigma' \ \alpha"
shows "otherwith \ P \ I \ Q0 \ \sigma \ \sigma' \ \alpha"
using `assms(1)` unfolding otherwith_def by (clarsimp elim!: `assms(2)`) (clarsimp elim!: `assms(2)`)

**Lemma**
all_but:
assumes "\( \forall \xi. S \ \xi \ \xi \)"
and "\( \sigma' \ i = \ sigma \ i \)"
and "\( \forall j. j \neq i \rightarrow S (\sigma j) (\sigma' j) \)"
shows "\( \forall j. S (\sigma j) (\sigma' j) \)"
using `assms` by (clarsimp elim!: `assms(2)`)

**Lemma**
all_but_eq [dest]:
assumes "\( \sigma' \ i = \ sigma \ i \)"
and "\( \forall j. j \neq i \rightarrow \sigma j = \sigma' j \)"
shows "\( \sigma = \sigma' \)"
using `assms` by (clarsimp elim!: `assms(2)`)

**Other**

**Definition**
other :: 
\( (('s \Rightarrow 's) \Rightarrow bool) \Rightarrow (('i \Rightarrow 's) \Rightarrow ('i \Rightarrow 's) \Rightarrow bool) \) 
where \( other \ P \ I \ \sigma \ \sigma' \ \equiv \forall i. \text{if } i \in I \text{ then } \sigma' \ i = \sigma \ i \text{ else } P (\sigma i) (\sigma' i) \)

**Lemma**
otherI [intro]:
assumes local: "\( \forall i. i \in I \Rightarrow \sigma' \ i = \sigma \ i \)"
and other: "\( \forall j. j \notin I \Rightarrow P (\sigma j) (\sigma' j) \)"
shows "other \ P \ I \ \sigma \ \sigma'"
using `assms` unfolding other_def by clarsimp
lemma otherE [elim]:
assumes "other P I σ σ'"
and "[∀ i∈I. σ' i = σ i; ∀ j. j∉I → P (σ j) (σ' j) ] \implies R σ σ'"
shows "R σ σ'"
using assms unfolding other_def by simp

lemma other_localD [dest]:
"other P {i} σ σ' \implies σ' i = σ i"
by auto

lemma other_otherD [dest]:
"other P {i} σ σ' \implies ∀ j. j \neq i \rightarrow P (σ j) (σ' j)"
by auto

lemma other_bothE [elim]:
assumes "other P {i} σ σ'"
obtains "σ' i = σ i" and "∀ j. j \neq i \rightarrow P (σ j) (σ' j)"
using assms by auto

lemma weaken_local [elim]:
assumes "other P I σ σ'"
and PQ: "\lambdax ξ ξ'. P ξ ξ' \implies Q ξ ξ'"
shows "other Q I σ σ'"
using assms unfolding other_def by auto

definition global :: "((nat ⇒ 's) ⇒ bool) ⇒ (nat ⇒ 's) × 'local ⇒ bool"
where "global P ≡ (λ(σ, _). P σ)"

lemma globalsimp [simp]: "global P s = P (fst s)"
unfolding global_def by (simp split: split_split)

definition globala :: "((nat ⇒ 's, 'action) transition ⇒ bool) ⇒ ((nat ⇒ 's) × 'local, 'action) transition ⇒ bool"
where "globala P ≡ (\lambdax a, (σ', _). P (σ, a, σ'))"

lemma globalasimp [simp]: "globala P s = P (fst (fst s), fst (snd s), fst (snd (snd s)))"
unfolding globala_def by (simp split: split_split)

end

5 Terms of the Algebra for Wireless Networks

theory AWN
imports Lib
begin

5.1 Sequential Processes

type_synonym ip = nat
type_synonym data = nat

Most of AWN is independent of the type of messages, but the closed layer turns newpkt actions into the arrival
of newpkt messages. We use a type class to maintain some abstraction (and independence from the definition of
particular protocols).

class msg =
  fixes newpkt :: "data × ip ⇒ 'a"
  and eq_newpkt :: "'a ⇒ bool"
  assumes eq_newpkt_eq [simp]: "eq_newpkt (newpkt (d, i))"

Sequential process terms abstract over the types of data states ('s), messages ('m), process names ('p), and labels ('l).

datatype ('s, 'm, 'p, 'l) seqp =
GUARD "'l" "'s ⇒ 's set" "('s, 'm, 'p, 'l) seqp"
| ASSIGN "'l" "'s ⇒ 's" "('s, 'm, 'p, 'l) seqp"
| CHOICE "('s, 'm, 'p, 'l) seqp" "('s, 'm, 'p, 'l) seqp"
| UCAST "'l" "'s ⇒ 'm" "('s, 'm, 'p, 'l) seqp"
| BCAST "'l" "'s ⇒ 'm" "('s, 'm, 'p, 'l) seqp"
| GCAST "'l" "'s ⇒ set" "('s, 'm, 'p, 'l) seqp"
| SEND "'l" "'s ⇒ 'm" "('s, 'm, 'p, 'l) seqp"
| DELIVER "'l" "'s ⇒ data" "('s, 'm, 'p, 'l) seqp"
| RECEIVE "'l" "'m ⇒ 's ⇒ 's" "('s, 'm, 'p, 'l) seqp"
| CALL 'p

syntax

"_guard" :: ["'a, ("'s, 'm, 'p, unit) seqp"] ⇒ ("'s, 'm, 'p, unit) seqp"
| "(00(_))/\" [0, 60] 60)

"_lguard" :: ["'a, 'a, ("'s, 'm, 'p, unit) seqp"] ⇒ ("'s, 'm, 'p, unit) seqp"
| "({}_{(00(_))/\" [0, 60] 60)

"_ifguard" :: ["pttrn, bool, ("'s, 'm, 'p, unit) seqp"] ⇒ ("'s, 'm, 'p, unit) seqp"
| "(00(_))/\" [0, 0, 60] 60)

"_bassign" :: ["'a, ("'s, 'm, 'p, unit) seqp"] ⇒ ("'s, 'm, 'p, unit) seqp"
| "((00[_,]])/\" [0, 60] 60)

"_lassign" :: ["'a, 'a, ("'s, 'm, 'p, 'a) seqp"] ⇒ ("'s, 'm, 'p, 'a) seqp"
| "(_)/\" [0, 0, 60] 60)

"_unicast" :: ["'a, 'a, ("'s, 'm, 'p, unit) seqp", ("'s, 'm, 'p, unit) seqp"] ⇒ ("'s, 'm, 'p, unit) seqp"
| "((3unicast'(1(3_),/ (3_))') .//(_)/ (2\_))" [0, 0, 60, 60] 60)

"_lunicast" :: ["'a, 'a, 'a, ("'s, 'm, 'p, 'a) seqp", ("'s, 'm, 'p, 'a) seqp"] ⇒ ("'s, 'm, 'p, 'a) seqp"
| "((3_{_}unicast'(1(3_),/ (3_))') .//(_)/ (2\_))" [0, 0, 0, 60, 60] 60)

"_bcast" :: ["'a, ("'s, 'm, 'p, unit) seqp"] ⇒ ("'s, 'm, 'p, unit) seqp"
| "((3broadcast'(1(_))') .)//_" [0, 60] 60)

"_lbcast" :: ["'a, 'a, ("'s, 'm, 'p, 'a) seqp"] ⇒ ("'s, 'm, 'p, 'a) seqp"
| "((3_{_}broadcast'(1(_))') .)//_" [0, 0, 60] 60)

"_gcast" :: ["'a, 'a, ("'s, 'm, 'p, unit) seqp"] ⇒ ("'s, 'm, 'p, unit) seqp"
| "((3groupcast'(1(_),/ (_))') .//(_))/ (2\_))" [0, 0, 0, 60] 60)

"_lgcast" :: ["'a, 'a, 'a, ("'s, 'm, 'p, 'a) seqp"] ⇒ ("'s, 'm, 'p, 'a) seqp"
| "((3_{_}groupcast'(1(_),/ (_))') .//(_))/ (2\_))" [0, 0, 0, 0, 60] 60)

"_send" :: ["'a, ("'s, 'm, 'p, unit) seqp"] ⇒ ("'s, 'm, 'p, unit) seqp"
| "((3send'(1(_))') .)//_" [0, 60] 60)

"_lsend" :: ["'a, 'a, ("'s, 'm, 'p, 'a) seqp"] ⇒ ("'s, 'm, 'p, 'a) seqp"
| "((3_{_}send'(1(_))') .)//_" [0, 0, 60] 60)

"_deliver" :: ["'a, ("'s, 'm, 'p, unit) seqp"] ⇒ ("'s, 'm, 'p, unit) seqp"
| "((3deliver'(1(_))') .)//_" [0, 60] 60)

"_ldeliver" :: ["'a, 'a, ("'s, 'm, 'p, 'a) seqp"] ⇒ ("'s, 'm, 'p, 'a) seqp"
| "((3_{_}deliver'(1(_))') .)//_" [0, 0, 60] 60)

"_receive" :: ["'a, ("'s, 'm, 'p, unit) seqp"] ⇒ ("'s, 'm, 'p, unit) seqp"
| "((3receive'(1(_))') .)//_" [0, 60] 60)

"_lreceive" :: ["'a, 'a, ("'s, 'm, 'p, 'a) seqp"] ⇒ ("'s, 'm, 'p, 'a) seqp"
| "((3_{_}receive'(1(_))') .)//_" [0, 0, 60] 60)

translations

"_guard f p" ⇒ "CONST GUARD () f p"

"_lguard l f p" ⇒ "CONST GUARD l f p"

"_ifguard ξ e p" ⇒ "CONST GUARD () (λξ. if e then {} else {}) p"
_assign f p  \Rightarrow \text{"CONST ASSIGN () f p"}

_lassign l f p  \Rightarrow \text{"CONST ASSIGN l f p"}

_bassign e p  \Rightarrow \text{"CONST ASSIGN () (\lambda e. ) p"}

_lbassign l e p  \Rightarrow \text{"CONST ASSIGN l (\lambda e. ) p"

_unicast fip fmsg p q  \Rightarrow \text{"CONST UCAST () fip fmsg p q"}

_lunicast l fip fmsg p q  \Rightarrow \text{"CONST UCAST l fip fmsg p q"}

_bcast fmsg p  \Rightarrow \text{"CONST BCAST () fmsg p"}

_lbcast l fmsg p  \Rightarrow \text{"CONST BCAST l fmsg p"}

_gcast fmsg p  \Rightarrow \text{"CONST GCAST () fmsg p"}

_lgcast l fmsg p  \Rightarrow \text{"CONST GCAST l fmsg p"}

_send fmsg p  \Rightarrow \text{"CONST SEND () fmsg p"}

_lsend l fmsg p  \Rightarrow \text{"CONST SEND l fmsg p"}

_deliver fdata p  \Rightarrow \text{"CONST DELIVER () fdata p"}

_ldeliver l fdata p  \Rightarrow \text{"CONST DELIVER l fdata p"}

_receive fmsg p  \Rightarrow \text{"CONST RECEIVE () fmsg p"}

_lreceive l fmsg p  \Rightarrow \text{"CONST RECEIVE l fmsg p"}

notation "CHOICE" ("((_)//(\_)//(\_))") [56, 55] 55

and "CALL" ("(\text{3call'((3_)')")} [0] 60

definition not_call :: 
  "\text{('s, 'm, 'p, 'l) seqp} \Rightarrow \text{bool}"

where 
  "\text{not_call p} \equiv \forall pn. p \neq \text{call(pn)}"

lemma not_call_simps [simp]:

"\text{\lambda f p. not_call ({l}fg p)}"  
"\text{\lambda fa p. not_call ({l}fa p)}"  
"\text{\lambda p1 p2. not_call (p1 \oplus p2)}"  
"\text{\lambda fip fmsg p q. not_call ({l}unicast(fip, fmsg).p \triangleright q)}"  
"\text{\lambda fmsg p. not_call ({l}broadcast(fmsg).p)}"  
"\text{\lambda fips fmsg p. not_call ({l}groupcast(fips, fmsg).p)}"  
"\text{\lambda fmsg p. not_call ({l}send(fmsg).p)}"  
"\text{\lambda fdatal fdata p. not_call ({l}deliver(fdatal).p)}"  
"\text{\lambda fmsg p. not_call ({l}receive(fmsg).p)}"  
"\text{\lambda pn. \neg (\text{not_call (call(pn))})"  

unfolding not_call_def by auto

definition not_choice :: "\text{('s, 'm, 'p, 'l) seqp} \Rightarrow \text{bool}"

where 
  "\text{not_choice p} \equiv \forall p1 p2. p \neq p1 \oplus p2"

lemma not_choice_simps [simp]:

"\text{\lambda f p. not_choice ({l}fg p)}"  
"\text{\lambda fa p. not_choice ({l}fa p)}"  
"\text{\lambda p1 p2. \neg (not_choice (p1 \oplus p2))}"  
"\text{\lambda fip fmsg p q. not_choice ({l}unicast(fip, fmsg).p \triangleright q)}"  
"\text{\lambda fmsg p. not_choice ({l}broadcast(fmsg).p)}"  
"\text{\lambda fips fmsg p. not_choice ({l}groupcast(fips, fmsg).p)}"  
"\text{\lambda fmsg p. not_choice ({l}send(fmsg).p)}"  
"\text{\lambda fdatal fdata p. not_choice ({l}deliver(fdatal).p)}"  
"\text{\lambda fmsg p. not_choice ({l}receive(fmsg).p)}"  
"\text{\lambda pn. not_choice (call(pn))}"  

unfolding not_choice_def by auto

lemma seqp_congs:

"\text{\lambda f p. {l}fg p = {l}fg p}"  
"\text{\lambda fa p. {l}fa p = {l}fa p}"  
"\text{\lambda p1 p2. p1 \oplus p2 = p1 \oplus p2}"  
"\text{\lambda fip fmsg p q. {l}unicast(fip, fmsg).p \triangleright q = {l}unicast(fip, fmsg).p \triangleright q}"

21
"\( \forall fmsg. \{1\}broadcast(fmsg).p = \{1\}broadcast(fmsg).p \)"

"\( \forall fips fmsg p. \{1\}groupcast(fips, fmsg).p = \{1\}groupcast(fips, fmsg).p \)"

"\( \forall fmsg p. \{1\}send(fmsg).p = \{1\}send(fmsg).p \)"

"\( \forall fdata p. \{1\}deliver(fdata).p = \{1\}deliver(fdata).p \)"

"\( \forall fmsg p. \{1\}receive(fmsg).p = \{1\}receive(fmsg).p \)"

"\( \forall pn. \text{call}(pn) = \text{call}(pn) \)"

by auto

Remove data expressions from process terms.

fun seqp_skeleton :: "('s, 'm, 'p, 'l) seqp \Rightarrow (unit, unit, 'p, 'l) seqp"
where
  "seqp_skeleton (\{\}_p) = \{\}_p"
| "seqp_skeleton (\[\]_p) = \[\]_p" 
| "seqp_skeleton (p \oplus q) = (seqp_skeleton p) \oplus (seqp_skeleton q)"
| "seqp_skeleton (\{\}_unicast(_, _). p \lhd q) = \{\}_unicast(_, _). (seqp_skeleton p) \lhd (seqp_skeleton q)"
| "seqp_skeleton (\{\}_broadcast(_). p) = \{\}_broadcast(_). (seqp_skeleton p)"
| "seqp_skeleton (\{\}_groupcast(_, _). p) = \{\}_groupcast(_, _). (seqp_skeleton p)"
| "seqp_skeleton (\{\}_send(_). p) = \{\}_send(_). (seqp_skeleton p)"
| "seqp_skeleton (\{\}_deliver(_). p) = \{\}_deliver(_). (seqp_skeleton p)"
| "seqp_skeleton (\{\}_receive(_). p) = \{\}_receive(_). (seqp_skeleton p)"
| "seqp_skeleton (\text{call}(pn)) = \text{call}(pn)"

Calculate the subterms of a term.

fun subterms :: "('s, 'm, 'p, 'l) seqp \Rightarrow ('s, 'm, 'p, 'l) seqp set"
where
  "subterms (\{\}_fg_p) = \{\}_fg_p \cup \text{subterms } p"
| "subterms (\[\]_fa_p) = \[\]_fa_p \cup \text{subterms } p" 
| "subterms (p1 \oplus p2) = \{p1 \oplus p2\} \cup \text{subterms } p1 \cup \text{subterms } p2"
| "subterms (\{\}_unicast(fip, fmsg). p \lhd q) = \{\}_unicast(fip, fmsg). p \lhd q \cup \text{subterms } p \cup \text{subterms } q"
| "subterms (\{\}_broadcast(fmsg). p) = \{\}_broadcast(fmsg). p \cup \text{subterms } p"
| "subterms (\{\}_groupcast(fips, fmsg). p) = \{\}_groupcast(fips, fmsg). p \cup \text{subterms } p"
| "subterms (\{\}_send(fmsg). p) = \{\}_send(fmsg). p \cup \text{subterms } p"
| "subterms (\{\}_deliver(fdata). p) = \{\}_deliver(fdata). p \cup \text{subterms } p"
| "subterms (\{\}_receive(fmsg). p) = \{\}_receive(fmsg). p \cup \text{subterms } p"
| "subterms (\text{call}(pn)) = \{\}_call(pn)"

lemma subterms_refl [simp]: "p \in \text{subterms } p" 
by (cases p) simp_all

lemma subterms_trans [elim]:
  assumes "q \in \text{subterms } p"
  and "r \in \text{subterms } q"
  shows "r \in \text{subterms } p"
using assms by (induction p) auto

lemma root_in_subterms [simp]:
  "\( \forall \Gamma pn. \exists \Gamma pn'. \Gamma pn \in \text{subterms } (\Gamma pn') \)"
by (rule_tac x=\Gamma pn in exI) simp

lemma deriv_in_subterms [elim, dest]:
  "\( \forall f p q. \{1\}f p q \in \text{subterms } p \Longrightarrow q \in \text{subterms } p \)"
| "\( \forall fa p q. \{1\}fa p q \in \text{subterms } p \Longrightarrow q \in \text{subterms } p \)"
| "\( \forall p1 p2 q. p1 \oplus p2 \in \text{subterms } p \Longrightarrow p1 \in \text{subterms } p \)"
| "\( \forall p1 p2 q. p1 \oplus p2 \in \text{subterms } p \Longrightarrow p2 \in \text{subterms } p \)"
| "\( \forall fmsg q r. \{1\}_unicast(fip, fmsg). q \lhd r \in \text{subterms } p \Longrightarrow q \in \text{subterms } p \)"
| "\( \forall fmsg q r. \{1\}_unicast(fip, fmsg). q \lhd r \in \text{subterms } p \Longrightarrow r \in \text{subterms } p \)"
| "\( \forall fmsg q p. \{1\}_broadcast(fmsg). q \in \text{subterms } p \Longrightarrow q \in \text{subterms } p \)"
| "\( \forall fmsg q p. \{1\}_groupcast(fips, fmsg). q \in \text{subterms } p \Longrightarrow q \in \text{subterms } p \)"
| "\( \forall fmsg q p. \{1\}_send(fmsg). q \in \text{subterms } p \Longrightarrow q \in \text{subterms } p \)"
| "\( \forall fdata q p. \{1\}_deliver(fdata). q \in \text{subterms } p \Longrightarrow q \in \text{subterms } p \)"
| "\( \forall fmsg q p. \{1\}_receive(fmsg). q \in \text{subterms } p \Longrightarrow q \in \text{subterms } p \)"
by auto
5.2 Actions

There are two sorts of $\tau$ actions in AWN: one at the level of individual processes (within nodes), and one at the network level (outside nodes). We define a class so that we can ignore this distinction whenever it is not critical.

```plaintext
class tau =  
  fixes tau :: "'a" ("$\tau$")
```

5.2.1 Sequential Actions (and related predicates)

```plaintext
datatype 'm seq_action =  
  broadcast 'm  
  | groupcast "ip set" 'm  
  | unicast ip 'm  
  | notunicast ip ("$\neg$unicast _" [1000] 60)  
  | send 'm  
  | receive 'm  
  | seq_tau ("$\tau_s$")
```

instantiation "seq_action" :: (type) tau
begin

definition step_seq_tau [simp]: "$\tau \equiv \tau_s$"
instance ..
end

definition recvmsg :: "('m ⇒ bool) ⇒ 'm seq_action ⇒ bool"
where "recvmsg P a ≡ case a of receive m ⇒ P m  
    | _ ⇒ True"

lemma recvmsg_simps [simp]:
  "\(\forall m.\) recvmsg P (broadcast m) = True"  
  "\(\forall ips m.\) recvmsg P (groupcast ips m) = True"  
  "\(\forall ip m.\) recvmsg P (unicast ip m) = True"  
  "\(\forall ip.\) recvmsg P (notunicast ip) = True"  
  "\(\forall m.\) recvmsg P (send m) = True"  
  "\(\forall d.\) recvmsg P (deliver d) = True"  
  "\(\forall m.\) recvmsg P (receive m) = P m"  
  "recvmsg P $\tau_s$ = True"

unfolding recvmsg_def by simp_all

lemma recvmsgTT [simp]: "recvmsg TT a"
by (cases a) simp_all

lemma recvmsgE [elim]:
  assumes "recvmsg (R $\sigma$) a"  
  and "\(\forall m.\) R $\sigma$ m ⇒ R $\sigma'$ m"  
  shows "recvmsg (R $\sigma'$) a"
using assms(1) by (cases a) (auto elim!: assms(2))

definition anycast :: "('m ⇒ bool) ⇒ 'm seq_action ⇒ bool"
where "anycast P a ≡ case a of broadcast m ⇒ P m  
    | groupcast _ m ⇒ P m  
    | unicast _ m ⇒ P m  
    | _ ⇒ True"

lemma anycast_simps [simp]:
  "\(\forall m.\) anycast P (broadcast m) = P m"  
  "\(\forall ips m.\) anycast P (groupcast ips m) = P m"  
  "\(\forall ip m.\) anycast P (unicast ip m) = P m"  
  "\(\forall ip.\) anycast P (notunicast ip) = True"  
  "\(\forall m.\) anycast P (send m) = True"  
  "\(\forall d.\) anycast P (deliver d) = True"  
  "\(\forall m.\) anycast P (receive m) = True"  
  "anycast P $\tau_s$ = True"
```
unfolding anycast_def by simp_all

definition orcvmsg :: "((ip ⇒ 's) ⇒ 'm ⇒ bool) ⇒ (ip ⇒ 's) ⇒ 'm seq_action ⇒ bool"
where "orecvmsg P σ a ≡ (case a of receive m ⇒ P σ m
          | _ ⇒ True)"

lemma orcvmsg_simps [simp]:
  "∀ m. orcvmsg P σ (broadcast m) = True"
"∀ ips m. orcvmsg P σ (groupcast ips m) = True"
"∀ ip m. orcvmsg P σ (unicast ip m) = True"
"∀ ip. orcvmsg P σ (notunicast ip) = True"
"∀ m. orcvmsg P σ (send m) = True"
"∀ d. orcvmsg P σ (deliver d) = True"
"∀ m. orcvmsg P σ (receive m) = P σ m"
" orcvmsg P σ τ_s = True"

unfolding orcvmsg_def by simp_all

lemma orcvmsgEI [elim]:
  "[ orcvmsg P σ a; \A σ a. P σ a ⇒ Q σ a ] \⇒ orcvmsg Q σ a"
by (cases a) simp_all

lemma orcvmsg_stateless_recvmsg [elim]:
  "orecvmsg (λ _. P) σ a =⇒ recvmsg P a"
by (cases a) simp_all

lemma orcvmsg_recv_weaken [elim]:
  "[ orcvmsg P σ a; \A σ a. P σ a ⇒ Q a ] \⇒ recvmsg Q a"
by (cases a) simp_all

definition sendmsg :: "('m ⇒ bool) ⇒ 'm seq_action ⇒ bool"
where "sendmsg P a ≡ (case a of send m ⇒ P m | _ ⇒ True)"

lemma sendmsg_simps [simp]:
  "∀ m. sendmsg P (broadcast m) = True"
"∀ ips m. sendmsg P (groupcast ips m) = True"
"∀ ip m. sendmsg P (unicast ip m) = True"
"∀ ip. sendmsg P (notunicast ip) = True"
"∀ m. sendmsg P (send m) = P m"
"∀ d. sendmsg P (deliver d) = True"
"∀ m. sendmsg P (receive m) = True"
" sendmsg P τ_s = True"

unfolding sendmsg_def by simp_all

unfolding step_node_tau by (simp_all)

instantiation "node_action" :: (type) tau
begin

definition step_node_tau [simp]: "τ ≡ τ_n"

instance ..
definition arrivemsg :: "ip ⇒ 'm node_action ⇒ bool" where "arrivemsg i P a ≡ case a of node_arrive ii ni m ⇒ ((ii = {i} → P m)) \_ ⇒ True"

lemma arrivemsg_simps[simp]:
"\R m. arrivemsg i P (R:*cast(m)) = True"
"\d m. arrivemsg i P (d:deliver(m)) = True"
"\i ii ni m. arrivemsg i P (ii:ni:arrive(m)) = (ii = {i} → P m)"
"\i i2. arrivemsg i P (connect(i1, i2)) = True"
"\i i2. arrivemsg i P (disconnect(i1, i2)) = True"
"\i i' d di. arrivemsg i P (i':newpkt(d, di)) = True"
" arrivemsg i P τn = True"

unfolding arrivemsg_def by simp_all

lemma arrivemsgTT [simp]: "arrivemsg i TT = TT"
by (rule ext) (clarsimp simp: arrivemsg_def split: node_action.split)

definition oarrivemsg :: "((ip ⇒ 's) ⇒ 'm ⇒ bool) ⇒ (ip ⇒ 's) ⇒ 'm node_action ⇒ bool"
where "oarrivemsg P σ a ≡ case a of node_arrive ii ni m ⇒ P σ (\_ ⇒ True"

lemma oarrivemsg_simps[simp]:
"\R m. oarrivemsg P σ (R:*cast(m)) = True"
"\d m. oarrivemsg P σ (d:deliver(m)) = True"
"\i ii ni m. oarrivemsg P σ (ii:ni:arrive(m)) = P σ m"
"\i i2. oarrivemsg P σ (connect(i1, i2)) = True"
"\i i2. oarrivemsg P σ (disconnect(i1, i2)) = True"
"\i i' d di. oarrivemsg P σ (i':newpkt(d, di)) = True"
" oarrivemsg P σ τn = True"

unfolding oarrivemsg_def by simp_all

lemma oarrivemsg_True [simp, intro]: "oarrivemsg (λ _ _. True) σ a"
by (cases a) auto

definition castmsg :: "('m ⇒ bool) ⇒ 'm node_action ⇒ bool"
where "castmsg P a ≡ case a of _:*cast(m) ⇒ P m
\_ ⇒ True"

lemma castmsg_simps[simp]:
"\R m. castmsg P (R:*cast(m)) = P m"
"\d m. castmsg P (d:deliver(m)) = True"
"\i ii ni m. castmsg P (ii:ni:arrive(m)) = True"
"\i i2. castmsg P (connect(i1, i2)) = True"
"\i i2. castmsg P (disconnect(i1, i2)) = True"
"\i i' d di. castmsg P (i':newpkt(d, di)) = True"
" castmsg P τn = True"

unfolding castmsg_def by simp_all

5.3 Networks

datatype net_tree =
  Node ip "ip set" "(\_; \_;)"
| Subnet net_tree net_tree (infixl "∥" 90)

declare net_tree.induct [[induct del]]
lemmas net_tree_induct [induct type: net_tree] = net_tree.induct [rename_abs i R p1 p2]

datatype 's net_state =
  NodeS ip 's "ip set"
| SubnetS 's net_state "'s net_state"

fun net_ips :: "'s net_state ⇒ ip set"
where
"net_ips (NodeS i R) = {i}"
/ "net_ips (SubnetS n1 n2) = net_ips n1 ∪ net_ips n2"

fun net_tree_ips :: "net_tree ⇒ ip set"
where
"net_tree_ips (p1 || p2) = net_tree_ips p1 ∪ net_tree_ips p2"
/ "net_tree_ips ((i; R)) = {i}"

lemma net_tree_ips_commute:
"net_tree_ips (p1 || p2) = net_tree_ips (p2 || p1)"
by simp (rule Un_commute)

fun wf_net_tree :: "net_tree ⇒ bool"
where
"wf_net_tree (p1 || p2) = (net_tree_ips p1 ∩ net_tree_ips p2 = {}
∧ wf_net_tree p1 ∧ wf_net_tree p2)"
/ "wf_net_tree ((i; R)) = True"

lemma wf_net_tree_children [elim]:
assumes "wf_net_tree (p1 || p2)"
obtains "wf_net_tree p1"
and "wf_net_tree p2"
using assms by simp

fun netmap :: "'s net_state ⇒ ip ⇒ 's option"
where
"netmap (NodeS i p R i) = [i ↦→ p]"
/ "netmap (SubnetS s t) = netmap s ++ netmap t"

lemma not_in_netmap [simp]:
assumes "i /∈ net_ips ns"
shows "netmap ns i = None"
using assms by (induction ns) simp_all

lemma netmap_none_not_in_net_ips:
assumes "netmap ns i = None"
shows "i /∈ net_ips ns"
using assms by (auto simp add: net_ips_is_dom_netmap)

lemma net_ips_is_dom_netmap: "net_ips s = dom(netmap s)"
proof (induction s)
fix i R, and p :: 's
show "net_ips (NodeS i p R i) = dom (netmap (NodeS i p R i))"
by auto
next
fix s1 s2 :: 's net_state
assume "net_ips s1 = dom (netmap s1)"
and "net_ips s2 = dom (netmap s2)"
thus "net_ips (SubnetS s1 s2) = dom (netmap (SubnetS s1 s2))"
by auto
qed

lemma in_netmap [simp]:
assumes "i ∈ net_ips ns"
shows "netmap ns i ≠ None"
using assms by (auto simp add: net_ips_is_dom_netmap)

lemma netmap_subnets_same:
assumes "netmap s1 i = x"
and "netmap s2 i = x"
shows "netmap (SubnetS s1 s2) i = x"
using assms by simp (metis map_add_dom_app_simps(1) map_add_dom_app_simps(3))

lemma netmap_subnets_samef:
assumes "netmap s1 = f" and "netmap s2 = f"
shows "netmap (SubnetS s1 s2) = f"
using assms by simp (metis map_add_le_mapI map_le_antisym map_le_map_add map_le_refl)

lemma netmap_add_disjoint [elim]:
assumes "\forall i \in net_ips s1 \cup net_ips s2. the ((netmap s1 ++ netmap s2) i) = \sigma i"
and "net_ips s1 \cap net_ips s2 = {}"
shows "\forall i \in net_ips s1. the (netmap s1 i) = \sigma i"
proof
  fix i
  assume "i \in net_ips s1"
  hence "i \in dom(netmap s1)" by (simp add: net_ips_is_dom_netmap)
  moreover with assms(2)
  have "i /\in dom(netmap s2)" by (auto simp add: net_ips_is_dom_netmap)
  ultimately have "\sigma (netmap s1 i) = \sigma ((netmap s1 ++ netmap s2) i)"
  by (simp add: map_add_dom_app_simps)
  with assms(1) and 'i \in net_ips s1' show "\sigma (netmap s1 i) = \sigma i" by simp
qed

lemma netmap_add_disjoint2 [elim]:
assumes "\forall i \in net_ips s1 \cup net_ips s2. the ((netmap s1 ++ netmap s2) i) = \sigma i"
shows "\forall i \in net_ips s2. the (netmap s2 i) = \sigma i"
using assms by (simp add: net_ips_is_dom_netmap)
  (metis Un_iff map_add_dom_app_simps(1))

lemma net_ips_netmap_subnet [elim]:
assumes "net_ips s1 \cap net_ips s2 = {}" and "\forall i \in net_ips (SubnetS s1 s2). the (netmap (SubnetS s1 s2) i) = \sigma i"
shows "\forall i \in net_ips s1. the (netmap s1 i) = \sigma i" and "\forall i \in net_ips s2. the (netmap s2 i) = \sigma i"
proof
  from assms(2)
  have "\forall i \in net_ips s1 \cup net_ips s2. the ((netmap s1 ++ netmap s2) i) = \sigma i" by auto
  with assms(1) show "\forall i \in net_ips s1. the (netmap s1 i) = \sigma i" by - (erule(1) netmap_add_disjoint)
next
  from assms(2)
  have "\forall i \in net_ips s1 \cup net_ips s2. the ((netmap s1 ++ netmap s2) i) = \sigma i" by auto
  thus "\forall i \in net_ips s2. the (netmap s2 i) = \sigma i" by - (erule netmap_add_disjoint2)
qed

fun inoclosed :: "'s \Rightarrow 'm::msg node_action \Rightarrow bool"
where
  "inoclosed _ (node_arrive ii ni m) = eq_newpkt m"
| "inoclosed _ (node_newpkt i d di) = False"
| "inoclosed _ _ = True"

lemma inoclosed_simps [simp]:
"\forall \sigma \ i d. inoclosed \sigma (i:deliver(d)) = True"
"\forall \sigma \ i i'. inoclosed \sigma (connect(i, i')) = True"
"\forall \sigma. inoclosed \sigma (\tau) = True"
by auto

definition netmask :: "ip set \Rightarrow ((ip \Rightarrow 's) \times 'l) \Rightarrow ((ip \Rightarrow 's option) \times 'l)"
where
  "netmask I s = \lambda i. if i\in I then Some (fst s i) else None, snd s"

lemma netmask_def'[simp]:
  "netmask I (\sigma, \zeta) = (\lambda i. if i\in I then Some (\sigma i) else None, \zeta)"
unfolding netmask_def by auto
fun netgmap :: "('s ⇒ 'g × 'l) ⇒ 's net_state ⇒ (nat ⇒ 'g option) × 'l net_state"

where

"netgmap sr (NodeS i s R) = ([(i ⇒ fst (sr s)), NodeS i (snd (sr s))]) R"

| "netgmap sr (SubnetS s1 s2) = (let (σ1, ss) = netgmap sr s1 in
  let (σ2, tt) = netgmap sr s2 in
    (σ1 ++ σ2, SubnetS ss tt))"

lemma dom_fst_netgmap [simp, intro]: "dom (fst (netgmap sr n)) = net_ips n"
using assms proof (induction n)
fix i s R
show "dom (fst (netgmap sr (NodeS i s R))) = net_ips (NodeS i s R)"
by simp
next
fix n1 n2
assume a1: "dom (fst (netgmap sr n1)) = net_ips n1"
and a2: "dom (fst (netgmap sr n2)) = net_ips n2"
obtain σ1 ζ1 σ2 ζ2 where nm1: "netgmap sr n1 = (σ1, ζ1)"
and nm2: "netgmap sr n2 = (σ2, ζ2)"
by (metis surj_pair)
hence "netgmap sr (SubnetS n1 n2) = (σ1 ++ σ2, SubnetS ζ1 ζ2)" by simp
hence "dom (fst (netgmap sr (SubnetS n1 n2))) = dom (σ1 ++ σ2)" by simp
also from a1 a2 nm1 nm2 have "dom (σ1 ++ σ2) = net_ips (SubnetS n1 n2)" by auto
finally show "dom (fst (netgmap sr (SubnetS n1 n2))) = net_ips (SubnetS n1 n2)" .
qed

lemma netgmap_pair_dom [elim]:
obtains σ ζ where "netgmap sr n = (σ, ζ)"
and "dom σ = net_ips n"
by (metis dom_fst_netgmap surjective_pairing)

lemma net_ips_netgmap [simp]:
"net_ips (snd (netgmap sr s)) = net_ips s"
proof (induction s)
fix s1 s2
assume "net_ips (snd (netgmap sr s1)) = net_ips s1"
and "net_ips (snd (netgmap sr s2)) = net_ips s2"
thus "net_ips (snd (netgmap sr (SubnetS s1 s2))) = net_ips (SubnetS s1 s2)"
by (cases "netgmap sr s1", cases "netgmap sr s2") auto
qed simp

lemma some_the_fst_netgmap:
assumes "i ∈ net_ips s"
shows "Some (the (fst (netgmap sr s) i)) = fst (netgmap sr s) i"
using assms by (metis domIff dom_fst_netgmap option.collapse)

lemma fst_netgmap_none [simp]:
assumes "i ∈ net_ips s"
shows "fst (netgmap sr s) i = None"
using assms by (metis domIff dom_fst_netgmap)

lemma fst_netgmap_subnet [simp]:
"fst (case netgmap sr s1 of (σ1, ss) ⇒
  case netgmap sr s2 of (σ2, tt) ⇒
    (σ1 ++ σ2, SubnetS ss tt)) = (fst (netgmap sr s1) ++ fst (netgmap sr s2))"
by (metis (mono_tags) fst_conv netgmap_pair_dom split_conv)

lemma snd_netgmap_subnet [simp]:
"snd (case netgmap sr s1 of (σ1, ss) ⇒
  case netgmap sr s2 of (σ2, tt) ⇒
    (σ1 ++ σ2, SubnetS ss tt)) = (SubnetS (snd (netgmap sr s1)) (snd (netgmap sr s2)))"
by (metis (lifting, no_types) Pair_inject split_beta' surjective_pairing)
lemma fst_netgmap_not_none [simp]:
  assumes "i ∈ net_ips s"
  shows "fst (netgmap sr s) i ≠ None"
  using assms by (induction s) auto

lemma netgmap_netgmap_not_rhs [simp]:
  assumes "i ∉ net_ips s2"
  shows "(fst (netgmap sr s1) ++ fst (netgmap sr s2)) i = (fst (netgmap sr s1)) i"
  proof -
    from assms(1) have "i ∉ dom (fst (netgmap sr s2))" by simp
    thus ?thesis by (simp add: map_add_dom_app_simps)
  qed

lemma netgmap_netgmap_rhs [simp]:
  assumes "i ∈ net_ips s2"
  shows "(fst (netgmap sr s1) ++ fst (netgmap sr s2)) i = (fst (netgmap sr s2)) i"
  using assms by (simp add: map_add_dom_app_simps)

lemma netgmap_netmask_subnets [elim]:
  assumes "netgmap sr s1 = netmask (net_tree_ips n1) (σ, snd (netgmap sr s1))"
  and "netgmap sr s2 = netmask (net_tree_ips n2) (σ, snd (netgmap sr s2))"
  shows "fst (netgmap sr (SubnetS s1 s2)) = fst (netmask (net_tree_ips (n1 || n2)) (σ, snd (netgmap sr (SubnetS s1 s2))))"
  proof (rule ext)
    fix i
    have "i ∈ net_tree_ips n1 ∨ i ∈ net_tree_ips n2 ∨ (i ∉ net_tree_ips n1 ∪ net_tree_ips n2)"
      by auto
    thus "fst (netgmap sr (SubnetS s1 s2)) i = fst (netmask (net_tree_ips (n1 || n2)) (σ, snd (netgmap sr (SubnetS s1 s2)))) i"
      proof (elim disjE)
        assume "i ∈ net_tree_ips n1"
        with 'netgmap sr s1 = netmask (net_tree_ips n1) (σ, snd (netgmap sr s1))'
        'netgmap sr s2 = netmask (net_tree_ips n2) (σ, snd (netgmap sr s2))'
        show ?thesis
          by (cases "netgmap sr s1", cases "netgmap sr s2", clarsimp)
            (metis (lifting, mono_tags) map_add_Some_iff)
        next
        assume "i ∈ net_tree_ips n2"
        with 'netgmap sr s1 = netmask (net_tree_ips n1) (σ, snd (netgmap sr s1))'
        'netgmap sr s2 = netmask (net_tree_ips n2) (σ, snd (netgmap sr s2))'
        show ?thesis
          by simp (metis (lifting, mono_tags) fst_conv map_add_find_right)
        next
        assume "i ∉ net_tree_ips n1 ∪ net_tree_ips n2"
        with 'netgmap sr s1 = netmask (net_tree_ips n1) (σ, snd (netgmap sr s1))'
        'netgmap sr s2 = netmask (net_tree_ips n2) (σ, snd (netgmap sr s2))'
        show ?thesis
          by simp (metis (lifting, mono_tags) fst_conv)
      qed
  qed

lemma netgmap_netmask_subnets' [elim]:
  assumes "netgmap sr s1 = netmask (net_tree_ips n1) (σ, snd (netgmap sr s1))"
  and "netgmap sr s2 = netmask (net_tree_ips n2) (σ, snd (netgmap sr s2))"
  and "s = SubnetS s1 s2"
  shows "netgmap sr s = netmask (net_tree_ips (n1 || n2)) (σ, snd (netgmap sr s))"
  by (simp only: assms(3))
    (rule prod_eqI [OF netgmap_netmask_subnets [OF assms(1-2)]], simp)

lemma netgmap_subnet_split1:
  assumes "netgmap sr (SubnetS s1 s2) = netmask (net_tree_ips (n1 || n2)) (σ, ζ)"
  and "net_tree_ips n1 ∩ net_tree_ips n2 = {}"
  and "net_ips s1 = net_tree_ips n1"
  and "net_ips s2 = net_tree_ips n2"
  shows "netgmap sr s1 = netmask (net_tree_ips n1) (σ, snd (netgmap sr s1))"
proof (rule prod_eqI)
  show "fst (netmap sr s1) = fst (netmask (net_tree_ips n1) (σ, snd (netmap sr s1)))"
  proof (rule ext, simp, intro conjI impI)
    fix i
    assume "i ∈ net_tree_ips n1"
    with 'net_tree_ips n1 ∩ net_tree_ips n2 = {}' have "i ∉ net_tree_ips n2"
      by auto
    from assms(1) [simplified prod_eq_iff]
    have "(fst (netmap sr s1) ++ fst (netmap sr s2)) i ="
      "(if i ∈ net_tree_ips n1 ∨ i ∈ net_tree_ips n2 then Some (σ i) else None)"
      by simp
    also from 'i ∉ net_tree_ips n2' and 'net_ips s2 = net_tree_ips n2'
    have "(fst (netmap sr s1) ++ fst (netmap sr s2)) i = fst (netmap sr s1) i"
      by (metis dom_fst_netgmap map_add_dom_app_simps(3))
    finally show "fst (netmap sr s1) i = Some (σ i)"
      using 'i ∈ net_tree_ips n1' by simp
  next
    fix i
    assume "i ∉ net_tree_ips n1"
    with 'net_ips s1 = net_tree_ips n1' have "i ∉ net_ips s1" by simp
    thus "fst (netmap sr s1) i = None" by simp
  qed
qed simp

lemma netgmap_subnet_split2:
  assumes "netgmap sr (SubnetS s1 s2) = netmask (net_tree_ips (n1 || n2)) (σ, ζ)"
  and "net_ips s1 = net_tree_ips n1"
  and "net_ips s2 = net_tree_ips n2"
  shows "netgmap sr s2 = netmask (net_tree_ips n2) (σ, snd (netmap sr s2))"
proof (rule prod_eqI)
  show "fst (netgmap sr s2) = fst (netmask (net_tree_ips n2) (σ, snd (netmap sr s2)))"
  proof (rule ext, simp, intro conjI impI)
    fix i
    assume "i ∉ net_tree_ips n2"
    from assms(1) [simplified prod_eq_iff]
    have "(fst (netmap sr s1) ++ fst (netmap sr s2)) i ="
      "(if i ∈ net_tree_ips n1 ∨ i ∈ net_tree_ips n2 then Some (σ i) else None)"
      by simp
    also from 'i ∉ net_tree_ips n2' and 'net_ips s2 = net_tree_ips n2'
    have "(fst (netmap sr s1) ++ fst (netmap sr s2)) i = fst (netmap sr s2) i"
      by (metis dom_fst_netgmap map_add_dom_app_simps(1))
    finally show "fst (netmap sr s2) i = Some (σ i)"
      using 'i ∉ net_tree_ips n2' by simp
  next
    fix i
    assume "i ∉ net_tree_ips n2"
    with 'net_ips s2 = net_tree_ips n2' have "i ∉ net_ips s2" by simp
    thus "fst (netmap sr s2) i = None" by simp
  qed
qed simp

lemma netmap_fst_netgmap_rel:
  shows "(λi. map_option (fst o sr) (netmap s i)) = fst (netgmap sr s)"
proof (induction s)
  fix ii s R
  show "(λi. map_option (fst o sr) (netmap (NodeS ii s R) i)) = fst (netgmap sr (NodeS ii s R))"
    by auto
next
  fix s1 s2
  assume a1: "(λi. map_option (fst o sr) (netmap s1 i)) = fst (netgmap sr s1)"
    and a2: "(λi. map_option (fst o sr) (netmap s2 i)) = fst (netgmap sr s2)"
  show "(λi. map_option (fst o sr) (netmap (SubnetS s1 s2) i)) = fst (netgmap sr (SubnetS s1 s2))"
  proof (rule ext)
    fix i
  qed
qed
from a1 a2 have "map_option (fst o sr) ((netmap s1 ++ netmap s2) i) = (fst (netgmap sr s1) ++ fst (netgmap sr s2)) i"
by (metis fst_conv map_add_dom_app_simps(1) map_add_dom_app_simps(3) net_ips_is_dom_netmap netgmap_pair_dom)
thus "map_option (fst o sr) (netmap (SubnetS s1 s2) i) = fst (netgmap sr (SubnetS s1 s2)) i"
by simp
qed

lemma netmap_is_fst_netgmap:
assumes "netmap s' = netmap s"
shows "fst (netgmap sr s') = fst (netgmap sr s)"
using assms by (metis netmap_fst_netgmap_rel)

lemma netmap_is_fst_netgmap':
assumes "netmap s' i = netmap s i"
shows "fst (netgmap sr s') i = fst (netgmap sr s) i"
using assms by (metis netmap_fst_netgmap_rel)

lemma fst_netgmap_pair_fst [simp]:
"fst (netgmap (λ(p, q). (fst p, snd p, q)) s) = fst (netgmap fst s)"
by (induction s) auto

Introduce streamlined alternatives to netgmap to simplify certain property statements and thus make them easier to understand and to present.

fun netlift :: "('s ⇒ 'g × 'l) ⇒ 's net_state ⇒ ('l ⇒ 'g option)"
where "netlift sr (NodeS i s R) = [i ↦ fst (sr s)]"
| "netlift sr (SubnetS s t) = (netlift sr s) ++ (netlift sr t)"

lemma fst_netgmap_netlift:
"fst (netgmap sr s) = netlift sr s"
by (induction s) simp_all

fun netliftl :: "('s ⇒ 'g × 'l) ⇒ 's net_state ⇒ 'l net_state"
where "netliftl sr (NodeS i s R) = NodeS i (snd (sr s)) R"
| "netliftl sr (SubnetS s t) = SubnetS (netliftl sr s) (netliftl sr t)"

lemma snd_netgmap_netliftl:
"snd (netgmap sr s) = netliftl sr s"
by (induction s) simp_all

lemma netgmap_netlift_netliftl: "netgmap sr s = (netlift sr s, netliftl sr s)"
by (rule simp_all add: fst_netgmap_netlift snd_netgmap_netliftl)

end

6 Semantics of the Algebra of Wireless Networks

theory AWN_SOS
imports TransitionSystems AWN
begin

6.1 Table 1: Structural operational semantics for sequential process expressions

inductive_set seqp_sos :: "('s, 'm, 'p, 'l) seqp_env ⇒ ('s × ('s, 'm, 'p, 'l) seqp, 'm seq_action) transition set"
for Λ :: "('s, 'm, 'p, 'l) seqp_env"
where
broadcastT: "(((ξ, {l}broadcast(s msg)).p), broadcast (s msg ξ), ((ξ, p)) ∈ seqp_sos Λ)"
| groupcastT: "(((ξ, {l}groupcast(s ips, s msg)).p), groupcast (s ips ξ) (s msg ξ), ((ξ, p)) ∈ seqp_sos Λ)"
indicative cases

\[
\text{seqp_callTE} [\text{elim}]: \quad "((\xi, (l)\text{call}(p\text{m}), a, ((\xi', p')) \in \text{seqp_sos} \Gamma) \implies ((\xi, \text{call}(pn)), a, ((\xi', p')) \in \text{seqp_sos} \Gamma"
\]

and \text{seqp_choiceTE} [\text{elim}]: \quad "((\xi, p1 \oplus p2), a, ((\xi', q)) \in \text{seqp_sos} \Gamma"\)

lemma \text{seqp_broadcastTE} [\text{elim}]:

\[
"((\xi, (l)\text{broadcast}(s_{m_{sg}}). p), a, ((\xi', q)) \in \text{seqp_sos} \Gamma; \\
[\tau = \text{broadcast}(s_{m_{sg}} \xi); \xi' = \xi; q = p] \implies P) \implies P"
\]

by \text{(ind_cases "((\xi, (l)\text{broadcast}(s_{m_{sg}}). p), a, ((\xi', q)) \in \text{seqp_sos} \Gamma") simp

lemma \text{seqp_groupcastTE} [\text{elim}]:

\[
"((\xi, (l)\text{groupcast}(s_{ips}, s_{m_{sg}}). p), a, ((\xi', q)) \in \text{seqp_sos} \Gamma; \\
[\tau = \text{groupcast}(s_{ips} \xi; s_{m_{sg}} \xi); \xi' = \xi; q = p] \implies P) \implies P"
\]

by \text{(ind_cases "((\xi, (l)\text{groupcast}(s_{ips}, s_{m_{sg}}). p), a, ((\xi', q)) \in \text{seqp_sos} \Gamma") simp

lemma \text{seqp_unicastTE} [\text{elim}]:

\[
"((\xi, (l)\text{unicast}(s_{ip}, s_{m_{sg}}). p \triangleright q), a, ((\xi', r)) \in \text{seqp_sos} \Gamma; \\
[\tau = \text{unicast}(s_{ip} \xi; s_{m_{sg}} \xi); \xi' = \xi; r = p] \implies P; \\
[a = \text{unicast}(s_{ip} \xi); \xi' = \xi; r = q] \implies P) \implies P"
\]

by \text{(ind_cases "((\xi, (l)\text{unicast}(s_{ip}, s_{m_{sg}}). p \triangleright q), a, ((\xi', r)) \in \text{seqp_sos} \Gamma") simp all

lemma \text{seqp_sendTE} [\text{elim}]:

\[
"((\xi, (l)\text{send}(s_{m_{sg}}). p), a, ((\xi', q)) \in \text{seqp_sos} \Gamma; \\
[\tau = \text{send}(s_{m_{sg}} \xi); \xi' = \xi; q = p] \implies P) \implies P"
\]

by \text{(ind_cases "((\xi, (l)\text{send}(s_{m_{sg}}). p), a, ((\xi', q)) \in \text{seqp_sos} \Gamma") simp

lemma \text{seqp_deliverTE} [\text{elim}]:

\[
"((\xi, (l)\text{deliver}(s_{data}). p), a, ((\xi', q)) \in \text{seqp_sos} \Gamma; \\
[\tau = \text{deliver}(s_{data} \xi); \xi' = \xi; q = p] \implies P) \implies P"
\]

by \text{(ind_cases "((\xi, (l)\text{deliver}(s_{data}). p), a, ((\xi', q)) \in \text{seqp_sos} \Gamma") simp

lemma \text{seqp_receiveTE} [\text{elim}]:

\[
"((\xi, (l)\text{receive}(u_{m_{sg}}). p), a, ((\xi', q)) \in \text{seqp_sos} \Gamma; \\
[\tau = \text{receive}(u_{m_{sg}} \xi; s_{m_{sg}} \xi); q = p] \implies P) \implies P"
\]

by \text{(ind_cases "((\xi, (l)\text{receive}(u_{m_{sg}}). p), a, ((\xi', q)) \in \text{seqp_sos} \Gamma") simp

lemma \text{seqp_assignTE} [\text{elim}]:

\[
"((\xi, (l)\text{assign}(p\text{m}). p), a, ((\xi', q)) \in \text{seqp_sos} \Gamma; \\
[\tau = \text{assign}(p\text{m}); \xi' = \xi; q = p] \implies P) \implies P"
\]

by \text{(ind_cases "((\xi, (l)\text{assign}(p\text{m}). p), a, ((\xi', q)) \in \text{seqp_sos} \Gamma") simp

lemma \text{seqp_guardTE} [\text{elim}]:

\[
"((\xi, (l)\text{guard}(p\text{m}). a, ((\xi', q)) \in \text{seqp_sos} \Gamma; \\
[\tau = \text{guard}(p\text{m}); \xi' = \xi; q = p] \implies P) \implies P"
\]

by \text{(ind_cases "((\xi, (l)\text{guard}(p\text{m}). a, ((\xi', q)) \in \text{seqp_sos} \Gamma") simp

lemmas \text{seqpTEs} =

\text{seqp_broadcastTE}
\text{seqp_groupcastTE}
\text{seqp_unicastTE}
\text{seqp_sendTE}
\text{seqp_deliverTE}
inductive_set

parable: 

\[ (s, a, s') \in S; \forall m. a \neq \text{receive } m \implies ((s, t), a, (s', t)) \in \text{parp sos } S T \]

parright: 

\[ (t, a, t') \in T; \forall m. a \neq \text{send } m \implies ((s, t), a, (s, t')) \in \text{parp sos } S T \]

parboth: 

\[ (s, \text{receive } m, s') \in S; (t, \text{send } m, t') \in T \implies (s, t), \tau, (s', t')) \in \text{parp sos } S T \]

lemma par_broadcastTE [elim]:

\[ [(s, t), \text{broadcast } m, (s', t')) \in \text{parp sos } S T; (s, \text{broadcast } m, s') \in S; t' = t \implies P; (t, \text{broadcast } m, t') \in T; s' = s \implies P] \implies P' \]

by (ind_cases "((s, t), broadcast m, (s', t')) \in \text{parp sos } S T") simp_all

lemma par_groupcastTE [elim]:

\[ [(s, t), \text{groupcast ips } m, (s', t')) \in \text{parp sos } S T; (s, \text{groupcast ips } m, s') \in S; t' = t \implies P; (t, \text{groupcast ips } m, t') \in T; s' = s \implies P] \implies P' \]

by (ind_cases "((s, t), groupcast ips m, (s', t')) \in \text{parp sos } S T") simp_all

lemma par_unicastTE [elim]:

\[ [(s, t), \text{unicast } i m, (s', t')) \in \text{parp sos } S T; (s, \text{unicast } i m, s') \in S; t' = t \implies P; (t, \text{unicast } i m, t') \in T; s' = s \implies P] \implies P' \]

by (ind_cases "((s, t), unicast i m, (s', t')) \in \text{parp sos } S T") simp_all

lemma par_notunicastTE [elim]:

\[ [(s, t), \text{notunicast } i, (s', t')) \in \text{parp sos } S T; (s, \text{notunicast } i, s') \in S; t' = t \implies P; (t, \text{notunicast } i, t') \in T; s' = s \implies P] \implies P' \]

by (ind_cases "((s, t), notunicast i, (s', t')) \in \text{parp sos } S T") simp_all

lemma par_sendTE [elim]:

\[ [(s, t), \text{send } m, (s', t')) \in \text{parp sos } S T; (s, \text{send } m, s') \in S; t' = t \implies P \]

by (ind_cases "((s, t), send m, (s', t')) \in \text{parp sos } S T") auto

lemma par_deliverTE [elim]:

\[ [(s, t), \text{deliver } d, (s', t')) \in \text{parp sos } S T; (s, \text{deliver } d, s') \in S; t' = t \implies P \]

by (ind_cases "((s, t), deliver d, (s', t')) \in \text{parp sos } S T") simp_all

lemma par_receiveTE [elim]:

\[ [(s, t), \text{receive } m, (s', t')) \in \text{parp sos } S T; (t, \text{receive } m, t') \in T; s' = s \implies P \]

by (ind_cases "((s, t), receive m, (s', t')) \in \text{parp sos } S T") auto

inductive_cases par_tauTE: "((s, t), \tau, (s', t')) \in \text{parp sos } S T"
lemmas parpTEs =
par_broadcastTE
par_groupcastTE
par_unicastTE
par_notunicastTE
par_sendTE
par_deliverTE
par_receiveTE

lemma parp_sos_cases [elim]:
assumes 
"((s, t), a, (s', t')) ∈ parp_sos S T"
and 
"[(s, a, s') ∈ S; \forall m. a ≠ receive m; t' = t] \Longrightarrow P"
and 
"[(t, a, t') ∈ T; \forall m. a ≠ send m; s' = s] \Longrightarrow P"
and 
"\forall m. [(s, receive m, s') ∈ S; (t, send m, t') ∈ T] \Longrightarrow P"
shows "P"
using assms by cases auto

definition
par_comp :: "('s1, 'm seq_action) automaton ⇒ ('s2, 'm seq_action) automaton ⇒ ('s1 × 's2, 'm seq_action) automaton"
"(_⟨⟨_\rangle⟩) [102, 103] 102)
where
"s ⟨⟨ t \equiv (init = init s × init t, trans = parp_sos (trans s) (trans t)) \rangle"  

lemma trans_par_comp [simp]:
"trans (s ⟨⟨ t) = parp_sos (trans s) (trans t)"
unfolding par_comp_def by simp

lemma init_par_comp [simp]:
"init (s ⟨⟨ t) = init s × init t"
unfolding par_comp_def by simp

6.3 Table 3: Structural operational semantics for node expressions

inductive_set
node_sos :: "('s, 'm seq_action) transition set ⇒ ('s net_state, 'm node_action) transition set" for S :: "('s, 'm seq_action) transition set"
where
node_bcast:
"(s, broadcast m, s') ∈ S \Longrightarrow (NodeS i s R, R:*cast(m), NodeS i s' R) ∈ node_sos S"
node_gcast:
"(s, groupcast D m, s') ∈ S \Longrightarrow (NodeS i s R, (R ∩ D):*cast(m), NodeS i s' R) ∈ node_sos S"
node_ucast:
"[(s, unicast d m, s') ∈ S; d ∈ R] \Longrightarrow (NodeS i s R, \{d\}:*cast(m), NodeS i s' R) ∈ node_sos S"
node_notucast:
"[(s, ¬unicast d, s') ∈ S; d ∉ R] \Longrightarrow (NodeS i s R, \{d\}:*cast(m), NodeS i s' R) ∈ node_sos S"
node_deliver:
"(s, deliver d, s') ∈ S \Longrightarrow (NodeS i s R, i:deliver(d), NodeS i s' R) ∈ node_sos S"
node_receive:
"(s, receive m, s') ∈ S \Longrightarrow (NodeS i s R, \{i\}:arrive(m), NodeS i s' R) ∈ node_sos S"
node_tau:
"(s, τ, s') ∈ S \Longrightarrow (NodeS i s R, τ, NodeS i s' R) ∈ node_sos S"
node_connect1:
"(NodeS i s R, connect(i, i'), NodeS i s (R ∪ \{i\})) ∈ node_sos S"
node_connect2:
"(NodeS i s R, connect(i', i), NodeS i s (R ∪ \{i\})) ∈ node_sos S"
node_disconnect1:
"(NodeS i s R, disconnect(i, i'), NodeS i s (R - \{i\})) ∈ node_sos S"
node_disconnect2:
"(NodeS i s R, disconnect(i', i), NodeS i s (R - \{i\})) ∈ node_sos S"
"[i ≠ i'; i ≠ i'' ] ⇒ (∏(Node i s R, connect(i', i''), Node i s R) ∈ node_sos S"
/ node_disconnect_other:
"[i ≠ i'; i ≠ i'' ] ⇒ (∏(Node i s R, disconnect(i', i''), Node i s R) ∈ node_sos S"

inductive_cases node_arriveTE: "(Node i s R, ii¬ni:arrive(m), Node i s' R) ∈ node_sos S"
and node_arriveTE': "(Node i s R, H¬K:arrive(m), Node i s' R) ∈ node_sos S"
and node_castTE: "(Node i s R, RM:*cast(m), Node i s' R) ∈ node_sos S"
and node_castTE': "(Node i s R, RM:*cast(m), s') ∈ node_sos S"
and node_deliverTE: "(Node i s R, i:deliver(d), Node i s' R) ∈ node_sos S"
and node_deliverTE': "(s, i:deliver(d), s') ∈ node_sos S"
and node_deliverTE'': "(Node ii s R, i:deliver(d), s') ∈ node_sos S"
and node_tauTE: "(Node i s R, τ, Node i s' R) ∈ node_sos S"
and node_tauTE': "(Node i s R, τ, s') ∈ node_sos S"
and node_connectTE: "(Node ii s R, connect(i, i'), Node ii s' R') ∈ node_sos S"
and node_connectTE': "(Node ii s R, connect(i, i'), s') ∈ node_sos S"
and node_disconnectTE: "(Node ii s R, disconnect(i, i'), Node ii s' R') ∈ node_sos S"
and node_disconnectTE': "(Node ii s R, disconnect(i, i'), s') ∈ node_sos S"

lemma node_sos_never_newpkt [simp]:
assumes "(s, a, s') ∈ node_sos S"
shows "a ≠ i:newpkt(d, di)"
using assms by cases auto

lemma arrives_or_not:
assumes "(Node i s R, ii¬ni:arrive(m), Node i' s' R') ∈ node_sos S"
shows "(ii = {i} ∧ ni = {}) ∨ (ii = {} ∧ ni = {i})"
using assms by rule simp_all

deinition node_comp :: "ip ⇒ ('s, 'm seq_action) automaton ⇒ ip set ⇒ ('s net_state, 'm node_action) automaton"
("⟨⟨ _ : (_) : _ ⟩⟩") [0, 0, 0] 104

where
"⟨i : np : R_i⟩ ≡ ⟨init = {Node i s R | s. s ∈ init np}, trans = node_sos (trans np)⟩"

lemma trans_node_comp:
"trans (⟨i : np : R_i⟩) = node_sos (trans np)"
unfolding node_comp_def by simp

lemma init_node_comp:
"init (⟨i : np : R_i⟩) = {Node i s R_i | s. s ∈ init np}"
unfolding node_comp_def by simp

lemmas node_comps = trans_node_comp init_node_comp

lemma trans_par_node_comp [simp]:
"trans ((i : s (⟨⟨ t : R ⟩⟩ = node_sos (parp_sos (trans s) (trans t)))"
unfolding node_comp_def by simp

lemma snd_par_node_comp [simp]:
"init ((i : s (⟨⟨ t : R ⟩⟩) = {Node i si R | s. st ∈ init s × init t}"
unfolding node_comp_def by simp

lemma node_sos_dest_is_net_state:
assumes "(s, a, s') ∈ node_sos S"
shows "∃i' P' R'. s' = Node i' P' R'"
using assms by induct auto

lemma node_sos_dest:
assumes "(Node i p R, a, s') ∈ node_sos S"
shows "∃ P' R'. s' = Node i P' R'"
using assms assms [THEN node_sos_dest_is_net_state]
by - (erule node_sos_cases, auto)
lemma node_sos_states [elim]:
assumes "(ns, a, ns') ∈ node_sos S"
obtains i s R s' R' where "ns = NodeS i s R"
and "ns' = NodeS i s' R'"

proof -
assume [intro!]: "∀ i s R s' R'. ns = NodeS i s R ⇒ ns' = NodeS i s' R' ⇒ thesis"
from assms(1) obtain i s R where "ns = NodeS i s R"
by (cases ns) auto
moreover with assms(1) obtain s' R' where "ns' = NodeS i s' R'"
by (metis node_sos_dest)
ultimately show thesis ..
qed

lemma node_sos_cases [elim]:
"(NodeS i p, NodeS i p' p') ∈ node_sos S =⇒
(∀ m. a = R:*cast(m); R' = R; (p, broadcast m, p') ∈ S =⇒ P) =⇒
(∀ D m. a = (R ∩ D):*cast(m); R' = R; (p, groupcast D m, p') ∈ S =⇒ P) =⇒
(∀ m. a = {i}:arrive(m); R' = R; (p, receive m, p') ∈ S =⇒ P) =⇒
(∀ D m. a = i:deliver(d); R' = R; (p, deliver d, p') ∈ S =⇒ P) =⇒
(∀ m. a = i; a = connect(i, i); R' = R ∪ {i}; p = p' =⇒ P) =⇒
(∀ m. a = disconnect(i, i); R' = R ∩ {i}; p = p' =⇒ P) =⇒
(∀ m. a = disconnect(i, i'); R' = R ∩ {i'}; p = p' =⇒ P) =⇒
(∀ m. a = {d}:unicast d, p = p' =⇒ P) =⇒
(∀ m. a = {d}:unicast d, p = p' =⇒ P) =⇒
(∀ m. a = (R ∩ K):unicast d, p = p' =⇒ P) =⇒
(∀ m. a = {d}:unicast d, p = p' =⇒ P) =⇒
(∀ m. a = (R ∩ K):unicast d, p = p' =⇒ P) =⇒
(∀ m. a = (R ∩ K):unicast d, p = p' =⇒ P) =⇒
(∀ m. a = {d}:unicast d, p = p' =⇒ P) =⇒
(∀ m. a = {d}:unicast d, p = p' =⇒ P) =⇒" by (rule node_sos_cases) simp_all

6.4 Table 4: Structural operational semantics for partial network expressions

inductive_set
pnet_sos :: "('s net_state, 'm node_action) transition set
⇒ ('s net_state, 'm node_action) transition set
⇒ ('s net_state, 'm node_action) transition set" for S :: "('s net_state, 'm node_action) transition set"
and T :: "('s net_state, 'm node_action) transition set"
where
pnet_cast1: "[(R ∪ {i}):arrive(m), t') ∈ T] =⇒ (SubnetS s t, R:*cast(m), SubnetS s' t') ∈ pnet_sos S T"
pnet_cast2: "[(R ∩ K):unicast d, p = p' =⇒ P] =⇒ (SubnetS s t, R:*cast(m), SubnetS s' t') ∈ pnet_sos S T"
pnet_arrive: "[(R ∩ K):unicast d, p = p' =⇒ P] =⇒ (SubnetS s t, (H ∪ H') ∩ (K ∪ K'):unicast d, SubnetS s' t') ∈ pnet_sos S T"
pnet_arrive1: "[(s, i:deliver(d), s') ∈ S] =⇒ (SubnetS s t, i:deliver(d), SubnetS s' t') ∈ pnet_sos S T"
pnet_arrive2: "[(t, i:deliver(d), t') ∈ T] =⇒ (SubnetS s t, i:deliver(d), SubnetS s' t') ∈ pnet_sos S T"
pnet_tau1: "[(s, τ, s') ∈ S] =⇒ (SubnetS s t, τ, SubnetS s' t') ∈ pnet_sos S T"
pnet_tau2: "[(t, τ, t') ∈ T] =⇒ (SubnetS s t, τ, SubnetS s' t') ∈ pnet_sos S T"
pnet_connect: "[(s, connect(i, i'), s') ∈ S] =⇒ (SubnetS s t, connect(i, i'), SubnetS s' t') ∈ pnet_sos S T"
pnet_disconnect: "[(s, disconnect(i, i'), s') ∈ S] =⇒ (SubnetS s t, disconnect(i, i'), SubnetS s' t') ∈ pnet_sos S T"
inductive_cases partial_castTE [elim]:  "(s, R:*cast(m), s') ∈ pnet_sos S T"
  and partial_arriveTE [elim]:  "(s, H=R:arrive(m), s') ∈ pnet_sos S T"
  and partial_deliverTE [elim]:  "(s, i:deliver(d), s') ∈ pnet_sos S T"
  and partial_tauTE [elim]:  "(s, τ, s') ∈ pnet_sos S T"
  and partial_connectTE [elim]:  "(s, connect(i, i'), s') ∈ pnet_sos S T"
  and partial_disconnectTE [elim]:  "(s, disconnect(i, i'), s') ∈ pnet_sos S T"

lemma pnet_sos_never_newpkt:
  assumes "(st, a, st') ∈ pnet_sos S T"
  and "⋀ i d di a s s'. (s, a, s') ∈ S =⇒ a ≠ i:newpkt(d, di)"
  and "⋀ i d di a t t'. (t, a, t') ∈ T =⇒ a ≠ i:newpkt(d, di)"
  shows "a ≠ i:newpkt(d, di)"
  using assms(1) by cases (auto dest!: assms(2-3))

fun pnet :: "(ip ⇒ ('s, 'm seq_action) automaton) ⇒ net_tree ⇒ ('s net_state, 'm node_action) automaton"
where
  "pnet np ((i; R) i) = (i : np i : R)"
| "pnet np (p1 ∥ p2) = (init = {SubnetS s1 s2 |s1 s2. s1 ∈ init (pnet np p1) ∧ s2 ∈ init (pnet np p2)},
                     trans = pnet_sos (trans (pnet np p1)) (trans (pnet np p2)))"

lemma pnet_node_init [elim, simp]:
  assumes "s ∈ init (pnet np (⟨i; R⟩))"
  shows "s ∈ {NodeS i s R |s. s ∈ init (np i)}"
  using assms by (simp add: node_comp_def)

lemma pnet_node_init' [elim]:
  assumes "s ∈ init (pnet np (⟨i; R⟩))"
  obtains ns where "s = NodeS i ns R"
  and "ns ∈ init (np i)"
  using assms by (auto simp add: node_comp_def)

lemma pnet_node_trans [elim, simp]:
  assumes "(s, a, s') ∈ trans (pnet np (⟨i; R⟩))"
  shows "(s, a, s') ∈ node_sos (trans (np i))"
  using assms by (simp add: trans_node_comp)

lemma pnet_never_newpkt':
  assumes "(s, a, s') ∈ trans (pnet np n)"
  shows "∀ i d di. a ≠ i:newpkt(d, di)"
  using assms proof (induction n arbitrary: s a s')
  fix n1 n2 s a s'
  assume IH1: "∀ s a s'. (s, a, s') ∈ trans (pnet np n1) =⇒ ∀ i d di. a ≠ i:newpkt(d, di)"
  and IH2: "∀ s a s'. (s, a, s') ∈ trans (pnet np n2) =⇒ ∀ i d di. a ≠ i:newpkt(d, di)"
  and "(s, a, s') ∈ trans (pnet np (n1 ∥ n2))"
  show "∀ i d di. a ≠ i:newpkt(d, di)"
  proof (intro allI)
    fix i d di
    from '(s, a, s') ∈ trans (pnet np (n1 ∥ n2))'
    have "(s, a, s') ∈ pnet_sos (trans (pnet np n1)) (trans (pnet np n2))"
    by simp
    thus "a ≠ i:newpkt(d, di)"
    by (rule pnet_sos_never_newpkt) (auto dest!: IH1 IH2)
  qed
  qed (simp add: node_comps)

lemma pnet_never_newpkt:
  assumes "(s, a, s') ∈ trans (pnet np n)"
  shows "a ≠ i:newpkt(d, di)"
  proof -
    from assms have "∀ i d di. a ≠ i:newpkt(d, di)"
    by (rule pnet_never_newpkt')
    thus ?thesis by clarsimp

37
6.5 Table 5: Structural operational semantics for complete network expressions

inductive_set

cnet_sos :: "('s, ('m::msg) node_action) transition set
            ⇒ ('s, 'm node_action) transition set"
for S :: "('s, 'm node_action) transition set"
where
  cnet_connect: "(s, connect(i, i'), s') ∈ S ⇒ (s, connect(i, i'), s') ∈ cnet_sos S"
  cnet_disconnect: "(s, disconnect(i, i'), s') ∈ S ⇒ (s, disconnect(i, i'), s') ∈ cnet_sos S"
  cnet_cast: "(s, R:*cast(m), s') ∈ S ⇒ (s, τ, s') ∈ cnet_sos S"
  cnet_tau: "(s, τ, s') ∈ S ⇒ (s, τ, s') ∈ cnet_sos S"
  cnet_deliver: "(s, i:deliver(d), s') ∈ S ⇒ (s, i:deliver(d), s') ∈ cnet_sos S"
  cnet_newpkt: "(s, {i}¬K:arrive(newpkt(d, di)), s') ∈ S ⇒ (s, i:newpkt(d, di), s') ∈ cnet_sos S"

inductive_cases
  connect_completeTE: "(s, connect(i, i'), s') ∈ cnet_sos S"
  and disconnect_completeTE: "(s, disconnect(i, i'), s') ∈ cnet_sos S"
  and tau_completeTE: "(s, τ, s') ∈ cnet_sos S"
  and deliver_completeTE: "(s, i:deliver(d), s') ∈ cnet_sos S"
  and newpkt_completeTE: "(s, i:newpkt(d, di), s') ∈ cnet_sos S"

lemmas completeTEs = connect_completeTE
disconnect_completeTE
tau_completeTE
deliver_completeTE
delpkt_completeTE

lemma complete_no_cast [simp]:
  "(s, R:*cast(m), s') ∉ cnet_sos T"
proof
  assume "(s, R:*cast(m), s') ∈ cnet_sos T"
  hence "R:*cast(m) ≠ R:*cast(m)"
  by (rule cnet_sos.cases) auto
  thus False by simp
qed

lemma complete_no_arrive [simp]:
  "(s, ii¬ni:arrive(m), s') ∉ cnet_sos T"
proof
  assume "(s, ii¬ni:arrive(m), s') ∈ cnet_sos T"
  hence "ii¬ni:arrive(m) ≠ ii¬ni:arrive(m)"
  by (rule cnet_sos.cases) auto
  thus False by simp
qed

abbreviation
closed :: "('s net_state, ('m::msg) node_action) automaton ⇒ ('s net_state, 'm node_action) automaton"
where
  "closed ≡ (λA. A (| trans := cnet_sos (trans A) |))"

end

7 Control terms and well-definedness of sequential processes

theory AWN_Cterms
imports AWN
begin

7.1 Microsteps

We distinguish microsteps from ‘external’ transitions (observable or not). Here, they are a kind of ‘hypothetical computation’, since, unlike τ-transitions, they do not make choices but rather ‘compute’ which choices are possible.
inductive
  microstep :: "('s, 'm, 'p, 'l) seqp_env
           ⇒ ('s, 'm, 'p, 'l) seqp
           ⇒ ('s, 'm, 'p, 'l) seqp
           ⇒ bool"
for Γ :: "('s, 'm, 'p, 'l) seqp_env"
where
  microstep_choiceI1 [intro, simp]: "microstep Γ (p1 ⊕ p2) p1"
  | microstep_choiceI2 [intro, simp]: "microstep Γ (p1 ⊕ p2) p2"
  | microstep_callI [intro, simp]: "microstep Γ (call(pn)) (Γ pn)"
abbreviation microstep_rtcl
where
  "microstep_rtcl Γ p q ≡ (microstep Γ)∗∗p q"
abbreviation microstep_tcl
where
  "microstep_tcl Γ p q ≡ (microstep Γ)++p q"
syntax
  "_microstep" :: "[(('s, 'm, 'p, 'l) seqp, ('s, 'm, 'p, 'l) seqp_env, ('s, 'm, 'p, 'l) seqp]
                   ⇒ bool" ("_microstep") 50
  "_microstep_rtcl" :: "[(('s, 'm, 'p, 'l) seqp, ('s, 'm, 'p, 'l) seqp_env, ('s, 'm, 'p, 'l) seqp]
                        ⇒ bool" ("_microstep_rtcl") 50
  "_microstep_tcl" :: "[(('s, 'm, 'p, 'l) seqp, ('s, 'm, 'p, 'l) seqp_env, ('s, 'm, 'p, 'l) seqp]
                       ⇒ bool" ("_microstep_tcl") 50
translations
  "p1 ⇝ Γ p2" ≡ "CONST microstep Γ p1 p2"
  "p1 ⇝ Γ p2" ≡ "CONST microstep_rtcl Γ p1 p2"
  "p1 ⇝ Γ p2" ≡ "CONST microstep_tcl Γ p1 p2"
lemma microstep_choiceD [dest]:
  "(p1 ⊕ p2) ⇝ Γ p ⇒ p = p1 ∨ p = p2"
  by (ind_cases "(p1 ⊕ p2) ⇝ Γ p") auto
lemma microstep_choiceE [elim]:
  "[(p1 ⊕ p2) ⇝ Γ p; p1 = Γ pn] =⇒ P"
  by (blast)
lemma microstep_callD [dest]:
  "(call(pn)) ⇝ Γ p ⇒ p = Γ pn"
  by (ind_cases "(call(pn)) ⇝ Γ p")
lemma microstep_callE [elim]:
  "[(call(pn)) ⇝ Γ p; p = Γ (pn)] =⇒ P"
  by auto
lemma no_microstep_guard: "¬ ((({l}{g}) p) ⇝ Γ q)"
  by (rule notI) (ind_cases "(({l}{g}) p) ⇝ Γ q")
lemma no_microstep_assign: "¬ (({l}[f] p) ⇝ Γ q)"
  by (rule notI) (ind_cases "({l}[f] p) ⇝ Γ q")
lemma no_microstepunicast: "¬ ((({l}unicast(sip, smsg).p ⋄ q) ⇝ Γ r)"
  by (rule notI) (ind_cases "({l}unicast(sip, smsg).p ⋄ q) ⇝ Γ r")
lemma no_microstep_broadcast: "¬ ((({l}broadcast(smsg).p) ⇝ Γ q)"
  by (rule notI) (ind_cases "({l}broadcast(smsg).p) ⇝ Γ q")
lemma no_microstep_groupcast: "¬ ((({l}groupcast(sips, smsg).p) ⇝ Γ q)"
lemma no_microstep_send: "¬ ((\{l\}send(s_{msg}).p) \xrightarrow{\Gamma} q)"
by (rule notI) (ind_cases "\{l\}send(s_{msg}).p \xrightarrow{\Gamma} q")

lemma no_microstep_deliver: "¬ ((\{l\}deliver(s_{data}).p) \xrightarrow{\Gamma} q)"
by (rule notI) (ind_cases "\{l\}deliver(s_{data}).p \xrightarrow{\Gamma} q")

lemma no_microstep_receive: "¬ ((\{l\}receive(u_{msg}).p) \xrightarrow{\Gamma} q)"
by (rule notI) (ind_cases "\{l\}receive(u_{msg}).p \xrightarrow{\Gamma} q")

lemma microstep_call_or_choice [dest]:
assumes "p \xrightarrow{\Gamma} q"
shows "\(\exists pn. p = call(pn)) \lor (\exists p1 p2. p = p1 \oplus p2)"
using assms by clarsimp (metis microstep.simps)

lemmas no_microstep [intro,simp] =
no_microstep_guard
no_microstep_assign
no_microstep_unicast
no_microstep_broadcast
no_microstep_groupcast
no_microstep_send
no_microstep_deliver
no_microstep_receive

7.2 Wellformed process specifications

A process specification \(\Gamma\) is wellformed if its microstep \(\Gamma\) relation is free of loops and infinite chains.

For example, these specifications are not wellformed:

\[\Gamma_1\]
P1 = call(p1)
\[\Gamma_2\]
\[\Gamma_3\]
P1 = send(msg) \cdot call(p1) \oplus call(p1)
\[\Gamma_3\]
P2 = call(p2)
P3 = call(p3)
P4 = call(p5) \ldots

definition

wellformed :: \('s, 'm, 'p, 'l\) seqp_env \Rightarrow bool
where

"wellformed \(\Gamma\) = \(\lambda q p. p \xrightarrow{\Gamma} q\)"

lemma wellformed_defP: "wellformed \(\Gamma\) = \(\lambda q p. p \xrightarrow{\Gamma} q\)"

unfolding wellformed_def wfP_def
by simp

The induction rule for wellformed \(\Gamma\) is stronger than \(\lambda \Gamma \in\): wellformed \(\Gamma\) = \(\lambda q p. p \xrightarrow{\Gamma} q\) because the case for call(pn) can be shown with the assumption on \(\Gamma\) pn.

lemma wellformed_induct
[consumes 1, case_names ASSIGN CHOICE CALL GUARD UCAST BCAST GCAST SEND DELIVER RECEIVE,
induct set: wellformed]:
assumes "wellformed \(\Gamma\)"
and ASSIGN: "\(\lambda f p. \) wellformed \(\Gamma\) = \(\lambda q p. \) P ((f) f p)"
and GUARD: "\(\lambda f p. \) wellformed \(\Gamma\) = \(\lambda q p. \) P ((f) f p)"
and UCAST: "\(\lambda fip fmsg p q. wellformed \(\Gamma\) = \(\lambda q p. \) P ((call) fip, fmsg) p q)"
and BCAST: "\(\lambda fmsg p. \) wellformed \(\Gamma\) = \(\lambda q p. \) P ((broadcast) fmsg) p q)"
and GCAST: "\(\lambda fips fmsg p. \) wellformed \(\Gamma\) = \(\lambda q p. \) P ((groupcast) fips, fmsg) p q)"
and SEND: "\(\lambda fmsg p. \) wellformed \(\Gamma\) = \(\lambda q p. \) P ((send) fmsg) p q)"
and DELIVER: "\(\lambda fdata p. \) wellformed \(\Gamma\) = \(\lambda q p. \) P ((deliver) fdata) p q)"
and RECEIVE: "\(\lambda fmsg p. \) wellformed \(\Gamma\) = \(\lambda q p. \) P ((receive) fmsg) p q)"
and CHOICE: "\(\lambda p1 p2. \) [ P (call(pn)) \Rightarrow P (p1 \oplus p2)"
and CALL: "\(\lambda pn. \) P (call(pn)) \Rightarrow P (call(pn))"
shows "P a"
using \texttt{assms(1)} unfolding \texttt{wellformed_defP}
proof (rule \texttt{wfP_induct_rule}, case_tac \texttt{x}, simp_all)
  fix \texttt{p1 p2}
  assume \texttt{"\forall q. (p1 \oplus p2) \rightsquigarrow \Gamma q \Rightarrow P q"}
  then obtain \texttt{"P p1"} and \texttt{"P p2"} by (auto intro!: \texttt{microstep.intros})
  thus \texttt{"P (p1 \oplus p2)"} by (rule \texttt{CHOICE [OF 'wellformed \Gamma']})
next
  fix \texttt{pn}
  assume \texttt{"\forall q. (call(pn)) \rightsquigarrow \Gamma q = \Rightarrow P q"}
  hence \texttt{"P (call(pn))"} by (auto intro!: \texttt{microstep.intros})
  thus \texttt{"P (call(pn))"} by (rule \texttt{CALL [OF 'wellformed \Gamma']})
qed (auto intro: \texttt{assms})

7.3 Start terms (sterms)

Formulate sets of local subterms from which an action is directly possible. Since the process specification $\Gamma$ is not considered, only choice terms $p1 \oplus p2$ are traversed, and not $call(p)$ terms.

\begin{verbatim}
fun stermsl :: ('s, 'm, 'p, 'l) seqp ⇒ ('s, 'm, 'p, 'l) seqp set
where
  "stermsl (p1 \oplus p2) = stermsl p1 \cup stermsl p2"
/ "stermsl p = \{p\}"

lemma stermsl_nobigger: "q ∈ stermsl p \Rightarrow \text{size } q \leq \text{size } p"
  by (induct p) auto

lemma stermsl_no_choice[simp]: "p1 \oplus p2 \notin stermsl p"
  by (induct p) simp_all

lemma stermsl_choice_disj[simp]:
  "p ∈ stermsl (p1 \oplus p2) = (p ∈ stermsl p1 \lor p ∈ stermsl p2)"
  by simp

lemma stermsl_in_branch[elim]:
  "[ [p ∈ stermsl (p1 \oplus p2); p ∈ stermsl p1 \Rightarrow P; p ∈ stermsl p2 \Rightarrow P] \Rightarrow P]"
  by auto

lemma stermsl_commute:
  "stermsl (p1 \oplus p2) = stermsl (p2 \oplus p1)"
  by simp (rule Un_commute)

lemma stermsl_not_empty:
  "stermsl p \neq \{}"
  by (induct p) auto

lemma stermsl_idem [simp]:
  "\bigcup q∈stermsl p. stermsl q = stermsl p"
  by (induct p) simp_all

lemma stermsl_in_wfpf:
  assumes AA: "A \subseteq \{ (q, p) \cdot p \rightsquigarrow \Gamma q \} '' A"
  and *: "p ∈ A"
  shows "\exists r\instermsl p. r \in A"
using *
proof (induction p)
  fix \texttt{p1 p2}
  assume IH1: "p1 ∈ A \Rightarrow \exists r\instermsl p1. r \in A"
  and IH2: "p2 ∈ A \Rightarrow \exists r\instermsl p2. r \in A"
  and *: "p1 \oplus p2 \in A"
  from * and AA have "p1 \oplus p2 ∈ \{ (q, p) \cdot p \rightsquigarrow \Gamma q \} '' A" by auto
  hence "p1 ∈ A \lor p2 ∈ A" by auto
  hence "(\exists r\instermsl p1. r \in A) \lor (\exists r\instermsl p2. r \in A)"
  proof
    assume "p1 ∈ A" hence "\exists r\instermsl p1. r \in A" by (rule IH1) thus \texttt{thesis} ..

41
\end{verbatim}
next
  assume "p2 ∈ A" hence "∃r ∈ stermsl p2. r ∈ A" by (rule IH2) thus ?thesis ..
  qed

hence "∃r ∈ stermsl p1 ∪ stermsl p2. r ∈ A" by blast
thus "∃r ∈ stermsl (p1 ⊕ p2). r ∈ A" by simp

next case UCAST from UCAST.prems show ?case by auto
  qed auto

lemma nocall_stermsl_max:
  assumes "r ∈ stermsl p"
  and "not_call r"
  shows "¬((q, p). (q, p)⇝Γ q) ≤ A" by blast

next case UCAST from UCAST.prems show ?case by auto
  qed auto

7.4 Start terms

The start terms are those terms, relative to a wellformed process specification Γ, from which transitions can occur directly.
": 

'(s, 'm, 'p, 'l) seqp_env ⇒ (s, 'm, 'p, 'l) seqp ⇒ (s, 'm, 'p, 'l) seqp set" where

sterms_choice: "sterms Γ (p1 ⊕ p2) = sterms Γ p1 ∪ sterms Γ p2"
sterms_call: "sterms Γ (call(pn)) = sterms Γ (Γ pn)"
sterms_other: "sterms Γ p = {p}"

by pat_completeness auto

lemma sterms_dom_basic[simp]:
assumes "not_call p"
  and "not_choice p"
shows "sterms_dom (Γ, p)"
proof (rule accpI)
  fix y
  assume "sterms_rel y (Γ, p)"
  with assms show "sterms_dom y"
  by (cases p) (auto simp: sterms_rel.simps)
qed

lemma sterms_termination:
assumes "wellformed Γ"
shows "sterms_dom (Γ, p)"
proof-
  have sterms_rel':
    "sterms_rel = (λgq gp. (gq, gp) ∈ {((Γ, q), (Γ', p)). Γ = Γ' ∧ p ↝Γ q})"
  by (rule ext)+ (auto simp: sterms_rel.simps elim: microstep.cases)

  from assms have "∀x. x ∈ Wellfounded.acc {(q, p). p ↝Γ q}".
    unfolding wellformed_def by (simp add: wf_acc_iff)
  hence "p ∈ Wellfounded.acc {(q, p). p ↝Γ q}" ..

  hence "((Γ, p) ∈ Wellfounded.acc {(Γ, q), (Γ', p)). Γ = Γ' ∧ p ↝Γ q}" by (rule acc_induct) (auto intro: accI)

  thus "sterms_dom (Γ, p)" unfolding sterms_rel' accp_acc_eq .
qed

declare sterms.psimp [simp]

lemmas sterms_psimp[simp] = sterms.psimp [OF sterms_termination]
  and sterms_pinduct = sterms.pinduct [OF sterms_termination]

lemma sterms_reflD [dest]:
assumes "q ∈ sterms Γ p"
  and "not_choice p" "not_call p"
shows "q = p"
using assms by (cases p) auto

lemma sterms_choice_disj [simp]:
assumes "wellformed Γ"
shows "p ∈ sterms Γ (p1 ⊕ p2) = (p ∈ sterms Γ p1 ∨ p ∈ sterms Γ p2)"
using assms by (simp)

lemma sterms_no_choice [simp]:
assumes "wellformed Γ"
shows "p1 ⊕ p2 /∈ sterms Γ p"
using assms by induction auto

lemma sterms_not_choice [simp]:
assumes "wellformed Γ"
  and "q ∈ sterms Γ p"
shows "not_choice q"
using assms unfolding not_choice_def
by (auto dest: sterms_no_choice)
lemma sterms_no_call [simp]:
  assumes "wellformed Γ"
  shows "call(pn) ∉ sterms Γ p"
  using assms by induction auto

lemma sterms_not_call [simp]:
  assumes "wellformed Γ"
    and "q ∈ sterms Γ p"
  shows "not_call q"
  using assms unfolding not_call_def by (auto dest: sterms_no_call)

lemma sterms_in_branch:
  assumes "wellformed Γ"
    and "p ∈ sterms Γ (p1 ⊕ p2)"
    and "p ∈ sterms Γ p1 ⇒ P"
    and "p ∈ sterms Γ p2 ⇒ P"
  shows "P"
  using assms by auto

lemma sterms_commute:
  assumes "wellformed Γ"
  shows "sterms Γ (p1 ⊕ p2) = sterms Γ (p2 ⊕ p1)"
  using assms by simp (rule Un_commute)

lemma sterms_not_empty:
  assumes "wellformed Γ"
  shows "sterms Γ p ≠ {}"
  using assms by (induct p rule: sterms_pinduct [OF 'wellformed Γ']) simp_all

lemma sterms_sterms [simp]:
  assumes "wellformed Γ"
  shows "(⋃x ∈ sterms Γ p. sterms Γ x) = sterms Γ p"
  using assms by induction simp_all

lemma sterms_stermsl:
  assumes "ps ∈ sterms Γ p"
    and "wellformed Γ"
  shows "ps ∈ stermsl p ∨ (∃pn. ps ∈ stermsl (Γ pn))"
  using assms by (induction p rule: sterms_pinduct [OF 'wellformed Γ']) auto

lemma sterms1_stermsl [elim]:
  assumes "q ∈ stermsl p"
    and "not_call q"
    and "wellformed Γ"
  shows "q ∈ sterms Γ p"
  using assms by (induct p) auto

lemma sterms1_stermsl_heads:
  assumes "ps ∈ sterms Γ (Γ pn)"
    and "wellformed Γ"
  shows "∃pn. ps ∈ stermsl (Γ pn)"
proof -
  from assms have "ps ∈ stermsl (Γ pn) ∨ (∃pn’. ps ∈ stermsl (Γ pn’))"
    by (rule sterms1_stermsl)
  thus ?thesis by auto
qed

lemma sterms_subterms [dest]:
  assumes "wellformed Γ"
    and "∃pn. p ∈ subterms (Γ pn)"
    and "q ∈ sterms Γ p"
  shows "∃pn. q ∈ subterms (Γ pn)"
using assms by (induct p) auto

lemma no_microsteps_sterms_refl:
assumes "wellformed Γ"
shows "(¬ (∃ q. p⇝ Γ q)) = (sterms Γ p = {p})"
proof (cases p)
  fix p1 p2
  assume 'wellformed Γ' have "p1 ⊕ p2 /∈ sterms Γ (p1 ⊕ p2)" by simp
  hence "sterms Γ (p1 ⊕ p2) ≠ {p1 ⊕ p2}" by auto
  moreover have "∃ q. (p1 ⊕ p2)⇝ Γ q" by auto
  ultimately show ?thesis
    using 'p = p1 ⊕ p2' by simp
next
  fix pn
  assume 'wellformed Γ' have "call(pn) /∈ sterms Γ (call(pn))" by simp
  hence "sterms Γ (call(pn)) ≠ {call(pn)}" by auto
  moreover have "∃ q. (call(pn))⇝ Γ q" by auto
  ultimately show ?thesis
    using 'p = call(pn)' by simp
qed simp_all

lemma sterms_maximal [elim]:
assumes "wellformed Γ" and "q ∈ sterms Γ p"
shows "sterms Γ q = {q}" using assms by (cases q) auto

lemma microstep_rtranscl_equal:
assumes "not_call p" and "not_choice p" and "p⇝ Γ∗ q"
shows "q = p" using assms(3) proof (rule converse_rtranclpE)
  fix p'
  assume "p⇝ Γ p'" with assms(1-2) show "q = p" by (cases p) simp_all
qed simp

lemma microstep_rtranscl_singleton [simp]:
assumes "not_call p" and "not_choice p"
shows "{q. p⇝ Γ∗ q ∧ sterms Γ q = {q}} = {p}" proof (rule set_eqI)
  fix p'
  show "(p' ∈ {q. p⇝ Γ∗ q ∧ sterms Γ q = {q}}) = (p' ∈ {p})"
  proof
    assume "p' ∈ {q. p⇝ Γ∗ q ∧ sterms Γ q = {q}}"
    hence "sterms Γ q = {q}" by auto
    from this(1) have "p' = p" proof
      fix q assume "p⇝ Γ q" with 'not_call p' and 'not_choice p' have False
      by (cases p) auto
      thus "p' = p" ..
    qed simp
    thus "p' ∈ {p}" by simp
  next
    assume "p' ∈ {p}" hence "p' = p" ..
    with 'not_call p' and 'not_choice p' show "p' ∈ {q. p⇝ Γ∗ q ∧ sterms Γ q = {q}}" by (cases p) simp_all
  qed

45
theorem stems_maximal_microstep:
  assumes "wellformed Γ" shows "sterms Γ p ⊆ {q. p ↭ Γ q}"
proof
  induction
  caseIH1: "sterms Γ p1 ⊆ {q. p ↭ Γ q} ∧ sterns Γ q = {q1}"
  caseIH2: "sterms Γ p2 ⊆ {q. p2 ↭ Γ q} ∧ sterns Γ q = {q2}"
  assume IH: "sterms Γ p ⊆ {q. p ↭ Γ q} ∧ sterns Γ q = {q}"
  shows "(p1 ⊕ p2) ↭ Γ q"
  proof
    fix p1 p2
    assume IH1: "sterms Γ p1 ⊆ {q. p1 ↭ Γ q} ∧ sterns Γ q = {q1}"
    and IH2: "sterms Γ p2 ⊆ {q. p2 ↭ Γ q} ∧ sterns Γ q = {q2}"
    have "sterms Γ p1 ⊆ {q. (p1 ⊕ p2) ↭ Γ q} ∧ sterns Γ q = {q}"
    proof
      assume p': "p' ∈ sterns Γ p" with IH1 have "p1 ↭ Γ p'" by auto
      moreover have "(p1 ⊕ p2) ↭ Γ p1" ..
      ultimately have "(p1 ⊕ p2) ↭ Γ p'" by - (rule converse_rtranclp_into_rtranclp)
      qed simp
    qed
    moreover have "sterms Γ p2 ⊆ {q. (p1 ⊕ p2) ↭ Γ q} ∧ sterns Γ q = {q}"
    proof
      assume p': "p' ∈ sterns Γ p2" with IH2 have "p2 ↭ Γ p'" by auto
      moreover have "(p1 ⊕ p2) ↭ Γ p2" ..
      ultimately have "(p1 ⊕ p2) ↭ Γ p'" by - (rule converse_rtranclp_into_rtranclp)
      qed simp
    qed
    ultimately show "sterms Γ (p1 ⊕ p2) ⊆ {q. (p1 ⊕ p2) ↭ Γ q} ∧ sterns Γ q = {q}"
      using 'wellformed Γ' by simp
next
  fix pn
  assume IH: "sterms Γ (Γ pn) ⊆ {q. Γ pn ↭ Γ q} ∧ sterns Γ q = {q1}"
  show "sterms Γ (call(pn)) ⊆ {q. (call(pn)) ↭ Γ q} ∧ sterns Γ q = {q}"
  proof
    fix p
    assume p': "p' ∈ sterns Γ (call(pn))" with IH have "Γ pn ↭ Γ p'" by auto
    note this(1)
    moreover have "(call(pn)) ↭ Γ pn" by simp
    ultimately have "(call(pn)) ↭ Γ p'" by - (rule converse_rtranclp_into_rtranclp)
    qed simp
  qed simp_all
  with 'wellformed Γ' show "sterms Γ p ⊆ {q. p ↭ Γ q} ∧ sterns Γ q = {q}"
    by (simp only: no_microsteps_sterns_refl)
next
  from 'wellformed Γ' have "{q. p ↭ Γ q} ⊆ sterns Γ p" proof (induction)
    caseIH1: "{q. p1 ↭ Γ q} ⊆ sterns Γ p1"
    caseIH2: "{q. p2 ↭ Γ q} ⊆ sterns Γ p2"
    show "{q. (p1 ⊕ p2) ↭ Γ q} ⊆ sterns Γ (p1 ⊕ p2)"
      proof (rule, drule CollectD, erule conjE)
fix q'
assume "(p1 ⊕ p2) ⇝ Γ⁺ q'" and "sterms Γ q' = {q'}"
with 'wellformed Γ' have "(p1 ⊕ p2) ⇝ Γ⁺ q'"
by (auto dest!: rtranclpD sterms_no_choice)

thus "q' ∈ sterms Γ (p1 ⊕ p2)"

proof
  assume "p1 ⇝ Γ⁺ q'"
  with IH1 and 'sterms Γ q' = {q'}' have "q' ∈ sterms Γ p1" by auto
  with 'wellformed Γ' show ?thesis by auto
  next
  assume "p2 ⇝ Γ⁺ q'"
  with IH2 and 'sterms Γ q' = {q'}' have "q' ∈ sterms Γ p2" by auto
  with 'wellformed Γ' show ?thesis by auto
qed

next
fix pn
assume IH: "{q. Γ pn ⇝ Γ⁺ q ∧ sterms Γ q = {q}} ⊆ sterms Γ (Γ pn)"

show "{q. (call(pn)) ⇝ Γ⁺ q ∧ sterms Γ q = {q}} ⊆ sterms Γ (call(pn))"
proof (rule, drule CollectD, erule conjE)
  fix q'
  assume "(call(pn)) ⇝ Γ⁺ q'" and "sterms Γ q' = {q'}"
  with 'wellformed Γ' have "(call(pn)) ⇝ Γ⁺ q'" by (auto dest!: rtranclpD sterms_no_call)
  moreover have "(call(pn)) ⇝ Γ pn" ..
  ultimately have "Γ pn ⇝ Γ⁺ q'" by (auto dest!: tranclpD)
  with 'sterms Γ q' = {q'}' and IH have "q' ∈ sterms Γ (Γ pn)" by auto
  with 'wellformed Γ' show "q' ∈ sterms Γ (call(pn))" by simp
qed

7.5 Derivative terms

The derivatives of a term are those sterms potentially reachable by taking a transition, relative to a wellformed process specification Γ. These terms overapproximate the reachable sterms, since the truth of guards is not considered.

function (domintros) dterms :: "('s, 'm, 'p, 'l) seqp_env ⇒ ('s, 'm, 'p, 'l) seqp ⇒ ('s, 'm, 'p, 'l) seqp set"
where
  "dterms Γ ({{l}}⟨g⟩ p)" = sterms Γ p
  / "dterms Γ ({{l}}[u] p)" = sterms Γ p
  / "dterms Γ (p1 ⊕ p2)" = dterms Γ p1 ∪ dterms Γ p2
  / "dterms Γ ({{l}}unicast(s_ip, s_msg).p ⊊ q)" = sterms Γ p ∪ sterms Γ q
  / "dterms Γ ({{l}}broadcast(s_msg). p)" = sterms Γ p
  / "dterms Γ ({{l}}groupcast(s_ip, s_msg). p)" = sterms Γ p
  / "dterms Γ ({{l}}send(s_msg).p)" = sterms Γ p
  / "dterms Γ ({{l}}deliver(s_data).p)" = sterms Γ p
  / "dterms Γ ({{l}}receive(u_msg).p)" = sterms Γ p
  / "dterms Γ (call(pn))" = dterms Γ (Γ pn)
by pat_completeness auto

lemma dterms_dom_basic [simp]:
  assumes "not_call p"
  and "not_choice p"
  shows "dterms_dom (Γ, p)"

proof (rule accpI)
  fix y
  assume "dterms_rel y (Γ, p)"
  with assms show "dterms_dom y"
    by (cases p) (auto simp: dterms_rel.simps)
qed

lemma dterms_termination:
  assumes "wellformed Γ"
  shows "dterms_dom (Γ, p)"
proof -
  have dterms_rel': "dterms_rel = (λgq gp. (gq, gp) ∈ {((Γ, q), (Γ', p)). Γ = Γ' ∧ p ⇒Γ q})"
    by (rule ext)+ (auto simp: dterms_rel.simps elim: microstep.cases)
  from 'wellformed(Γ)'
  have "∀ x. x ∈ Wellfounded.acc {(q, p). p ⇒Γ q}"
    unfolding wellformed_def
    by (simp add: wf_acc_iff)
  hence "p ∈ Wellfounded.acc {(Γ, p). p ⇒Γ q}"
    by (rule acc_induct) (auto intro: accI)
  thus "dterms_dom (Γ, p)"
    unfolding dterms_rel'
    by (subst accp_acc_eq)
qed

lemmas dterms_psimp [simp] = dterms.psimp [OF dterms_termination]
and dterms_pinduct = dterms.pinduct [OF dterms_termination]

lemma stterms_after_dterms [simp]:
  assumes "wellformed Γ"
  shows "⋃x ∈ dterms Γ p. stterms Γ x = dterms Γ p"
using assms by (induction p) simp_all

lemma stterms_before_dterms [simp]:
  assumes "wellformed Γ"
  shows "⋃x ∈ stterms Γ p. dterms Γ x = dterms Γ p"
using assms by (induction p) simp_all

lemma dterms_choice_disj [simp]:
  assumes "wellformed Γ"
  shows "p ∈ dterms Γ (p1 ⊕ p2) = (p ∈ dterms Γ p1 ∨ p ∈ dterms Γ p2)"
using assms by (simp)

lemma dterms_in_branch:
  assumes "wellformed Γ"
  and "p ∈ dterms Γ (p1 ⊕ p2)"
  and "p ∈ dterms Γ p1 ⇒ P"
  and "p ∈ dterms Γ p2 ⇒ P"
  shows "P"
using assms by auto

lemma dterms_no_choice:
  assumes "wellformed Γ"
  shows "p1 ⊕ p2 ∉ dterms Γ p"
using assms by induction simp_all

lemma dterms_not_choice [simp]:
  assumes "wellformed Γ"
  and "q ∈ dterms Γ p"
  shows "not_choice q"
using assms unfolding not_choice_def
by (auto dest: dterms_no_choice)

lemma dterms_no_call:
  assumes "wellformed Γ"
  shows "call(pn) ∉ dterms Γ p"
using assms by induction simp_all
lemma dterms_not_call [simp]:
assumes "wellformed Γ" and "q ∈ dterms Γ p"
shows "not_call q"
using assms unfolding not_call_def
by (auto dest: dterms_no_call)

lemma dterms_subterms:
assumes wf: "wellformed Γ" and "∃ pn. p ∈ subterms (Γ pn)"
and "q ∈ dterms Γ p"
shows "∃ pn. q ∈ subterms (Γ pn)"
using assms
proof (induct p)
fix p1 p2
assume IH1: "∃ pn. p1 ∈ subterms (Γ pn) =⇒ q ∈ dterms Γ p1 =⇒ ∃ pn. q ∈ subterms (Γ pn)"
and IH2: "∃ pn. p2 ∈ subterms (Γ pn) =⇒ q ∈ dterms Γ p2 =⇒ ∃ pn. q ∈ subterms (Γ pn)"
and *: "∃ pn. p1 ⊕ p2 ∈ subterms (Γ pn)"
and "q ∈ dterms Γ (p1 ⊕ p2)"
from * obtain pn where "p1 ⊕ p2 ∈ subterms (Γ pn)"
by auto
hence "p1 ∈ subterms (Γ pn)" and "p2 ∈ subterms (Γ pn)"
by auto
from 'q ∈ dterms Γ (p1 ⊕ p2)'
wf have "q ∈ dterms Γ p1 ∨ q ∈ dterms Γ p2"
by auto
thus "∃ pn. q ∈ subterms (Γ pn)"
proof
assume "q ∈ dterms Γ p1"
with 'p1 ∈ subterms (Γ pn)'
show ?thesis
by (auto intro: IH1)
next
assume "q ∈ dterms Γ p2"
with 'p2 ∈ subterms (Γ pn)'
show ?thesis
by (auto intro: IH2)
qed auto
Note that the converse of [wellformed Γ; ∃ pn. ?p ∈ subterms (?Γ pn); ?q ∈ dterms ?Γ ?p] =⇒ ∃ pn. ?q ∈ subterms (?Γ pn) is not true because dterms are an over-approximation; i.e., we cannot show, in general, that guards return a non-empty set of post-states.

7.6 Control terms

The control terms of a process specification Γ are those subterms from which transitions are directly possible. We can omit call(pn) terms, since the root terms of all processes are considered, and also p1 ⊕ p2 terms since they effectively combine the transitions of the subterms p1 and p2.

It will be shown that only the control terms, rather than all subterms, need be considered in invariant proofs.

inductive_set
cterms :: "('s, 'm, 'p, 'l) seqp_env ⇒ ('s, 'm, 'p, 'l) seqp set"
for Γ :: "('s, 'm, 'p, 'l) seqp_env"
where
ctermsSI[intro]: "p ∈ stersms Γ (Γ pn) =⇒ p ∈ cterms Γ"
ctermsDI[intro]: "[ pp ∈ cterms Γ; p ∈ dterms Γ pp ] =⇒ p ∈ cterms Γ"

lemma cterms_not_choice [simp]:
assumes "wellformed Γ" and "p ∈ cterms Γ"
shows "not_choice p"
using assms
proof (cases p)
case CHOICE from 'p ∈ cterms Γ'
show ?thesis
using 'wellformed Γ' by cases simp_all
lemma cterms_no_choice [simp]:
assumes "wellformed Γ"
shows "p1 ⊕ p2 ∉ cterms Γ"
using assms by (auto dest: cterms_not_choice)

lemma cterms_not_call [simp]:
assumes "wellformed Γ"
and "p ∈ cterms Γ"
shows "not_call p"
using assms

proof (cases p)
case CALL from 'p ∈ cterms Γ' show ?thesis
  using 'wellformed Γ' by cases simp_all
qed simp_all

lemma cterms_no_call [simp]:
assumes "wellformed Γ"
shows "call(pn) /∈ cterms Γ"
using assms by (auto dest: cterms_not_call)

lemma stterms_cterms [elim]:
assumes "p ∈ cterms Γ"
and "q ∈ stterms Γ p"
and "wellformed Γ"
shows "q ∈ cterms Γ"
using assms by (cases p, auto)

lemma dterms_cterms [elim]:
assumes "p ∈ cterms Γ"
and "q ∈ dterms Γ p"
and "wellformed Γ"
shows "q ∈ cterms Γ"
using assms by (cases p) auto

lemma derivs_in_cterms [simp]:

\[\begin{align*}
& \forall l f p. \{l\} f p \in cterms Γ =\Rightarrow sterms Γ p \subseteq cterms Γ \\
& \forall l f p. \{l\} \left[\begin{array}{c}
\end{array}\right] p \in cterms Γ =\Rightarrow sterms Γ p \subseteq cterms Γ \\
& \forall l fip fmsg q p. \{l\} unicast(fip, fmsg). p ⊕ q \in cterms Γ =\Rightarrow sterms Γ p \subseteq cterms Γ \land sterms Γ q \subseteq cterms Γ \\
& \forall l fmsg p. \{l\} broadcast(fmsg). p \in cterms Γ =\Rightarrow sterms Γ p \subseteq cterms Γ \\
& \forall l fips fmsg p. \{l\} groupcast(fips, fmsg). p \in cterms Γ =\Rightarrow sterms Γ p \subseteq cterms Γ \\
& \forall l fdata p. \{l\} deliver(fdata). p \in cterms Γ =\Rightarrow sterms Γ p \subseteq cterms Γ \\
& \forall l fmsg p. \{l\} receive(fmsg). p \in cterms Γ =\Rightarrow sterms Γ p \subseteq cterms Γ \\
\end{align*}\]
by (auto simp: dterms.psimp)

7.7 Local control terms

We introduce a ‘local’ version of cterms that does not step through calls and, thus, that is defined independently of a process specification Γ. This allows an alternative, terminating characterisation of cterms as a set of subterms. Including call(pn)s in the set makes for a simpler relation with stterms1, even if they must be filtered out for the desired characterisation.

function
ctermsl :: "('s, 'm, 'p, 'l) seqp ⇒ ('s, 'm, 'p , 'l) seqp set"
where

"ctermsl ((1)g) p" = insert ((1)g) p (ctermsl p)"  "ctermsl ((1)u) p" = insert ((1)u) p (ctermsl p)"  "ctermsl ((1)unicast(s_ip, s_msg). p ⊕ q) = insert ((1)unicast(s_ip, s_msg). p ⊕ q) (ctermsl p)"  "ctermsl ((1)broadcast(s_msg). p) = insert ((1)broadcast(s_msg). p) (ctermsl p)"  "ctermsl ((1)groupcast(s_ips, s_msg). p) = insert ((1)groupcast(s_ips, s_msg). p) (ctermsl p)"
ctermsl \((\{1\}\text{send}(s_{msg}).\ p)\) = \(\text{insert}(\{1\}\text{send}(s_{msg}).\ p)\) (ctermsl p)

ctermsl \((\{1\}\text{deliver}(s_{data}).\ p)\) = \(\text{insert}(\{1\}\text{deliver}(s_{data}).\ p)\) (ctermsl p)

ctermsl \((\{1\}\text{receive}(u_{msg}).\ p)\) = \(\text{insert}(\{1\}\text{receive}(u_{msg}).\ p)\) (ctermsl p)

ctermsl \((p_1 \oplus p_2)\) = ctermsl \(p_1\) \(\cup\) ctermsl \(p_2\)

ctermsl \((\text{call}(pn))\) = \(\{\text{call}(pn)\}\)

by \(\text{pat\_completeness\ auto}\)
termination by (relation "measure(size)") (auto dest: stermsl_nobigger)

lemmas ctermsl_induct =
ctermsl.induct [case_names GUARD ASSIGN UCAST BCAST GCAST SEND DELIVER RECEIVE CHOICE CALL]

lemma ctermsl_refl [intro]: "not\_choice\ p \implies p \in ctermsl\ p"
by (cases p) auto

lemma ctermsl_subterms:
"ctermsl\ p = \{q. q \in \text{subterms}\ p \land \text{not\_choice}\ q\}\" (is "?lhs = ?rhs")
proof
  show "?lhs \subseteq \?rhs" by (induct p, auto) next
  show "?rhs \subseteq \?lhs" by (induct p, auto)
qed

lemma ctermsl_trans [elim]:
assumes "q \in ctermsl\ p"
  and "r \in ctermsl\ q"
shows "r \in ctermsl\ p"
using assms
proof (induction p rule: ctermsl_induct)
case (CHOICE p1 p2)
have "(q \in ctermsl\ p1) \lor (q \in ctermsl\ p2)" using CHOICE.prems(1) by simp
  hence "r \in ctermsl\ p1 \lor r \in ctermsl\ p2" proof (rule disj_forward)
    assume "q \in ctermsl\ p1"
    thus "r \in ctermsl\ p1" using 'r \in ctermsl\ q' by (rule CHOICE.IH)
    next
    assume "q \in ctermsl\ p2"
    thus "r \in ctermsl\ p2" using 'r \in ctermsl\ q' by (rule CHOICE.IH)
    qed
  thus "r \in ctermsl\ (p1 \oplus p2)" by simp
  qed auto

lemma ctermsl_ex_trans [elim]:
assumes "\exists q \in ctermsl\ p. r \in ctermsl\ q"
shows "r \in ctermsl\ p"
using assms by auto

lemma call_ctermsl_empty [elim]:
"[ p \in ctermsl\ p'; \text{not\_call}\ p ] \implies \text{not\_call}\ p'"
unfolding not_call_def by (cases p) auto

lemma stermsl_ctermsl_choice1 [simp]:
assumes "q \in stermsl\ p1"
shows "q \in ctermsl\ (p1 \oplus p2)"
using assms by (induction p1) auto

lemma stermsl_ctermsl_choice2 [simp]:
assumes "q \in stermsl\ p2"
shows "q \in ctermsl\ (p1 \oplus p2)"
using assms by (induction p2) auto

lemma stermsl_ctermsl [elim]:
assumes "q \in stermsl\ p"
shows "q \in ctermsl\ p"

51
using assms

proof (cases p)
  case (CHOICE p1 p2)
  hence "q ∈ stermsl (p1 ⊕ p2)" using assms by simp
  hence "q ∈ stermsl p1 ∨ q ∈ stermsl p2" by simp
  hence "q ∈ ctermsl (p1 ⊕ p2)" by (rule) (simp_all del: ctermsl.simps)
  thus "q ∈ ctermsl p" using CHOICE by simp
qed simp_all

lemma stermsl_after_ctermsl [simp]:
  "(⋃x ∈ ctermsl p. stermsl x) = ctermsl p"
  using assms by (induction p) auto

lemma stermsl_before_ctermsl [simp]:
  "(⋃x ∈ stermsl p. ctermsl x) = ctermsl p"
  using assms by (induction p) simp_all

lemma ctermsl_no_choice: "p1 ⊕ p2 /∈ ctermsl p"
  by (induct p) simp_all

lemma ctermsl_ex_stermsl: "q ∈ ctermsl p =⇒ ∃ps ∈ stermsl p. q ∈ ctermsl ps"
  by (induct p) auto

lemma dterms_ctermsl [intro]:
  assumes "q ∈ dterms Γ p" and "wellformed Γ"
  shows "q ∈ ctermsl p ∨ (∃pn. q ∈ ctermsl (Γ pn))"
  using assms(1-2)
proof (induction p rule: dterms_pinduct [OF 'wellformed Γ'])
  fix Γ l fg p
  assume "q ∈ dterms Γ ({l}⟨fg⟩ p)" and "wellformed Γ"
  hence "q ∈ sterms Γ p" by simp
  hence "q ∈ stermsl p ∨ (∃pn. q ∈ stermsl (Γ pn))" using 'wellformed Γ' by (rule sterms_stermsl)
  thus "q ∈ ctermsl ({l}⟨fg⟩ p) ∨ (∃pn. q ∈ ctermsl (Γ pn))" proof
    assume "q ∈ stermsl p"
    hence "q ∈ ctermsl p" by (rule stermsl_ctermsl)
    hence "q ∈ ctermsl ({l}⟨fg⟩ p)" by simp
    thus ?thesis ..
  next
    assume "∃pn. q ∈ stermsl (Γ pn)"
    then obtain pn where "q ∈ stermsl (Γ pn)" by auto
    hence "q ∈ ctermsl (Γ pn)" by (rule stermsl_ctermsl)
    hence "∃pn. q ∈ ctermsl (Γ pn)" ..
    thus ?thesis ..
  qed
next
  fix Γ p1 p2
  assume "q ∈ dterms Γ (p1 ⊕ p2)"
  and IH1: "[ q ∈ dterms Γ p1; wellformed Γ ] =⇒ q ∈ ctermsl p1 ∨ (∃pn. q ∈ ctermsl (Γ pn))"
  and IH2: "[ q ∈ dterms Γ p2; wellformed Γ ] =⇒ q ∈ ctermsl p2 ∨ (∃pn. q ∈ ctermsl (Γ pn))"
  and "wellformed Γ"
  thus "q ∈ ctermsl (p1 ⊕ p2) ∨ (∃pn. q ∈ ctermsl (Γ pn))" by auto
next
  fix Γ pn
  assume "q ∈ dterms Γ (call(pn))" and "wellformed Γ"
  and "[ q ∈ dterms Γ (Γ pn); wellformed Γ ] =⇒ q ∈ ctermsl (Γ pn) ∨ (∃pn. q ∈ ctermsl (Γ pn))"
  thus "q ∈ ctermsl (call(pn)) ∨ (∃pn. q ∈ ctermsl (Γ pn))" by auto
qed (simp_all, (metis sterms_stermsl stermsl_ctermsl)+)
lemma ctermsl_cterms [elim]:
assumes "q ∈ ctermsl p"
and "not_call q"
and "sterms Γ p ⊆ cterms Γ"
and "wellformed Γ"
shows "q ∈ cterms Γ"
using assms by (induct p rule: ctermsl.induct) auto

7.8 Local derivative terms

We define local dterms for use in the theorem that relates cterms and sets of ctermsl.

function dtermsl :: 
('s, 'm, 'p, 'l) seqp 
⇒ 
('s, 'm, 'p, 'l) seqp set
where
"dtermsl ({l}⟨fg⟩ p)" = stermsl p
"dtermsl ({l}⟨fa⟩ p)" = stermsl p
"dtermsl (p1 ⊕ p2)" = dtermsl p1 ∪ dtermsl p2
"dtermsl ({l}unicast(fip, fmsg). p ⊥ q)" = stermsl p ∪ stermsl q
"dtermsl ({l}broadcast(fmsg). p)" = stermsl p
"dtermsl ({l}groupcast(fips, fmsg). p)" = stermsl p
"dtermsl ({l}send(fmsg). p)" = stermsl p
"dtermsl ({l}deliver(fdata). p)" = stermsl p
"dtermsl ({l}receive(fmsg). p)" = stermsl p
"dtermsl (call(pn))" = {}
by pat_completeness auto
termination by (relation "measure(size)") (auto dest: stermsl_nobigger)

lemma stermsl_after_dtermsl [simp]:
shows "⋃x ∈ dtermsl p. stermsl x = dtermsl p"
using assms by (induct p) simp_all

lemma stermsl_before_dtermsl [simp]:
"⋃x ∈ stermsl p. dtermsl x = dtermsl p"
using assms by (induct p) simp_all

lemma dtermsl_no_choice [simp]: "p1 ⊕ p2 ∉ dtermsl p"
by (induct p) simp_all

lemma dtermsl_choice_disj [simp]:
"p ∈ dtermsl (p1 ⊕ p2) = (p ∈ dtermsl p1 ∨ p ∈ dtermsl p2)"
by simp

lemma dtermsl_in_branch [elim]:
"[p ∈ dtermsl (p1 ⊕ p2); p ∈ dtermsl p1 ⇒ P; p ∈ dtermsl p2 ⇒ P] ⇒ P"
by auto

lemma ctermsl_dtermsl [elim]:
assumes "q ∈ dtermsl p"
shows "q ∈ ctermsl p"
using assms by (induct p) (simp_all, (metis stermsl_ctermsl)+)

lemma dtermsl_dtermsl [elim]:
assumes "q ∈ dtermsl p"
and "not_call q"
and "wellformed Γ"
shows "q ∈ dterms Γ p"
using assms
using assms by (induct p) (simp_all, (metis stermsl_stermsl)+)

lemma ctermsl_stermsl_or_dtermsl:
assumes "q ∈ ctermsl p"
shows "q ∈ stermsl p ∨ (∃p′ ∈ dtermsl p. q ∈ ctermsl p′)"
using assms by (induct p) (auto dest: ctermsl_ex_stermsl)
lemma dtermsl_add_stermsl_beforeD:
assumes "q ∈ dtermsl p"
shows "∃ ps∈stermsl p. q ∈ dtermsl ps"
proof -
  from assms have "q ∈ (⋃ x∈stermsl p. dtermsl x)" by auto
  thus ?thesis
    by (rule UN_E) auto
qed

lemma call_dtermsl_empty [elim]:
"q ∈ dtermsl p ⇒ not_call p"
by (cases p) simp_all

7.9 More properties of control terms

We now show an alternative definition of cterms based on sets of local control terms. While the original definition
has convenient induction and simplification rules, useful for proving properties like cterms_includes_sterms_of_seq_reachabl,
this definition makes it easier to systematically generate the set of control terms of a process specification.

theorem cterms_def':
assumes wfg: "wellformed Γ"
shows "cterms Γ = { p |p pn. p ∈ ctermsl (Γ pn) ∧ not_call p }"
(is "_=?ctermsl_set")
proof (rule iffI [THEN set_eqI])
  fix p
  assume "p ∈ cterms Γ"
  thus "p ∈ ?ctermsl_set"
    proof (induction p)
      fix p pn
      assume "p ∈ sterms Γ (Γ pn)"
      then obtain pn' where "p ∈ ctermsl (Γ pn)" using wfg
        by (blast dest: sterms_stermsl_heads)
      hence "p ∈ ctermsl (Γ pn)" ..
      moreover from 'p ∈ sterms Γ (Γ pn)' wfg have "not_call p" by simp
      ultimately show "p ∈ ?ctermsl_set" by auto
    next
      fix pp p
      assume "pp ∈ cterms Γ"
      and IH: "pp ∈ ?ctermsl_set"
      and *: "p ∈ dterms Γ pp"
      from * have "p ∈ ctermsl pp ∨ (∃pn. p ∈ ctermsl (Γ pn))" using wfg by (rule dterms_ctermsl)
      hence "∃pn. p ∈ ctermsl (Γ pn)" proof
        assume "p ∈ ctermsl pp"
        from 'pp ∈ cterms Γ' and IH obtain pn' where "pp ∈ ctermsl (Γ pn)" by auto
        with 'p ∈ ctermsl pp' have "p ∈ ctermsl (Γ pn)" by auto
        thus "∃pn. p ∈ ctermsl (Γ pn)" ..
      qed -
      moreover from 'p ∈ dterms Γ pp' wfg have "not_call p" by simp
      ultimately show "p ∈ ?ctermsl_set" by auto
    qed
next
  fix p
  assume "p ∈ ?ctermsl_set"
  then obtain pn where *: "p ∈ ctermsl (Γ pn)" and "not_call p" by auto
  from * have "p ∈ stermsl (Γ pn) ∨ (∃p'∈dtermsl (Γ pn). p ∈ ctermsl p')" by (rule ctermsl_stermsl_or_dtermsl)
  hence "p ∈ cterms Γ" proof
    assume "p ∈ sterms Γ (Γ pn)"
    hence "p ∈ sterms Γ (Γ pn)" using 'not_call p' wfg ..
thus \textit{"p \in cterms \Gamma"} ..

next

assume \textit{\"\exists p' \in dtermsl (\Gamma \ pn). p \in ctermsl p'\"}

then obtain \(p\) where \(p'1: p' \in dtermsl (\Gamma \ pn)\)

and \(p'2: p \in ctermsl p'\"

from \(p'2\) and \textit{\"not_call p\"} have \textit{\"not_call p'\"} ..

from \(p'1\) obtain \(ps\) where \(ps1: ps \in stermsl (\Gamma \ pn)\)

and \(ps2: p' \in dtermsl ps\)

by (blast dest: dtermsl_add_stermsl_beforeD)

from \(ps2\) have \textit{\"not_call ps\"} ..

from \(ps1\) obtain \(ps\) where \(ps1: ps \in ctermsl (\Gamma \ pn)\)

and \(ps2: p' \in dtermsl ps\)

hence \textit{\"sterms \Gamma p' \subseteq cterms \Gamma\"}

using \textit{\"wfg\"} by auto

with \(p' \in dtermsl ps'\) and \textit{\"not_call p''\"}

show \textit{\"p \in cterms \Gamma\"}

using \textit{\"wfg\"} ..

qed

\textbf{lemma \textsc{ctermsE [elim]}:}

\textbf{assumes \textit{\"wellformed \Gamma\"}

\textbf{and \textit{\"p \in cterms \Gamma\"}}}

\textbf{obtains \(pn\) where \textit{\"p \in ctermsl (\Gamma \ pn)\"}

\textbf{and \textit{\"not_call p\"}}}

\textbf{using \textit{\"assms(2)\"} unfolding \textit{\"cterms_def\" [OF \textit{\"assms(1)\"}] by auto}}

\textbf{corollary \textsc{cterms_subterms}:}

\textbf{assumes \textit{\"wellformed \Gamma\"}

\textbf{shows \textit{\"cterms \Gamma \{p|p pn. p \in subterms (\Gamma \ pn) \land not_call p \land not_choice p\}\"}}

\textbf{by \textit{\"(subst \textit{\"cterms_def\" [OF \textit{\"assms(1)\"}]}, subst \textit{\"ctermsl_subterms\"}) auto}}

\textbf{lemma \textsc{subterms_in_cterms [elim]}:}

\textbf{assumes \textit{\"wellformed \Gamma\"}

\textbf{and \textit{\"p \in subterms \Gamma\"}

\textbf{and \textit{\"not_call \ p\"}

\textbf{and \textit{\"not_choice \ p\"}}}

\textbf{shows \textit{\"p \in cterms \Gamma\"}}

\textbf{using \textit{\"assms\" unfolding \textit{\"cterms_subterms\" [OF \textit{\"wellformed \Gamma\"}] by auto}}}

\textbf{lemma \textsc{subterms_stermsl_ctermsl}:}

\textbf{assumes \textit{\"q \in subterms \ p\"}

\textbf{and \textit{\"r \in stermsl q\"}}}

\textbf{shows \textit{\"r \in ctermsl \ p\"}}

\textbf{using \textit{\"assms\"}

\textbf{proof \textit{\"(induct \ p\"}})

\textbf{fix \(p1 \ p2\)

\textbf{assume \textit{\"IH1: \"q \in subterms \ p1 \implies r \in stermsl \ q \implies r \in ctermsl \ p1\"}}

\textbf{and \textit{\"IH2: \"q \in subterms \ p2 \implies r \in stermsl \ q \implies r \in ctermsl \ p2\"}}

\textbf{and \textit{\"*: \"q \in subterms \ (p1 \oplus \ p2)\"}}

\textbf{and \textit{\"r \in stermsl \ q\"}}

\textbf{from \textit{\"*\"} have \textit{\"q \in \{p1 \oplus \ p2\} \cup subterms \ p1 \cup subterms \ p2\"} by simp

\textbf{thus \textit{\"r \in ctermsl \ (p1 \oplus \ p2)\"}}

\textbf{proof \textit{\"(elim \ UnE)\"}}

\textbf{assume \textit{\"q \in \{p1 \oplus \ p2\}\"} with \textit{\"r \in stermsl \ q\"} show \textit{\"thesis\"}

\textbf{by simp (metis stermsl_ctermsl)}

\textbf{next}

\textbf{assume \textit{\"q \in subterms \ p1\"} hence \textit{\"r \in ctermsl \ p1\"} using \textit{\"r \in stermsl \ q\"} by \textit{\"(rule \ IH1)\"}

\textbf{thus \textit{\"thesis\"} by simp}

\textbf{next}

\textbf{assume \textit{\"q \in subterms \ p2\"} hence \textit{\"r \in ctermsl \ p2\"} using \textit{\"r \in stermsl \ q\"} by \textit{\"(rule \ IH2)\"}

\textbf{thus \textit{\"thesis\"} by simp}

\textbf{qed}

\textbf{qed}

\textbf{auto}

\textbf{lemma \textsc{subterms_sterms_cterms}:}

\textbf{assumes \textit{\"wf: \"wellformed \Gamma\"\"}}

55
and "p ∈ subterms (Γ pn)"
shows "sterms Γ p ⊆ cterms Γ"
using assms(2)
proof (induct p)
  fix p
  assume "call(p) ∈ subterms (Γ pn)"
  from wf have "sterms Γ (call(p)) = sterms Γ (Γ p)" by simp
  thus "sterms Γ (call(p)) ⊆ cterms Γ" by auto
next
  fix p1 p2
  assume IH1: "p1 ∈ subterms (Γ pn) =⇒ sterms Γ p1 ⊆ cterms Γ" and IH2: "p2 ∈ subterms (Γ pn) =⇒ sterms Γ p2 ⊆ cterms Γ"
  moreover from * have "p2 ∈ subterms (Γ pn)" by auto
  hence "sterms Γ p1 ⊆ cterms Γ" by (rule IH1)
  ultimately show "sterms Γ (p1 ⊕ p2 ) ⊆ cterms Γ" using wf by simp
qed (auto elim!: subterms_in_cterms [OF 'wellformed Γ'])

lemma subterms_sterms_in_cterms:
  assumes "wellformed Γ"
  and "p ∈ subterms (Γ pn)"
  and "q ∈ sterms Γ p"
  shows "q ∈ cterms Γ"
  using assms by (auto dest!: subterms_sterms_cterms [OF 'wellformed Γ'])
end

8 Labelling sequential processes

theory AWN_Labels
imports AWN AWN_Cterms
begin

8.1 Labels

Labels serve two main purposes. They allow the substitution of sterms in invariant proofs. They also allow the strengthening (control state dependent) of invariants.

function (domintros) labels :: "('s, 'm, 'p, 'l) seqp_env ⇒ ('s, 'm, 'p, 'l) seqp ⇒ 'l set"
where
  "labels Γ ({{l}(fg) p)" = {l}"
  | "labels Γ ({{l}[fa] p)" = {l}"
  | "labels Γ (p1 ⊕ p2)" = labels Γ p1 ∪ labels Γ p2"
  | "labels Γ ({{l}unicast(fip, fmsg).p ⊗ q)" = {l}"
  | "labels Γ ({{l}broadcast(fmsg). p)" = {l}"
  | "labels Γ ({{l}groupcast(fips, fmsg). p)" = {l}"
  | "labels Γ ({{l}send(fmsg).p)" = {l}"
  | "labels Γ ({{l}deliver(fdata).p)" = {l}"
  | "labels Γ ({{l}receive(fmsg).p)" = {l}"
  | "labels Γ (call(pn))" = labels Γ (Γ pn)"

by pat_completeness auto

lemma labels_dom_basic [simp]:
  assumes "not_call p"
  and "not_choice p"
  shows "labels_dom (Γ, p)"
proof (rule accpI)
  fix y
  assume "labels_rel y (Γ, p)"
  with assms show "labels_dom y"
end
by (cases p) (auto simp: labels_rel.simps)

qed

lemma labels_termination:
  fixes Γ p
  assumes "wellformed(Γ)"
  shows "labels_dom (Γ, p)"
proof -
  have labels_rel': "labels_rel = (λgq gp. (gq, gp) ∈ {{(Γ, q), (Γ', p)). Γ = Γ' ∧ p ~Γ q})"
    by (rule ext)+ (auto simp: labels_rel.simps intro: microstep.intros elim: microstep.cases)
  from 'wellformed(Γ)'
    have ∀ x. x ∈ Wellfounded.acc {{(q, p). p ~Γ q}}
    unfolding wellformed_def
    by (simp add: wf_acc_iff)
  hence "p ∈ Wellfounded.acc {{(q, p). p ~Γ q}}"
    ..
  hence "(Γ, p) ∈ Wellfounded.acc {{(Γ, q), (Γ', p). Γ = Γ' ∧ p ~Γ q}}"
    by (rule acc_induct) (auto intro: accI)
  thus "labels_dom (Γ, p)"
    unfolding labels_rel'
    by (subst accp_acc_eq)
  qed

declare labels.psimps[simp]

lemmas labels_pinduct = labels.pinduct [OF labels_termination]
  and labels_psimps[simp] = labels.psimps [OF labels_termination]

lemma labels_not_empty:
  fixes Γ p
  assumes "wellformed Γ"
  shows "labels Γ p ≠ {}"
by (induct p rule: labels_pinduct [OF 'wellformed Γ']) simp_all

lemma has_label [dest]:
  fixes Γ p
  assumes "wellformed Γ"
  shows "∃ l. l ∈ labels Γ p"
using labels_not_empty [OF assms] by auto

lemma singleton_labels [simp]:
  "(1 l' f p. 1 ∈ labels Γ ((1l){f}(p) = (l = l'))"
  "(1 l' f p. 1 ∈ labels Γ ((1l)[f](p) = (l = l'))"
  "(1 l' fmsg p q. 1 ∈ labels Γ ((1l)unicast(fip, fmsg).p ⊢ q) = (l = l'))"
  "(1 l' fmsg p. 1 ∈ labels Γ ((1l)broadcast(fmsg).p) = (l = l'))"
  "(1 l' fmsg p. 1 ∈ labels Γ ((1l)groupcast(fips, fmsg).p) = (l = l'))"
  "(1 l' fmsg p. 1 ∈ labels Γ ((1l)send(fmsg).p) = (l = l'))"
  "(1 l' fmsg p. 1 ∈ labels Γ ((1l)receive(fmsg).p) = (l = l'))"
by auto

lemma in_labels-singletons [dest!]:
  "(1 l' f p. 1 ∈ labels Γ ((1l){f}(p) = (l = l'))"
  "(1 l' f p. 1 ∈ labels Γ ((1l)[f](p) = (l = l'))"
  "(1 l' fmsg p q. 1 ∈ labels Γ ((1l)unicast(fip, fmsg).p ⊢ q) = (l = l'))"
  "(1 l' fmsg p. 1 ∈ labels Γ ((1l)broadcast(fmsg).p) = (l = l'))"
  "(1 l' fmsg p. 1 ∈ labels Γ ((1l)groupcast(fips, fmsg).p) = (l = l'))"
  "(1 l' fmsg p. 1 ∈ labels Γ ((1l)send(fmsg).p) = (l = l'))"
  "(1 l' fmsg p. 1 ∈ labels Γ ((1l)receive(fmsg).p) = (l = l'))"
by auto

definition simple_labels :: "('s, 'm, 'p, 'l) seqp_env ⇒ bool"
where
  "simple_labels Γ ≡ ∀ pn. ∀ p ∈ subterms (Γ pn). (∃ l. labels Γ p = (1l))"

lemma simple_labelsI [intro]:
assumes \( \forall pn. p \in \text{subterms} (\Gamma \ pn) \implies \exists l. \text{labels} \ \Gamma \ p = \{l\} \)"
shows "simple_labels \( \Gamma \)"
using assms unfolding simple_labels_def by auto

The simple_labels \( \Gamma \) property is necessary to transfer results shown over the cterms of a process specification \( \Gamma \) to the reachable actions of that process.

Consider the process \( \{l_1\} \text{send}(m_1) . \ p_1 \uplus \{l_2\} \text{send}(m_2) . \ p_2 \). The iteration over cterms \( \Gamma \) will cover the two transitions \( (l_1, \text{send} m_1, p_1) \) and \( (l_2, \text{send} m_2, p_2) \), but reachability requires the four transitions \( (l_1, \text{send} m_1, p_1), (l_1, \text{send} m_2, p_2), (l_2, \text{send} m_1, p_1), \) and \( (l_2, \text{send} m_2, p_2) \).

In a simply labelled process, the former is sufficient to show the latter, since it is by design; we try to treat call sites as "syntactic pastings" of process terms, giving rise, ultimately have "\( l \in \text{labels} \ \Gamma \ p_1 \)" and "\( l \in \text{labels} \ \Gamma \ p_2 \)"
by (metis \emph{Un_iff} \ \emph{insert_iff} \ \emph{set_eqI}+)
moreover from \( \text{ein} \) have "\( \exists pn. p_1 \in \text{subterms} (\Gamma \ pn) \)"
and "\( \exists pn. p_2 \in \text{subterms} (\Gamma \ pn) \)"
by auto
ultimately show "\( \forall p' \in \text{sterms} \ \Gamma \ (p_1 \uplus p_2) \). l \in \text{labels} \ \Gamma \ p' \)"
using \( \text{wf} \ \text{IH1} \ [\text{OF} \ \text{sl} \ \text{wf}] \ \text{IH2} \ [\text{OF} \ \text{sl} \ \text{wf}] \) by auto
qed auto

lemma labels_in_sterms:
fixes \( \Gamma \ p \)
assumes "wellformed \( \Gamma \)"
and "\( l \in \text{labels} \ \Gamma \ p \)"
shows "\( \exists p' \in \text{sterms} \ \Gamma \ p. l \in \text{labels} \ \Gamma \ p' \)"
using assms by (induct \( p \) rule: labels_pinduct [OF 'wellformed \( \Gamma \)']) (auto intro: Un_iff)

lemma labels_sterms_labels:
fixes \( \Gamma \ p \ p' \ l \)
assumes "wellformed \( \Gamma \)"
and "p' ∈ terms Γ p"
and "l ∈ labels Γ p'"
shows "l ∈ labels Γ p"
using assms
by (induct p rule: labels_pinduct [OF 'wellformed Γ']) auto

primrec labelfrom :: "int ⇒ int ⇒ ('s, 'm, 'p, 'a) seqp ⇒ int × ('s, 'm, 'p, int) seqp"
where
"labelfrom n nn (\{f\} p) =
(let (nn', p') = labelfrom nn (nn + 1) p in
(nn', \{n\}f p'))"
| "labelfrom n nn (\[f\] p) =
(let (nn', p') = labelfrom nn (nn + 1) p in
(nn', \{n\}[f] p'))"
| "labelfrom n nn (p ⊕ q) =
(let (nn', p') = labelfrom n nn p in
let (nn'', q') = labelfrom n nn' q in
(nn'', p' ⊕ q'))"
| "labelfrom n nn (\{unicast(fip, fmsg). p \[\} p) =
(let (nn', p') = labelfrom nn (nn + 1) p in
(let (nn'', q') = labelfrom nn' (nn' + 1) q in
(nn'', \{n\}unicast(fip, fmsg). p' ⊕ q'))"
| "labelfrom n nn (\{broadcast(fmsg). p) =
(let (nn', p') = labelfrom nn (nn + 1) p in
(nn', \{n\}broadcast(fmsg). p'))"
| "labelfrom n nn (\{groupcast(fipset, fmsg). p) =
(let (nn', p') = labelfrom nn (nn + 1) p in
(nn', \{n\}groupcast(fipset, fmsg). p'))"
| "labelfrom n nn (\{send(fmsg). p) =
(let (nn', p') = labelfrom nn (nn + 1) p in
(nn', \{n\}send(fmsg). p'))"
| "labelfrom n nn (\{deliver(fdata). p) =
(let (nn', p') = labelfrom nn (nn + 1) p in
(nn', \{n\}deliver(fdata). p'))"
| "labelfrom n nn (\{receive(fmsg). p) =
(let (nn', p') = labelfrom nn (nn + 1) p in
(nn', \{n\}receive(fmsg). p'))"
| "labelfrom n nn (call(fargs)) = (nn - 1, call(fargs))"

primrec labelmap :: "('a ⇒ 'b) ⇒ ('s, 'm, 'p, 'a) seqp ⇒ ('s, 'm, 'p, 'b) seqp"
where
"labelmap lf (\{f\} p) = \{lf \}f (labelmap lf p)"
| "labelmap lf (\[f\] p) = \{lf \}[f] (labelmap lf p)"
| "labelmap lf (p ⊕ q) = (labelmap lf p) ⊕ (labelmap lf q)"
| "labelmap lf (\{unicast(fip, fmsg). p \[\} p) = \{lf \}unicast(fip, fmsg). (labelmap lf p) ⊕ (labelmap lf q)"
| "labelmap lf (\{broadcast(fmsg). p) = \{lf \}broadcast(fmsg). (labelmap lf p)"
| "labelmap lf (\{groupcast(fipset, fmsg). p) = \{lf \}groupcast(fipset, fmsg). (labelmap lf p)"
| "labelmap lf (\{send(fmsg). p) = \{lf \}send(fmsg). (labelmap lf p)"
| "labelmap lf (\{deliver(fdata). p) = \{lf \}deliver(fdata). (labelmap lf p)"
| "labelmap lf (\{receive(fmsg). p) = \{lf \}receive(fmsg). (labelmap lf p)"
| "labelmap lf (call(fargs)) = call(fargs)"

datatype 'pn label =
LABEL 'pn int ("_-:" [1000, 1000] 999)

instantiation "label" :: (ord) ord
begin

fun less_eq_label :: "'a label ⇒ 'a label ⇒ bool"
where "(\ll::n1) ≤ (\ll::n2) = (\ll = \ll ∧ n1 ≤ n2)"

definition less_label: "(\ll::'a label) < \ll ←→ \ll ≤ \ll ∧ ¬ (\ll ≤ \ll)"

59
instance .. end

abbreviation labelled :: 
"'p ⇒ ('s, 'm, 'p, 'a) seqp ⇒ ('s, 'm, 'p, 'p label) seqp"
where "labelled pn p ≡ labelmap (λl. LABEL pn l) (snd (labelfrom 0 1 p))"
end

9 A custom tactic for showing invariants via control terms

theory Inv_Cterms
importsAWN_Labels
begin

This tactic tries to solve a goal by reducing it to a problem over (local) cterms (using one of the cterms_intros intro rules); expanding those to consider all process names (using one of the ctermsl_cases destruction rules); simplifying each (using the cterms_env simplification rules); splitting them up into separate subgoals; replacing the derivative term with a variable; ‘executing’ a transition of each term; and then simplifying.

The tactic can stop after applying introduction rule ("inv_cterms (intro_only)"), or after having generated the verification condition subgoals and before having simplified them ("inv_cterms (vcs_only)"). It takes arguments to add or remove simplification rules ("simp add: lemmas names"); to add forward rules on assumptions (to introduce previously proved invariants; "inv add: lemmas names"); or to add elimination rules that solve any remaining subgoals ("solve: lemmas names").

To configure the tactic for a set of transition rules:

1. add elimination rules: declare seqpTEs [cterms seqte]
2. add rules to replace derivative terms: declare elimders [cterms_elimders]

To configure the tactic for a process environment (Γ):

1. add simp rules: declare Γ.simps [cterms.env]
2. add case rules: declare aodv_proc_cases [ctermsl_cases]
3. add invariant intros declare seq.invariant.ctermsI [OF aodv_wf aodv_control_within aodv_simple_labels, ctermsl_intros]
   seq.step.invariant.ctermsI [OF aodv_wf aodv_control_within aodv_simple_labels, ctermsl_intros]

lemma has.ctermsI: "p ∈ ctermsI Γ ⇒ p ∈ ctermsI Γ".

ML {*
structure CtermsElimders = Named_Thms
 (val name = @binding "cterms_elimders"
 val description = "Rules for truncating sequential process terms")
val cterms_elimders_add = Thm.declaration_attribute CtermsElimders.add_thm
val cterms_elimders_del = Thm.declaration_attribute CtermsElimders.del_thm

structure CtermsTE = Named_Thms
 (val name = @binding "cterms_seqte"
 val description = "Elimination rules for sequential process terms")
val cterms_seqte_add = Thm.declaration_attribute CtermsTE.add_thm
val cterms_seqte_del = Thm.declaration_attribute CtermsTE.del_thm

structure CtermsEnvRules = Named_Thms
 (val name = @binding "cterms_env"
 val description = "Simplification rules for sequential process environments")
val cterms_env_add = Thm.declaration_attribute CtermsEnvRules.add_thm
val cterms_env_del = Thm.declaration_attribute CtermsEnvRules.del_thm

structure CtermslCases = Named_Thms
 (val name = @binding "ctermsl_cases"
 val description = "Destruction rules for case splitting ctermsl")
val ctermsl_cases_add = Thm.declaration_attribute CtermslCases.add_thm
val ctermsl_cases_del = Thm.declaration_attribute CtermslCases.del_thm

structure CtermsIntros = Named_Thms
  (val name = @{binding "cterms_intros"}
   val description = "Introduction rules from cterms")
val cterms_intros_add = Thm.declaration_attribute CtermsIntros.add_thm
val cterms_intros_del = Thm.declaration_attribute CtermsIntros.del_thm

structure CtermsInvs = Named_Thms
  (val name = @{binding "cterms_invs"}
   val description = "Invariants to try to apply at each vc")
val cterms_invs_add = Thm.declaration_attribute CtermsInvs.add_thm
val cterms_invs_del = Thm.declaration_attribute CtermsInvs.del_thm

structure CtermsFinal = Named_Thms
  (val name = @{binding "cterms_final"}
   val description = "Elimination rules to try on each vc after simplification")
val cterms_final_add = Thm.declaration_attribute CtermsFinal.add_thm
val cterms_final_del = Thm.declaration_attribute CtermsFinal.del_thm

fun simp_only thms ctxt =
  asm_full_simp_tac
  (Context.proof_map
   (Simplifier.map_ss (Raw_Simplifier.clear_simpset
     #> fold Simplifier.add_simp thms))
    ctxt)

(* shallow_simp is useful for mopping up assumptions before really trying to simplify. Perhaps surprisingly, this saves minutes in some of the proofs that use a lot of invariants of the form (l = P(_:n --> P). *)
fun shallow_simp ctxt =
  let val ctxt' = Config.put simp_depth_limit 2 ctxt in
   TRY o safe_asm_full_simp_tac ctxt'
  end

fun create_vcs ctxt i =
  let val main_simp_thms = CtermsEnvRules.get ctxt
    val ctermsl_cases = CtermslCases.get ctxt
  in
    dtac @{thm has_ctermsl} i
    THEN_ELSE (dmatch_tac ctermsl_cases i
      THEN
      TRY (REPEAT_ALL_NEW (ematch_tac [@{thm disjE}] i)
        THEN
        (PARALLEL_GOALS (ALLGOALS
          (fn i => simp_only main_simp_thms ctxt i
            THEN
            TRY (REPEAT_ALL_NEW (ematch_tac [@{thm disjE}] i)))))
        , all_tac)
    end

fun try_invs ctxt =
  let val inv_thms = CtermsInvs.get ctxt
    fun fapp thm = TRY o (EVERY' (ftac thm :: replicate (Thm.nprems_of thm - 1) assume_tac))
  in
    EVERY' (map fapp inv_thms)
  end

fun try_final ctxt =
  let val final_thms = CtermsFinal.get ctxt
    fun eapp thm = EVERY' (etac thm :: replicate (Thm.nprems_of thm - 1) assume_tac)
  in
    TRY o (FIRST' (map eapp final_thms))
  end
fun each ctxt =
  (EVERY' ((ematch_tac (CtermsElimders.get ctxt) :: replicate 2 assume_tac))
    THEN' simp_only @{thms labels_psimps} ctxt
    THEN' (ematch_tac (CtermsTE.get ctxt)
      THEN_ALL_NEW
      (fn j => simp_only [@{thm mem_Collect_eq}] ctxt j
        THEN REPEAT (etac @{thm exE} j)
        THEN REPEAT (etac @{thm conjE} j)))
    ORELSE' (SOLVED' (clarsimp_tac ctxt))
  )

fun simp_all ctxt =
  let val ctxt' =
    ctxt |> fold Splitter.add_split [@{thm split_if_asm}]
  in
    (PARALLEL.Goals o ALLGOALS o shallow_simp) ctxt
    THEN
    TRY ((CHANGED_PROP (PARALLEL.Goals (ALLGOALS (asm_full_simp_tac ctxt' THEN' (try_final ctxt)))))
  end

fun intro_and_invs ctxt i =
  let val cterms_intros = CtermsIntros.get ctxt in
    match_tac cterms_intros i
    THEN (PARALLEL.Goals (ALLGOALS (try_invs ctxt)))
  end

fun process_vcs ctxt _ =
  ALLGOALS (create_vcs ctxt ORELSE' (SOLVED' (clarsimp_tac ctxt)))
  THEN (PARALLEL.Goals (ALLGOALS (TRY o (each ctxt))))

val intro_onlyN = "intro_only"
val vcs_onlyN = "vcs_only"
val invN = "inv"
val solveN = "solve"

val inv_cterms_options =
  (Args.parens (Args.$$$ intro_onlyN) >> K intro_and_invs ||
   Args.parens (Args.$$$ vcs_onlyN) >> K (fn ctxt => intro_and_invs ctxt
     THEN' (PARALLEL.Goals (ALLGOALS (try_invs ctxt))))
   || Scan.succeed (fn ctxt => intro_and_invs ctxt
     THEN' process_vcs ctxt
     THEN' K (simp_all ctxt)))

val inv_cterms_setup =
  Method.setup @{binding inv_cterms}
  (Scan.lift inv_cterms_options --| Method.sections
   (((Args.$$ invN -- Args.add -- Args.colon >> K (I, cterms_invs_add))
   :: (Args.$$ solveN -- Args.colon >> K (I, cterms_final_add))
   :: Simplifier.simp_modifiers)
   >> (fn tac => SIMPLE_METHOD' o tac))
  "Solve invariants by considering all (interesting) control terms."

declare
  insert_iff [cterms_env]
  Un_insert_right [cterms_env]
  sup_bot_right [cterms_env]
10 Configure the inv-cterms tactic for sequential processes

theory AWN_SOS_Labels
imports AWN_SOS Inv_Cterms
begin

lemma elimder_guard:
  assumes "p = {l}⟨fg⟩ qq"
  and "l' ∈ labels Γ q"
  and "((ξ, p), a, (ξ', q)) ∈ seqp_sos Γ"
  obtains p' where "p = {l}⟨fg⟩ p'"
    and "l' ∈ labels Γ qq"
  using assms by auto

lemma elimder_assign:
  assumes "p = {l}[fa] qq"
  and "l' ∈ labels Γ q"
  and "((ξ, p), a, (ξ', q)) ∈ seqp_sos Γ"
  obtains p' pp' where "p = {l}[fa] p'"
    and "l' ∈ labels Γ qq"
  using assms by auto

lemma elimder_ucast:
  assumes "p = {l}unicast(fip, fmsg).q1 ⊿ q2"
  and "l' ∈ labels Γ q"
  and "((ξ, p), a, (ξ', q)) ∈ seqp_sos Γ"
  obtains p' pp' where "p = {l}unicast(fip, fmsg).p' ⊿ pp'"
    and "case a of unicast _ _ ⇒ l' ∈ labels Γ q1 
      _ ⇒ l' ∈ labels Γ q2"
  using assms by simp (erule seqpTEs, auto)

lemma elimder_bcast:
  assumes "p = {l}broadcast(fmsg).qq"
  and "l' ∈ labels Γ q"
  and "((ξ, p), a, (ξ', q)) ∈ seqp_sos Γ"
  obtains p' where "p = {l}broadcast(fmsg). p'"
    and "l' ∈ labels Γ qq"
  using assms by auto

lemma elimder_gcast:
  assumes "p = {l}groupcast(fips, fmsg).qq"
  and "l' ∈ labels Γ q"
  and "((ξ, p), a, (ξ', q)) ∈ seqp_sos Γ"
  obtains p' where "p = {l}groupcast(fips, fmsg). p'"
    and "l' ∈ labels Γ qq"
  using assms by auto

lemma elimder_send:
  assumes "p = {l}send(fmsg).qq"
  and "l' ∈ labels Γ q"
  and "((ξ, p), a, (ξ', q)) ∈ seqp_sos Γ"
  obtains p' where "p = {l}send(fmsg). p'"
    and "l' ∈ labels Γ qq"
  using assms by auto

lemma elimder_deliver:
  assumes "p = {l}deliver(fdata).qq"
  and "l' ∈ labels Γ q"
  and "((ξ, p), a, (ξ', q)) ∈ seqp_sos Γ"
  obtains p' where "p = {l}deliver(fdata). p'"
    and "l' ∈ labels Γ qq"
  using assms by auto
obtains $p'$ where "$p = \{l\} deliver(fdata).p'\$
and "$l' \in \text{labels } \Gamma \; qq"
using \text{assms by auto}

lemma elimder_receive:
assumes "$p = \{l\} receive(fmsg).qq"
and "$l' \in \text{labels } \Gamma \; q"
and "((\xi, p), a, (\xi', q)) \in \text{seqp_sos } \Gamma\"
obtains $p'$ where "$p = \{l\} receive(fmsg).p'\$
and "$l' \in \text{labels } \Gamma \; qq"
using \text{assms by auto}

lemmas elimders =
elimder_guard
elimder_assign
elimder_uctast
elimder_bcast
elimder_gcast
elimder_send
elimder_deliver
elimder_receive
declare
seqpTEs [cterms_seqte]
elimders [cterms_elimders]
end

11 Lemmas for partial networks

theory Pnet
importsAWN_SOS Invariants
begin

These lemmas mostly concern the preservation of node structure by pnet_sos transitions.

lemma pnet_maintains_dom:
assumes "\((s, a, s')\) \in \text{trans } (pnet np p)"
shows "net_ips s = net_ips s'"
using \text{assms proof (induction } p \text{ arbitrary: } s \; a \; s')
fix \; i \; R \; s \; a \; s'
assume "\((s, a, s')\) \in \text{trans } (pnet np \langle i; R \rangle)"
hence "\((s, a, s')\) \in \text{node_sos } \text{trans } (np \langle i \rangle)" ..
thus "net_ips s = net_ips s'"
by (rule node_sos.cases) simp_all
next
fix \; p1 \; p2 \; s \; a \; s'
assume "\forall s \; a \; s'. \; \text{\((s, a, s')\) } \in \text{trans } (pnet np \langle p1 \mid p2 \rangle) \implies \text{net_ips } s = \text{net_ips } s'"
and "\forall a \; s'. \; \text{\((s, a, s')\) } \in \text{trans } (pnet np p1) \implies \text{net_ips } s = \text{net_ips } s'"
and "\forall a \; s'. \; \text{\((s, a, s')\) } \in \text{trans } (pnet np p2) \implies \text{net_ips } s = \text{net_ips } s'"
thus "net_ips s = net_ips s'"
by simp (erule pnet_sos.cases, simp_all)
qed

lemma pnet_net_ips_net_tree_ips [elim]:
assumes "s \in \text{reachable } (pnet np p) \; I"
shows "net_ips s = net_tree_ips p"
using \text{assms proof induction}
fix \; s
assume "s \in \text{init } (pnet np p)"
thus "net_ips s = net_tree_ips p"
proof (induction \; p \text{ arbitrary: } s)
fix \; i \; R \; s
assume "s \in \text{init } (pnet np \langle i; R \rangle)"
then obtain ns where "s = \text{NodeS } i \; ns \; R" ..
thus "net_ips s = net_tree_ips (i; R)"
    by simp
next
  fix p1 p2 s
  assume IH1: "\s. s ∈ init (pnet np p1) ⟹ net_ips s = net_tree_ips p1"
    and IH2: "\s. s ∈ init (pnet np p2) ⟹ net_ips s = net_tree_ips p2"
    and "s ∈ init (pnet np (p1 || p2))"
  from this(3) obtain s1 s2 where "s1 ∈ init (pnet np p1)"
    and "s2 ∈ init (pnet np p2)"
    and "s = SubnetS s1 s2" by auto
  from this(1-2) have "net_ips s1 = net_tree_ips p1"
    and "net_ips s2 = net_tree_ips p2"
    using IH1 IH2 by auto
  with 's = SubnetS s1 s2' show "net_ips s = net_tree_ips (p1 || p2)" by auto
qed
next
  fix s a s'
  assume "(s, a, s') ∈ trans (pnet np p)"
    and "net_ips s = net_tree_ips p"
  from this(1) have "net_ips s = net_ips s'"
    by (rule pnet_maintains_dom)
  with 'net_ips s = net_tree_ips p' show "net_ips s' = net_tree_ips p" by simp
qed

lemma pnet_init_net_ips_net_tree_ips:
  assumes "s ∈ init (pnet np p)"
  shows "net_ips s = net_tree_ips p"
  using assms(1) by (rule reachable_init [THEN pnet_net_ips_net_tree_ips])

lemma pnet_init_in_net_ips_in_net_tree_ips [elim]:
  assumes "s ∈ init (pnet np p)"
    and "i ∈ net_ips s"
  shows "i ∈ net_tree_ips p"
  using assms by (clarsimp dest!: pnet_init_net_ips_net_tree_ips)

lemma pnet_init_in_net_tree_ips_in_net_ips [elim]:
  assumes "s ∈ init (pnet np p)"
    and "i ∈ net_tree_ips p"
  shows "i ∈ net_ips s"
  using assms by (clarsimp dest!: pnet_init_net_ips_net_tree_ips)

lemma pnet_init_not_in_net_tree_ips_not_in_net_ips [elim]:
  assumes "s ∈ init (pnet np p)"
    and "i ∉ net_tree_ips p"
  shows "i ∉ net_ips s"
proof
  assume "i ∈ net_ips s"
  with assms(1) have "i ∈ net_tree_ips p" ..
  with assms(2) show False ..
qed

lemma net_node_reachable_is_node:
  assumes "st ∈ reachable (pnet np ⟨ii; R⟩) I"
  shows "∃ns R. st = NodeS ii ns R"
  using assms proof induct
    fix s
    assume "s ∈ init (pnet np ⟨ii; R⟩)"
    thus "∃ns R. s = NodeS ii ns R"
      by (rule pnet_node_init') simp
next
  fix s a s'
  assume "s ∈ reachable (pnet np ⟨ii; R⟩) I"
  and "∃ns R. s = NodeS ii ns R"
and "(s, a, s') \in \text{trans} \ (\text{pnet np } (i; R))"
and "I a"
thus "\exists ns R. s' = NodeS ii ns R"
by (auto simp add: trans_node_comp dest!: node_sos_dest)
qed

lemma partial_net_preserves_subnets:
assumes "(SubnetS s t, a, st') \in \text{pnet_sos } (\text{trans} \ (\text{pnet np } p1)) (\text{trans} \ (\text{pnet np } p2))"
shows "\exists s' t'. st' = SubnetS s' t'"
using assms by cases simp_all

lemma net_par_reachable_is_subnet:
assumes "st \in \text{reachable} \ (\text{pnet np } (p1 \parallel p2)) I"
shows "\exists s t. st = SubnetS s t"
using assms by (induct (auto dest!: partial_net_preserves_subnets)

lemma reachable_par_subnet_induct [consumes, case_names init step]:
assumes "SubnetS s t \in \text{reachable} \ (\text{pnet np } (p1 \parallel p2)) I"
and init: "\forall s t. SubnetS s t \in \text{init} \ (\text{pnet np } (p1 \parallel p2)) \Longrightarrow P s t"
and step: "\forall s t s' t' a. [
SubnetS s t \in \text{reachable} \ (\text{pnet np } (p1 \parallel p2)) I; 
P s t; (SubnetS s t, a, SubnetS s' t') \in \text{trans} \ (\text{pnet np } (p1 \parallel p2))) I a] 
\Longrightarrow P s' t'"
shows "P s t"
using assms(1) proof (induction "SubnetS s t" arbitrary: s t)
fix s t
assume "SubnetS s t \in \text{init} \ (\text{pnet np } (p1 \parallel p2))"
with init show "P s t".

next
fix st a s' t'
assume "st \in \text{reachable} \ (\text{pnet np } (p1 \parallel p2)) I"
and tr: "(st, a, SubnetS s' t') \in \text{trans} \ (\text{pnet np } (p1 \parallel p2))"
and "I a"
and IH: "\forall s t. st = SubnetS s t \Longrightarrow P s t"
from this(1) obtain s t where "st = SubnetS s t"
and str: "SubnetS s t \in \text{reachable} \ (\text{pnet np } (p1 \parallel p2)) I"
by (metis net_par_reachable_is_subnet)
note this(2)
moreover from IH and 'st = SubnetS s t' have "P s t" .
moreover from 'st = SubnetS s t' and tr
have "(SubnetS s t, a, SubnetS s' t') \in \text{trans} \ (\text{pnet np } (p1 \parallel p2))" by simp
ultimately show "P s' t'"
using 'I a' by (rule step)
qed

lemma subnet_reachable:
assumes "SubnetS s1 s2 \in \text{reachable} \ (\text{pnet np } (p1 \parallel p2)) TT"
shows "s1 \in \text{reachable} \ (\text{pnet np } p1) TT"
"s2 \in \text{reachable} \ (\text{pnet np } p2) TT"
proof -
from assms have "s1 \in \text{reachable} \ (\text{pnet np } p1) TT"
\land s2 \in \text{reachable} \ (\text{pnet np } p2) TT"
proof (induction rule: reachable_par_subnet_induct)
fix s1 s2
assume "SubnetS s1 s2 \in \text{init} \ (\text{pnet np } (p1 \parallel p2))"
thus "s1 \in \text{reachable} \ (\text{pnet np } p1) TT"
\land s2 \in \text{reachable} \ (\text{pnet np } p2) TT"
by (auto dest: reachable_init)

next
case (step s1 s2 s1' s2' a)
hence "SubnetS s1 s2 \in \text{reachable} \ (\text{pnet np } (p1 \parallel p2)) TT"
and sr1: "s1 \in \text{reachable} \ (\text{pnet np } p1) TT"
and sr2: "s2 \in \text{reachable} \ (\text{pnet np } p2) TT"
and "(SubnetS s1 s2, a, SubnetS s1' s2') \in \text{trans} \ (\text{pnet np } (p1 \parallel p2))" by auto
from this have "(SubnetS s1 s2, a, SubnetS s1' s2') ∈ pnet sos (trans (pnet np p1)) (trans (pnet np p2))" by simp
thus "s1' ∈ reachable (pnet np p1) TT ∧ s2' ∈ reachable (pnet np p2) TT" by cases (insert sr1 sr2, auto elim: reachable_step)

by simp
thus "s1 ∈ reachable (pnet np p1) TT" "s2 ∈ reachable (pnet np p2) TT" by auto

lemma delivered_to_node [elim]:
assumes "s ∈ reachable (pnet np (ii; R)) TT" and "(s, i:deliver(d), s') ∈ trans (pnet np (ii; R))"
shows "i = ii"
proof -
from assms(1) obtain P R where "s = NodeS ii P R"
with assms(2) show "i = ii"
by (clarsimp simp add: trans_node_comp elim!: node_deliverTE')

qed

lemma delivered_to_net_ips:
assumes "s ∈ reachable (pnet np p) TT" and "(s, i:deliver(d), s') ∈ trans (pnet np p)"
shows "i ∈ net_ips s"
using assms proof (induction p arbitrary: s s')
fix ii R, i s s'
assume sr: "s ∈ reachable (pnet np (ii; R)) TT" and "(s, i:deliver(d), s') ∈ trans (pnet np (ii; R))"
from this(2) have tr: "(s, i:deliver(d), s') ∈ node sos (trans (np ii))" by simp
from sr obtain P R where [simp]: "s = NodeS ii P R"
by (metis net_node reachable_is_node)
moreover from tr obtain P' R' where [simp]: "s' = NodeS ii P' R'"
by (metis node sos dest)
ultimately have "i = ii" using tr by auto
thus "i ∈ net_ips s" by simp

next
fix p1 p2 s s'
assume IH1: "∀ s s'. [ s ∈ reachable (pnet np p1) TT; (s, i:deliver(d), s') ∈ trans (pnet np p1) ] ⇒ i ∈ net_ips s"
and IH2: "∀ s s'. [ s ∈ reachable (pnet np p2) TT; (s, i:deliver(d), s') ∈ trans (pnet np p2) ] ⇒ i ∈ net_ips s"
and sr: "s ∈ reachable (pnet np (p1 || p2)) TT" and tr: "(s, i:deliver(d), s') ∈ trans (pnet np (p1 || p2))"
from tr have "(s, i:deliver(d), s') ∈ pnet sos (trans (pnet np p1)) (trans (pnet np p2))" by simp
thus "i ∈ net_ips s" using (rule partial deliverTE)

proof (rule partial deliverTE)
fix s1 s1' s2
assume "s = SubnetS s1 s2" and "s' = SubnetS s1' s2"
and tr: "(s1, i:deliver(d), s1') ∈ trans (pnet np p1)"
from sr have "s1 ∈ reachable (pnet np p1) TT" by (auto simp only: 's = SubnetS s1 s2' elim: subnet reachable)

hence "i ∈ net_ips s" using sr by (rule IH1)
thus "i ∈ net_ips s" by (simp add: 's = SubnetS s1 s2')

next
fix s2 s2' s1
assume "s = SubnetS s1 s2" and "s' = SubnetS s1 s2'" and tr: "(s2, i:deliver(d), s2') ∈ trans (pnet np p2)"
from sr have "s2 ∈ reachable (pnet np p2) TT" by (auto simp only: 's = SubnetS s1 s2' elim: subnet reachable)
hence \( i \in \text{net_ips s2} \) using \( \text{tr by (rule IH2)} \)
thus \( i \in \text{net_ips s} \) by (simp add: \( 's = \text{SubnetS s1 s2'} \))
qed

lemma \( \text{wf_net_tree_net_ips_disjoint [elim]} \):
assumes \( \text{"wf_net_tree (p1 \parallel p2)"} \)
and \( "s1 \in \text{reachable (pnet np p1) S}" \)
and \( "s2 \in \text{reachable (pnet np p2) S}" \)
shows \( "\text{net_ips s1 \cap net_ips s2 = {}}" \)
proof -
from \( \text{"wf_net_tree (p1 \parallel p2)"} \) have \( "\text{net_tree_ips p1 \cap net_tree_ips p2 = {}}" \) by auto
moreover from \assm{2} have \( "\text{net_ips s1 = net_tree_ips p1}" .. \)
moreover from \assm{3} have \( "\text{net_ips s2 = net_tree_ips p2}" .. \)
ultimately show \( ?\text{thesis by simp} \)
qed

lemma \( \text{init_mapstate_Some_aodv_init [elim]} \):
assumes \( "s \in \text{init (pnet np p)}" \)
and \( "\text{netmap s i = Some v}" \)
shows \( "v \in \text{init (np i)}" \)
proof (induction \( p \) arbitrary: \( s \))
fix \( ii R s \)
assume \( "s \in \text{init (pnet np \langle ii; R \rangle)}" \)
and \( "\text{netmap s i = Some v}" \)
from \assm{1} obtain \( ns \) where \( "s = \text{NodeS ii ns R}" \)
and \( "ns \in \text{init (np ii)}" .. \)
moreover from \assm{1} and \( \text{'netmap s i = Some v'} \) have \( "i = ii" \)
by simp (metis domI domIff)
ultimately show \( "v \in \text{init (np i)}" \)
using \( \text{'netmap s i = Some v'} \) by simp
next
fix \( p1 p2 s \)
assume \( \text{IH1: } \[\forall s. s \in \text{init (pnet np p1) \implies netmap s i = Some v \implies v \in \text{init (np i)}}\] \)
and \( \text{IH2: } \[\forall s. s \in \text{init (pnet np p2) \implies netmap s i = Some v \implies v \in \text{init (np i)}}\] \)
and \( "s \in \text{init (pnet np (p1 \parallel p2))}" \)
and \( "\text{netmap s i = Some v}" \)
from \assm{3} obtain \( s1 s2 \) where \( "s = \text{SubnetS s1 s2}" \)
and \( "s1 \in \text{init (pnet np p1)}" \)
and \( "s2 \in \text{init (pnet np p2)}" \) by auto
from \assm{1} and \( \text{'netmap s i = Some v'} \) have \( "\text{netmap s1 i = Some v \lor netmap s2 i = Some v}" \) by auto
thus \( "v \in \text{init (np i)}" \)
proof
assume \( "\text{netmap s1 i = Some v}" \)
with \( \text{\( s1 \in \text{init (pnet np p1)} \)'} \) show \( ?\text{thesis by (rule IH1)} \)
next
assume \( "\text{netmap s2 i = Some v}" \)
with \( \text{\( s2 \in \text{init (pnet np p2)} \)'} \) show \( ?\text{thesis by (rule IH2)} \)
qed

lemma \( \text{reachable_connect_netmap [elim]} \):
assumes \( "s \in \text{reachable (pnet np n) TT}" \)
and \( "\langle s, \text{connect(i, i'), s'} \rangle \in \text{trans (pnet np n)}" \)
shows \( "\text{netmap s'} = \text{netmap s}" \)
proof (induction \( n \) arbitrary: \( s s' \))
fix \( ii R i s s' \)
assume \( \text{sr: } \[\forall s. s \in \text{reachable (pnet np \langle ii; R_i \rangle) TT} \implies netmap s' = \text{netmap s}\] \)
and \( "\langle s, \text{connect(i, i'), s'} \rangle \in \text{trans (pnet np \langle ii; R_i \rangle)}" \)
from \assm{2} have \( \text{tr: } \[\langle s, \text{connect(i, i'), s'} \rangle \in \text{node_sos (trans (np ii))}\] .. \)
from \( \text{sr} \) obtain \( R \) where \( "s = \text{NodeS ii p R}" \)
by (metis \text{net_node_reachable_is_node})
with \( \text{tr} \) show \( "\text{netmap s'} = \text{netmap s}" \)
by (auto elim!: node_sos.cases)

next
fix p1 p2 s s'
assume IH1: "∀ s s'. [ s ∈ reachable (pnet np p1) TT; (s, connect(i, i'), s') ∈ trans (pnet np p1) ] ⇒ netmap s' = netmap s"
and IH2: "∀ s s'. [ s ∈ reachable (pnet np p2) TT; (s, connect(i, i'), s') ∈ trans (pnet np p2) ] ⇒ netmap s' = netmap s"
and sr: "s ∈ reachable (pnet np (p1 || p2)) TT"
and tr: "(s, connect(i, i'), s') ∈ trans (pnet np (p1 || p2))"
from tr have "(s, connect(i, i'), s') ∈ pnet_sos (trans (pnet np p1)) (trans (pnet np p2))"
by simp
thus "netmap s' = netmap s"
proof cases
fix s1 s1' s2 s2'
assume "s = SubnetS s1 s2"
and "s' = SubnetS s1' s2''" and tr1: "(s1, connect(i, i'), s1') ∈ trans (pnet np p1)"
and tr2: "(s2, connect(i, i'), s2') ∈ trans (pnet np p2)"
from this(1) and sr
have "SubnetS s1 s2 ∈ reachable (pnet np (p1 || p2)) TT" by simp
hence sr1: "s1 ∈ reachable (pnet np p1) TT"
and sr2: "s2 ∈ reachable (pnet np p2) TT"
by (auto intro: subnet_reachable)
from sr1 tr1 have "netmap s1' = netmap s1" by (rule IH1)
moreover from sr2 tr2 have "netmap s2' = netmap s2" by (rule IH2)
ultimately show "netmap s' = netmap s"
using 's = SubnetS s1 s2' and 's' = SubnetS s1' s2'' by simp
qed simp_all
qed

lemma reachable_disconnect_netmap [elim]:
assumes "s ∈ reachable (pnet np n) TT"
and "(s, disconnect(i, i'), s') ∈ trans (pnet np n)"
shows "netmap s' = netmap s"
using assms proof (induction n arbitrary: s s')
fix ii R, s s'
assume sr: "s ∈ reachable (pnet np (⟨ii; R⟩)) TT"
and "(s, disconnect(i, i'), s') ∈ trans (pnet np (⟨ii; R⟩))"
from this(2) have tr: "(s, disconnect(i, i'), s') ∈ node_sos (trans (np ii))" ..
from sr obtain p R where "s = NodeS ii p R" by (metis net_node_reachable_is_node)
with tr show "netmap s' = netmap s" by (auto elim!: node_sos.cases)
next
fix p1 p2 s s'
assume IH1: "∀ s s'. [ s ∈ reachable (pnet np p1) TT; (s, disconnect(i, i'), s') ∈ trans (pnet np p1) ] ⇒ netmap s' = netmap s"
and IH2: "∀ s s'. [ s ∈ reachable (pnet np p2) TT; (s, disconnect(i, i'), s') ∈ trans (pnet np p2) ] ⇒ netmap s' = netmap s"
and sr: "s ∈ reachable (pnet np (p1 || p2)) TT"
and tr: "(s, disconnect(i, i'), s') ∈ trans (pnet np (p1 || p2))"
from tr have "(s, disconnect(i, i'), s') ∈ pnet_sos (trans (pnet np p1)) (trans (pnet np p2))"
by simp
thus "netmap s' = netmap s"
proof cases
fix s1 s1' s2 s2'
assume "s = SubnetS s1 s2"
and "s' = SubnetS s1' s2''" and tr1: "(s1, disconnect(i, i'), s1') ∈ trans (pnet np p1)"
and tr2: "(s2, disconnect(i, i'), s2') ∈ trans (pnet np p2)"
from this(1) and sr
have "SubnetS s1 s2 ∈ reachable (pnet np (p1 || p2)) TT" by simp
hence sr1: "s1 ∈ reachable (pnet np p1) TT"
and sr2: "s2 ∈ reachable (pnet np p2) TT"
by (auto intro: subnet_reachable)
from sr1 tr1 have "netmap s1' = netmap s1" by (rule IH1)
moreover from sr2 tr2 have "netmap s2' = netmap s2" by (rule IH2)
ultimately show "netmap s' = netmap s"
  using 's = SubnetS s1 s2' and 's' = SubnetS s1' s2'' by simp
qed simp_all

fun net_ip_action :: "('s, 'm seq_action) automaton ⇒ 'm node_action ⇒ ip ⇒ net_tree ⇒ 's net_state ⇒ 's net_state ⇒ bool"
where
"net_ip_action np a i (p1 || p2) (SubnetS s1 s2) (SubnetS s1' s2') =
((i ∈ net_ips s1 → ((s1, a, s1') ∈ trans (pnet np p1)
∧ s2' = s2 ∧ net_ip_action np a i p1 s1 s1'))
∧ (i ∈ net_ips s2 → ((s2, a, s2') ∈ trans (pnet np p2))
∧ s1' = s1 ∧ net_ip_action np a i p2 s2 s2'))"

lemma pnet_tau_single_node [elim]:
assumes "wf_net_tree p"
and "s ∈ reachable (pnet np p) TT"
and "(s, τ, s') ∈ trans (pnet np p)"
shows "∃i∈net_ips s. ((∀j. j ≠ i → netmap s' j = netmap s j)
∧ net_ip_action np τ i p s s')"
using assms proof (induction p arbitrary: s s')
  fix i
  assume "s ∈ reachable (pnet np (ii; R_i)) TT"
  and "(s, τ, s') ∈ trans (pnet np (ii; R_i))"
  from this obtain p R p' R' where "s = NodeS ii p R" and "s' = NodeS ii p' R'"
  by (metis (hide_lams, no_types) TT_True net_node_reachable_is_node
        reachable_step)
  hence "net_ips s = {ii}" and "net_ips s' = {ii}" by simp_all
  hence "∃i∈dom (netmap s). ∀j. j ≠ i → netmap s' j = netmap s j"
    by (simp add: net_ips_is_dom_netmap)
  thus "∃i∈net_ips s. ((∀j. j ≠ i → netmap s' j = netmap s j)
    ∧ net_ip_action np τ i (⟨ii; R_i⟩) s s')"
    by (simp add: net_ips_is_dom_netmap)
next
fix p1 p2 s s'
assume IH1: "∀s s'. [ wf_net_tree p1;
    s ∈ reachable (pnet np p1) TT;
    (s, τ, s') ∈ trans (pnet np p1) ]
⇒ ∃i∈net_ips s. ((∀j. j ≠ i → netmap s' j = netmap s j)
    ∧ net_ip_action np τ i p1 s s')"
and IH2: "∀s s'. [ wf_net_tree p2;
    s ∈ reachable (pnet np p2) TT;
    (s, τ, s') ∈ trans (pnet np p2) ]
⇒ ∃i∈net_ips s. ((∀j. j ≠ i → netmap s' j = netmap s j)
    ∧ net_ip_action np τ i p2 s s')"
and sr: "s ∈ reachable (pnet np (p1 || p2)) TT"
and "wf_net_tree (p1 || p2)"
and tr: "(s, τ, s') ∈ trans (pnet np (p1 || p2))"
from 'wf_net_tree (p1 || p2)′ have "net_tree_ips p1 ∩ net_tree_ips p2 = {}"
  and "wf_net_tree p1" and "wf_net_tree p2" by auto
from tr have "(s, τ, s') ∈ pnet_sos (trans (pnet np p1)) (trans (pnet np p2))" by simp
thus "∃i∈net_ips s. ((∀j. j ≠ i → netmap s' j = netmap s j)
    ∧ net_ip_action np τ i (p1 || p2) s s')"
proof cases
  fix s1 s1' s2
  assume subs: "s = SubnetS s1 s2"
  and subs': "s' = SubnetS s1' s2''
  and tr1: "(s1, τ, s1') ∈ trans (pnet np p1)"
  next
  fix s1 s1' s2
  assume subs: "s = SubnetS s1 s2"
  and subs': "s' = SubnetS s1' s2"
from sr have sr1: "s1 ∈ reachable (pnet np p1) TT"
  and "s2 ∈ reachable (pnet np p2) TT"
  by (simp_all only: subs) (erule subnet_reachable)+
  with 'net_tree_ips p1 ∩ net_tree_ips p2 = {'} have "dom(netmap s1) ∩ dom(netmap s2) = {'}
  by (metis net_ips_is_dom_netmap pnet_net_ips_net_tree_ips)
from 'wf_net_tree p1' sr1 tr1 obtain i where "i∈dom(netmap s1)"
  and "∀ j. j ≠ i → netmap s1' j = netmap s1 j"
  and "net_ip_action np τ i p1 s1 s1'"
  by (auto simp add: net_ips_is_dom_netmap dest!: IH1)
from this(1) and 'dom(netmap s1) ∩ dom(netmap s2) = {}'
  have "i ∉ dom(netmap s2)"
  by auto
with subs subs' tr1 'net_ip_action np τ i p1 s1 s1''
  have "net_ip_action np τ i (p1 ∥ p2) s s'"
  by (simp add: net_ips_is_dom_netmap)
moreover have "∀ j. j ≠ i → (netmap s1' ++ netmap s2) j = (netmap s1 + netmap s2) j"
proof (intro allI impl1)
fix j
assume "j ≠ i"
with * have "netmap s1' j = netmap s1 j" by simp
thus "(netmap s1' ++ netmap s2) j = (netmap s1 ++ netmap s2) j"
by (metis (hide_lams, mono_tags) map_add_Some_iff map_add_dom_app_simps(3))
qed
ultimately show ?thesis using 'i∈dom(netmap s1)'
  by (auto simp add: net_ips_is_dom_netmap)
next
fix s2 s2' s1
assume subs: "s = SubnetS s1 s2"
  and subs': "s' = SubnetS s1 s2'"
  and tr2: "(s2, τ, s2') ∈ trans (pnet np p2)"
from sr have "s1 ∈ reachable (pnet np p1) TT"
  and sr2: "s2 ∈ reachable (pnet np p2) TT"
  by (simp_all only: subs) (erule subnet_reachable)+
  with 'net_tree_ips p1 ∩ net_tree_ips p2 = {'} have "dom(netmap s1) ∩ dom(netmap s2) = {'}
  by (metis net_ips_is_dom_netmap pnet_net_ips_net_tree_ips)
from 'wf_net_tree p2' sr2 tr2 obtain i where "i∈dom(netmap s2)"
  and "∀ j. j ≠ i → (netmap s2' ++ netmap s2) j = (netmap s1 + netmap s2) j"
  by (auto simp add: net_ips_is_dom_netmap)
moreover have "∀ j. j ≠ i → (netmap s1' ++ netmap s2') j = (netmap s1 + netmap s2') j"
proof (intro allI impl1)
fix j
assume "j ≠ i"
with * have "netmap s2' j = netmap s2 j" by simp
thus "(netmap s1' ++ netmap s2') j = (netmap s1 + netmap s2') j"
by (metis (hide_lams, mono_tags) domD map_add_Some_iff map_add_dom_app_simps(3))
qed
ultimately show ?thesis using 'i∈dom(netmap s2)'
  by (clarsimp simp add: net_ips_is_dom_netmap)
  (metis domI dom_map_add map_add_find_right)
next
fix ii R s s'
assume sr: "s ∈ reachable (pnet np (ii; R)) TT"
lemma pnet_deliver_single_node [elim]:
  assumes "wf_net_tree p"
  and "s ∈ reachable (pnet np p) TT"
  and "(s, i:deliver(d), s') ∈ trans (pnet np p)"
  shows "∀ j. j ≠ i → netmap s' j = netmap s j ∧ net_ip_action np (i:deliver(d)) i p s s'"
and tr: "(s, i:deliver(d), s') ∈ trans (pnet np (ii; R))"
from this obtain p R p' R' where "s = NodeS ii p R" and "s' = NodeS ii p' R"
by (metis (hide_lams, no_types) TT_True net_node_reachable_is_node_reachable_step)

hence "net_ips s = {ii}" and "net_ips s' = {ii}" by simp_all
hence "∀ j. j ≠ ii → netmap s' j = netmap s j"
by simp

moreover from sr tr have "i = ii" by (rule delivered_to_node)
ultimately show "(∀ j. j ≠ i → netmap s' j = netmap s j)
∧ net_ip_action np (i:deliver(d)) i ((ii; R)) s s''" by simp

next
fix p1 p2 s s'
assume IH1: "∀ s s'. [ wf_net_tree p1;
s ∈ reachable (pnet np p1) TT;
(s, i:deliver(d), s') ∈ trans (pnet np p1) ]
⇒ (∀ j. j ≠ i → netmap s' j = netmap s j)
∧ net_ip_action np (i:deliver(d)) i p1 s s'"
and IH2: "∀ s s'. [ wf_net_tree p2;
s ∈ reachable (pnet np p2) TT;
(s, i:deliver(d), s') ∈ trans (pnet np p2) ]
⇒ (∀ j. j ≠ i → netmap s' j = netmap s j)
∧ net_ip_action np (i:deliver(d)) i p2 s s'"
and sr: "s ∈ reachable (pnet np (p1 ∥ p2)) TT"
and "wf_net_tree (p1 ∥ p2)"
and tr: "(s, i:deliver(d), s') ∈ trans (pnet np (p1 ∥ p2))"
from 'wf_net_tree (p1 ∥ p2)' have "net_tree_ips p1 ∩ net_tree_ips p2 = {}"
and "wf_net_tree p1"
and "wf_net_tree p2" by auto
from tr have "(s, i:deliver(d), s') ∈ pnet_sos (trans (pnet np p1)) (trans (pnet np p2))" by simp
thus "(∀ j. j ≠ i → netmap s' j = netmap s j)
∧ net_ip_action np (i:deliver(d)) i (p1 ∥ p2) s s''" by simp

proof cases
fix s1 s1' s2
assume subs: "s = SubnetS s1 s2"
and subs': "s' = SubnetS s1' s2"
and tr1: "(s1, i:deliver(d), s1') ∈ trans (pnet np p1)"
from sr have sr1: "s1 ∈ reachable (pnet np p1) TT"
and "s2 ∈ reachable (pnet np p2) TT"
by (simp_all only: subs) (erule subnet_reachable)+
with 'net_tree_ips p1 ∩ net_tree_ips p2 = {}' have "dom(netmap s1) ∩ dom(netmap s2) = {}"
by (metis net_ips_is_dom_netmap pnet_net_ips_net_tree_ips)
moreover from sr1 tr1 have "i ∈ net_ips s1" by (rule delivered_to_net_ips)
ultimately have "i ∉ dom(netmap s2)" by (auto simp add: net_ips_is_dom_netmap)
from 'wf_net_tree p1' sr1 tr1 have "∀ j. j ≠ i → netmap s1' j = netmap s1 j"
and "net_ip_action np (i:deliver(d)) i p1 s1 s1''"
by (auto dest!: IH1)
from subs subs' tr1 this(2) 'i ∉ dom(netmap s2)'
have "net_ip_action np (i:deliver(d)) i (p1 ∥ p2) s s''" by simp
moreover have "∀ j. j ≠ i → (netmap s1' ++ netmap s2) j = (netmap s1 ++ netmap s2) j"
by (auto simp add: net_ips_is_dom_netmap)

proof (intro allI_impl)
fix j
assume "j ≠ i"
with * have "netmap s1' j = netmap s1 j" by simp
thus "(netmap s1' ++ netmap s2) j = (netmap s1 ++ netmap s2) j"
by (metis (hide_lams, mono_tags) map_add_dom_app_simp (map_add_dom_app_simps (map_add_dom_app_simps (1))))
qed
ultimately show ?thesis using 'i ∉ net_ips s1' subs subs' by auto

next
fix s2 s2' s1
assume subs: "s = SubnetS s1 s2"
and \( s' = \text{SubnetS} \ s1 \ s2' \)

and \( \text{tr2}: (s2, i: \text{deliver}(d), s2') \in \text{trans} (\text{pnet np p2}) \)

from sr have \( s1 \in \text{reachable} (\text{pnet np p1} \ TT) \)

and \( \text{sr2}: s2 \in \text{reachable} (\text{pnet np p2} \ TT) \)

by (simp_all only: subs) (erule subnet_reachable)+

with \( \text{'net_tree_ips p1} \cap \text{net_tree_ips p2} = \{\} \)

have \( \text{dom}(\text{netmap s1}) \cap \text{dom}(\text{netmap s2}) = \{\} \)

by (metis net_ips_is_dom_netmap pnet_net_ips_net_tree_ips)

moreover from \( \text{sr2} \ \text{tr2} \)

have \( i \in \text{net_ips s2} \)

by (rule delivered_to_net_ips)

ultimately have \( i \not\in \text{dom}(\text{netmap s1}) \)

by (auto simp add: net_ips_is_dom_netmap)

from \( \text{wf_net_tree p2} \ \text{sr2} \ \text{tr2} \)

have \( \forall j. j \neq i \rightarrow \text{netmap s2'} \ j = \text{netmap s2} \ j \)

and \( \text{net_ip_action np (i: deliver(d)) i p2 s2 s2'} \)

by (auto dest!: IH2)

from \( \text{subs subs'} \ \text{tr2} \ this(2) \ 'i \not\in \text{dom(\text{netmap s1})}' \)

have \( \text{net_ip_action np (i: deliver(d)) i (p1} \parallel p2) s s' \)

by (simp add: net_ips_is_dom_netmap)

moreover have \( \forall j. j \neq i \rightarrow (\text{netmap s1} ++ \text{netmap s2'}) \ j = (\text{netmap s1} ++ \text{netmap s2}) \ j \)

proof (intro allI impI)

fix \ j \)

assume \( j \neq i \)

with \( * \)

have \( \text{netmap s2'} \ j = \text{netmap s2} \ j \)

by simp

thus \( (\text{netmap s1} ++ \text{netmap s2'}) \ j = (\text{netmap s1} ++ \text{netmap s2}) \ j \)

by (metis (hide_lams, mono_tags) domD map_add_Some_iff map_add_dom_app_simps(3))

qed

ultimately show \( ?\text{thesis} \) using \( i \in \text{net_ips s2'} \ \text{subs subs'} \) by auto

qed simp_all

qed

end

12 Lemmas for closed networks

theory Closed

imports Pnet

begin

lemma complete_net_preserves_subnets:

assumes \( (\text{SubnetS} \ s t, a, st') \in \text{cnet_sos (pnet_sos (trans (\text{pnet np p1}) (trans (\text{pnet np p2})))}) \)

shows \( \exists s' t'. st' = \text{SubnetS} \ s' t' \)

using assms by cases (auto dest: partial_net_preserves_subnets)

lemma complete_net_reachable_is_subnet:

assumes \( st \in \text{reachable (closed (\text{pnet np (p1} \parallel p2))) I} \)

shows \( \exists s t. st = \text{SubnetS} \ s t \)

using assms by induction (auto dest!: complete_net_preserves_subnets)

lemma closed_reachable_par_subnet_induct [consumes, case_names init step]:

assumes \( \text{SubnetS} \ s t \in \text{reachable (closed (\text{pnet np (p1} \parallel p2))) I} \)

and init: \( \forall s t. \text{SubnetS} \ s t \in \text{init (closed (\text{pnet np (p1} \parallel p2)))} \Rightarrow P \ s t \)

and step: \( \forall s t s' t' \ a. \]

\( \text{SubnetS} \ s t \in \text{reachable (closed (\text{pnet np (p1} \parallel p2))) I}; \]

\( P \ s t; (\text{SubnetS} \ s t, a, \text{SubnetS} s' t') \in \text{trans (closed (\text{pnet np (p1} \parallel p2)))}; \ I a \]

\( \Rightarrow P \ s' t' \)

shows \( P \ s t \)

using assms(1) proof (induction "\text{SubnetS} \ s t" arbitrary: s t)

fix s t

assume "\text{SubnetS} \ s t \in \text{init (closed (\text{pnet np (p1} \parallel p2)))}"

with init show "P s t" .

next

fix \ s t a s' t'

assume "st \in \text{reachable (closed (\text{pnet np (p1} \parallel p2))) I}"

and tr: "(st, a, \text{SubnetS} s' t') \in \text{trans (closed (\text{pnet np (p1} \parallel p2)))}"

and "I a"

and IH: "\forall s t. st = \text{SubnetS} \ s t \Rightarrow P \ s t"
from this(1) obtain \( st = \text{SubnetS } s \ t \) 
and "\( \text{SubnetS } s \ t \in \text{reachable } (\text{closed } (\text{pnet np (p1 || p2)))) \)" 
by (metis complete_net_reachable_is_subnet) 

note this(2) 
moreover from IH and 'st = SubnetS s t' have "P s t" 
moreover from tr and 'st = SubnetS s t' 
have "(SubnetS s t, a, SubnetS s t') \in \text{trans } (\text{closed } (\text{pnet np (p1 || p2))))" by simp 
ultimately show "P s t'" 
using 'I a' by (rule assms(3)) 

qed 

lemma reachable_closed_reachable_pnet [elim]: 
assumes "s \in \text{reachable } (\text{closed } (\text{pnet np n})) \ TT" 
shows "s \in \text{reachable } (\text{pnet np n}) \ TT" 
using assms proof (induction rule: reachable.induct) 
fix s s' a 
assume sr: "s \in \text{reachable } (\text{pnet np n}) \ TT" 
and "(s, a, s') \in \text{trans } (\text{closed } (\text{pnet np n}))" 
from this(2) have "(s, a, s') \in \text{cnet sos } (\text{trans } (\text{pnet np n}))" by simp 
thus "s' \in \text{reachable } (\text{pnet np n}) \ TT" 
by cases (insert sr, auto elim!: reachable_step) 
qed (auto elim: reachable_init) 

lemma closed_node_net_state [elim]: 
assumes "st \in \text{reachable } (\text{closed } (\text{pnet np ⟨ii; R⟩})) \ TT" 
obtains ξ p q R where "st = \text{NodeS ii ((ξ, p), q) R}" 
using assms by (metis net_node_reachable_is_node reachable_closed_reachable_pnet surj_pair) 

lemma closed_subnet_net_state [elim]: 
assumes "st \in \text{reachable } (\text{closed } (\text{pnet np (p1 || p2)))) \ TT" 
obtains s t where "st = \text{SubnetS } s \ t" 
using assms by (metis reachable_closed_reachable_pnet net_par_reachable_is_subnet) 

lemma closed_imp_pnet_trans [elim, dest]: 
assumes "(s, a, s') \in \text{trans } (\text{closed } (\text{pnet np n}))" 
shows "\exists a'. (s, a', s') \in \text{trans } (\text{pnet np n})" 
using assms by (auto elim!: cnet_sos.cases) 

lemma reachable_not_in_net_tree_ips [elim]: 
assumes "s \in \text{reachable } (\text{closed } (\text{pnet np n})) \ TT" 
and "i /∈ \text{net_tree_ips n}" 
shows "\text{netmap } s \ i = \text{None}" 
using assms proof induction 
fix s 
assume "s \in \text{init } (\text{closed } (\text{pnet np n}))" 
and "i /∈ \text{net_tree_ips n}" 
thus "\text{netmap } s \ i = \text{None}" 
proof (induction n arbitrary: s) 
fix ii R s 
assume "s \in \text{init } (\text{closed } (\text{pnet np ⟨ii; R⟩}))" 
and "i /∈ \text{net_tree_ips ⟨ii; R⟩}" 
from this(2) have "i \neq ii" by simp 
moreover from 's \in \text{init } (\text{closed } (\text{pnet np ⟨ii; R⟩}))' obtain p where "s = \text{NodeS ii p R}" 
by simp (metis pnet.simps(1) pnet_node_init') 
ultimately show "\text{netmap } s \ i = \text{None}" by simp 
next 
fix p1 p2 s 
assume IH1: "∀s. s \in \text{init } (\text{closed } (\text{pnet np p1})) \implies i /∈ \text{net_tree_ips p1} \implies \text{netmap } s \ i = \text{None}" 
and IH2: "∀s. s \in \text{init } (\text{closed } (\text{pnet np p2})) \implies i /∈ \text{net_tree_ips p2} \implies \text{netmap } s \ i = \text{None}" 
and "s \in \text{init } (\text{closed } (\text{pnet np (p1 || p2)))" 
and "i /∈ \text{net_tree_ips (p1 || p2)}" 
from this(3) obtain s1 s2 where "s = \text{SubnetS } s1 \ s2"
and "s1 ∈ init (closed (pnet np p1))"
and "s2 ∈ init (closed (pnet np p2))" by simp metis
moreover from 'i ∉ net_tree_ips (p1 || p2)' have "i ∉ net_tree_ips p1"
and "i ∉ net_tree_ips p2" by auto
ultimately have "netmap s1 i = None"
and "netmap s2 i = None"
using IH1 IH2 by auto
with 's = SubnetS s1 s2' show "netmap s i = None" by simp
qed

next
fix s a s'
assume sr: "s ∈ reachable (closed (pnet np n)) TT"
and tr: "(s, a, s') ∈ trans (closed (pnet np n))"
and IH: "i ∉ net_tree_ips n =⇒ netmap s i = None"
and "i ∉ net_tree_ips n"
from this(3-4) have "i ∉ net_ips s" by auto
with tr have "i ∉ net_ips s'" by simp (erule cnet_sos.cases, (metis net_ips_is_dom_netmap pnet_maintains_dom)+)
thus "netmap s' i = None" by simp
qed

lemma closed_pnet_aodv_init [elim]:
assumes "s ∈ init (closed (pnet np n))" and "i ∈ net_tree_ips n"
shows "the (netmap s i) ∈ init (np i)"
using assms proof (induction n arbitrary: s)
fix ii R s
assume "s ∈ init (closed (pnet np (ii; R)))"
and "i ∈ net_tree_ips (ii; R)"
hence "s ∈ init (pnet np (i; R))" by simp
then obtain p where "s = NodeS i p R"
and "p ∈ init (np i)" ..
with 's = NodeS i p R' have "netmap s = [i ↦ p]" by simp
with 'p ∈ init (np i)' show "the (netmap s i) ∈ init (np i)" by simp
next
fix p1 p2 s
assume IH1: "∀ s. s ∈ init (closed (pnet np p1)) =⇒ i ∈ net_tree_ips p1 =⇒ the (netmap s i) ∈ init (np i)"
and IH2: "∀ s. s ∈ init (closed (pnet np p2)) =⇒ i ∈ net_tree_ips p2 =⇒ the (netmap s i) ∈ init (np i)"
and "s ∈ init (closed (pnet np (p1 || p2)))"
and "i ∈ net_tree_ips (p1 || p2)"
from this(3) obtain s1 s2 where "s = SubnetS s1 s2"
and "s1 ∈ init (closed (pnet np p1))"
and "s2 ∈ init (closed (pnet np p2))"
by auto
from (2) have "net_tree_ips p1 = net_ips s1"
by (clarsimp dest!: pnet_init_net_ips_net_tree_ips)
from 's2 ∈ init (closed (pnet np p2))' have "net_tree_ips p2 = net_ips s2"
by (clarsimp dest!: pnet_init_net_ips_net_tree_ips)
show "the (netmap s i) ∈ init (np i)"
proof (cases "i ∈ net_tree_ips p2")
assume "i ∈ net_tree_ips p2"
with 's2 ∈ init (closed (pnet np p2))' have "the (netmap s2 i) ∈ init (np i)" by (rule IH2)
moreover from 'i ∈ net_tree_ips p2' and 'net_tree_ips p2 = net_ips s2'
have "i ∈ net_ips s2" by simp
ultimately show thesis
using 's = SubnetS s1 s2' by (auto simp add: net_ips_is_dom_netmap)
next
assume "i ∉ net_tree_ips p2"
with 'i ∈ net_tree_ips (p1 || p2)' have "i ∈ net_tree_ips p1" by simp
with 's1 ∈ init (closed (pnet np p1))' have "the (netmap s1 i) ∈ init (np i)"
by (rule IH1)
moreover from \( i \in \text{net_tree_ips p1} \) and \( \text{net_tree_ips p1 = net_ips s1} \)

have \( i \in \text{net_ips s1} \) by simp

moreover from \( i \not\in \text{net_tree_ips p2} \) and \( \text{net_tree_ips p2 = net_ips s2} \)

have \( i \not\in \text{net_ips s2} \) by simp

ultimately show \( \text{thesis} \)

using \( s = \text{SubnetS} s1 s2 \)

by (simp add: map_add_dom_app_simps net_ips_is_dom_netmap)

qed

qed

end

13 Open semantics of the Algebra of Wireless Networks

theory OAWN_SOS

imports TransitionSystems AWN

begin

These are variants of the SOS rules that work against a mixed global/local context, where the global context is represented by a function \( \sigma \) mapping ip addresses to states.

13.1 Open structural operational semantics for sequential process expressions

inductive_set
oseq_sos
:: "('s, 'm, 'p, 'l) seqp_env \Rightarrow ip \Rightarrow (ip \Rightarrow 's) \times (('s, 'm, 'p, 'l) seqp, 'm seq_action) transition set"

for \( \Gamma :: ('s, 'm, 'p, 'l) seqp_env \)
and \( i :: ip \)

where

obroadcastT: \( \sigma' \cdot i = \sigma \cdot i \Rightarrow ((\sigma, \{l\} broadcast(s_m msg).p), broadcast (s_m msg (\sigma \cdot i)), (\sigma', p)) \in oseq_sos \Gamma i \)

/ ogroupcastT: \( \sigma' \cdot i = \sigma \cdot i \Rightarrow ((\sigma, \{l\} groupcast(s_ips, s_m msg).p), groupcast (s_ips (\sigma \cdot i)) (s_m msg (\sigma \cdot i)), (\sigma', p)) \in oseq_sos \Gamma i \)

/ ounicastT: \( \sigma' \cdot i = \sigma \cdot i \Rightarrow ((\sigma, \{l\} unicast(s_ip, s_m msg).p \oplus q), unicast (s_ip (\sigma \cdot i)) (s_m msg (\sigma \cdot i)), (\sigma', p)) \in oseq_sos \Gamma i \)

/ onotunicastT: \( \sigma' \cdot i = \sigma \cdot i \Rightarrow ((\sigma, \{l\} unicast(s_ip, s_m msg).p \oplus q), \neg unicast (s_ip (\sigma \cdot i)), (\sigma', q)) \in oseq_sos \Gamma i \)

/ osendT: \( \sigma' \cdot i = \sigma \cdot i \Rightarrow ((\sigma, \{l\} send(s_m msg).p), send (s_m msg (\sigma \cdot i)), (\sigma', p)) \in oseq_sos \Gamma i \)

/ odeliverT: \( \sigma' \cdot i = \sigma \cdot i \Rightarrow ((\sigma, \{l\} deliver(s_data).p), deliver (s_data (\sigma \cdot i)), (\sigma', p)) \in oseq_sos \Gamma i \)

/ oreceiveT: \( \sigma' \cdot i = \text{u_mmsg} \; \text{msg} \; (\sigma \cdot i) \Rightarrow ((\sigma, \{l\} receive(u_m msg).p), receive msg, (\sigma', p)) \in oseq_sos \Gamma i \)

/ oassignT: \( \sigma' \cdot i = \text{u} \; (\sigma \cdot i) \Rightarrow ((\sigma, \{l\}[\text{u}] p), \tau, (\sigma', p)) \in oseq_sos \Gamma i \)

/ ocallT: \( ((\sigma, \Gamma \text{pn}), a, (\sigma', p')) \in oseq_sos \Gamma i \Rightarrow ((\sigma, \text{call(pn)}), a, (\sigma', p')) \in oseq_sos \Gamma i \)

/ ochoiceT1: \( ((\sigma, p), a, (\sigma', p')) \in oseq_sos \Gamma i \Rightarrow ((\sigma, p \oplus q), a, (\sigma', p')) \in oseq_sos \Gamma i \)

/ ochoiceT2: \( ((\sigma, q), a, (\sigma', q')) \in oseq_sos \Gamma i \Rightarrow ((\sigma, p \oplus q), a, (\sigma', q')) \in oseq_sos \Gamma i \)
declare oseq_sos.intros [intro]

13.2 Open structural operational semantics for parallel process expressions
\begin{align*}
&\Rightarrow \text{('s2, 'm seq_action) transition set} \\
&\Rightarrow ((\text{ip} \Rightarrow 's) \times ('s1 \times 's2), 'm seq_action) transition set
\end{align*}

for i :: ip

and S :: "((\text{ip} \Rightarrow 's) \times 's1, 'm seq_action) transition set"

and T :: "('s2, 'm seq_action) transition set"

where

\begin{itemize}
  \item \text{oparleft:} \\
  \quad 
  \begin{align*}
  &\left[\begin{array}{l}
  ((\sigma, (s, t)), a, (\sigma', (s', t'))) \in \text{oparp_sos i S T} \\
  (\sigma, (s, t)), a, (\sigma', (s', t')) \in \text{oparp_sos i S T}
  \end{array}\right] \Rightarrow \\
  &\left[\begin{array}{l}
  \sigma \in T; s' = s; \sigma' = \sigma; i = \sigma i \Rightarrow P
  \end{array}\right] \Rightarrow P
  \end{align*}

  \\
  \item \text{oparright:} \\
  \quad 
  \begin{align*}
  &\left[\begin{array}{l}
  (t, a, t') \in T; \exists m. a \neq \text{send m}; \sigma' \neq \sigma i \Rightarrow \\
  (\sigma, (s, t)), a, (\sigma', (s', t')) \in \text{oparp_sos i S T}
  \end{array}\right] \Rightarrow \\
  &\left[\begin{array}{l}
  (t, broadcast m, t') \in T; \exists s. t' = t; \exists \sigma' \neq \sigma i \Rightarrow P
  \end{array}\right] \Rightarrow P
  \end{align*}

  \\
  \item \text{oparboth:} \\
  \quad 
  \begin{align*}
  &\left[\begin{array}{l}
  ((\sigma, (s, t)), receive m, (\sigma', (s', t'))) \in \text{oparp_sos i S T} \\
  ((\sigma, (s, t)), \tau, (\sigma', (s', t'))) \in \text{oparp_sos i S T}
  \end{array}\right] \Rightarrow \\
  &\left[\begin{array}{l}
  \tau \Rightarrow P
  \end{array}\right] \Rightarrow P
  \end{align*}
\end{itemize}

\text{lemma opar broadcastTE [elim]:}

\begin{align*}
&\left[\begin{array}{l}
  ((\sigma, (s, t)), broadcast m, (\sigma', (s', t'))) \in \text{oparp_sos i S T} \\
  (\sigma, (s, t)), broadcast m, (\sigma', (s', t')) \in \text{oparp_sos i S T}
  \end{array}\right] \Rightarrow \\
&\left[\begin{array}{l}
  (t, broadcast m, t') \in T; \exists s. t' = t; \exists \sigma' \neq \sigma i \Rightarrow P
  \end{array}\right] \Rightarrow P
\end{align*}

by (ind_cases "((\sigma, (s, t)), broadcast m, (\sigma', (s', t'))) \in \text{oparp_sos i S T" simp_all

\text{lemma opar groupcastTE [elim]:}

\begin{align*}
&\left[\begin{array}{l}
  ((\sigma, (s, t)), groupcast ips m, (\sigma', (s', t'))) \in \text{oparp_sos i S T} \\
  (\sigma, (s, t)), groupcast ips m, (\sigma', (s', t')) \in \text{oparp_sos i S T}
  \end{array}\right] \Rightarrow \\
&\left[\begin{array}{l}
  (t, groupcast ips m, t') \in T; \exists s. t' = t; \exists \sigma' \neq \sigma i \Rightarrow P
  \end{array}\right] \Rightarrow P
\end{align*}

by (ind_cases "((\sigma, (s, t)), groupcast ips m, (\sigma', (s', t'))) \in \text{oparp_sos i S T" simp_all

\text{lemma opar unicastTE [elim]:}

\begin{align*}
&\left[\begin{array}{l}
  ((\sigma, (s, t)), unicast i m, (\sigma', (s', t'))) \in \text{oparp_sos i S T} \\
  (\sigma, (s, t)), unicast i m, (\sigma', (s', t')) \in \text{oparp_sos i S T}
  \end{array}\right] \Rightarrow \\
&\left[\begin{array}{l}
  (t, unicast i m, t') \in T; \exists s. t' = t; \exists \sigma' \neq \sigma i \Rightarrow P
  \end{array}\right] \Rightarrow P
\end{align*}

by (ind_cases "((\sigma, (s, t)), unicast i m, (\sigma', (s', t'))) \in \text{oparp_sos i S T" simp_all

\text{lemma opar notunicastTE [elim]:}

\begin{align*}
&\left[\begin{array}{l}
  ((\sigma, (s, t)), notunicast i, (\sigma', (s', t'))) \in \text{oparp_sos i S T} \\
  (\sigma, (s, t)), notunicast i, (\sigma', (s', t')) \in \text{oparp_sos i S T}
  \end{array}\right] \Rightarrow \\
&\left[\begin{array}{l}
  (t, notunicast i, t') \in T; \exists s. t' = t; \exists \sigma' \neq \sigma i \Rightarrow P
  \end{array}\right] \Rightarrow P
\end{align*}

by (ind_cases "((\sigma, (s, t)), notunicast i, (\sigma', (s', t'))) \in \text{oparp_sos i S T" simp_all

\text{lemma opar sendTE [elim]:}

\begin{align*}
&\left[\begin{array}{l}
  ((\sigma, (s, t)), send m, (\sigma', (s', t'))) \in \text{oparp_sos i S T} \\
  (\sigma, (s, t)), send m, (\sigma', (s', t')) \in \text{oparp_sos i S T}
  \end{array}\right] \Rightarrow \\
&\left[\begin{array}{l}
  (t, send m, t') \in T; \exists s. t' = t; \exists \sigma' \neq \sigma i \Rightarrow P
  \end{array}\right] \Rightarrow P
\end{align*}

by (ind_cases "((\sigma, (s, t)), send m, (\sigma', (s', t'))) \in \text{oparp_sos i S T" auto

\text{lemma opar deliverTE [elim]:}

\begin{align*}
&\left[\begin{array}{l}
  ((\sigma, (s, t)), deliver d, (\sigma', (s', t'))) \in \text{oparp_sos i S T} \\
  (\sigma, (s, t)), deliver d, (\sigma', (s', t')) \in \text{oparp_sos i S T}
  \end{array}\right] \Rightarrow \\
&\left[\begin{array}{l}
  (t, deliver d, t') \in T; \exists s. t' = t; \exists \sigma' \neq \sigma i \Rightarrow P
  \end{array}\right] \Rightarrow P
\end{align*}

by (ind_cases "((\sigma, (s, t)), deliver d, (\sigma', (s', t'))) \in \text{oparp_sos i S T" simp_all

\text{lemma opar receiveTE [elim]:}

\begin{align*}
&\left[\begin{array}{l}
  ((\sigma, (s, t)), receive m, (\sigma', (s', t'))) \in \text{oparp_sos i S T} \\
  (t, receive m, t') \in T; \exists s. t' = t; \exists \sigma' \neq \sigma i \Rightarrow P
  \end{array}\right] \Rightarrow P
\end{align*}

by (ind_cases "((\sigma, (s, t)), receive m, (\sigma', (s', t'))) \in \text{oparp_sos i S T" auto

\text{inductive cases opar tauTE:} \\
"((\sigma, (s, t)), \tau, (\sigma', (s', t'))) \in \text{oparp_sos i S T"}

\text{lemmas oparTEs =}

opar_broadcastTE
opar_groupcastTE
opar_unicastTE
opar_notunicastTE
opar_sendTE
opar_deliverTE
opar_receiveTE
lemma oparp_sos_cases [elim]:
  assumes "[((σ, (s, t)), a, (σ', (s', t'))) ∈ oparp_sos i S T]
  and "[(σ, (s, σ', s'))) ∈ S; \( \land m. a \neq \text{receive } m; t' = t \] \(\Rightarrow\) P"
  and "[(t, a, t') ∈ T; \( \land m. a \neq \text{send } m; s' = s; σ' i = σ i \] \(\Rightarrow\) P"
  and "\(\land m. [ a = τ; ((σ, s), \text{receive } m, (σ', s')) ∈ S; (t, \text{send } m, t') ∈ T \] \(\Rightarrow\) P"
  shows "P" using assms by cases auto

definition extg :: "(('a × 'b) × 'c ⇒ 'a × 'b × 'c)
  where "extg ≡ λ((σ, l1), l2). (σ, (l1, l2))"

lemma extgsimp [simp]: "extg ((σ, l1), l2) = (σ, (l1, l2))"
unfolding extg_def by simp

lemma extg_range_prod: "extg ' (i1 × i2) = {((σ, s1, s2))|(σ, s1) ∈ i1 ∧ s2 ∈ i2}"
unfolding image_def extg_def by (rule Collect_cong) (auto split: split_split)

definition opar_comp :: "((ip ⇒ 's) × 'l, 'm seq_action) automaton
⇒ ((ip ⇒ 's) × 'l net_state, 'm node_action) automaton"
("\langle\langle\_\_\rangle\rangle" [102, 0, 103] 102)
where "s ⟨⟨i t⟩ = ((init i t = extg ' (init s × init t), trans = oparp_sos i (trans s) (trans t))]

lemma opar_comp_def': "s ⟨⟨i t = ((init i t = {((σ, s l, t l))|σ s l t l. (σ, s l) ∈ init s l ∧ t l ∈ init t},
trans = oparp_sos i (trans s) (trans t))]
unfolding opar_comp_def extg_def image_def by (auto split: split_split)

lemma trans_opar_comp [simp]: "trans (s ⟨⟨i t) = oparp_sos i (trans s) (trans t)"
unfolding opar_comp_def by simp

lemma init_opar_comp [simp]: "init (s ⟨⟨i t) = extg ' (init s × init t)"
unfolding opar_comp_def by simp

13.3 Open structural operational semantics for node expressions

inductive_set
  onode_sos :: "((ip ⇒ 's) × 'l, 'm seq_action) transition set
⇒ ((ip ⇒ 's) × 'l net_state, 'm node_action) transition set"
for S :: "((ip ⇒ 's) × 'l, 'm seq_action) transition set"
where
  onode_bcast: "((σ, s), broadcast m, (σ', s')) ∈ S \(\Rightarrow\) ((σ, NodeS i s R), R:*cast(m), (σ', NodeS i s' R)) ∈ onode_sos S"
  onode_gcast: "((σ, s), groupcast D m, (σ', s')) ∈ S \(\Rightarrow\) ((σ, NodeS i s R), (R ∩ D):*cast(m), (σ', NodeS i s' R)) ∈ onode_sos S"
  onode_ucast: "[((σ, s), unicast d m, (σ', s')) ∈ S; d ∈ R] \(\Rightarrow\) ((σ, NodeS i s R), (d):*cast(m), (σ', NodeS i s' R)) ∈ onode_sos S"
  onode_notucast: "[((σ, s), ¬unicast d, (σ', s')) ∈ S; d \notin R; \forall j. j \neq i \rightarrow σ' j = σ j] \(\Rightarrow\) ((σ, NodeS i s R), τ, (σ', NodeS i s' R)) ∈ onode_sos S"
| onode_deliver: "[(σ, s), deliver d, (σ', s')] ∈ S; ∀ j. j' ≠ i → σ' j = σ j] \n⇒ ((σ, NodeS i s R), i:deliver(d), (σ', NodeS i' s' R)) ∈ onode_sos S" |
| onode_tau: "[(σ, s), τ, (σ', s')] ∈ S; ∀ j. j' ≠ i → σ' j = σ j] \n⇒ ((σ, NodeS i s R), τ, (σ', NodeS i' s' R)) ∈ onode_sos S" |
| onode_receive: "((σ, s), receive m, (σ', s')) ∈ S \n⇒ ((σ, NodeS i s R), {i} ¬ {i}:arrive(m), (σ', NodeS i' s' R)) ∈ onode_sos S" |
| onode_arrive: "σ' i = σ i \n⇒ ((σ, NodeS i s R), ii ¬ {i}:arrive(m), (σ', NodeS i' s' R)) ∈ onode_sos S" |
| onode_connect1: "σ' i = σ i \n⇒ ((σ, NodeS i s R), connect(i, i'), (σ', NodeS i s (R ∪ {i'}))) ∈ onode_sos S" |
| onode_connect2: "σ' i = σ i \n⇒ ((σ, NodeS i s R), connect(i', i), (σ', NodeS i s (R ∪ {i'}))) ∈ onode_sos S" |
| onode_disconnect1: "σ' i = σ i \n⇒ ((σ, NodeS i s R), disconnect(i, i'), (σ', NodeS i s (R - {i'}))) ∈ onode_sos S" |
| onode_disconnect2: "σ' i = σ i \n⇒ ((σ, NodeS i s R), disconnect(i', i), (σ', NodeS i s (R - {i'}))) ∈ onode_sos S" |
| onode_connect_other: "i' ≠ i; i' ≠ i''; σ' i = σ i \n⇒ ((σ, NodeS i s R), connect(i', i''), (σ', NodeS i s R)) ∈ onode_sos S" |
| onode_disconnect_other: "i' ≠ i; i' ≠ i''; σ' i = σ i \n⇒ ((σ, NodeS i s R), disconnect(i', i''), (σ', NodeS i s R)) ∈ onode_sos S" |

inductive cases
| onode_arriveTE [elim]: "((σ, NodeS i s R), ii ¬ {i}:arrive(m), (σ', NodeS i' s' R)) ∈ onode_sos S" |
| and onode_castTE [elim]: "((σ, NodeS i s R), RR:*cast(m), (σ', NodeS i' s' R)) ∈ onode_sos S" |
| and onode_deliverTE [elim]: "((σ, NodeS i s R), ii:deliver(d), (σ', NodeS i' s' R)) ∈ onode_sos S" |
| and onode_connectTE [elim]: "((σ, NodeS i s R), connect(ii, ii'), (σ', NodeS i' s' R)) ∈ onode_sos S" |
| and onode_disconnectTE [elim]: "((σ, NodeS i s R), disconnect(ii, ii'), (σ', NodeS i' s' R)) ∈ onode_sos S" |
| and onode_newpktTE [elim]: "((σ, NodeS i s R), ii:newpkt(d, di), (σ', NodeS i' s' R)) ∈ onode_sos S" |
| and onode_tauTE [elim]: "((σ, NodeS i s R), τ, (σ', NodeS i' s' R)) ∈ onode_sos S" |

lemma carries_or_not:
| assumes "((σ, NodeS i s R), ii ¬ {i}:arrive(m), (σ', NodeS i' s' R)) ∈ onode_sos S" |
| shows "(ii = {i} ∧ ni = {}) ∨ (ii = {} ∧ ni = {i})" |
| using assms by rule simp_all |

definition
| onode_comp :: "ip \n⇒ ((ip ⇒ 's) × 'l, 'm seq_action) automaton \n⇒ ip set \n⇒ ((ip ⇒ 's) × 'l net_state, 'm node_action) automaton" |
| "(⟨_ : (_) : _⟩ o)" [0, 0, 0] 104 |
| where
| "⟨i : onp : Ri⟩ o ≡ {init = {(σ, NodeS i s R), σ s. (σ, s) ∈ init onp}, \ntrans = onode_sos (trans onp)}"
lemma trans_onode_comp:
"trans ((i : S : R),) = onode_sos (trans S)"
unfolding onode_comp_def by simp

lemma init_onode_comp:
"init ((i : S : R),) = {((σ, NodeS i s R) | σ s. (σ, s) ∈ init S)}"
unfolding onode_comp_def by simp

lemmas onode_comps = trans_onode_comp init_onode_comp

lemma fst_par_onode_comp [simp]:
"trans ((i : s (I t : R),) = onode_sos (oparp_sos I (trans s) (trans t))"
unfolding onode_comp_def by simp

lemma init_par_onode_comp [simp]:
"init ((i : s (I t : R),) = {(σ, NodeS i (s1, s2) R) | σ s1 s2. ((σ, s1), s2) ∈ init s × init t}"
unfolding onode_comp_def by (simp add: extg_range_prod)

lemma onode_sos_dest_is_net_state:
assumes "((σ, p), a, s') ∈ onode_sos S"
shows "∃σ' i' ζ' R'. s' = (σ', NodeS i' ζ' R')"
using assms proof -
  assume "((σ, p), a, s') ∈ onode_sos S"
  then obtain σ' i' ζ' R' where "s' = (σ', NodeS i' ζ' R')"
    by (auto elim!: onode_sos.cases)
  hence ?thesis by simp
qed

lemma onode_sos_dest_is_net_state':
assumes "((σ, NodeS i p R), a, s') ∈ onode_sos S"
shows "∃σ' ζ' R'. s' = NodeS i ζ' R'"
proof -
  def ns' ≡ "(σ', s')"
  with assms have "((σ, NodeS i p R), a, ns') ∈ onode_sos S" by simp
  then obtain σ' i' ζ' R' where "ns' = (σ', NodeS i' ζ' R')"
    by (metis onode_sos_dest_is_net_state')
  hence "ns' = NodeS i ζ' R'" by (simp add: ns'_def)
  thus ?thesis by simp
qed

lemma onode_sos_dest_is_net_state'':
assumes "((σ, NodeS i p R), a, s') ∈ onode_sos S"
shows "∃ζ' R'. s' = NodeS i ζ' R'
proof -
  def ns' ≡ "(σ', s')"
  with assms have "((σ, NodeS i p R), a, ns') ∈ onode_sos S" by simp
  then obtain σ' ζ' R' where "ns' = (σ', NodeS i ζ' R')"
    by (metis onode_sos_dest_is_net_state')
  hence "ns' = NodeS i ζ' R'" by (simp add: ns'_def)
  thus ?thesis by simp
qed

lemma onode_sos_src_is_net_state:
assumes "((σ, p), a, s') ∈ onode_sos S"
shows "∃i ζ R. p = NodeS i ζ R"
proof -
  assume "((σ, p), a, s') ∈ onode_sos S"
  then obtain i ζ R where "p = NodeS i ζ R"
    by (cases s') (auto elim!: onode_sos.cases)
  thus ?thesis by simp
qed

lemma onode_sos_net_states:
assumes "((σ, s), a, (σ', s')) ∈ onode_sos S"
shows "∃i ζ R ζ' R'. s = NodeS i ζ R ∧ s' = NodeS i ζ' R'"
proof -
from assms obtain i ⊥ R where "s = NodeS i ⊥ R"
  by (metis onode_sos_src_is_net_state)
moreover with assms obtain i′ ⊥ R where "s′ = NodeS i′ ⊥ R"
  by (auto dest!: onode_sos_dest_is_net_state')
ultimately show ?thesis by simp
qed

lemma node_sos_cases [elim]:
  "((σ, NodeS i p) ∈ onode_sos S) ⊃
   (∀m. a = R:*cast(m); R' = R; ((σ, p), broadcast m, (σ', p')) ∈ S) ⊃ P
     "
  "((σ, NodeS i p) ∈ onode_sos S) ⊃
   (∀D m. a = (R ∩ D):*cast(m); R' = R; ((σ, p), groupcast D m, (σ', p')) ∈ S) ⊃ P
     "
  "((σ, NodeS i p) ∈ onode_sos S) ⊃
   (∀m. a = (d):*cast(m); R' = R; ((σ, p), unicast d m, (σ', p')) ∈ S; d ∈ R') ⊃ P
     
   ⊃ ((∀a. a = i:deliver(d); R' = R; ((σ, p), deliver d, (σ', p')) ∈ S) ⊃ P)

  "((σ, i : deliver(d); R' = R; p = p'; σ' i = σ i) ∈ S ⊃ P)"

  "((∀a. a = i:deliver(d); R' = R; ((σ, p), receive m, (σ', p')) ∈ S) ⊃ P)
   "
  "((σ, i : deliver(d); R' = R; p = p'; σ' i = σ i) ∈ S ⊃ P)"

  "((∀a. a = i:deliver(d); R' = R; p = p'; σ' i = σ i) ∈ S ⊃ P)"

  by (erule onode_sos_cases) (simp | metis)+

13.4 Open structural operational semantics for partial network expressions

inductive_set

opnet_sos :: "((ip ⇒ 's) × 'l net_state, 'm node_action) transition set"
⇒ ((ip ⇒ 's) × 'l net_state, 'm node_action) transition set
⇒ ((ip ⇒ 's) × 'l net_state, 'm node_action) transition set"

for S :: "((ip ⇒ 's) × 'l net_state, 'm node_action) transition set"
and T :: "((ip ⇒ 's) × 'l net_state, 'm node_action) transition set"
where

  "[ [ ((σ, s), R:*cast(m), (σ', s')) ∈ S; ((σ, t), H:=K:arrive(m), (σ', t')) ∈ T; H ⊆ R; K ∩ R = {} ]
     ⊃ ((σ, SubnetS s t), R:*cast(m), (σ', SubnetS s' t')) ∈ opnet_sos S T"

| opnet_cast2:
  "[ [ ((σ, s), H:=K:arrive(m), (σ', s')) ∈ S; ((σ, t), R:*cast(m), (σ', t')) ∈ T; H ⊆ R; K ∩ R = {} ]
     ⊃ ((σ, SubnetS s t), R:*cast(m), (σ', SubnetS s' t')) ∈ opnet_sos S T"

| opnet_arrive:
  "[ [ ((σ, s), H:=K:arrive(m), (σ', s')) ∈ S; ((σ, t), H:=K:arrive(m), (σ', t')) ∈ T ]
     ⊃ ((σ, SubnetS s t), (H ∪ H'):=K:arrive(m), (σ', SubnetS s' t')) ∈ opnet_sos S T"

| opnet_deliver1:
  "[ [ ((σ, s), i:deliver(d), (σ', s')) ∈ S
     ⊃ ((σ, SubnetS s t), i:deliver(d), (σ', SubnetS s' t)) ∈ opnet_sos S T"

| opnet_deliver2:
  "[ [ ((σ, t), i:deliver(d), (σ', t')) ∈ T ]
     ⊃ ((σ, SubnetS s t), i:deliver(d), (σ', SubnetS s' t')) ∈ opnet_sos S T"

| opnet_tau1:
  "[ [ ((σ, s), τ, (σ', s')) ∈ S
     ⊃ ((σ, SubnetS s t), τ, (σ', SubnetS s' t)) ∈ opnet_sos S T"

| opnet_tau2:
  "[ [ ((σ, t), τ, (σ', t')) ∈ T
     ⊃ ((σ, SubnetS s t), τ, (σ', SubnetS s' t')) ∈ opnet_sos S T"

82
13.5 Open structural operational semantics for complete network expressions

inductive_set

ocnet_sos :: "((ip ⇒ 's) × 'l net_state, 'm::msg node_action) transition set" 
⇒ ((ip ⇒ 's) × 'l net_state, 'm node_action) transition set"

for S :: "((ip ⇒ 's) × 'l net_state, 'm node_action) transition set"

where

ocnet_connect: "[((s, s), connect(i, i'), (s', s')) ∈ S; ∀j. j /∈ net_ips s → (s' j = σ j)]
⇒ ((s, s), connect(i, i'), (s', s')) ∈ ocnet_sos S"

/ ocnet_disconnect:

"[((s, s), disconnect(i, i'), (s', s')) ∈ S; ∀j. j /∈ net_ips s → (s' j = σ j)]
⇒ ((s, s), disconnect(i, i'), (s', s')) ∈ ocnet_sos S"

/ ocnet_cast:

"[((s, s), R:*cast(m), (s', s')) ∈ S; ∀j. j /∈ net_ips s → (s' j = σ j)]
⇒ ((s, s), τ, (s', s')) ∈ ocnet_sos S"

/ ocnet_tau:

"[((s, s), τ, (s', s')) ∈ S; ∀j. j /∈ net_ips s → (s' j = σ j)]
"
\[(\sigma, s, \tau, (\sigma', s')) \in \text{ocnet}_sos S\]

| ocat_deliver: |
| "\[ ((\sigma, s), i:\text{deliver}(d), (\sigma', s')) \in S; \forall j. j \notin \text{net_ips} s \rightarrow (\sigma' j = \sigma j) \] \]
| \[ \rightarrow ((\sigma, s), i:\text{deliver}(d), (\sigma', s')) \in \text{ocnet}_sos S\]

| ocat_newpkt: |
| "\[ ((\sigma, s), (i \neg K:\text{arrive}(\text{newpkt}(d, di)), (\sigma', s')) \in S; \forall j. j \notin \text{net_ips} s \rightarrow (\sigma' j = \sigma j) \] \]
| \[ \rightarrow ((\sigma, s), i:\text{newpkt}(d, di), (\sigma', s')) \in \text{ocnet}_sos S\]

**inductive cases** oconnect_completeTE: "((\sigma, s), \text{connect}(i, i'), (\sigma', s')) \in \text{ocnet}_sos S"

and odisconnect_completeTE: "((\sigma, s), \text{disconnect}(i, i'), (\sigma', s')) \in \text{ocnet}_sos S"

and otau_completeTE: "((\sigma, s), \tau, (\sigma', s')) \in \text{ocnet}_sos S"

and odeliver_completeTE: "((\sigma, s), i:\text{deliver}(d), (\sigma', s')) \in \text{ocnet}_sos S"

and onewpkt_completeTE: "((\sigma, s), i:\text{newpkt}(d, di), (\sigma', s')) \in \text{ocnet}_sos S"

**lemmas** ocompleteTEs = oconnect_completeTE

odisconnect_completeTE

otau_completeTE

odeliver_completeTE

onewpkt_completeTE

**lemma** ocomplete_no_cast [simp]:

"((\sigma, s), R:*\text{cast}(m), (\sigma', s')) \notin \text{ocnet}_sos T"

proof

assume "((\sigma, s), R:*\text{cast}(m), (\sigma', s')) \in \text{ocnet}_sos T"

hence "R:*\text{cast}(m) \neq R:*\text{cast}(m)"

by (rule ocnet_sos.cases) auto

thus False by simp

qed

**lemma** ocomplete_no_arrive [simp]:

"((\sigma, s), ii \neg ni:\text{arrive}(m), (\sigma', s')) \notin \text{ocnet}_sos T"

proof

assume "((\sigma, s), ii \neg ni:\text{arrive}(m), (\sigma', s')) \in \text{ocnet}_sos T"

hence "ii \neg ni:\text{arrive}(m) \neq ii \neg ni:\text{arrive}(m)"

by (rule ocnet_sos.cases) auto

thus False by simp

qed

**lemma** ocomplete_no_change [elim]:

assumes "((\sigma, s), a, (\sigma', s')) \in \text{ocnet}_sos T"

and "j \notin \text{net_ips} s"

shows "\sigma' j = \sigma j"

using assms by cases simp_all

**lemma** ocomplete_transE [elim]:

assumes "((\sigma, \zeta), a, (\sigma', \zeta')) \in \text{ocnet}_sos (\text{trans} (\text{opnet onp} n))"

obtains a' where "((\sigma, \zeta), a', (\sigma', \zeta')) \in \text{trans} (\text{opnet onp} n)"

using assms by (cases a) (auto elim!: ocompleteTEs [simplified])

**abbreviation**

oclosed :: "((\text{ip} \Rightarrow 's) \times 'l \text{net_state}, ('m::\text{msg}) \text{node_action}) \text{automaton}

\Rightarrow ((\text{ip} \Rightarrow 's) \times 'l \text{net_state}, 'm \text{node_action}) \text{automaton}"

where

"\text{oclosed} \equiv (\lambda A. A (| \text{trans} := \text{ocnet}_sos \text{ (trans} A) |))"

end

### 14 Configure the inv-cterms tactic for open sequential processes

theory OAWN_SOS_Labels

imports OAWN_SOS Inv_Cterms

begin

end
lemma oelimder_guard:
assumes "p = {l}⟨fg⟩ qq"
and "l' \in \text{labels } \Gamma q"
and "((\sigma, p), a, (\sigma', q)) \in \text{oseqp sos } \Gamma i"
obtains p' where "p = {l'}\langle fg \rangle p''"
and "l' \in \text{labels } \Gamma q"
using assms by auto

lemma oelimder_assign:
assumes "p = {l}[fa] qq"
and "l' \in \text{labels } \Gamma q"
and "((\sigma, p), a, (\sigma', q)) \in \text{oseqp sos } \Gamma i"
obtains p' where "p = {l'}[fa] p''"
and "l' \in \text{labels } \Gamma q"
using assms by auto

lemma oelimder_ucast:
assumes "p = {l}unicast(fip, fmsg).q1 \oplus q2"
and "l' \in \text{labels } \Gamma q"
and "((\sigma, p), a, (\sigma', q)) \in \text{oseqp sos } \Gamma i"
obtains p' pp' where "p = {l'}unicast(fip, fmsg). p''"
and "\text{case } a \text{ of unicast } _ _ \Rightarrow l' \in \text{labels } \Gamma q1"
and "\text{case } a \text{ of unicast } _ _ \Rightarrow l' \in \text{labels } \Gamma q2"
using assms by simp (erule oseqpTEs, auto)

lemma oelimder_bcast:
assumes "p = {l}broadcast(fmsg).qq"
and "l' \in \text{labels } \Gamma q"
and "((\sigma, p), a, (\sigma', q)) \in \text{oseqp sos } \Gamma i"
obtains p' where "p = {l}broadcast(fmsg). p''"
and "l' \in \text{labels } \Gamma qq"
using assms by auto

lemma oelimder_gcast:
assumes "p = {l}groupcast(fips, fmsg).qq"
and "l' \in \text{labels } \Gamma q"
and "((\sigma, p), a, (\sigma', q)) \in \text{oseqp sos } \Gamma i"
obtains p' where "p = {l}groupcast(fips, fmsg). p''"
and "l' \in \text{labels } \Gamma qq"
using assms by auto

lemma oelimder_send:
assumes "p = {l}send(fmsg).qq"
and "l' \in \text{labels } \Gamma q"
and "((\sigma, p), a, (\sigma', q)) \in \text{oseqp sos } \Gamma i"
obtains p' where "p = {l}send(fmsg). p''"
and "l' \in \text{labels } \Gamma qq"
using assms by auto

lemma oelimder_deliver:
assumes "p = {l}deliver(fdata).qq"
and "l' \in \text{labels } \Gamma q"
and "((\sigma, p), a, (\sigma', q)) \in \text{oseqp sos } \Gamma i"
obtains p' where "p = {l}deliver(fdata). p''"
and "l' \in \text{labels } \Gamma qq"
using assms by auto

lemma oelimder_receive:
assumes "p = {l}receive(fmsg).qq"
and "l' \in \text{labels } \Gamma q"
and "((\sigma, p), a, (\sigma', q)) \in \text{oseqp sos } \Gamma i"
obtains p' where "p = {l}receive(fmsg). p''"
and "l' \in \text{labels } \Gamma qq"
lemmas oelimders =
  oelimder_guard
  oelimder_assign
  oelimder_uCAST
  oelimder_bcast
  oelimder_gcast
  oelimder_send
  oelimder_deliver
  oelimder_receive

declare
  oseqpTEs [cterms_seqte]
  oelimders [cterms_elimders]

end

15 Lemmas for open partial networks

theory OPnet
imports OAWN_SOS OInvariants
begin

These lemmas mostly concern the preservation of node structure by \textit{opnet_sos} transitions.

lemma opnet_maintains_dom:
  assumes "((\sigma, \ns), a, (\sigma', \ns')) \in \text{trans (opnet np p)}"
  shows "\text{net_ips ns = net_ips ns'}"
  using \textit{assms} proof (induction p arbitrary: \sigma \ns a \sigma' \ns')
  fix i R \sigma \ns a \sigma' \ns'
  assume "((\sigma, \ns), a, (\sigma', \ns')) \in \text{trans (opnet np \langle i; R \rangle)}"
  hence "((\sigma, \ns), a, (\sigma', \ns')) \in \text{onode_sos (trans (np i))}" ..
  thus "\text{net_ips ns = net_ips ns'}"
  by (simp add: net_ips_is_dom_netmap, erule onode_sos.cases, simp_all)

next
  fix p1 p2 \sigma \ns a \sigma' \ns'
  assume "\forall \sigma \ns a \sigma' \ns'. ((\sigma, \ns), a, (\sigma', \ns')) \in \text{trans (opnet np p1)} \Rightarrow \text{net_ips ns = net_ips ns'}" 
  and "\forall \sigma \ns a \sigma' \ns'. ((\sigma, \ns), a, (\sigma', \ns')) \in \text{trans (opnet np p2)} \Rightarrow \text{net_ips ns = net_ips ns'}"
  hence "((\sigma, \ns), a, (\sigma', \ns')) \in \text{trans (opnet np (p1 || p2))}"
  thus "\text{net_ips ns = net_ips ns'}"
  by simp (erule opnet_sos.cases, simp_all)
qed

lemma opnet_net_ips_net_tree_ips:
  assumes "((\sigma, \ns)) \in \text{oreachable (opnet np p) S U}"
  shows "\text{net_ips ns = net_tree_ips p}"
  using \textit{assms} proof (induction rule: \textit{oreachable_pair_induct})
  fix \sigma s
  assume "((\sigma, \ns)) \in \text{init (opnet np p)}"
  thus "\text{net_ips s = net_tree_ips p}"
  proof (induction p arbitrary: \sigma s)
  fix p1 p2 \sigma s
  assume IH1: "((\sigma s. (\sigma, s)) \in \text{init (opnet np p1)} \Rightarrow \text{net_ips s = net_tree_ips p1})"
  and IH2: "((\sigma s. (\sigma, s)) \in \text{init (opnet np p2)} \Rightarrow \text{net_ips s = net_tree_ips p2})"
  hence "((\sigma, s)) \in \text{init (opnet np (p1 || p2))}"
  thus "\text{net_ips s = net_tree_ips (p1 || p2)}"
  by (clarsimp simp add: net_ips_is_dom_netmap, erule onode_sos_cases, metis Un_commute)
qed (clarsimp simp add: onode_comps)

next
  fix \sigma s \sigma' s' a
  assume "((\sigma, s)) \in \text{oreachable (opnet np p) S U}"
  and "\text{net_ips s = net_tree_ips p}"


and \(((\sigma, s), a, (\sigma', s')) \in \text{trans (opnet np p)}\)
and \("S \sigma \sigma' a"\)
thus "net_ips s' = net_tree_ips p"
by (simp add: net_ips_is_dom_netmap)
\((\text{metis net_ips_is_dom_netmap opnet_maintains_dom})\)
qed simp

lemma opnet_net_ips_net_tree_ips_init:
assumes "((\sigma, ns), a, ((\sigma', s')) \in \text{init (opnet np p)}"
shows "net_ips ns = net_tree_ips p"
using assms(1) by (rule oreachable_init [THEN opnet_net_ips_net_tree_ips])

lemma opartial_net_preserves_subnets:
assumes "((\sigma, \text{SubnetS s t}), a, ((\sigma', st')) \in \text{opnet_sos (trans (opnet np p1)) (trans (opnet np p2)})"
shows "\exists s' t'. st' = \text{SubnetS s' t'}"
using assms by cases simp_all

lemma net_par_oreachable_is_subnet:
assumes "((\sigma, st), a, (\sigma', st')) \in \text{oreachable (opnet np (p1 \parallel p2)) S U}"
shows "\exists s t. st = \text{SubnetS s t}"
proof
- def p \equiv "((\sigma, st)"
with assms have "p \in \text{oreachable (opnet np (p1 \parallel p2)) S U}" by simp
hence "\exists \sigma s t. p = ((\sigma, \text{SubnetS s t})"
by induct (auto dest!: opartial_net_preserves_subnets)
with p_def show \(?thesis by simp
qed

end

16 Lifting rules for (open) nodes

theory ONode_Lifting
importsAWN OAWN_SOS OInvariants
begin

lemma node_net_state':
assumes "s \in \text{oreachable (\langle i : T : R \rangle o) S U}"
shows "\exists \sigma \zeta R. s = (\sigma, \text{NodeS i \zeta R})"
using assms proof induction
fix s
assume "s \in \text{init (\langle i : T : R \rangle o)}"
then obtain \sigma \zeta where "s = (\sigma, \text{NodeS i \zeta R})"
by (auto simp: onode_comps)
thus "\exists \sigma \zeta R. s = (\sigma, \text{NodeS i \zeta R})" by auto
next
fix s a \sigma'
assume rt: "s \in \text{oreachable (\langle i : T : R_i \rangle o) S U}"
and ih: "\exists \sigma \zeta R. s = (\sigma, \text{NodeS i \zeta R})"
and "U (fst s) \sigma'"
then obtain \sigma \zeta R
where "(\sigma, \text{NodeS i \zeta R}) \in \text{oreachable (\langle i : T : R_i \rangle o) S U}"
and "U \sigma \sigma'" and "snd s = \text{NodeS i \zeta R}" by auto
from this(1-2)
have "(\sigma', \text{NodeS i \zeta R}) \in \text{oreachable (\langle i : T : R_i \rangle o) S U}"
by - (erule(1) oreachable_other')
with 'snd s = NodeS i \zeta R' show "\exists \sigma \zeta R. (\sigma', snd s) = (\sigma, \text{NodeS i \zeta R})" by simp
next
fix s a s'
assume rt: "s \in \text{oreachable (\langle i : T : R_i \rangle o) S U}"
and ih: "\exists \sigma \zeta R. s = (\sigma, \text{NodeS i \zeta R})"
and tr: "(s, a, s') \in \text{trans (\langle i : T : R_i \rangle o)}"
and "S (fst s) (fst s') a"
from ih obtain \sigma \zeta R where "s = (\sigma, \text{NodeS i \zeta R})" by auto
87
with tr have "((σ, Nodes i ζ R), a, s') ∈ onode_sos (trans T)"
  by (simp add: onode_comps)
then obtain σ' ζ' R' where "s' = (σ', Nodes i ζ' R')"
  using onode_sos_dest_is_net_state' by metis
with tr 's = (σ, Nodes i ζ R)' show "∃ σ ζ R. s' = (σ, Nodes i ζ R)"
  by simp

qd

lemma node_net_state:
  assumes "(σ, s) ∈ ooreachable ((i : T : R_i) o) S U"
  shows "∃ σ ζ R. s = Nodes i ζ R"
  using assms
  by (metis Pair_inject node_net_state')

lemma node_net_state_trans [elim]:
  assumes sor: "(σ, s) ∈ ooreachable ((i : ζ : R) o) S U"
  and str: "((σ, s), a, (σ', s')) ∈ trans ((i : ζ : R) o)"
  obtains ζ R ζ' R' where "s = Nodes i ζ R"
  and "s' = Nodes i ζ' R'"
  proof
    assume *: "∀ζ R ζ' R'. s = Nodes i ζ R → s' = Nodes i ζ' R' → thesis"
    from sor obtain ζ R where "s = Nodes i ζ R"
      by (metis node_net_state)
    moreover with str obtain ζ' R' where "s' = Nodes i ζ' R'"
      by (simp only: onode_comps)
    ultimately show thesis by (rule *)
  qed

lemma nodemap_induct' [consumes, case_names init other local]:
  assumes "(σ, Nodes ii ζ R) ∈ ooreachable ((ii : T : R_i) o) S U"
  and init: "∀σ ζ. (σ, Nodes ii ζ R) ∈ init ((ii : T : R_i) o) → P (σ, Nodes ii ζ R)"
  and other: "∀σ ζ R σ'. ζ σ' a. [ (σ, Nodes ii ζ R) ∈ ooreachable ((ii : T : R_i) o) S U; U ζ σ'; P (σ, Nodes ii ζ R) ] → P (σ', Nodes ii ζ R)"
  and local: "∀σ ζ R σ' ζ' R' a. [ (σ, Nodes ii ζ R) ∈ ooreachable ((ii : T : R_i) o) S U; (σ, Nodes ii ζ R), a, (σ', Nodes ii ζ' R')) ∈ trans ((ii : T : R_i) o); S σ σ' a; P (σ, Nodes ii ζ R) ] → P (σ', Nodes ii ζ' R')"
  shows "P (σ, Nodes ii ζ R)"
  using assms(1) proof induction
  fix s
  assume "s ∈ init ((ii : T : R_i) o)"
  hence "s ∈ ooreachable ((ii : T : R_i) o) S U"
    by (rule ooreachable_init)
  with 's ∈ init ((ii : T : R_i) o)' obtain σ ζ where "s = (σ, Nodes ii ζ R)"
    using node_net_state by (simp add: onode_comps) metis
  with 's ∈ init ((ii : T : R_i) o)' and init show "P s" by simp
  next
  fix s a σ'
  assume sr: "s ∈ ooreachable ((ii : T : R_i) o) S U"
  and "U (fst s) σ'"
  and "P s"
  from sr obtain σ ζ R where "s = (σ, Nodes ii ζ R)"
    using node_net_state' by metis
  with sr 'U (fst s) σ' 'P s' show "P (σ', snd s)"
    by simp (metis other)
  next
  fix s a s'
  assume sr: "s ∈ ooreachable ((ii : T : R_i) o) S U"
  and tr: "(s, a, s') ∈ trans ((ii : T : R_i) o)"
  and "S (fst s) (fst s') a"
  and "P s"
from this(1-3) have "s' ∈ orreachable ((ii : T : R_i) o) S U"
by - (erule(2) orreachable_local)
then obtain σ' ζ' R' where [simp]: "s' = (σ', NodeS ii ζ' R')"
using node_net_state' by metis
from sr and 'P s' obtain σ ζ R
where [simp]: "s = (σ, NodeS ii ζ R)"
and A1: "(σ, NodeS ii ζ R) ∈ orreachable ((ii : T : R_i) o) S U"
and A4: "P (σ, NodeS ii ζ R)"
using node_net_state' by metis
with tr and 'S (fst s) (fst s') a'
have A2: "(((σ, NodeS ii ζ R), a, (σ', NodeS ii ζ' R')) ∈ trans ((ii : T : R_i) o)"
and A3: "S (σ σ' a)" by simp_all
from A1 A2 A3 A4 have "P (σ', NodeS ii ζ' R')" by (rule local)
thus "P s'" by simp

qed

lemma nodemap_induct [consumes, case_names init step]:
assumes "(σ, NodeS ii ζ R) ∈ orreachable ((ii : T : R_i) o) S U"
and init: "∀σ ζ. (σ, NodeS ii ζ R) ∈ init (ii : T : R_i) o → P σ ζ R_i"
and other: "∀σ ζ. (σ, NodeS ii ζ R) ∈ orreachable ((ii : T : R_i) o) S U;
U σ ζ_i; P σ ζ R_i ] → P σ' ζ R"
and local: "∀σ ζ R σ' ζ' R'. a.
[(σ, NodeS ii ζ R) ∈ orreachable ((ii : T : R_i) o) S U;
(σ, NodeS ii ζ' R') ∈ trans ((ii : T : R_i) o);
S σ ζ a; P σ ζ R_i ] → P σ' ζ' R'
shows "P σ ζ R" using assms(1) proof (induction "(σ, NodeS ii ζ R)" arbitrary: σ ζ R)
fix σ ζ R
assume a1: "(σ, NodeS ii ζ R) ∈ init (ii : T : R_i) o"
hence "R = R_i" by (simp add: init_onode_comp)
with a1 have "((σ, NodeS ii ζ R), a, (σ', NodeS ii ζ' R')) ∈ trans ((ii : T : R_i) o);
S σ ζ a; P σ ζ R_i ] → P σ' ζ' R'
next
fix st a a' ζ R'
assume "st ∈ orreachable ((ii : T : R_i) o) S U"
and tr: "(st, a, (σ', NodeS ii ζ' R')) ∈ trans ((ii : T : R_i) o)"
and IH: "∀σ R. st = (σ, NodeS ii ζ R) → P σ R"
from this(1) obtain σ ζ R where "st = (σ, NodeS ii ζ R)"
and "(σ, NodeS ii ζ R) ∈ orreachable ((ii : T : R_i) o) S U"
by (metis node_net_state')

note this(2)

moreover from tr and 'st = (σ, NodeS ii ζ R)'
have "((σ, NodeS ii ζ R), a, (σ', NodeS ii ζ' R')) ∈ trans ((ii : T : R_i) o)" by simp
moreover from 'S (fst st) (fst (σ', NodeS ii ζ' R')) a' and 'st = (σ, NodeS ii ζ R)'
have "S σ σ' a" by simp
moreover from IH and 'st = (σ, NodeS ii ζ R)' have "P σ ζ R".
ultimately show "P σ ζ R'" by (rule local)

next
fix st σ' ζ R
assume "st ∈ orreachable ((ii : T : R_i) o) S U"
and "U (fst st) σ'"
and "snd st = NodeS ii ζ R"
and IH: "∀σ R. st = (σ, NodeS ii ζ R) → P σ R"
from this(1,3) obtain σ where "st = (σ, NodeS ii ζ R)"
and "(σ, NodeS ii ζ R) ∈ orreachable ((ii : T : R_i) o) S U"
by (metis surjective_pairing)

note this(2)

moreover from 'U (fst st) σ' and 'st = (σ, NodeS ii ζ R)' have "U σ σ'" by simp
moreover from IH and 'st = (σ, NodeS ii ζ R)' have "P σ ζ R".
ultimately show "P σ ζ R'" by (rule other)

qed
lemma node_addressD [dest, simp]:
assumes "("σ, NodeS i ζ R") ∈ ororeachable ((ii : T : R)i) U"
shows "i = ii"
using assms by (clarsimp dest!: node_net_state')

lemma node_proc_reachable [dest]:
assumes "("σ, NodeS i ζ R") ∈ ororeachable (⟨ii : T : R⟩o)
(otherwith S {ii} (oarrivemsg I)) (other U {ii})"
and sgivesu: "∀ ξ ξ'. S ξ ξ' =⇒ U ξ ξ'"
sows "("σ, ζ") ∈ ororeachable T (otherwith S {ii} (orecvmsg I)) (other U {ii})"
proof -
from assms(1) have "("σ, NodeS ii ζ R") ∈ ororeachable (⟨ii : T : R⟩o)
(otherwith S {ii} (oarrivemsg I)) (other U {ii})"
by - (frule node_addressD, simp)
thus ?thesis
proof (induction rule: nodemap_induct)
  fix σ ζ
  assume "("σ, NodeS ii ζ R") ∈ init (⟨ii : T : R⟩o)"
  hence "("σ, ζ") ∈ init T" by (auto simp: onode_comps)
  thus "("σ, ζ") ∈ ororeachable T (otherwith S {ii} (orecvmsg I)) (other U {ii})"
  by (rule oreachable_init)
next
  fix σ ζ R σ' ζ' R' a
  assume "other U {ii} σ σ''" and "("σ, ζ") ∈ ororeachable T (otherwith S {ii} (orecvmsg I)) (other U {ii})"
  thus "("σ', ζ") ∈ ororeachable T (otherwith S {ii} (orecvmsg I)) (other U {ii})"
  by - (rule oreachable_other')
next
  fix σ ζ R σ' ζ' R' a
  assume rs: "("σ, NodeS ii ζ R") ∈ ororeachable (⟨ii : T : R⟩o)
(otherwith S {ii} (oarrivemsg I)) (other U {ii})"
  and tr: "("σ, NodeS ii ζ R"), a, (σ', NodeS ii ζ' R')") ∈ trans (⟨ii : T : R⟩o)"
  and ow: "otherwith S {ii} (oarrivemsg I) σ σ' a"
  and ih: "("σ, ζ") ∈ ororeachable T (otherwith S {ii} (orecvmsg I)) (other U {ii})"
  from ow have *: "σ' ii = σ ii =⇒ other U {ii} σ σ''"
  by (clarsimp elim!: otherwithE) (rule otherI, simp_all, metis sgivesu)
  from tr have "("σ, NodeS ii ζ R"), a, (σ', NodeS ii ζ' R')") ∈ onode_sos (trans T)"
  by (simp add: onode_comps)
  thus "("σ', ζ'") ∈ ororeachable T (otherwith S {ii} (orecvmsg I)) (other U {ii})"
proof cases
  case onode_bcast
  with ih and ow show ?thesis
  by (auto elim!: oreachable_local' otherwithE)
next
  case onode_gcast
  with ih and ow show ?thesis
  by (auto elim!: oreachable_local' otherwithE)
next
  case onode_ucast
  with ih and ow show ?thesis
  by (auto elim!: oreachable_local' otherwithE)
next
  case onode_notucast
  with ih and ow show ?thesis
  by (auto elim!: oreachable_local' otherwithE)
next
  case onode_deliver
  with ih and ow show ?thesis
  by (auto elim!: oreachable_local' otherwithE)
next
  case onode_tau
  with ih and ow show ?thesis
  by (auto elim!: oreachable_local' otherwithE)
next
case onode_receive
with ih and ow show ?thesis
  by (auto elim!: oreachable_local' otherwithE)
next
case (onode_arrive m)
hence "ζ' = ζ" and "σ' ii = σ ii" by auto
from this(2) have "other U {ii} σ σ'" by (rule *)
with ih and 'ζ' = ζ' show ?thesis by auto
next
case onode_connect1
hence "ζ' = ζ" and "σ' ii = σ ii" by auto
from this(2) have "other U {ii} σ σ'" by (rule *)
with ih and 'ζ' = ζ' show ?thesis by auto
next
case onode_connect2
hence "ζ' = ζ" and "σ' ii = σ ii" by auto
from this(2) have "other U {ii} σ σ'" by (rule *)
with ih and 'ζ' = ζ' show ?thesis by auto
next
case onode_connect_other
hence "ζ' = ζ" and "σ' ii = σ ii" by auto
from this(2) have "other U {ii} σ σ'" by (rule *)
with ih and 'ζ' = ζ' show ?thesis by auto
next
case onode_disconnect1
hence "ζ' = ζ" and "σ' ii = σ ii" by auto
from this(2) have "other U {ii} σ σ'" by (rule *)
with ih and 'ζ' = ζ' show ?thesis by auto
next
case onode_disconnect2
hence "ζ' = ζ" and "σ' ii = σ ii" by auto
from this(2) have "other U {ii} σ σ'" by (rule *)
with ih and 'ζ' = ζ' show ?thesis by auto
next
case onode_disconnect_other
hence "ζ' = ζ" and "σ' ii = σ ii" by auto
from this(2) have "other U {ii} σ σ'" by (rule *)
with ih and 'ζ' = ζ' show ?thesis by auto
qed
qed

lemma node_proc_reachable_statelessasm [dest]:
assumes "(σ, NodeS i ζ R) ∈ oreachable ((ii : T : R_i) o)
  (otherwith (λ_. True) {ii} (oarrivemsg I))
  (other (λ_. True) {ii})"
shows "(σ, ζ) ∈ oreachable T
  (otherwith (λ_. True) {ii} (orecvmsg I)) (other (λ_. True) {ii})"
using assms
by (rule node_proc_reachable) simp_all

lemma node_lift:
assumes "T |= (otherwith S {ii} (orecvmsg I), other U {ii} →) global P"
and "S ξ ξ' |= U ξ ξ'" shows "(ii : T : R_i) o |= (otherwith S {ii} (oarrivemsg I), other U {ii} →) global P"
proof (rule oinvariant_oreachableI)
fix σ ξ
assume "(σ, ξ) ∈ oreachable ((ii : T : R_i) o) (otherwith S {ii} (oarrivemsg I)) (other U {ii})"
moreover then obtain i s R where "ζ = NodeS i s R"
  by (metis node_net_state)
ultimately have "(σ, NodeS i s R) ∈ oreachable ((ii : T : R_i) o)
  (otherwith S {ii} (oarrivemsg I)) (other U {ii})"
  by simp
hence \( (\sigma, s) \in \text{oreachable} \ T \ (\text{otherwith} \ S \ \{i\} \ (\text{orecvmsg} \ I)) \ (\text{other} \ U \ \{i\}) \) 
by - (erule node\_proc\_reachable, erule assms(2))
with assms(1) show "global \( P (\sigma, \zeta) \)"
by (metis fst\_conv globalsimp oinvariantD)
qed

lemma node\_lift\_step [intro]:
assumes pinv: "T \models_A (\text{otherwith} \ S \ \{i\} \ (\text{orecvmsg} \ I), \text{other} \ U \ \{i\} \rightarrow) \text{globala} \ (\lambda(\sigma, _, \sigma'). \ Q \ (\sigma, \sigma'))"
and other: "\( \nexists \sigma', \text{other} \ U \ \{i\} \ (\sigma, \sigma' \Rightarrow Q \ (\sigma, \sigma')) \)"
and sgivesu: "\( \nexists \zeta', S \ (\zeta, \zeta' \Rightarrow U \ (\zeta, \zeta')) \)"
shows "(i : T : R_i)_o \models_A (\text{otherwith} \ S \ \{i\} \ (\text{cararrivemsg} \ I), \text{other} \ U \ \{i\} \rightarrow) \text{globala} \ (\lambda(\sigma, _, \sigma'). \ Q \ (\sigma, \sigma'))"
(is "\_ \_A (\zeta, \zeta' \Rightarrow U \ (\zeta, \zeta'))")
proof (rule ostep\_invariantI, simp)
fix \( \sigma \) \( a' \) \( s' \)
assume rs: "(\sigma, s) \in \text{oreachable} \ ((i : T : R_i)_o) \ ?S \ ?U"
and tr: "((\sigma, s), a, (\sigma', s')) \in \text{trans} \ ((i : T : R_i)_o)"
and ow: "?S \ (\sigma, \sigma') \ a"
from ow have *: "\sigma' \ i = \sigma \ i \Rightarrow \text{other} \ U \ \{i\} \ (\sigma, \sigma')" 
by (clarsimp elim!: otherwithE) (rule otherI, simp_all, metis sgivesu)
from rs tr obtain \( \zeta \)
where [simp]: "\( s = \text{NodeS} \ i : T : R \)"
and "\( (\sigma, \text{NodeS} \ i : T : R) \in \text{oreachable} \ ((i : T : R_i)_o) \ ?S \ ?U) \)"
by (metis node\_net\_state)
from this(2) have or: "\( (\sigma, \zeta) \in \text{oreachable} \ T \ (\text{otherwith} \ S \ \{i\} \ (\text{orecvmsg} \ I)) \ ?U \)"
by (rule node\_proc\_reachable [OF _ assms(3)])
from tr have "((\sigma, \text{NodeS} \ i : T : R), a, (\sigma', s')) \in \text{onode\_sos} \ (\text{trans} \ T)"
by (simp add: onode\_comps)
thus "Q \ (\sigma, \sigma')"
proof cases
fix \( m \) \( \zeta' \)
assume "\( a = R : \text{cast} \ m \)"
and tr': "(\( \sigma, \zeta \), \text{broadcast} \ m, (\sigma', \zeta')) \in \text{trans} \ T"
from this(1) and \( ?S \ (\sigma, \sigma') \ a' \) have "otherwith \ S \ \{i\} \ (\text{orecvmsg} \ I) \ (\sigma, \sigma') \ (\text{broadcast} \ m)"
by (auto elim!: otherwithE)
with or tr' show ?thesis by (rule ostep\_invariantD [OF pinv, simplified])
next
fix \( D \) \( m \) \( \zeta' \)
assume "\( a = (R \cap D) : \text{cast} \ m \)"
and tr': "(\( \sigma, \zeta \), \text{groupcast} \ D \ m, (\sigma', \zeta')) \in \text{trans} \ T"
from this(1) and \( ?S \ (\sigma, \sigma') \ a' \) have "otherwith \ S \ \{i\} \ (\text{orecvmsg} \ I) \ (\sigma, \sigma') \ (\text{groupcast} \ D \ m)"
by (auto elim!: otherwithE)
with or tr' show ?thesis by (rule ostep\_invariantD [OF pinv, simplified])
next
fix \( d \) \( m \) \( \zeta' \)
assume "\( a = \{d\} : \text{cast} \ m \)"
and tr': "(\( \sigma, \zeta \), \text{unicast} \ d \ m, (\sigma', \zeta')) \in \text{trans} \ T"
from this(1) and \( ?S \ (\sigma, \sigma') \ a' \) have "otherwith \ S \ \{i\} \ (\text{orecvmsg} \ I) \ (\sigma, \sigma') \ (\text{unicast} \ d \ m)"
by (auto elim!: otherwithE)
with or tr' show ?thesis by (rule ostep\_invariantD [OF pinv, simplified])
next
fix \( d \) \( \zeta' \)
assume "\( a = \tau \)"
and tr': "(\( \sigma, \zeta \), \text{unicast} \ - \text{unicast} \ d, (\sigma', \zeta')) \in \text{trans} \ T"
from this(1) and \( ?S \ (\sigma, \sigma') \ a' \) have "otherwith \ S \ \{i\} \ (\text{orecvmsg} \ I) \ (\sigma, \sigma') \ (\text{unicast} \ - \text{unicast} \ d)"
by (auto elim!: otherwithE)
with or tr' show ?thesis by (rule ostep\_invariantD [OF pinv, simplified])
next
fix \( d \) \( \zeta' \)
assume "\( a = i : \text{deliver} \ (d) \)"
and tr': "(\( \sigma, \zeta \), \text{deliver} \ d, (\sigma', \zeta')) \in \text{trans} \ T"
from this(1) and \( ?S \ (\sigma, \sigma') \ a' \) have "otherwith \ S \ \{i\} \ (\text{orecvmsg} \ I) \ (\sigma, \sigma') \ (\text{deliver} \ d)"
by (auto elim!: otherwithE)
with or tr' show ?thesis by (rule ostep\_invariantD [OF pinv, simplified])
next
fix $\zeta'$
assume "$a = \tau$"
and $tr': (((\sigma, \zeta), \tau, (\sigma', \zeta')) \in \text{trans } T$
from this(1) and 'S $\sigma \sigma'$ a' have "otherwith $S \{i\} \text{orecvmsg } I \sigma \sigma' \tau"
by (auto elim!: otherwithE)
with or $tr'$ show ?thesis by (rule ostep_invariantD [OF pinv, simplified])

next
fix $m \zeta'$
assume "$a = \{i\}-\{i\}:\text{arrive}(m)"
and "$\sigma' i = \sigma i$"
from this(2) have "other $U \{i\} \sigma \sigma'$" by (rule *)
thus ?thesis by (rule other)

next
fix $i'$
assume "$a = \text{connect}(i, i')""
and "$\sigma' i = \sigma i$"
from this(2) have "other $U \{i\} \sigma \sigma'$" by (rule *)
thus ?thesis by (rule other)

next
fix $i'$
assume "$a = \text{connect}(i', i')""
and "$\sigma' i = \sigma i$"
from this(2) have "other $U \{i\} \sigma \sigma'$" by (rule *)
thus ?thesis by (rule other)

next
fix $i'$
assume "$a = \text{disconnect}(i', i')""
and "$\sigma' i = \sigma i$"
from this(2) have "other $U \{i\} \sigma \sigma'$" by (rule *)
thus ?thesis by (rule other)

next
fix $i'$
assume "$a = \text{disconnect}(i, i')""
and "$\sigma' i = \sigma i$"
from this(2) have "other $U \{i\} \sigma \sigma'$" by (rule *)
thus ?thesis by (rule other)

next
fix $i'$
assume "$a = \text{disconnect}(i', i')""
and "$\sigma' i = \sigma i$"
from this(2) have "other $U \{i\} \sigma \sigma'$" by (rule *)
thus ?thesis by (rule other)

next
fix $i'$
assume "$a = \text{disconnect}(i', i')""
and "$\sigma' i = \sigma i$"
from this(2) have "other $U \{i\} \sigma \sigma'$" by (rule *)
thus ?thesis by (rule other)

next
fix $i'$$i''$
assume "$a = \text{disconnect}(i', i'')""
and "$\sigma' i = \sigma i$"
from this(2) have "other $U \{i\} \sigma \sigma'$" by (rule *)
thus ?thesis by (rule other)

next
fix $i'$
assume "$a = \text{connect}(i', i')""
and "$\sigma' i = \sigma i$"
from this(2) have "other $U \{i\} \sigma \sigma'$" by (rule *)
thus ?thesis by (rule other)

next
assume "$T \models_A (\lambda \sigma \_ \text{orecvmsg } I \sigma, \text{other } (\lambda \_ \_. \text{True}) \{i\} \rightarrow)$
and "$(\lambda Q \xi. Q \xi \xi)"
shows "$(i : T : R_i) \models_A (\lambda \sigma \_ \text{arrivemsg } I \sigma, \text{other } (\lambda \_ \_. \text{True}) \{i\} \rightarrow)"
lemma node_lift_anycast [intro]:

proof -
  from assms(1)
  have "T \models_A (\text{otherwith} (\lambda_\_. \text{True}) \{i\} (\text{orecmsg} I), \text{other} (\lambda_\_. \text{True}) \{i\} \rightarrow)
    \text{globala} (\lambda(\sigma, a, \sigma'). \text{anycast} (Q (\sigma \sigma') a))"
    by rule auto
  with assms(2) have "(i : T : R_i)_o \models_A (\text{otherwith} (\lambda_\_. \text{True}) \{i\} (\text{oarrivmsg} I),
    \text{other} (\lambda_\_. \text{True}) \{i\} \rightarrow)
    \text{globala} (\lambda(\sigma, a, \sigma'). \text{castmsg} (Q (\sigma \sigma') a))"
    by (rule node_lift_step, auto)
  thus ?thesis by rule auto
qed

lemma node_lift_anycast_statelessassm [intro]:

proof (rule ostep_invariantI, simp)
  fix \sigma s a \sigma' s'
  assume "\sigma, s \in \text{oreachable} ((i : T : R_i)_o) S U"
  and tr: "((\sigma, s), a, (\sigma', s')) \in \text{trans} ((i : T : R_i)_o)"
  and "S \sigma \sigma' a"
  from this(1-2) obtain \zeta R
  where [simp]: "s = NodeS \i : T : R\" 
    and [simp]: "NodeS \i : T : R \in \text{oreachable} ((i : T : R_i)_o) S U"
  by (metis node_net_state)
  from this(2) have "(\sigma, \zeta) \in \text{oreachable} T (\text{otherwith} S \{i\} (\text{orecmsg} I)) U"
    by (rule node_proc_reachable [OF \_ assms(2)])
  moreover from tr have "((\sigma, NodeS \i : T : R), a, (\sigma', s')) \in \text{onode_sos} (\text{trans} T)"
    by (simp add: onode_comps)
  ultimately show "\text{castmsg} (Q \sigma \sigma') a" using 'S \sigma \sigma' a'
    by (erule onode_sos_cases, auto elim!: ostep_invariantD [OF \_ assms])
qed

lemma node_local_deliver:

"(i : \zeta : T : R_i)_o \models_A (S, U \rightarrow) \text{globala} (\lambda(\_, a, \_). \forall j. j \neq i \rightarrow (\forall d. a \neq j: \text{deliver}(d)))"

proof (rule ostep_invariantI, simp)
  fix \sigma s a \sigma' s'
  assume "\sigma, s \in \text{oreachable} ((i : \zeta : T : R_i)_o) S U"
  and "((\sigma, s), a, (\sigma', s')) \in \text{trans} ((i : \zeta : T : R_i)_o)"
  and "S \sigma \sigma' a"
  moreover from this(1-2) obtain \zeta R \zeta' R'
  where "s = NodeS \i : T : R\" and "s' = NodeS \i : T : R'\" ..
  ultimately show "\forall j. j \neq i \rightarrow (\forall d. a \neq j: \text{deliver}(d))"
by (cases a) (auto simp add: onode_comps)

qed

lemma node_tau_deliver_unchanged:
"(i : ζi : R_i)_o \models_A (S, U \rightarrow) \text{globala } (\lambda(\sigma, a, \sigma'). a = \tau \lor (\exists i d. a = i:deliver(d))
\rightarrow (\forall j. j \neq i \rightarrow \sigma' = j))"
proof (rule ostep_invariantI, clarsimp simp only: globalasimp snd_conv fst_conv)
fix σ s a σ' s' j
assume "((i : ζi : R_i)_o) S U"
and "((i : ζi : R_i)_o) S U"
and "s = NodeS i σ σ' a"
and "s = NodeS i σ σ' a"
and "a = τ \lor (\exists i d. a = i:deliver(d))"
and "j \neq i"
moreover from this(1-2) obtain ζ R ζ' R' where "s = NodeS i ζ R" and "s' = NodeS i ζ' R'" ..
ultimately show "σ' j = σ j"
by (cases a) (auto simp del: step_node_tau simp add: onode_comps)

qed

end

17 Lifting rules for (open) partial networks

theory OPnet_Lifting
imports ONode_Lifting OAWN_SOS OPnet
begin

lemma oreachable_par_subnet_induct [consumes, case_names init other local]:
assumes "(σ, SubnetS s t) \in oreachable (opnet onp (p_1 || p_2)) S U"
and init: "\\(\land \sigma s t. (\sigma, SubnetS s t) \in init (opnet onp (p_1 || p_2)) \rightarrow P \sigma s t"
and other: "\\(\land \sigma s t \sigma'. [ (\sigma, SubnetS s t) \in oreachable (opnet onp (p_1 || p_2)) S U; U \sigma \sigma'; P \sigma s t ] \rightarrow P \sigma' s' t'"
and local: "\\(\land \sigma s t \sigma' s' t' a. [ (\sigma, SubnetS s t) \in oreachable (opnet onp (p_1 || p_2)) S U; ((\sigma, SubnetS s t), a, (\sigma', SubnetS s' t')) \in trans (opnet onp (p_1 || p_2)); S \sigma \sigma' a; P \sigma s t ] \rightarrow P \sigma' s' t'"

shows "P \sigma s t"
using assms(1) proof (induction "(\sigma, SubnetS s t)" arbitrary: s t σ)
fix s t σ
assume "(\sigma, SubnetS s t) \in init (opnet onp (p_1 || p_2))"
with init show "P \sigma s t".

next
fix st a s' t' σ'
assume "st \in oreachable (opnet onp (p_1 || p_2)) S U"
and tr: "((st, a, (σ', SubnetS s' t')) \in trans (opnet onp (p_1 || p_2)))"
and "S (fst st) (fst (σ', SubnetS s' t')) a"
and IH: "\\(\land s t σ. st = (σ, SubnetS s t) \rightarrow P \sigma s t"
from this(1) obtain s t σ where "st = (σ, SubnetS s t)"
and "(σ, SubnetS s t) \in oreachable (opnet onp (p_1 || p_2)) S U"
and (metis net_par_oreachable_is_subnet pair Collapse)

note this(2)
moreover from tr and 'st = (σ, SubnetS s t)'
have "((σ, SubnetS s t), a, (σ', SubnetS s' t')) \in trans (opnet onp (p_1 || p_2))" by simp
moreover from 'S (fst st) (fst (σ', SubnetS s' t')) a' and 'st = (σ, SubnetS s t)'
have "S σ σ' a" by simp
moreover from IH and 'st = (σ, SubnetS s t)'
ultimately show "P \sigma s t".

next
fix st σ' s t
assume "st \in oreachable (opnet onp (p_1 || p_2)) S U"
and "U (fst st) σ'"
and "snd st = SubnetS s t"
and IH: "\\(\land s t σ. st = (σ, SubnetS s t) \rightarrow P \sigma s t"
from this(1,3) obtain σ where "st = (σ, SubnetS s t)"
and "(σ, SubnetS s t) \in oreachable (opnet onp (p_1 || p_2)) S U"
and (metis pair Collapse)
moreover from IH and \( 'st = (\sigma, \text{SubnetS } s t) \)' have \( 'P \sigma s t' \).
ultimately show \( 'P \sigma' s t' \) by (rule other)
and "\(A. U \xi \xi\)"

and act1: "opnet onp \(p_1\) \(\vdash_A (\lambda \sigma \cdot \text{oarrivemsg } I \sigma, \text{other } U \,(\text{net_tree_ips } p_1) \rightarrow)\)

globa (\(\lambda(\sigma, a, \sigma'). \text{castmsg } (I \sigma) a\)

\(\wedge (a = \tau \vee (\exists i. d. a = i: \text{deliver}(d)) \rightarrow (\forall i. \text{net_tree_ips } p_1. U (\sigma i) (\sigma' i))\)

\(\wedge (\forall i. i \notin \text{net_tree_ips } p_1 \rightarrow (\sigma' i = \sigma i)))\)"

and act2: "opnet onp \(p_2\) \(\vdash_A (\lambda \sigma \cdot \text{oarrivemsg } I \sigma, \text{other } U \,(\text{net_tree_ips } p_2) \rightarrow)\)

globa (\(\lambda(\sigma, a, \sigma'). \text{castmsg } (I \sigma) a\)

\(\wedge (a = \tau \vee (\exists i. d. a = i: \text{deliver}(d)) \rightarrow (\forall i. \text{net_tree_ips } p_2. U (\sigma i) (\sigma' i))\)

\(\wedge (\forall i. i \notin \text{net_tree_ips } p_2 \rightarrow (\sigma' i = \sigma i)))\)"

shows "\((\sigma, s) \in \text{oreachable } (\text{opnet onp } p_1) \,(\lambda \sigma \cdot \text{oarrivemsg } I \sigma) \,(\text{other } U \,(\text{net_tree_ips } p_1))\)

\(\wedge (\sigma, t) \in \text{oreachable } (\text{opnet onp } p_2) \,(\lambda \sigma \cdot \text{oarrivemsg } I \sigma) \,(\text{other } U \,(\text{net_tree_ips } p_2))\)

\(\text{net_tree_ips } p_1 \cap \text{net_tree_ips } p_2 = \{\}\)" using assms(1)

proof (induction rule: \text{oreachable_par_subnet_induct})

case (init \(\sigma s t\))

hence sinit: "\((\sigma, s) \in \text{init } (\text{opnet onp } p_1)\)"

and tinit: "\((\sigma, t) \in \text{init } (\text{opnet onp } p_2)\)"

and "\(\text{net_ips } s \cap \text{net_ips } t = \{\}\)" by auto

moreover from sinit have "\(\text{net_ips } s = \text{net_tree_ips } p_1\)"

by (rule \text{opnet_net_ips_net_tree_ips_init})

moreover from tinit have "\(\text{net_ips } t = \text{net_tree_ips } p_2\)"

by (rule \text{opnet_net_ips_net_tree_ips_init})

ultimately show ?case by (auto elim: \text{oreachable_init})

next

case (other \(\sigma s t \sigma'\))

hence "\(\text{other } U \,(\text{net_tree_ips } (p_1 \parallel p_2)) \sigma'\)"

and IHs: "\((\sigma, s) \in \text{oreachable } (\text{opnet onp } p_1) \,(?S p_1) \,(?U p_1)\)"

and IHt: "\((\sigma, t) \in \text{oreachable } (\text{opnet onp } p_2) \,(?S p_2) \,(?U p_2)\)"

and "\(\text{net_tree_ips } p_1 \cap \text{net_tree_ips } p_2 = \{\}\)" by auto

have "\((\sigma', s) \in \text{oreachable } (\text{opnet onp } p_1) \,(?S p_1) \,(?U p_1)\)"

proof -

from '?'\(U (p_1 \parallel p_2) \sigma'\) and '?'\(A. U \xi \xi\) have "?U p_1 \sigma'\"

by (rule other\_net_tree_ips\_par_left)

with IHs show ?thesis by (erule(1) \text{oreachable_other'})

qed

moreover have "\((\sigma', t) \in \text{oreachable } (\text{opnet onp } p_2) \,(?S p_2) \,(?U p_2)\)"

proof -

from '?U (p_1 \parallel p_2) \sigma'\' and '?A. U \xi \xi' have "?U p_2 \sigma'\"

by (rule other\_net_tree_ips\_par_right)

with IHt show ?thesis by (erule(1) \text{oreachable_other'})

qed

ultimately show ?case using 'net_tree_ips p_1 \cap net_tree_ips p_2 = \{\}' by simp

next

case (local \(\sigma s t \sigma' s' t' a\))

hence stor: "\((\sigma, \text{SubnetS } s t) \in \text{oreachable } (\text{opnet onp } (p_1 \parallel p_2)) \,(?S (p_1 \parallel p_2)) \,(?U (p_1 \parallel p_2))\)"

and tr: "\((\sigma, \text{SubnetS } s t), a, (\sigma', \text{SubnetS } s' t') \in \text{trans } (\text{opnet onp } (p_1 \parallel p_2))\)"

and "\text{oarrivemsg } I \sigma a\"

and sor: "\((\sigma, s) \in \text{oreachable } (\text{opnet onp } p_1) \,(?S p_1) \,(?U p_1)\)"

and tor: "\((\sigma, t) \in \text{oreachable } (\text{opnet onp } p_2) \,(?S p_2) \,(?U p_2)\)"

and "\(\text{net_tree_ips } p_1 \cap \text{net_tree_ips } p_2 = \{\}\)" by auto

from tr have "\((\sigma, \text{SubnetS } s t), a, (\sigma', \text{SubnetS } s' t')\) \in \text{opnet_sos } (\text{trans } (\text{opnet onp } p_1)) \,(\text{trans } (\text{opnet onp } p_2))" by simp

hence "\((\sigma', s') \in \text{oreachable } (\text{opnet onp } p_1) \,(?S p_1) \,(?U p_1)\)

\(\wedge (\sigma', t') \in \text{oreachable } (\text{opnet onp } p_2) \,(?S p_2) \,(?U p_2)\)"
proof (cases)
  fix H K m H' K'
  assume "a = (H ∪ H') ∖ (K ∪ K') : arrive(m)"
  and str: "((σ, s), H¬K: arrive(m), (σ', s')) ∈ trans (opnet onp p1)"
  and ttr: "((σ, t), H¬K: arrive(m), (σ', t')) ∈ trans (opnet onp p2)"
  from this(1) and 'oarrivemsg I σ a' have "I σ m" by simp

  with sor str
  have "(σ', s') ∈ or reachable (opnet onp p1) (?S p1) (?U p1)"
    by - (erule(1) or reachable_local, auto)
  moreover from 'I σ m' tor ttr
  have "(σ', t') ∈ or reachable (opnet onp p2) (?S p2) (?U p2)"
    by - (erule(1) or reachable_local, auto)
  ultimately show ?thesis ..

next
  fix R m H K
  assume str: "((σ, s), R:*cast(m), (σ', s')) ∈ trans (opnet onp p1)"
    and ttr: "((σ, t), R¬K: arrive(m), (σ', t')) ∈ trans (opnet onp p2)"
  from sor str have "I σ m"
    by - (drule(1) ostep invariantD [OF act1], simp_all)
  with sor str
  have "(σ', s') ∈ or reachable (opnet onp p1) (?S p1) (?U p1)"
    by - (erule(1) or reachable_local, auto)
  moreover from 'I σ m' tor ttr
  have "(σ', t') ∈ or reachable (opnet onp p2) (?S p2) (?U p2)"
    by - (erule(1) or reachable_local, auto)
  ultimately show ?thesis ..

next
  fix i i'
  assume str: "((σ, s), connect(i, i'), (σ', s')) ∈ trans (opnet onp p1)"
    and ttr: "((σ, t), connect(i, i'), (σ', t')) ∈ trans (opnet onp p2)"
  with sor str
  have "(σ', s') ∈ or reachable (opnet onp p1) (?S p1) (?U p1)"
    by - (erule(1) or reachable_local, auto)
  moreover from 'I σ m' tor ttr
  have "(σ', t') ∈ or reachable (opnet onp p2) (?S p2) (?U p2)"
    by - (erule(1) or reachable_local, auto)
  ultimately show ?thesis ..

next
  fix i i'
  assume str: "((σ, s), disconnect(i, i'), (σ', s')) ∈ trans (opnet onp p1)"
    and ttr: "((σ, t), disconnect(i, i'), (σ', t')) ∈ trans (opnet onp p2)"
  with sor str
  have "(σ', s') ∈ or reachable (opnet onp p1) (?S p1) (?U p1)"
    by - (erule(1) or reachable_local, auto)
  moreover from 'I σ m' tor ttr
  have "(σ', t') ∈ or reachable (opnet onp p2) (?S p2) (?U p2)"
    by - (erule(1) or reachable_local, auto)
  ultimately show ?thesis ..

next
  fix i d
assume "t' = t"
    and str: "((σ, s), i:deliver(d), (σ', s')) ∈ trans (opnet onp p1)"

from sor str have "∀ j. j ∊ net_tree_ips p1 → σ' j = σ j"
    by (drule(1) ostep_invariantD [OF act1], simp_all)
moreover with 'net_tree_ips p1 ∩ net_tree_ips p2 = {}' have "∀ j. j ∊ net_tree_ips p1 → σ' j = σ j" by auto
moreover from sor str have "∀ j ∊ net_tree_ips p1. U (σ j) (σ' j)"
    by (drule(1) ostep_invariantD [OF act1], simp_all)
ultimately have "(σ', t') ∈ oreachable (opnet onp p2) (?[S p2] (?[U p2])" using tor 't' = t' by (clarsimp elim!: oreachable_other') (metis otherI '¬(ξ. U ξ ξ')'

moreover from sor str
have "(σ', s') ∈ oreachable (opnet onp p1) (?[S p1] (?[U p1])" by - (erule(1) oreachable_local, auto)
ultimately show ?thesis by (rule conjI [rotated])

next
fix i d
assume "s' = s"
    and str: "((σ, s'), i:deliver(d), (σ', t')) ∈ trans (opnet onp p2)"

from tor ttr have "∀ j. j ∊ net_tree_ips p2 → σ' j = σ j"
    by (drule(1) ostep_invariantD [OF act2], simp_all)
moreover with 'net_tree_ips p1 ∩ net_tree_ips p2 = {}' have "∀ j. j ∊ net_tree_ips p1 → σ' j = σ j" by auto
moreover from tor ttr have "∀ j ∊ net_tree_ips p2. U (σ j) (σ' j)"
    by (drule(1) ostep_invariantD [OF act2], simp_all)
ultimately have "(σ', t') ∈ oreachable (opnet onp p2) (?[S p2] (?[U p2])" using sor 's' = s' by (clarsimp elim!: oreachable_other') (metis otherI '¬(ξ. U ξ ξ')'

moreover from tor ttr
have "(σ', s') ∈ oreachable (opnet onp p1) (?[S p1] (?[U p1])" by - (erule(1) oreachable_local, auto)
ultimately show ?thesis ..

next
assume "t' = t"
    and str: "((σ, s), τ, (σ', s')) ∈ trans (opnet onp p1)"

from sor str have "∀ j. j ∊ net_tree_ips p1 → σ' j = σ j"
    by (drule(1) ostep_invariantD [OF act1], simp_all)
moreover with 'net_tree_ips p1 ∩ net_tree_ips p2 = {}' have "∀ j. j ∊ net_tree_ips p1 → σ' j = σ j" by auto
moreover from sor str have "∀ j ∊ net_tree_ips p1. U (σ j) (σ' j)"
    by (drule(1) ostep_invariantD [OF act1], simp_all)
ultimately have "(σ', t') ∈ oreachable (opnet onp p2) (?[S p2] (?[U p2])" using tor 't' = t' by (clarsimp elim!: oreachable_other') (metis otherI '¬(ξ. U ξ ξ')'

moreover from sor str
have "(σ', s') ∈ oreachable (opnet onp p1) (?[S p1] (?[U p1])" by - (erule(1) oreachable_local, auto)
ultimately show ?thesis by (rule conjI [rotated])

next
assume "s' = s"
    and ttr: "((σ, t), (σ', t')) ∈ trans (opnet onp p2)"

from tor ttr have "∀ j. j ∊ net_tree_ips p2 → σ' j = σ j"
    by (drule(1) ostep_invariantD [OF act2], simp_all)
moreover with 'net_tree_ips p1 ∩ net_tree_ips p2 = {}' have "∀ j. j ∊ net_tree_ips p2 → σ' j = σ j" by auto
moreover from tor ttr have "∀ j ∊ net_tree_ips p2. U (σ j) (σ' j)"
    by (drule(1) ostep_invariantD [OF act2], simp_all)
ultimately have \( (\sigma', s') \in \text{oreachable (opnet onp } p_1) (\exists s_1) (\forall U \ p_1) \)
using sor 's' = s' by (clarsimp elim!: oreachable_other')
(metis (no_types) oreachable_local' \( 's' = s' \))

moreover from tor ttr
have \( (\sigma', t') \in \text{oreachable (opnet onp } p_2) (\exists s_2) (\forall U \ p_2) \)
by - (erule(1) oreachable_local, auto)
ultimately show \( \text{thesis } .. \)

\( \text{Splitting'} \) reachability is trivial when there are no assumptions on interleavings, but this is useless for showing non-trivial properties, since the interleaving steps can do anything at all. This lemma is too weak.

lemma subnet_oreachable_true_true:
assumes \( (\sigma, \text{SubnetS } s_1 \ s_2) \in \text{oreachable (opnet onp } (p_1 \parallel p_2)) (\lambda_\_ \ _ \ True) (\lambda_\_ \ _ \ True) \)
shows \( (\sigma, s_1) \in \text{oreachable (opnet onp } p_1) (\lambda_\_ \ _ \ True) (\lambda_\_ \ _ \ True) \)
\( (\sigma, s_2) \in \text{oreachable (opnet onp } p_2) (\lambda_\_ \ _ \ True) (\lambda_\_ \ _ \ True) \)
(is \_ \ _ \ _ \ _ \ _ \ _ \ _ \ _)
using assms proof -
from assms have \( (\sigma, s_1) \in \text{oreachable p}_1 \land (\sigma, s_2) \in \text{oreachable p}_2 \)
proof (induction rule: oreachable_par_subnet_induct)
fix \( s, s_1 \ s_2 \)
assume \( (\sigma, \text{SubnetS } s_1 \ s_2) \in \text{init (opnet onp } p_1 \parallel p_2) \)
thus \( (\sigma, s_1) \in \text{oreachable p}_1 \land (\sigma, s_2) \in \text{oreachable p}_2 \)
by (auto dest: oreachable_init)
next
case (local \( \sigma, s_1 \ s_2 \ ' \ s', s_1' \ s_2', a')
hence \( (\sigma, \text{SubnetS } s_1 \ s_2) \in \text{oreachable (p}_1 \parallel p_2) \)
and sr1: \( (\sigma, s_1) \in \text{oreachable p}_1 \)
and sr2: \( (\sigma, s_2) \in \text{oreachable p}_2 \)
and \( (\sigma, \text{SubnetS } s_1 \ s_2), a, (\sigma', \text{SubnetS } s_1' \ s_2') \in \text{trans (opnet onp } (p_1 \parallel p_2)) \)
by auto from this(4)

have \( (\sigma, \text{SubnetS } s_1 \ s_2), a, (\sigma', \text{SubnetS } s_1' \ s_2') \in \text{opnet_sos (trans (opnet onp } p_1)) (\text{trans (opnet onp } p_2)) \)
by simp
thus \( (\sigma', s_1') \in \text{oreachable p}_1 \land (\sigma', s_2') \in \text{oreachable p}_2 \)
proof cases
fix \( R = H K \)
assume \("a = R:*cast(m)"
and tr1: \("(\sigma, s_1), R:*cast(m), (\sigma', s_1') \in \text{trans (opnet onp } p_1)"
and tr2: \("(\sigma, s_2), H:arrive(m), (\sigma', s_2') \in \text{trans (opnet onp } p_2)"
from sr1 and tr1 and TrueI have \("(\sigma', s_1') \in \text{oreachable p}_1 \)
by (rule oreachable_local')
moreover from sr2 and tr2 and TrueI have \("(\sigma', s_2') \in \text{oreachable p}_2 \)
by (rule oreachable_local')
ultimately show \( \text{thesis } .. \)
next
assume \("a = \tau"
and \("s_2' = s_2"
and tr1: \("(\sigma, s_1), \tau, (\sigma', s_1') \in \text{trans (opnet onp } p_1)"
from sr2 and this(2) have \("(\sigma', s_2') \in \text{oreachable p}_2 \)
by auto
moreover have \("(\lambda_\_ \ _ \ True) \sigma 's' \) by (rule TrueI)
ultimately have \("(\sigma', s_2') \in \text{oreachable p}_2 \)
by (rule oreachable_other')
moreover from sr1 and tr1 and TrueI have \("(\sigma', s_1') \in \text{oreachable p}_1 \)
by (rule oreachable_local')
qed (insert sr1 sr2, simp_all, (metis (no_types) oreachable_local'
oreachable_other')+)\)

qed auto
thus \( (\sigma, s_1) \in \text{oreachable p}_1 \)
\( (\sigma, s_2) \in \text{oreachable p}_2 \)
by auto

qed

It may also be tempting to try splitting from the assumption \((\sigma, \text{SubnetS } s_1 \ s_2) \in \text{oreachable (opnet onp } p_1 \parallel p_2) \)
"lift_opnet_sync_action:

lemma "\( \lambda_\xi. U_\xi \xi \)"
and act1: "\( \lambda i. R. \{ i : \text{onp } i : R \} \o \Vdash_A (\lambda_\sigma. \text{oarrivemsg I } \sigma, \text{other } U \{ i \} \rightarrow)\)
globala (\( \lambda (\sigma, a, \sigma'). \text{castmsg } (I \sigma) a \))
and act2: "\( \lambda i. R. \{ i : \text{onp } i : R \} \o \Vdash_A (\lambda_\sigma. \text{oarrivemsg I } \sigma, \text{other } U \{ i \} \rightarrow)\)
globala (\( \lambda (\sigma, a, \sigma'). (a \neq \tau \land (\forall i \in \text{net tree_ips } p. \text{S } (\sigma i) (\sigma' i)))\))
and act3: "\( \lambda i. R. \{ i : \text{onp } i : R \} \o \Vdash_A (\lambda_\sigma. \text{oarrivemsg I } \sigma, \text{other } U \{ i \} \rightarrow)\)
globala (\( \lambda (\sigma, a, \sigma'). (a = \tau \lor (\exists d. a = i: \text{deliver}(d)) \rightarrow U (\sigma i) (\sigma' i))\))
shows "\( \text{opnet onp } p \Vdash_A (\lambda_\sigma. \text{oarrivemsg I } \sigma, \text{other } U \{ \text{net tree_ips } p \} \rightarrow)\)
globala (\( \lambda (\sigma, a, \sigma'). \text{castmsg } (I \sigma) a \))
and (a \neq \tau \land (\forall i \in \text{net tree_ips } p. \text{S } (\sigma i) (\sigma' i)))
\land (a = \tau \lor (\exists i \in \text{net tree_ips } p. U (\sigma i) (\sigma' i)))
\land (\forall i \in \text{net tree_ips } p \rightarrow \sigma')\)
\((\lambda_\xi. U_\xi \xi)\)

(is "\( \text{opnet onp } p \Vdash_A (?I, ?U \rightarrow)\)"
proof (induction p)
fix \( i \in R \)
show "\( \text{opnet onp } \{ i; R \} \o \Vdash_A (?I, ?U \{ i; R \} \rightarrow)\)"
proof (rule ostep_invariantI, simp only: opnet.simps net_tree_ips.simps)
fix \( a \) a \( a' \) s'
assume sor: "\( (\sigma, a) \in \text{oreachable } \{ i : \text{onp } i : R \} \)
\( \lambda_\sigma. \text{oarrivemsg I } \sigma \) (other U \{ i \})" 
and str: "\( (\sigma, a, a', (\sigma', s')) \in \text{trans } \{ i : \text{onp } i : R \} \)"
and oam: "\( \text{oarrivemsg I } \sigma a \)"
hence "\( \text{castmsg } (I \sigma) a \)"
by - (drule(2) ostep_invariantD [OF act1], simp)
moreover from sor str oam have "\( a \neq \tau \land (\forall i \in \text{net tree_ips } p. S (\sigma i) (\sigma' i)) \)"
by - (drule(2) ostep_invariantD [OF act2], simp)
moreover have "\( a = \tau \lor (\exists i \in \text{net tree_ips } p. U (\sigma i) (\sigma' i)) \)"
proof 
from sor str oam have "\( a = \tau \lor (\exists d. a = i: \text{deliver}(d)) \rightarrow U (\sigma i) (\sigma' i) \)"
by - (drule(2) ostep_invariantD [OF node_local_deliver], simp)
ultimately show \( \text{thesis} \)
by clarsimp metis
qed
moreover from sor str oam have "\( \forall j. j \neq i \rightarrow (\forall d. a \neq j: \text{deliver}(d)) \)"
by - (drule(2) ostep_invariantD [OF node_local_deliver], simp)
moreover from sor str oam have "\( a = \tau \lor (\exists i \in \text{net tree_ips } p. (\sigma j \neq i \rightarrow \tau' j = \sigma) ) \)"
by - (drule(2) ostep_invariantD [OF node_tau_deliver_unchanged], simp)
ultimately show "\( \text{inv } \{ i \} ((\sigma, a), (\sigma', s')) \)" by simp
qed

next
fix \( p_1 \) \( p_2 \)
assume inv1: "\( \text{opnet onp } p_1 \o \Vdash_A (?I, ?U \rightarrow) \)"
and inv2: "\( \text{opnet onp } p_2 \o \Vdash_A (?I, ?U \rightarrow) \)"
show "\( \text{opnet onp } (p_1 || p_2) \o \Vdash_A (?I, ?U (p_1 || p_2) \rightarrow) \)"
proof (rule ostep_invariantI)
fix \( s \) st a \( \sigma \) \( \sigma' \) st'
assume "\( (\sigma, \text{st}) \in \text{oreachable } \{ \text{opnet } (p_1 || p_2) \} \)"
and "\( ((\sigma, \text{st}), a, (\sigma', \text{st'})') \in \text{trans } \{ \text{opnet } (p_1 || p_2) \} \)"
and "\( \text{oarrivemsg I } \sigma a \)"
from this(1) obtain s t
where "\( \text{st = SubnetS s t} \)"
and "\( (\sigma, \text{SubnetS s t}) \in \text{oreachable } \{ \text{opnet } (p_1 || p_2) \} \)"
by - (frule net_par_oreachable_is_subnet, metis)
from this(2) and inv1 and inv2
obtain sor: "\( (\sigma, s) \in \text{oreachable } \{ \text{opnet } p_1 \} \)"
and tor: "(s, t) ∈ orreachable (opnet onp p2) \(\exists i (\gamma \cup p2)\)"
and "net_tree_ips p1 \(\cap\) net_tree_ips p2 = {}"
by - (drule opnet_sync_action_subnet_orreachable [OF _ \(\forall \xi. \ U \xi \xi\) ], auto)

determine * and "((s, st), a, (s', st')) \(\in\) trans (opnet onp (p1 || p2))'
and 'st = SubnetS s t'

obtain s' t' where "st' = SubnetS s' t''

ultimately show \(\forall \xi. \ U \xi \xi\)"

by clarsimp (frule opartial_net_preserves_subnets, metis)

determine this(2)

have "castmsg (I a) a
∧ (a ≠ t \(\land\) \(\forall \xi. \ d\) a ≠ i:deliver(d)) \(\rightarrow\) (\(\forall \xi\in\text{net_tree_ips (p1 || p2) . S (\xi i) (\xi' i)}\))
∧ (a = t \(\lor\) \(\exists i\ d. \ a = i:deliver(d)) \(\rightarrow\) (\(\forall \xi\in\text{net_tree_ips (p1 || p2) . U (\xi i) (\xi' i)}\))
∧ (\(\forall i. i \notin\text{net_tree_ips (p1 || p2) \rightarrow \xi' i = \xi i}\))"

proof cases

fix \(\xi \in H K\)

assume "a = R:*cast(m)"

and str: "((s, s), R:*cast(m), (s', s')) \(\in\) trans (opnet onp p1)"

and ttr: "((s, t), H~K:arrive(m), (s', t')) \(\in\) trans (opnet onp p2)"

from sor and str have "I \(\forall \xi\in\text{net_tree_ips p1 . S (\xi i) (\xi' i)}\)
by (auto dest: ostep_invariantD [OF inv1])

moreover with tor and ttr have "\(\forall i\in\text{net_tree_ips p2 . S (\xi i) (\xi' i)}\)
by (auto dest: ostep_invariantD [OF inv2])

ultimately show ?thesis

using 'a = R:*cast(m)' by auto

next

fix \(\xi \in H K\)

assume "a = R:*cast(m)"

and str: "((s, s), H~H':arrive(m), (s', s')) \(\in\) trans (opnet onp p1)"

and ttr: "((s, t), R:*cast(m), (s', t')) \(\in\) trans (opnet onp p2)"

from tor and ttr have "I \(\forall i\in\text{net_tree_ips p2 . S (\xi i) (\xi' i)}\)
by (auto dest: ostep_invariantD [OF inv2])

moreover with sor and str have "\(\forall i\in\text{net_tree_ips p1 . S (\xi i) (\xi' i)}\)
by (auto dest: ostep_invariantD [OF inv1])

ultimately show ?thesis

using 'a = R:*cast(m)' by auto

next

fix \(\xi \in H K\)

assume "a = (H \(\cup\) H') ~ (K \(\cup\) K'):arrive(m)"

and str: "((s, s), H~H':arrive(m), (s', s')) \(\in\) trans (opnet onp p1)"

and ttr: "((s, t), H~K':arrive(m), (s', t')) \(\in\) trans (opnet onp p2)"

from this(1) and 'arrivemsg I \(\forall \xi\ a\)' have "I \(\forall \xi\ m\)" by simp

with sor and str have "\(\forall i\in\text{net_tree_ips p1 . S (\xi i) (\xi' i)}\)
by (auto dest: ostep_invariantD [OF inv1])

moreover from tor and ttr and 'I \(\forall i\ m\) have "\(\forall i\in\text{net_tree_ips p2 . S (\xi i) (\xi' i)}\)
by (auto dest: ostep_invariantD [OF inv2])

ultimately show ?thesis

using 'a = (H \(\cup\) H') ~ (K \(\cup\) K'):arrive(m)' by auto

next

fix i d

assume "a = i:deliver(d)"

with sor have "((\(\forall i\in\text{net_tree_ips p1 . U (\xi i) (\xi' i)}\))
∧ (\(\forall i. i \notin\text{net_tree_ips p1 \rightarrow \xi' i = \xi i}\))"
by (auto dest!: ostep_invariantD [OF inv1])

with 'a = i:deliver(d)' and '\(\forall \xi. \ U \xi \xi\) show ?thesis
by auto

next

fix i d

assume "a = i:deliver(d)"

with ttor have "((\(\forall i\in\text{net_tree_ips p2 . U (\xi i) (\xi' i)}\))
∧ (\(\forall i. i \notin\text{net_tree_ips p2 \rightarrow \xi' i = \xi i}\))"
by (auto dest!: ostep_invariantD [OF inv2])
with 'a = i:deliver(d)' and '\land \xi. U \xi \xi' show \?thesis
by auto

next
assume "a = \tau"
and str: "((\sigma, s), \tau, (\sigma', s')) \in trans (opnet onp p_1)"
with sor have "((\forall i \in \text{net_tree_ips} p_1. U (\sigma i) (\sigma' i))
\wedge (\forall i. i \in \text{net_tree_ips} p_1 \rightarrow \sigma' i = \sigma i))"
by (auto dest!: ostep_invariantD [OF inv1])
with 'a = \tau' and '\land \xi. U \xi \xi' show \?thesis
by auto

next
fix i i'
assume "a = connect(i, i')"
and str: "((\sigma, s), connect(i, i'), (\sigma', s')) \in trans (opnet onp p_1)"
and ttr: "((\sigma, t), connect(i, i'), (\sigma', t')) \in trans (opnet onp p_2)"
from sor and str have "\forall i \in \text{net_tree_ips} p_1. S (\sigma i) (\sigma' i)"
by (auto dest: ostep_invariantD [OF inv1])
moreover from tor and ttr have "\forall i \in \text{net_tree_ips} p_2. S (\sigma i) (\sigma' i)"
by (auto dest: ostep_invariantD [OF inv2])
ultimately show \?thesis
using 'a = connect(i, i')' by auto

next
fix i i'
assume "a = disconnect(i, i')"
and str: "((\sigma, s), disconnect(i, i'), (\sigma', s')) \in trans (opnet onp p_1)"
and ttr: "((\sigma, t), disconnect(i, i'), (\sigma', t')) \in trans (opnet onp p_2)"
from sor and str have "\forall i \in \text{net_tree_ips} p_1. S (\sigma i) (\sigma' i)"
by (auto dest: ostep_invariantD [OF inv1])
moreover from tor and ttr have "\forall i \in \text{net_tree_ips} p_2. S (\sigma i) (\sigma' i)"
by (auto dest: ostep_invariantD [OF inv2])
ultimately show \?thesis
using 'a = disconnect(i, i')' by auto

qed

thus "\?inv (\text{net_tree_ips} (p_1 || p_2)) ((\sigma, st), a, (\sigma', st'))" by simp

qed

theorem subnet_oreachable:
assumes "(\sigma, \text{SubnetS s t}) \in \text{oreachable} (opnet onp (p_1 || p_2))"
(\text{otherwith S (\text{net_tree_ips} (p_1 || p_2)) (oarrivemsg I)})
(\text{other U (\text{net_tree_ips} (p_1 || p_2))})"
(is "\_ \in \text{oreachable} _ (\?S (p_1 || p_2)) (?U (p_1 || p_2))")

and "\land \xi. S \xi \xi"

and "\land \xi. U \xi \xi"

and node1: "\land i R. (i : onp i : R)_0 \models _A (\lambda \sigma __. oarrivemsg I \sigma, other U \{i\} \rightarrow)
globala (\lambda (\sigma, a, \_). castmsg (I \sigma) a)"

and node2: "\land i R. (i : onp i : R)_0 \models _A (\lambda \sigma __. oarrivemsg I \sigma, other U \{i\} \rightarrow)
globala (\lambda (\sigma, a, \sigma'). (a \neq \tau \wedge (\forall i d. a \neq i:deliver(d)) \rightarrow S (\sigma i) (\sigma' i)))"

and node3: "\land i R. (i : onp i : R)_0 \models _A (\lambda \sigma __. oarrivemsg I \sigma, other U \{i\} \rightarrow)
globala (\lambda (\sigma, a, \sigma'). (a = \tau \vee (\exists d. a = i:deliver(d)) \rightarrow U (\sigma i) (\sigma' i)))"

shows "(\sigma, s) \in \text{oreachable} (opnet onp p_1)
(\text{otherwith S (\text{net_tree_ips} p_1)} (oarrivemsg I))"
\( \text{assms(1)} \) proof (induction rule: oreachable_par_subnet_induct)

case (init \( \sigma \ s \ t \) )

hence \( "(\sigma', s) \in \text{init} \text{(opnet onp p)}" \)

and \( tinit: "(\sigma', t) \in \text{init} \text{(opnet onp p)}" \)

and \( \text{net_tree_ips p} \cap \text{net_tree_ips p} = \{\} \) by auto

moreover from \( \text{sinit: "(\sigma, s) \in \text{oreachable (opnet onp p)}" \) \)

by (rule opnet_net_ips_net_tree_ips_init)

moreover from \( \text{tinit have "net_ips t = net_tree_ips p"} \) \)

by (rule opnet_net_ips_net_tree_ips_init)

ultimately show \( ?\text{case by (auto elim: oreachable_init)} \)

next 

\( \text{case (other } \sigma \ s \ t \sigma' \) \)

hence \( "\text{other } \text{U (net_tree_ips p1 || p2)} \) \( \sigma \ \sigma'' \)" \)

and \( \text{IHs: "}(\sigma, s) \in \text{oreachable (opnet onp p1)} \ (\text{?S p1)} (\text{?U p1)}" \)

and \( \text{IHT: "}(\sigma, t) \in \text{oreachable (opnet onp p2)} \ (\text{?S p2)} (\text{?U p2)}" \)

and \( \text{net_tree_ips p1} \cap \text{net_tree_ips p2} = \{\} \) by auto

have \( "(\sigma', s) \in \text{oreachable (opnet onp p1)} \ (\text{?S p1)} (\text{?U p1)}" \)

proof - 

from \( "(\text{?U (p1 || p2)} \) \( \sigma \ \sigma'' \)" \) and \( "\text{'A} \xi. U \xi \xi' \) have \( "(\sigma_2 \sigma \sigma')" \)

by (rule other_net_tree_ips_par_left)

with \( \text{IHs show } \text{thesis by (erule(1) oreachable_other')} \)

qed

moreover have \( "(\sigma', t) \in \text{oreachable (opnet onp p2)} \ (\text{?S p2)} (\text{?U p2)}" \)

proof - 

from \( "(\text{?U (p1 || p2)} \) \( \sigma \ \sigma'' \)" \) and \( "\text{'A} \xi. U \xi \xi' \) have \( "(\sigma_2 \sigma \sigma')" \)

by (rule other_net_tree_ips_par_right)

with \( \text{IHT show } \text{thesis by (erule(1) oreachable_other')} \)

qed

ultimately show \( ?\text{case using 'net_tree_ips p1} \cap \text{net_tree_ips p2} = \{\}' \) by simp

next 

\( \text{case (local } \sigma \ s \ t \sigma' \ s' \ t' \ a \) \)

hence \( "\text{stor: "}(\sigma, \text{SubnetS s} \ s) \in \text{oreachable (opnet onp p1 || p2)} (\text{?S (p1 || p2)} (\text{?U (p1 || p2)})" \)

and \( \text{tr: "}(\sigma, \text{SubnetS s} \ t, a, (\sigma', \text{SubnetS s'} t')) \in \text{trans (opnet onp (p1 || p2)})" \)

and \( "\text{S (p1 || p2)} \) \( \sigma \ \sigma' \ a \)" \)

and \( \text{sor: "}(\sigma, s) \in \text{oreachable (opnet onp p1)} \ (\text{?S p1)} (\text{?U p1)}" \)

and \( \text{tor: "}(\sigma, t) \in \text{oreachable (opnet onp p2)} \ (\text{?S p2)} (\text{?U p2)}" \)

and \( "\text{net_tree_ips p1} \cap \text{net_tree_ips p2} = \{\} \) by auto

have act: "\( \forall p. \text{opnet onp p} \models (\lambda (\sigma \ _,. \ \text{oarrivemsg I} \ \sigma, \text{other U (net_tree_ips p)} \rightarrow) \) \)

\( \text{globala} (\lambda (\sigma, a, \sigma'). \text{castmsg (I} \ \sigma) \ a \)

\( \land (a \neq \tau \land (\forall i \ d. \ a \neq i: \text{deliver(d)}) \rightarrow \) \)

\( (\forall i \in \text{net_tree_ips p}. S (\sigma i) (\sigma' i))) \)

\( \land (a = \tau \lor (\exists i \ d. \ a = i: \text{deliver(d)}) \rightarrow \) \)

\( ((\forall i \in \text{net_tree_ips p}. U (\sigma i) (\sigma' i))) \)

\( \land (\forall i. i \in \text{net_tree_ips p} \rightarrow \sigma' i = \sigma i)))) \)"

by (rule lift_opnet_sync_action [OF \( \text{assms(3-6)})])

from \( "(\text{?S (p1 || p2) \sigma \sigma' a'} \) have "(\forall j. j \in \text{net_tree_ips (p1 || p2)} \rightarrow S (\sigma j) (\sigma' j))" \)

and "oarrivemsg I \( \sigma \ a" \)

by (auto elim!: otherwithE)

from \( \text{tr have "}(\sigma, \text{SubnetS s} \ s, t) \in \text{opnet sos (trans (opnet onp p1)) (trans (opnet onp p2))" by simp \) 

hence "(\sigma', s') \in \text{oreachable (opnet onp p1)} (\text{?S p1)} (\text{?U p1)} 

\( \land (\sigma', t') \in \text{oreachable (opnet onp p2)} (\text{?S p2)} (\text{?U p2)}" \)

proof (cases)

fix \( H \ K \equiv H' \ K' \)
assume \( a = (H \cup H') \cap (K \cup K') : \text{arrive}(m) \)

and \( \text{str} : "((\sigma, s), H \cap K: \text{arrive}(m), (\sigma', s')) \in \text{trans}(\text{opnet onp } p_1)" \)

and \( \text{ttr} : "((\sigma, t), H' \cap K': \text{arrive}(m), (\sigma', t')) \in \text{trans}(\text{opnet onp } p_2)" \)

from this(1) and \( ?S \ (p_1 \parallel p_2) \sigma \sigma' \ a' \) have "I \( \sigma \ m \)" by auto

with \( \text{sor str} \) have "\( \forall i \in \text{net_tree_ips } p_1. \ S (\sigma i) (\sigma i)" 

by - (drule(1) ostep_arrive_invariantD \[OF act\], simp_all)

moreover from \( 'I \sigma m \) tor ttr have "\( \forall i \in \text{net_tree_ips } p_2. \ S (\sigma i) (\sigma i)" 

by - (drule(1) ostep_arrive_invariantD \[OF act\], simp_all)

ultimately have "\( \forall i. S (\sigma i) (\sigma i)" 

using "\( \forall j. j \notin \text{net_tree_ips } (p_1 \parallel p_2) \rightarrow S (\sigma j) (\sigma j)" by auto

with \( 'I \sigma m' \) sor str

have "\( \sigma' , s' \) \in \text{oreachable}(\text{opnet onp } p_1) \ (\sigma S p_1) \ (\sigma U p_1)" 

by - (erule(1) oreachable, auto)

moreover from "\( \forall i. S (\sigma i) (\sigma i)" 'I \sigma m' tor ttr

have "\( \sigma', t' \) \in \text{oreachable}(\text{opnet onp } p_2) \ (\sigma S p_2) \ (\sigma U p_2)" 

by - (erule(1) oreachable, auto)

ultimately show ?thesis ..

next

fix \( R \equiv H K \)

assume str: "\( ((\sigma, s), R: \ast \text{cast}(m), (\sigma', s')) \in \text{trans}(\text{opnet onp } p_1)" 

and ttr: "\( ((\sigma, t), R: \ast \text{cast}(m), (\sigma', t')) \in \text{trans}(\text{opnet onp } p_2)" 

from sor str have "'I \sigma m" 

by - (drule(1) ostep_arrive_invariantD \[OF act\], simp_all)

with sor str tor ttr have "\( \forall i. S (\sigma i) (\sigma i)" 

using "\( \forall j. j \notin \text{net_tree_ips} \ (p_1 \parallel p_2) \rightarrow S (\sigma j) (\sigma j)" 

by (fastforce dest!: ostep_arrive_invariantD \[OF act\] ostep_arrive_invariantD \[OF act\])

with 'I \sigma m' sor str

have "\( \sigma' , s' \) \in \text{oreachable}(\text{opnet onp } p_1) \ (\sigma S p_1) \ (\sigma U p_1)" 

by - (erule(1) oreachable, auto)

moreover from '\( \forall i. S (\sigma i) (\sigma i)" 'I \sigma m' tor ttr

have "\( \sigma', t' \) \in \text{oreachable}(\text{opnet onp } p_2) \ (\sigma S p_2) \ (\sigma U p_2)" 

by - (erule(1) oreachable, auto)

ultimately show ?thesis ..

next

fix \( R \equiv H K \)

assume str: "\( ((\sigma, s), H \cap K: \text{arrive}(m), (\sigma', s')) \in \text{trans}(\text{opnet onp } p_1)" 

and ttr: "\( ((\sigma, t), R: \ast \text{cast}(m), (\sigma', t')) \in \text{trans}(\text{opnet onp } p_2)" 

from tor ttr have "'I \sigma m" 

by - (drule(1) ostep_arrive_invariantD \[OF act\], simp_all)

with sor str tor ttr have "\( \forall i. S (\sigma i) (\sigma i)" 

using "\( \forall j. j \notin \text{net_tree_ips} \ (p_1 \parallel p_2) \rightarrow S (\sigma j) (\sigma j)" 

by (fastforce dest!: ostep_arrive_invariantD \[OF act\] ostep_arrive_invariantD \[OF act\])

with 'I \sigma m' sor str

have "\( \sigma' , s' \) \in \text{oreachable}(\text{opnet onp } p_1) \ (\sigma S p_1) \ (\sigma U p_1)" 

by - (erule(1) oreachable, auto)

moreover from '\( \forall i. S (\sigma i) (\sigma i)" 'I \sigma m' tor ttr

have "\( \sigma', t' \) \in \text{oreachable}(\text{opnet onp } p_2) \ (\sigma S p_2) \ (\sigma U p_2)" 

by - (erule(1) oreachable, auto)

ultimately show ?thesis ..

next

fix \( i \equiv i' \)

assume str: "\( ((\sigma, s), \text{connect}(i, i'), (\sigma', s')) \in \text{trans}(\text{opnet onp } p_1)" 

and ttr: "\( ((\sigma, t), \text{connect}(i, i'), (\sigma', t')) \in \text{trans}(\text{opnet onp } p_2)" 

with sor tor have "\( \forall i. S (\sigma i) (\sigma i)" 

using "\( \forall j. j \notin \text{net_tree_ips} \ (p_1 \parallel p_2) \rightarrow S (\sigma j) (\sigma j)" 

by (fastforce dest!: ostep_arrive_invariantD \[OF act\] ostep_arrive_invariantD \[OF act\])

with sor str

have "\( \sigma' , s' \) \in \text{oreachable}(\text{opnet onp } p_1) \ (\sigma S p_1) \ (\sigma U p_1)" 

by - (erule(1) oreachable, auto)

moreover from "\( \forall i. S (\sigma i) (\sigma i)" 'I \sigma m' tor ttr

have "\( \sigma', t' \) \in \text{oreachable}(\text{opnet onp } p_2) \ (\sigma S p_2) \ (\sigma U p_2)" 

by - (erule(1) oreachable, auto)
ultimately show ?thesis ..

next
fix i i'
assume str: "((σ, s), disconnect(i, i'), (σ', s')) ∈ trans (opnet onp p_1)"
and ttr: "((σ, t), disconnect(i, i'), (σ', t')) ∈ trans (opnet onp p_2)"
with sor tor have "∀ i. S (σ i) (σ' i)"
using "∀ j. j ∉ net_tree_ips p_1 ∥ p_2 → S (σ j) (σ' j)"
by (fastforce dest!: ostep_arrive_invariantD [OF act] ostep_arrive_invariantD [OF act])
with sor str
have "(σ', s') ∈ ororeachable (opnet onp p_1) (S p_1) (?U p_1)"
by - (erule(1) ororeachable_local, auto)
moreover from "∀ i. S (σ i) (σ' i)" tor ttr
have "(σ', t') ∈ ororeachable (opnet onp p_2) (S p_2) (?U p_2)"
by - (erule(1) ororeachable_local, auto)
ultimately show ?thesis ..

next
fix i d
assume "t' = t"
and str: "((σ, s), i:deliver(d), (σ', s')) ∈ trans (opnet onp p_1)"
from sor str have "∀ j. j ∉ net_tree_ips p_1 → σ' j = σ j"
by - (drule(1) ostep_arrive_invariantD [OF act], simp_all)
hence "∀ j. j ∉ net_tree_ips p_1 → S (σ j) (σ' j)"
by (auto intro: "∀ j. j ∉ net_tree_ips p_1 → S (σ j) (σ' j)"
with sor str
have "(σ', s') ∈ ororeachable (opnet onp p_1) (S p_1) (?U p_1)"
by - (erule(1) ororeachable_local, auto)
moreover have "(σ', t') ∈ ororeachable (opnet onp p_2) (S p_2) (?U p_2)"
proof -
from "∀ j. j ∉ net_tree_ips p_1 → σ' j = σ j" and "net_tree_ips p_1 ∩ net_tree_ips p_2 = {}"
have "∀ j. j ∉ net_tree_ips p_2 → σ' j = σ j" by auto
moreover from sor str have "∀ j ∉ net_tree_ips p_1. U (σ j) (σ' j)"
by - (drule(1) ostep_arrive_invariantD [OF act], simp_all)
ultimately show ?thesis
using tor "t' = t" "∀ j. j ∉ net_tree_ips p_1 → σ' j = σ j"
by (clarsimp elim!: ororeachable_other)
(auto intro: "∀ j. j ∉ net_tree_ips p_1 → S (σ j) (σ' j)"
with tor ttr
have "(σ', t') ∈ ororeachable (opnet onp p_2) (S p_2) (?U p_2)"
by - (erule(1) ororeachable_local, auto)
moreover have "(σ', s') ∈ ororeachable (opnet onp p_1) (S p_1) (?U p_1)"
proof -
from "∀ j. j ∉ net_tree_ips p_2 → σ' j = σ j" and "net_tree_ips p_1 ∩ net_tree_ips p_2 = {}"
have "∀ j. j ∉ net_tree_ips p_1 → σ' j = σ j" by auto
moreover from tor ttr have "∀ j ∉ net_tree_ips p_2. U (σ j) (σ' j)"
by - (drule(1) ostep_arrive_invariantD [OF act], simp_all)
ultimately show ?thesis
using sor "s' = s" "∀ j. j ∉ net_tree_ips p_2 → σ' j = σ j"
by (clarsimp elim!: ororeachable_other)
(auto intro: "∀ j. j ∉ net_tree_ips p_2 → S (σ j) (σ' j)"
qed
ultimately show ?thesis by - (rule conjI)
next
assume "s' = s"
   and ttr: "((σ, t), τ, (σ', t')) ∈ trans (opnet onp p2)"
from tor ttr have "∀ j. j ∈ net_tree_ips p2 → σ' j = σ j"
   by (drule(1) ostep_arrive_invariantD [OF act], simp_all)

hence "∀ j. j ∈ net_tree_ips p2 → S (σ j) (σ' j)"
   by (auto intro: ') ∧ S ξ ξ')

with tor ttr
have "(σ', t') ∈ oreachable (opnet onp p2) (S p2) (U p2)"
   by - (erule(1) oreachable_local, auto)

moreover have "(σ', s') ∈ oreachable (opnet onp p1) (S p1) (U p1)"

proof -
   from "∀ j. j ∈ net_tree_ips p2 → σ' j = σ j" and 'net_tree_ips p1 ∩ net_tree_ips p2 = {}'
   have "∀ j. j ∈ net_tree_ips p1 → σ' j = σ j" by auto
moreover from tor ttr have "∀ j ∈ net_tree_ips p2. U (σ j) (σ' j)"
   by (drule(1) ostep_arrive_invariantD [OF act], simp_all)
ultimately show ?thesis
   using sor 's' = s' "∀ j. j ∉ net_tree_ips p2 → σ' j = σ j"
   by (clarsimp elim!: oreachable_other')
   (metis otherI ') ∧ S ξ ξ')
qdocs
   qed ultimately show ?thesis by - (rule conjI)
next

assume "t' = t"
   and str: "((σ, s), τ, (σ', s')) ∈ trans (opnet onp p1)"
from tor str have "∀ j. j ∈ net_tree_ips p1 → σ' j = σ j"
   by (drule(1) ostep_arrive_invariantD [OF act], simp_all)

hence "∀ j. j ∈ net_tree_ips p1 → S (σ j) (σ' j)"
   by (auto intro: ') ∧ S ξ ξ')

with tor str
have "(σ', s') ∈ oreachable (opnet onp p1) (S p1) (U p1)"
   by - (erule(1) oreachable_local, auto)

moreover have "(σ', t') ∈ oreachable (opnet onp p2) (S p2) (U p2)"

proof -
   from "∀ j. j ∈ net_tree_ips p1 → σ' j = σ j" and 'net_tree_ips p1 ∩ net_tree_ips p2 = {}'
   have "∀ j. j ∈ net_tree_ips p2 → σ' j = σ j" by auto
moreover from tor str have "∀ j ∈ net_tree_ips p1. U (σ j) (σ' j)"
   by (drule(1) ostep_arrive_invariantD [OF act], simp_all)
ultimately show ?thesis
   using tor 't' = t' "∀ j. j ∉ net_tree_ips p2 → σ' j = σ j"
   by (clarsimp elim!: oreachable_other')
   (metis otherI ') ∧ S ξ ξ')
qdocs
   qed ultimately show ?thesis ..
qdocs
   qed with 'net_tree_ips p1 ∩ net_tree_ips p2 = {}' show ?case by simp
qdocs

lemmas subnet_oreachable1 [dest] = subnet_oreachable [THEN conjunct1, rotated 1]
lemmas subnet_oreachable2 [dest] = subnet_oreachable [THEN conjunct2, THEN conjunct1, rotated 1]
lemmas subnet_oreachable_disjoint [dest] = subnet_oreachable
   [THEN conjunct2, THEN conjunct2, rotated 1]
corollary pnet_lift:
assumes "∀ i : onp ii : R_i".
   |= (otherwith S {ii} (oarrivemsg I), other U {ii} →) global (P ii)

and "∧ ξ. S ξ ξ"
and "∧ ξ. U ξ ξ"

and node1: "∀ i. (i : onp i : R_i) = (λ σ _, oarrivemsg I σ, other U {i} →)
globala (λ(σ, a, _). castmsg (I σ) a)"
and node2: "∀ i. (i : onp i : R_i) = (λ σ _, oarrivemsg I σ, other U {i} →)"
18 Lifting rules for (open) closed networks

theory OClosed_Lifting
imports OPnet_Lifting
begin

lemma trans_fst_oclosed_fst1 [dest]:
"(s, connect(i, i'), s') ∈ ocnet_sos (trans p) ⟹ (s, connect(i, i'), s') ∈ trans p"
  by (metis PairE oconnect_completeTE)

lemma trans_fst_oclosed_fst2 [dest]:
"(s, disconnect(i, i'), s') ∈ ocnet_sos (trans p) ⟹ (s, disconnect(i, i'), s') ∈ trans p"
  by (metis PairE odisconnect_completeTE)

lemma trans_fst_oclosed_fst3 [dest]:
"(s, i: deliver(d), s') ∈ ocnet_sos (trans p) ⟹ (s, i: deliver(d), s') ∈ trans p"
  by (metis PairE odeliver_completeTE)

lemma oclosed_oreachable_inclosed:
  assumes "("σ, ζ") ∈ oreachable (oclosed (opnet np p)) (λ_: _ True) U"
  shows "("σ, ζ") ∈ oreachable (opnet np p) (otherwith (op=) (net_tree_ips p) inclosed) U"
  (is ") ∈ oreachable _ ?owS")"
  using assms proof (induction rule: oreachable_pair_induct)
    fix σ ζ
    assume "("σ, ζ") ∈ init (oclosed (opnet np p))"
hence \( (\sigma, \zeta) \in \text{init} \ (\text{opnet np } p) \) by simp
thus \( (\sigma, \zeta) \in \text{oreachable} \ (\text{opnet np } p) ?os \ U \) ..

next
fix \( \sigma \preceq \sigma' \)
assume \( (\sigma, \zeta) \in \text{oreachable} \ (\text{opnet np } p) ?os \ U \)
and \( U \sigma \sigma' \)
thus \( (\sigma', \zeta) \in \text{oreachable} \ (\text{opnet np } p) ?os \ U \)
by - (rule \text{oreachable}\_other')

next
fix \( \sigma \preceq \sigma' \preceq \zeta' a \)
assume zor: \( (\sigma, \zeta) \in \text{oreachable} \ (\text{opnet np } p) ?os \ U \)
and ztr: \( ((\sigma, \zeta), a, (\sigma', \zeta')) \in \text{trans} \ (\text{oclosed} \ (\text{opnet np } p)) \)
from this(1) have [simp]: \( \text{net}\_\text{ips} \zeta = \text{net}\_\text{tree}\_\text{ips} \ p \)
by (rule \text{opnet}\_\text{net}\_\text{ips}\_\text{net}\_\text{tree}\_\text{ips})
from ztr have \( \text{trans} \ (\text{oclosed} \ (\text{opnet np } p)) \) by simp
thus \( (\sigma', \zeta') \in \text{oreachable} \ (\text{opnet np } p) ?os \ U \)
proof cases
fix \( i K d di \)
assume \( a = i: \text{newpkt} \ (d, di) \)
and tr: \( ((\sigma, \zeta), \{i\} \neg K: \text{arrive} \ (\text{msg}\_\text{class}\_. \text{newpkt} \ (d, di)), (\sigma', \zeta')) \in \text{trans} \ (\text{opnet np } p) \)
and \( \forall \ j. j \not\in \text{net}\_\text{ips} \zeta \rightarrow \sigma' j = \sigma j \)
from this(3) have \( \forall \ j. j \not\in \text{net}\_\text{tree}\_\text{ips} \ p \rightarrow \sigma' j = \sigma j \)
using \( \text{net}\_\text{ips} \zeta = \text{net}\_\text{tree}\_\text{ips} \ p \) by auto
hence \( \text{otherwith} \ (\text{op=} \ (\text{net}\_\text{tree}\_\text{ips} \ p) \ \text{inoclosed} \ \sigma \sigma' \ (\{i\} \neg K: \text{arrive} \ (\text{msg}\_\text{class}\_. \text{newpkt} \ (d, di))) \)
by auto
with zor tr show \( \text{?thesis} \)
by - (rule \text{oreachable}\_local')
next
assume \( a = \tau \)
and tr: \( ((\sigma, \zeta), \tau, (\sigma', \zeta')) \in \text{trans} \ (\text{opnet np } p) \)
and \( \forall \ j. j \not\in \text{net}\_\text{ips} \zeta \rightarrow \sigma' j = \sigma j \)
from this(3) have \( \forall \ j. j \not\in \text{net}\_\text{tree}\_\text{ips} \ p \rightarrow \sigma' j = \sigma j \)
using \( \text{net}\_\text{ips} \zeta = \text{net}\_\text{tree}\_\text{ips} \ p \) by auto
hence \( \text{otherwith} \ (\text{op=} \ (\text{net}\_\text{tree}\_\text{ips} \ p) \ \text{inoclosed} \ \sigma \sigma' \ \tau \)
by auto
with zor tr show \( \text{?thesis} \)
by - (rule \text{oreachable}\_local')
qed (insert \( \text{\text{net}\_\text{ips} \zeta = \text{net}\_\text{tree}\_\text{ips} \ p}' \),
auto \elim! \: \text{oreachable}\_local' \ [DF zor])

qed

lemma \text{oclosed}\_\text{oreachable}\_\text{oreachable} \ [elim]:
assumes \( (\sigma, \zeta) \in \text{oreachable} \ (\text{oclosed} \ (\text{opnet onp } p)) \ (\lambda\_\_\_. \ True) \ U \)
shows \( (\sigma, \zeta) \in \text{oreachable} \ (\text{opnet onp } p) \ (\lambda\_\_\_. \ True) \ U \)
using assms by (rule \text{oclosed}\_\text{oreachable}\_\text{inclosed} \ [THEN \text{oreachable}\_\text{weakenE}]) \simp

lemma \text{inclosed}\_\text{closed} \ [intro]:
assumes \( \text{cinv}: \ (\text{opnet np } p \models (\text{otherwith} \ (\text{op=} \ (\text{net}\_\text{tree}\_\text{ips} \ p) \ \text{inoclosed}, U \rightarrow) \ P) \)
shows \( \text{\text{oclosed} \ (\text{opnet np } p) \models (\lambda\_\_\_. \ True, U \rightarrow) \ P} \)
using assms unfolding \text{oinvariant}\_\text{def}
by (clarsimp dest!: \text{oclosed}\_\text{oreachable}\_\text{inclosed})

end

19 Generic invariants on sequential AWN processes

theory \text{AWN}\_\text{Invariants}
imports \text{Invariants} \text{AWN}\_\text{SOS} \text{AWN}\_\text{Labels}
begin

19.1 Invariants via labelled control terms

Used to state that the initial control-state of an automaton appears within a process specification \( \Gamma \), meaning that its transitions, and those of its subterms, are subsumed by those of \( \Gamma \).
**definition**

control_within :: "(‘s, ‘m, ‘p, ‘l) seqp_env ⇒ (‘z × (‘s, ‘m, ‘p, ‘l) seqp) set ⇒ bool" where
"control_within Γ σ = ∨ (ξ, p)∈σ. ∃ pn. p ∈ subterms (Γ pn)"

**lemma** control_withinI [intro]:

assumes "∀ p ∈ Range σ ⇒ ∃ pn. p ∈ subterms (Γ pn)"

shows "control_within Γ σ"

using assms unfolding control_within_def by auto

**lemma** control_withinD [dest]:

assumes "control_within Γ σ" and "(ξ, p) ∈ σ"

shows "∃ pn. p ∈ subterms (Γ pn)"

using assms unfolding control_within_def by blast

**lemma** control_within_topI [intro]:

assumes "∀ p ∈ Range σ =⇒ ∃ pn. p = Γ pn"

shows "control_within Γ σ"

using assms unfolding control_within_def by clarsimp (metis Range.RangeI subterms_refl)

**lemma** seqp_sos_subterms:

assumes "wellformed Γ" and "∃ pn. p ∈ subterms (Γ pn)" and "((ξ, p), a, (ξ′, p′)) ∈ seqp_sos Γ"

shows "∃ pn. p′ ∈ subterms (Γ pn)"

using assms proof (induct p)

fix p1 p2

assume IH1: "∃ pn. p1 ∈ subterms (Γ pn) =⇒ ((ξ, p1), a, (ξ′, p′)) ∈ seqp_sos Γ =⇒ ∃ pn. p′ ∈ subterms (Γ pn)"

and IH2: "∃ pn. p2 ∈ subterms (Γ pn) =⇒ ((ξ, p2), a, (ξ′, p′)) ∈ seqp_sos Γ =⇒ ∃ pn. p′ ∈ subterms (Γ pn)"

and "∃ pn. p1 ⊕ p2 ∈ subterms (Γ pn)"

and "((ξ, p1 ⊕ p2), a, (ξ′, p′)) ∈ seqp_sos Γ"

from "∃ pn. p1 ⊕ p2 ∈ subterms (Γ pn)" obtain pn where "p1 ∈ subterms (Γ pn)"

and "p2 ∈ subterms (Γ pn)" by auto

from "((ξ, p1 ⊕ p2), a, (ξ′, p′)) ∈ seqp_sos Γ" have "((ξ, p1), a, (ξ′, p′)) ∈ seqp_sos Γ" ∨ "((ξ, p2), a, (ξ′, p′)) ∈ seqp_sos Γ" by auto

thus "∃ pn. p′ ∈ subterms (Γ pn)"

proof

assume "((ξ, p1), a, (ξ′, p′)) ∈ seqp_sos Γ"

with 'p1 ∈ subterms (Γ pn)' show ?thesis by (auto intro: IH1)

next

assume "((ξ, p2), a, (ξ′, p′)) ∈ seqp_sos Γ"

with 'p2 ∈ subterms (Γ pn)' show ?thesis by (auto intro: IH2)

qed

qed auto

**lemma** reachable_subterms:

assumes "wellformed Γ" and "control_within Γ (init A)" and "trans A = seqp_sos Γ" and "(ξ, p) ∈ reachable A Γ"

shows "∃ pn. p ∈ subterms (Γ pn)"

using assms(4)

proof (induct rule: reachable_pair_induct)

fix ξ p

assume "(ξ, p) ∈ init A"
with ‘control_within Γ (init A)’ show "∃ pn. p ∈ subterms (Γ pn)"

next

fix ξ p a ξ’ p'
assume "((ξ, p), a, (ξ’, p’)) ∈ trans A"
and "∃ pn. p ∈ subterms (Γ pn)"
and "I a"

moreover from this(3) and assms(3) have "((ξ, p), a, (ξ’, p’)) ∈ seqp sos Γ" by simp
ultimately show "∃ pn. p’ ∈ subterms (Γ pn)" using 'wellformed Γ'
by (auto elim: seqp sos_subterms)

qed

definition

onl :: "('s, 'm, 'p, 'l) seqp_env ⇒ ('z × 'l ⇒ bool) ⇒ ('z × ('s, 'm, 'p, 'l) seqp ⇒ bool)"

where

"onl Γ P ≡ (λ (ξ, p). ∀ l ∈ labels Γ p. P (ξ, l))"

lemma onlI [intro]:
assumes "∀ l ∈ labels Γ p. P (ξ, l)
shows "onl Γ P (ξ, p)"
using assms unfolding onl_def by simp

lemmas onlI' [intro] = onlI [simplified atomize_ball]

lemma onlD [dest]:
assumes "onl Γ P (ξ, p)
shows "∀ l ∈ labels Γ p. P (ξ, l)
using assms unfolding onl_def by simp

lemma onl_invariantI [intro]:
assumes init: "∀ l ∈ labels Γ p. (ξ, p) ∈ init A; l ∈ labels Γ p \implies P (ξ, l)"
and step: "∀ l ∈ labels Γ p'. P (ξ', l')\implies P (ξ', p')"\implies (I ∨) onl Γ P"
shows A \models (I ∨) onl Γ P"
proof (rule invariant_pairI)
fix ξ p
assume "(ξ, p) ∈ init A"
hence "∀ l ∈ labels Γ p. P (ξ, l)" using init by simp
thus "onl Γ P (ξ, p)"

next

fix ξ p a ξ’ p'
assume rp: "((ξ, p), a, (ξ’, p’)) ∈ trans A"
and "I a"

from 'onl Γ P (ξ, p)' have "∀ l ∈ labels Γ p. P (ξ, l)" ..
with rp tr 'I a' have "∀ l' ∈ labels Γ p'. P (ξ', l')" by (auto elim: step)
thus "onl Γ P (ξ', p')" ..

qed

lemma onl_invariantD [dest]:
assumes A \models (I ∨) onl Γ P" and "((ξ, p), a, (ξ’, p’)) ∈ trans A" and "I ∈ labels Γ p"
shows "P (ξ, l)"
using assms unfolding onl_def by auto
lemma onl_invariant_initD [dest]:
assumes invP: "A ⟱ (I →) onl Γ P"
  and init: "(ξ, p) ∈ init A"
  and pnl: "l ∈ labels Γ p"
shows "P (ξ, l)"
proof -
  from init have "(ξ, p) ∈ reachable A I" ..
  with invP show ?thesis using pnl ..
qed

lemma onl_invariant_sterms:
assumes wf: "wellformed Γ"
  and il: "A ⟱ (I →) onl Γ P"
  and rp: "(ξ, p) ∈ reachable A I"
  and "p' ∈ terms Γ p''
  and "l ∈ labels Γ p''"
shows "P (ξ, l)"
proof -
  from wf 'p' ∈ terms Γ 'l ∈ labels Γ p'' have "l ∈ labels Γ p''" by (rule labels_sterms_labels)
  with il rp show "P (ξ, l)" ..
qed

lemma onl_invariant_sterms_weaken:
assumes wf: "wellformed Γ"
  and il: "A ⟱ (I →) onl Γ P"
  and rp: "(ξ, p) ∈ reachable A I''
  and "p' ∈ terms Γ p''
  and "l ∈ labels Γ p''"
  and weaken: "∀ a. I' a ⟱ I a"
shows "P (ξ, l)"
proof -
  from '(ξ, p) ∈ reachable A I''' have "(ξ, p) ∈ reachable A I''" by (rule reachable_weakenE)
  (erule weaken)
  with assms(1-2) show ?thesis using assms(4-5) by (rule onl_invariant_sterms) simp
qed

lemma onl_invariant_sterms_TT:
assumes wf: "wellformed Γ"
  and il: "A ⟱ onl Γ P"
  and rp: "(ξ, p) ∈ reachable A I"
  and "p' ∈ terms Γ p"
  and "l ∈ labels Γ p"
shows "P (ξ, l)"
using assms by (rule onl_invariant_sterms_weaken) simp

lemma trans_from_sterms:
assumes "(((ξ, p), a, (ξ', q)) ∈ seqp_sos Γ)"
  and "wellformed Γ"
shows "p' ∈ terms Γ. (((ξ, p'), a, (ξ', q)) ∈ seqp_sos Γ)
using assms by (induction p rule: sterms_pinduct [OF 'wellformed Γ']) auto

lemma trans_from_sterms':
assumes "(((ξ, p'), a, (ξ', q)) ∈ seqp_sos Γ)"
  and "wellformed Γ"
and "p' ∈ terms Γ p" shows "(((ξ, p), a, (ξ', q)) ∈ seqp_sos Γ)
using assms by (induction p rule: sterms_pinduct [OF 'wellformed Γ']) auto

lemma trans_to_dterms:
assumes "(((ξ, p), a, (ξ', q)) ∈ seqp_sos Γ)"
  and "wellformed Γ"
shows "∀ r ∈ sterms Γ. r ∈ dterms Γ p"
using assms by (induction q) auto

**Theorem** cterms_includes_sterms_of_seq_reachable:
assumes "wellformed Γ"
and "control_within Γ (init A)"
and "trans A = seqp_sos Γ"
shows "⋃(sterms Γ ' snd ' reachable A I) ⊆ cterms Γ"

**Proof**
fix qs
assume "qs ∈ ⋃(sterms Γ ' snd ' reachable A I)"
then obtain ξ and q where *: "((ξ, q) ∈ reachable A I)"
and **: "qs ∈ sterms Γ q" by auto
from * have "⋀x. x ∈ sterms Γ q =⇒ x ∈ cterms Γ" (proof)
fix ξ p q
assume "((ξ, p) ∈ init A)"
and "q ∈ sterms Γ p" from 'control_within Γ (init A)' and '((ξ, p) ∈ init A)'
obtain pn where "p ∈ subterms (Γ pn)" by auto
with 'wellformed Γ' show "q ∈ cterms Γ" using 'q ∈ sterms Γ p' by (rule subterms_sterms_in_cterms)
next
fix p ξ a ξ' q x
assume "((ξ, p) ∈ reachable A I)"
and IH: "⋀x. x ∈ sterms Γ p =⇒ x ∈ cterms Γ" (from this(3) and 'trans A = seqp_sos Γ' have "((ξ, p), a, (ξ', q)) ∈ seqp_sos Γ" by simp)
from this and 'wellformed Γ' obtain ps
where ps: "ps ∈ sterms Γ p" (where)
and step: "((ξ, ps), a, (ξ', q)) ∈ seqp_sos Γ" (by (rule trans_from_sterms [THEN bexE]))
from ps have "ps ∈ cterms Γ" by (rule IH)
moreover from step 'wellformed Γ' 'x ∈ sterms Γ q' have "x ∈ dterms Γ ps" (by (rule trans_to_dterms [rule_format]))
ultimately show "x ∈ cterms Γ" by (rule ctermsDI)
qed
thus "qs ∈ cterms Γ" using ** .
qed

**Corollary** seq_reachable_in_cterms:
assumes "wellformed Γ"
and "control_within Γ (init A)"
and "trans A = seqp_sos Γ"
and "((ξ, p) ∈ reachable A I)"
and "p' ∈ sterms Γ p"
shows "p' ∈ cterms Γ" using assms(1-3)

**Proof** (rule cterms_includes_sterms_of_seq_reachable [THEN set_mp])
from assms(4-5) show "p' ∈ ⋃(sterms Γ ' snd ' reachable A I)"
by (auto elim!: rev_bexI)
qed

**Lemma** seq_invariant_ctermI:
assumes wf: "wellformed Γ"
and cw: "control_within Γ (init A)"
and sl: "simple_labels Γ"
and sp: "trans A = seqp_sos Γ"
and init: "∀ξ p l_1. [ (ξ, p) ∈ init A; l_1 ∈ labels Γ p ] =⇒ P (ξ, l_1)"
and step: "∀p l ξ a q l'. ξ' pp. [
proof

fix \( \xi \ p \ l \)
assume "\((\xi, p) \in \text{init } A"'
and "\((l \in \text{labels } \Gamma p) \land \text{init} \) show "P (\( \xi, l)"" by auto

next

fix \( \xi \ p \ a \ \xi' \ q \ l' \)
assume sr: "\((\xi, p) \in \text{reachable } A \ I"'
and pl: "\((\forall l \in \text{labels } \Gamma p. P (\( \xi, l)) \land \text{trans } A"'
and tr: "\((\xi, p), a, (\xi', q)) \in \text{trans } A"" and "\"I a"'

from this(3) and \(\text{trans } A = \text{seqp_sos } \Gamma'\) have tr': "\((\xi, p), a, (\xi', q)) \in \text{seqp_sos } \Gamma\"" by simp

show "P (\( \xi', l'))" proof -

from sr and tr and 'I a' have A7: "(\( \xi', q) \in \text{reachable } A \ I"" ..
from tr' obtain p' where "p' \in \text{sterms } \Gamma p" and "((\xi, p'), a, (\xi', q)) \in \text{seqp_sos } \Gamma"" by blast

by (blast dest: trans_from_sterms [OF _ wf])
from wf cv sp sr this(1) have A1: "p' \in \text{cterms } \Gamma" by (rule seq_reachable_in_cterms)
from labels_not_empty [OF wf] obtain ll where A2: "ll \in \text{labels } \Gamma p" by blast

with 'p' \in \text{sterms } \Gamma p' have "ll \in \text{labels } \Gamma p" by (rule labels_sterms_labels [OF wf])

with pl have A3: "P (\( \xi, ll)"" by simp
from 'p' \in \text{sterms } \Gamma p' have A4: "((\xi, p'), a, (\xi', q)) \in \text{seqp_sos } \Gamma"" by simp

from sr tr 'I a' have A9: "(\( \xi', q) \in \text{reachable } A \ I"" ..
from A1 A2 A3 A4 A5 A6 A7 A8 A9 'I a' show ?thesis by (rule step)

qed

lemma seq_invariant_ctermsI:
assumes wf: "\text{wellformed } \Gamma"'
and "\text{control_within } \Gamma (\text{init } A)"'
and "\text{simple_labels } \Gamma"'
and "\text{trans } A = \text{seqp_sos } \Gamma"" and init: "\(\\forall \xi \ p \ l. \ [ (\xi, p) \in \text{init } A; l \in \text{labels } \Gamma p \implies P (\( \xi, l))""

and step: "\(\\forall p \ l \ \xi \ a \ q \ l' \ \xi' \ pp \ pn. \ [ \text{wellformed } \Gamma; p \in \text{cterms } \Gamma (\text{pn}); \text{not_call } p; l \in \text{labels } \Gamma p; \text{init} \) show "P (\( \xi, l'))" by (rule step)

qed
\[ P (\xi, 1); \]
\[ ((\xi, p), a, (\xi', q)) \in \text{seqp sos } \Gamma; \]
\[ ((\xi, p), a, (\xi', q)) \in \text{trans } A; \]
\[ \xi' \in \text{labels } \Gamma q; \]
\[ (\xi, pp) \in \text{reachable } A I; \]
\[ p \in \text{sterms } \Gamma pp; \]
\[ (\xi', q) \in \text{reachable } A I; \]
\[ I a \]

shows \( A \models (I \rightarrow \text{onl } \Gamma P) \)

using \text{assms(1-4)} proof (rule \text{seq invariant ctermI})

fix \( \xi \) p l

assume \( ((\xi, p), a, (\xi', q)) \in \text{seqp sos } \Gamma; \)

thus \( P (\xi, l) \) by \( \text{rule init} \)

next

fix p l \( \xi a q l' \) \( \xi' pp \)

assume \( p \in \text{cterms } \Gamma \)

and \text{otherassms}: \( l \in \text{labels } \Gamma p \)

\( P (\xi, l) \)
\( ((\xi, p), a, (\xi', q)) \in \text{seqp sos } \Gamma \)
\( ((\xi, p), a, (\xi', q)) \in \text{trans } A \)
\( \xi' \in \text{labels } \Gamma q \)
\( (\xi, pp) \in \text{reachable } A I \)
\( p \in \text{sterms } \Gamma pp \)
\( (\xi', q) \in \text{reachable } A I \)
\( I a \)

from this(1) obtain \( pn \) where \( p \in \text{cterms}(\Gamma pn) \)

and \( \text{not_call } p \)

unfolding \text{cterms def'} [OF \text{wf}] by \text{auto}

with \text{wf} show \( P (\xi', l') \)

using \text{otherassms} by \( \text{rule step} \)

qed

19.2 Step invariants via labelled control terms

\text{definition}
\( \text{onll} :: \langle(\xi, l) \rangle \text{seqp_env} \)
\( \Rightarrow (\xi, l) \text{ transition } \Rightarrow \text{bool} \)
\( \Rightarrow (\xi, l) \text{ seqp, } a \text{ transition } \Rightarrow \text{bool} \)

where
\( \text{onll } \Gamma P \equiv (\lambda((\xi, p), a, (\xi', q))). \forall l \in \text{labels } \Gamma p. \forall l' \in \text{labels } \Gamma p'. P ((\xi, l), a, (\xi', l')) \)

\text{lemma onllI [intro]}:

assumes \( \lambda l l'. [ l \in \text{labels } \Gamma p; l' \in \text{labels } \Gamma p' ] \Rightarrow P ((\xi, l), a, (\xi', l')) \)

shows \( \text{onll } \Gamma P ((\xi, p), a, (\xi', p')) \)

using \text{assms unfolding onll_def by simp}

\text{lemma onllI [intro]}:

assumes \( \forall l \in \text{labels } \Gamma p. \forall l' \in \text{labels } \Gamma p'. P ((\xi, l), a, (\xi', l')) \)

shows \( \text{onll } \Gamma P ((\xi, p), a, (\xi', p')) \)

using \text{assms by auto}

\text{lemma onllD [dest]}:

assumes \( \text{onll } \Gamma P ((\xi, p), a, (\xi', p')) \)

shows \( \forall l \in \text{labels } \Gamma p. \forall l' \in \text{labels } \Gamma p'. P ((\xi, l), a, (\xi', l')) \)

using \text{assms unfolding onll_def by simp}

\text{lemma onll weaken [elim!]}: \( \lambda l \Gamma P Q s. [ \text{onll } \Gamma P s; \land s. P s \Rightarrow Q s ] \Rightarrow \text{onll } \Gamma Q s \)

by (\text{clarsimp dest!: onllD intro!: onll})

\text{lemma onll weaken [elim!]}: \( \lambda l \Gamma P Q s. [ \text{onll } \Gamma P s; \land s. P s \Rightarrow Q s ] \Rightarrow \text{onll } \Gamma Q s \)

by (\text{clarsimp dest!: onllD intro!: onll})
lemma onll_weaken [elim!]: "∀Γ P Q s. [ onll Γ P ((ξ, p), a, (ξ', p'))];
  ∃l1. P ((ξ, l), a, (ξ', l')) ⇒ Q ((ξ, l), a, (ξ', l')) ]
⇒ onll Γ Q ((ξ, p), a, (ξ', p'))"
by (clarsimp dest!: onllD intro!: onllI)

lemma onll_step_invariantI [intro]:
assumes *: "∀ξ p l a ξ' p' l'. P ((ξ, p), a, (ξ', p'));
  l ∈ labels Γ p;
  l' ∈ labels Γ p' ]
⇒ P ((ξ, l), a, (ξ', l'))"
shows "A |= A (I →) onll Γ P"
proof
  fix ξ p ξ' p' a
  assume "((ξ, p), a, (ξ', p')) ∈ trans A" and "I a"
  hence "∀l ∈ labels Γ p. ∀l' ∈ labels Γ p'. P ((ξ, l), a, (ξ', l'))" by (rule *)
  thus "onll Γ P ((ξ, p), a, (ξ', p'))" ..
  qed

lemma onll_step_invariantE [elim]:
assumes "A |= A (I →) onll Γ P"
and "((ξ, p), a, (ξ', p')) ∈ trans A" and "I a"
shows "∀l ∈ labels Γ p. ∀l' ∈ labels Γ p'. P ((ξ, l), a, (ξ', l'))"
proof -
  from assms(1-4) have "onll Γ P ((ξ, p), a, (ξ', p'))" ..
  with lp lp' show "P ((ξ, l), a, (ξ', l'))" by auto
  qed

lemma onll_step_invariantD [dest]:
assumes "A |= A (I →) onll Γ P"
and "((ξ, p), a, (ξ', p')) ∈ trans A" and "I a"
shows "∀l ∈ labels Γ p. ∀l' ∈ labels Γ p'. P ((ξ, l), a, (ξ', l'))"
using assms by auto

lemma onll_step_to_invariantI [intro]:
assumes sinv: "A |= A (I →) onll Γ Q"
and wf: "wellformed Γ"
and init: "∀ξ l p. [ (ξ, p) ∈ init A; l ∈ labels Γ p ] ⇒ P ((ξ, l), a, (ξ', l'))"
and step: "∀ξ l p l' a. [ (ξ, p) ∈ reachable A I; l ∈ labels Γ p; Q ((ξ, l), a, (ξ', l')); I a ] ⇒ P ((ξ', l'))"
shows "A |= (I →) onll Γ P"
proof
  fix ξ p l
  assume "((ξ, p), a, (ξ', p')) ∈ trans A" and "I a"
  hence "∀l ∈ labels Γ p. ∀l' ∈ labels Γ p'. P ((ξ, l), a, (ξ', l'))" by (rule *)
  thus "onll Γ P ((ξ, p), a, (ξ', p'))" by (rule init)
next
  fix ξ p a ξ' p' l'
  assume sr: "((ξ, p), a, (ξ', p')) ∈ trans A" and "I a"
  hence "∀l ∈ labels Γ p. P ((ξ, l), a, (ξ', l'))" by (rule *)
  thus "onll Γ Q ((ξ, p), a, (ξ', p'))" by (rule *)
and \( l' \in \text{labels } \Gamma p' \)

show "\( P ((\xi', l')) \)"

proof -
from \( l p' \) obtain \( l \) where "\( l \in \text{labels } \Gamma p \)" and "\( P (\xi, l) \)"
using \( \text{labels not empty [OF wf]} \) by auto
from \( \text{sinv sr tr 'I a' this(1) lp'} \) have "\( Q (((\xi, l), a, (\xi', l'))) \) ..
with \( sr 'l' \in \text{labels } \Gamma p' \) have "\( 'P ((\xi, l), a, (\xi', l')) \)"
by (rule \( \text{step} \))
qed

lemma \( \text{onll_step_invariant_sterms} \):
assumes "\( \text{wf}: \text{wellformed } \Gamma \)"
and \( \text{si}: 'A \models A (I \rightarrow) \text{onll } \Gamma P \)"
and \( \text{sr}: '((\xi, p), a, (\xi', q)) \in \text{trans } A ' \)
and "\( 'I a' \)"
and "\( l' \in \text{labels } \Gamma q \)"
and "\( p' \in \text{sterms } \Gamma p \)"
and "\( l' \in \text{labels } \Gamma p' \)"
shows "\( P ((\xi', l'), a, (\xi', l')) \)"

proof -
from \( \text{tr} 'p' \in \text{sterms } \Gamma p 'l' \in \text{labels } \Gamma p' \)
by (rule \( \text{labels_sterms_labels} \))
with \( \text{si sr sos 'I a'} \) have "\( P ((\xi, l), a, (\xi', l')) \)"
by (rule \( \text{onll_step_invariant_sterms} \))
qed

lemma \( \text{seq_step_invariant_sterms} \):
assumes "\( \text{inv}: 'A \models A (I \rightarrow) \text{onll } \Gamma P \)"
and "\( \text{wf}: \text{wellformed } \Gamma \)"
and \( \text{sp}: '\text{trans } A = \text{seqp_sos } \Gamma P ' \)
and "\( l' \in \text{labels } \Gamma q \)"
and \( \text{sr}: '((\xi, p), a, (\xi', q)) \in \text{trans } A ' \)
and "\( 'I a' \)"
and "\( p' \in \text{sterms } \Gamma p \)"
and "\( l' \in \text{labels } \Gamma p' \)"
shows "\( \forall l \in \text{labels } \Gamma p'. P ((\xi, l), a, (\xi', l')) \)"

proof -
from \( \text{tr} 'p' \in \text{sterms } \Gamma p 'l' \in \text{labels } \Gamma p' \)
by (rule \( \text{trans_from_sterms} ' \))
with \( \text{sp} '((\xi, p), a, (\xi', q)) \in \text{trans } A ' \)
fix \( l 'I' a' \) assume "\( l \in \text{labels } \Gamma p ' \)"
with \( \text{sp} \) have "\( '((\xi, p'), a, (\xi', q)) \in \text{trans } A ' \)
by (rule \( \text{onll_step_invariant_sterms} \))
show "\( \forall l \in \text{labels } \Gamma p'. P ((\xi, l), a, (\xi', l')) \)"

lemma \( \text{seq_step_invariant_sterms_weaken} \):
assumes "\( 'A \models A (I \rightarrow) \text{onll } \Gamma P \)"
and "\( \text{wellformed } \Gamma ' \)
and "\( '\text{trans } A = \text{seqp_sos } \Gamma P ' \)
and "\( l' \in \text{labels } \Gamma q ' \)
and "\( ((\xi, p), a, (\xi', q)) \in \text{trans } A ' \)
and "\( 'I a' \)"
and "\( p' \in \text{sterms } \Gamma p ' \)
and "\( 'l' \in \text{labels } \Gamma q ' \)

shows "\( \forall l \in \text{labels } \Gamma p'. P ((\xi, l), a, (\xi', l')) \)"

proof -
from \( \text{tr} 'p' \in \text{sterms } \Gamma p 'l' \in \text{labels } \Gamma p' \)
and \( 'l' \in \text{labels } \Gamma p' 'l' \in \text{labels } \Gamma q' 'p' \in \text{sterms } \Gamma p ' \)

by (rule \( \text{onll_step_invariant_sterms} ' \))
qed
lemma seq_step_invariant_sterms_TT:
assumes "A \models A onll \Gamma P"
and "wellformed \Gamma"
and "trans A = seqp_sos \Gamma"
and "l' \in labels \Gamma q"
and "((\xi, p'), a, (\xi', q)) \in trans A"
and "I a"
and "p' \in sterms \Gamma p"
shows "\forall l \in labels \Gamma p'. P (\xi, l)
using assms by (rule seq_step_invariant_sterms_weaken) simp

lemma onll_step_invariant_any_sterms:
assumes "wellformed \Gamma"
and "A \models \Gamma P"
and "((\xi, p), a, (\xi', q)) \in trans A"
and "I a"
and "l' \in labels \Gamma q"
shows "\forall p' \in sterms \Gamma p. \forall l \in labels \Gamma p'. P (\xi, l)" by (intro ballI) (rule onll_step_invariant_sterms [OF assms])

lemma seq_step_invariant_ctermI [intro]:
assumes wf: "wellformed \Gamma"
and cw: "control_within \Gamma (init A)"
and sl: "simple_labels \Gamma"
and sp: "trans A = seqp_sos \Gamma"
and step: "\forall p pp l \xi a q l' \xi'. [ p \in cterms \Gamma;
  l \in labels \Gamma p;
  ((\xi, p), a, (\xi', q)) \in seqp_sos \Gamma;
  ((\xi, p), a, (\xi', q)) \in trans A;
  l' \in labels \Gamma q;
  (\xi, pp) \in reachable A I;
  p' \in sterms \Gamma pp;
  (\xi', q) \in reachable A I;
  I a ] \Longrightarrow P (\xi, l)
shows "A \models (I \rightarrow) onll \Gamma P"
proof
  fix \xi p l a \xi' q l'
  assume sr: "((\xi, p), a, (\xi', q)) \in reachable A I"
  and tr: "((\xi, p), a, (\xi', q)) \in trans A"
  and "I a"
  and pl: "l \in labels \Gamma p"
  and A5: "l' \in labels \Gamma q"
from this(2) and sp have tr': "((\xi, p), a, (\xi', q)) \in seqp_sos \Gamma" by simp
then obtain p' where "p' \in sterms \Gamma p"
  by (blast dest: trans_from_sterms [OF \_ wf])
from wf cw sp sr this(1) have A1: "p' \in cterms \Gamma"
  by (rule seq_reachable_in_cterms)
from "((\xi, p'), a, (\xi', q)) \in seqp_sos \Gamma'" and sp
  have A4: "((\xi, p'), a, (\xi', q)) \in trans A" by simp
from sr 'p' \in sterms \Gamma p' obtain pp where A6: "((\xi, pp) \in reachable A I"
  and A7: "p' \in sterms \Gamma pp"
  by auto
from sr tr 'I a' have A8: "(\xi', q) \in reachable A I" ..
from wf cw sp sr have "\exists pn. p \in subterms (\Gamma pn)"
  by (rule reachable_subterms)
with sl wf have "\forall p' \in sterms \Gamma p. l \in labels \Gamma p'"
  using pl by (rule simple_labels_in_sterms)
with \( p' \in \text{sterms} \Gamma \) have "l \in \text{labels} \Gamma \) p" by simp
with A1 show "p ((\xi, l), a, (\xi', l'))" using A3 A4 A5 A6 A7 A8 'I a' by (rule step)
qed

lemma seq_step_invariant_ctermsI [intro]:
assumes "\text{wellformed} \Gamma"
and \( \text{cw: "control\_within} \Gamma (\text{init} A)"\)
and \( \text{sl: "simple\_labels} \Gamma"\)
and \( \text{sp: "trans} A = \text{seqp\_sos} \Gamma"\)
and step: "\( \forall p \ l \ a \ q \ l' \xi \ p \ p' \ \xi' \ pp pn. [\)
wellformed \( \Gamma; \)
p \( \in \text{cterms} \Gamma \) pn);
not_call p;
l \( \in \text{labels} \Gamma \) p;
((\xi, p), a, (\xi', q)) \( \in \text{seqp\_sos} \Gamma; \)
((\xi, p), a, (\xi', q)) \( \in \text{trans} A; \)
l' \( \in \text{labels} \Gamma \) q;
(\xi, pp) \( \in \text{reachable} A I; \)
p \( \in \text{sterms} \Gamma \) pp;
((\xi', q) \( \in \text{reachable} A I; \)
I a
] \implies p ((\xi, l), a, (\xi', l'))"
shows "A \models_{A} (I \to) \text{onli} \Gamma P"
using assms(1-4) proof (rule seq_step_invariant_ctermI)
fix p pp l \( \xi \ a \ q \ l' \xi' \)
assume "p \( \in \text{cterms} \Gamma"\)
and otherassms: "l \( \in \text{labels} \Gamma \) p"
"((\xi, p), a, (\xi', q)) \( \in \text{seqp\_sos} \Gamma; \)
"((\xi, p), a, (\xi', q)) \( \in \text{trans} A; \)
"l' \( \in \text{labels} \Gamma \) q;
((\xi, pp) \( \in \text{reachable} A I; \)
"p \( \in \text{sterms} \Gamma \) pp;
"((\xi', q) \( \in \text{reachable} A I; \)
"I a"
from this(1) obtain pn where "p \( \in \text{cterms} \Gamma \) pn"
and "not\_call p"
unfolding cterms_def' [OF wf] by auto
with wf show "p ((\xi, l), a, (\xi', l'))"
using otherassms by (rule step)
qed

end

20 Generic open invariants on sequential AWN processes

theory OAWN_Invariants
imports Invariants OInvariants
AWN_Cterms AWN_Labels AWN_Invariants
OAWN_SOS
begin

20.1 Open invariants via labelled control terms

lemma oseqp_sos_subterms:
assumes "\text{wellformed} \Gamma"
and "\exists pn. p \( \in \text{subterms} \Gamma \) pn"
and "((\sigma, p), a, (\sigma', p')) \( \in \text{oseqp\_sos} \Gamma i"n"
shows "\exists pn. p' \( \in \text{subterms} \Gamma \) pn"
using assms
proof (induct p)
fix p1 p2
assume IH1: "\exists pn. p1 \( \in \text{subterms} \Gamma \) pn \implies ((\sigma, p1), a, (\sigma', p')) \( \in \text{oseqp\_sos} \Gamma i \implies"
∃pn. p' ∈ subterms (Γ pn)"
and IH2: "∃pn. p2 ∈ subterms (Γ pn) \implies ((σ, p2), (σ', p')) ∈ oseqp_sos Γ i \implies
∃pn. p' ∈ subterms (Γ pn)"
and "∃pn. p1 ⊕ p2 ∈ subterms (Γ pn)"
and "((σ, p1 ⊕ p2), (σ', p')) ∈ oseqp_sos Γ i"
from '∃pn. p1 ⊕ p2 ∈ subterms (Γ pn)’ obtain pn
where "p1 ∈ subterms (Γ pn)"
and "p2 ∈ subterms (Γ pn)" by auto
from '[((σ, p1), (σ', p')) ∈ oseqp_sos Γ i’ have "((σ, p1 ⊕ p2), (σ', p')) ∈ oseqp_sos Γ i" by auto
thus "∃pn. p' ∈ subterms (Γ pn)"
proof
assume "((σ, p1), (σ', p')) ∈ oseqp_sos Γ i”
with 'p1 ∈ subterms (Γ pn)’ show ?thesis by (auto intro: IH1)
next
assume "((σ, p2), (σ', p')) ∈ oseqp_sos Γ i”
with 'p2 ∈ subterms (Γ pn)’ show ?thesis by (auto intro: IH2)
qed

lemma oreachable_subterms:
assumes "wellformed Γ"
and "control_within Γ (init A)"
and "trans A = oseqp_sos Γ i"
and "(σ, p) ∈ oreachable A S U"
shows "∃pn. p ∈ subterms (Γ pn)" using assms(4)
proof (induct rule: oreachable_pair_induct)
fix σ p
assume "(σ, p) ∈ init A”
with ‘control_within Γ (init A)’ show "∃pn. p ∈ subterms (Γ pn)” ..
next
fix σ p a σ’ p’
assume "(σ, p) ∈ oreachable A S U"
and "∃pn. p ∈ subterms (Γ pn)"
and "((σ, p), a, (σ’, p’)) ∈ trans A”
and "S σ σ’ a”
moreover from this(3) and ‘trans A = oseqp_sos Γ i’
and "((σ, p), a, (σ’, p’)) ∈ oseqp_sos Γ i” have "((σ, p), a, (σ’, p’)) ∈ oseqp_sos Γ i“ by simp
ultimately show "∃pn. p' ∈ subterms (Γ pn)"
using ‘wellformed Γ’
by (auto elim: oseqp_sos_subterms)
qed

lemma onl_invariantI [intro]:
assumes init: "∀σ p l. [ (σ, p) ∈ init A; l ∈ labels Γ p ] \implies P (σ, l)"
and other: "∀σ σ’ p l. [ (σ, p) ∈ oreachable A S U;
∀l ∈ labels Γ p. P (σ, l);
U σ σ’ ] \implies ∀l ∈ labels Γ p. P (σ’, l)”
and step: "∀σ p a σ’ p’ l’. 
[ (σ, p) ∈ oreachable A S U;
∀l ∈ labels Γ p. P (σ, l);
((σ, p), a, (σ’, p’)) ∈ trans A;
l' ∈ labels Γ p';
S σ σ’ a ] \implies P (σ’, l’)"
shows "A |≡ (S, U →) onl Γ P”
proof
fix σ p
assume "(σ, p) ∈ init A”
hence "∀l ∈ labels Γ p. P (σ, l)” using init by simp
thus "onl Γ P (σ, p)” ..
next
fix \( \sigma \ p \ a \ a' \ p' \)
assume \( \text{rp}: \ "(\sigma, p) \in \text{oreachable} \ A \ S \ U" \)
and "onl \( \Gamma \) \( P \ (\sigma, p) \)"
and tr: "\(((\sigma, p), a, (\sigma', p')) \in \text{trans} \ A" 
and "S \ a \ a'
from 'onl \( \Gamma \) \( P \ (\sigma, p) \)'
have "\( \forall l \in \text{labels} \ \Gamma \ p. \ P \ (\sigma, l) \)" ..

next
fix \( \sigma \ a \ a' \ p' \)
assume "\( (\sigma, p) \in \text{oreachable} \ A \ S \ U \)"
and "onl \( \Gamma \) \( P \ (\sigma, p) \)"
and "U \( \sigma \ a \ a' \)"
from 'onl \( \Gamma \) \( P \ (\sigma, p) \)'
have "\( \forall l \in \text{labels} \ \Gamma \ p. \ P \ (\sigma, l) \)" by simp

qed

lemma global_oinvariantI [intro]:
assumes init: "\( \forall \sigma \ p. \ (\sigma, p) \in \text{init} \ A \Rightarrow P \ (\sigma) \)"
and other: "\( \forall \sigma \ a \ p. \ [ (\sigma, p) \in \text{oreachable} \ A \ S \ U; \ P \ (\sigma) ; \ U \ a \ a' ] \Rightarrow P \ a' \)"
and step: "\( \forall \sigma \ p \ a \ a' \ p'. \ [ (\sigma, p) \in \text{oreachable} \ A \ S \ U; \ P \ (\sigma) ; \ ((\sigma, p), a, (\sigma', p')) \in \text{trans} \ A; \ S \ a \ a' ] \Rightarrow P \ a' \)"

shows "A \( \models (S, U \rightarrow) \ (\lambda (\sigma, _). \ P \ (\sigma)) \)"

proof

fix \( \sigma \ p \)
assume "\( (\sigma, p) \in \text{init} \ A \)"
thus "(\( \lambda (\sigma, _). \ P \ (\sigma) \) \( (\sigma, p) \)"
  by simp (erule init)

next
fix \( \sigma \ p \ a \ a' \ p' \)
assume rp: "\( (\sigma, p) \in \text{oreachable} \ A \ S \ U \)"
and "(\( \lambda (\sigma, _). \ P \ (\sigma) \) \( (\sigma, p) \)"
and tr: "\(((\sigma, p), a, (\sigma', p')) \in \text{trans} \ A" 
and "S \ a \ a'
from '(\( \lambda (\sigma, _). \ P \ (\sigma) \) \( (\sigma, p) \)'
have "P \( \sigma \)" by simp
with rp have "P \( \sigma' \)"
  using tr 'S \ a \ a' by (rule step)
thus "(\( \lambda (\sigma, _). \ P \ (\sigma) \) \( (\sigma', p') \)" by simp

next
fix \( \sigma \ p \)
assume "\( (\sigma, p) \in \text{oreachable} \ A \ S \ U \)"
and "(\( \lambda (\sigma, _). \ P \ (\sigma) \) \( (\sigma, p) \)"
and "U \( \sigma \ a \ a' \)"
hence "P \( \sigma' \)" by simp (erule other)
thus "(\( \lambda (\sigma, _). \ P \ (\sigma) \) \( (\sigma', p') \)" by simp

qed

lemma onl_oinvariantD [dest]:
assumes "A \( \models (S, U \rightarrow) \ \text{onl} \ \Gamma \ P \)"
and "\( (\sigma, p) \in \text{oreachable} \ A \ S \ U \)"
and "\( l \in \text{labels} \ \Gamma \ p \)"
shows "P \( (\sigma, l) \)"
using assms unfolding onl_def by auto

lemma onl_oinvariant_weakenD [dest]:
assumes "A \( \models (S', U' \rightarrow) \ \text{onl} \ \Gamma \ P \)"
and "\( (\sigma, p) \in \text{oreachable} \ A \ S \ U \)"
and "\( l \in \text{labels} \ \Gamma \ p \)"
and weakenS: "\( \forall s s' a. \ S \ s s' a \Rightarrow S' \ s s' a \)"
and weakenU: "\( \land s s'. U s s' \Rightarrow U' s s' \)"
shows "P (\( \sigma, l \))"

proof -
from \( '(\sigma, p) \in \text{oreachable} A S U' \) have "(\( \sigma, p \) \in \text{oreachable} A S U')"
by (rule oreachable_weakenE)
(erule weakenS, erule weakenU)
with 'A \models (S', U' \rightarrow) onl \Gamma P' show "P (\( \sigma, l \))"
using 'l \in labels \Gamma p' ..

qed

lemma onl_oinvariant_initD [dest]:
assumes invP: "A \models (S, U \rightarrow) onl \Gamma P"
and init: "(\( \sigma, p \) \in init A"
and pnl: "l \in labels \Gamma p"
shows "P (\( \sigma, l \))"
proof -
from init have "(\( \sigma, p \) \in \text{oreachable} A S U"
with invP show ?thesis using pnl ..

qed

lemma onl_oinvariant_sterms:
assumes wf: "wellformed \Gamma"
and il: "A \models (S, U \rightarrow) onl \Gamma P"
and rp: "(\( \sigma, p \) \in \text{oreachable} A S U"
and "p'\in\text{sterms} \Gamma p"
and "l\in\text{labels} \Gamma p'"
shows "P (\( \sigma, l \))"
proof -
from wf 'p'\in\text{sterms} \Gamma p' 'l\in\text{labels} \Gamma p'' have "l\in\text{labels} \Gamma p"
by (rule labels_sterms_labels)
with il rp show "P (\( \sigma, l \))" ..

qed

lemma onl_oinvariant_sterms_weaken:
assumes wf: "wellformed \Gamma"
and il: "A \models (S', U' \rightarrow) onl \Gamma P"
and rp: "(\( \sigma, p \) \in \text{oreachable} A S U"
and "p'\in\text{sterms} \Gamma p"
and "l\in\text{labels} \Gamma p'"
and weakenS: "\( \land \sigma \sigma' a. S \sigma \sigma' a \Rightarrow S' \sigma \sigma' a""
and weakenU: "\( \land \sigma \sigma'. U \sigma \sigma' \Rightarrow U' \sigma \sigma'""
shows "P (\( \sigma, l \))"
proof -
from \( '(\sigma, p) \in \text{oreachable} A S U' \) have "(\( \sigma, p \) \in \text{oreachable} A S U')"
by (rule oreachable_weakenE)
(erule weakenS, erule weakenU)
with assms(1-2) show ?thesis using assms(4-5)
by (rule onl_oinvariant_sterms)

qed

lemma otrans_from_sterms:
assumes "(\( (\sigma, p), a, (\sigma', q) \) \in \text{oseqp_sos} \Gamma i"
and "wellformed \Gamma"
shows "\( \exists p'\in\text{sterms} \Gamma p. ((\sigma, p'), a, (\sigma', q)) \in \text{oseqp_sos} \Gamma i""
using assms by (induction p rule: sterms_pinduct [OF 'wellformed \Gamma']) auto

lemma otrans_from_sterms':
assumes "(\( (\sigma, p'), a, (\sigma', q) \) \in \text{oseqp_sos} \Gamma i"
and "wellformed \Gamma"
and "p' \in \text{sterms} \Gamma p"
shows "((\sigma, p), a, (\sigma', q)) \in \text{oseqp_sos} \Gamma i""
using assms by (induction p rule: sterms_pinduct [OF 'wellformed \Gamma']) auto

lemma otrans_to_dterms:
assumes "[((σ, p), a, (σ', q)) ∈ oseqp sos Γ i"
          and "wellformed Γ"
shows "∀ r∈sterms Γ q. r ∈ dterms Γ p"
using assms by (induction q) auto

theorem cterms_includes_sterms_of_oseq_reachable:
assumes "wellformed Γ"
          and "control_within Γ (init A)"
          and "trans A = oseqp sos Γ i"
shows "∪ (sterms Γ ' snd ' oreachable A S U) ⊆ cterms Γ "
proof
  fix qs
  assume "qs ∈ ∪ (sterms Γ ' snd ' oreachable A S U)"
  then obtain ξ and q where *: "((ξ, q) ∈ oreachable A S U"
                               and **: "qs ∈ cterms Γ q" by auto
  from * have "∀ x. x ∈ cterms Γ q "
    proof (induction rule: oreachable_pair_induct)
      fix σ p q
      assume "((σ, p) ∈ init A"
      and "q ∈ cterms Γ p"
      from 'control_within Γ (init A)' and '(σ, p) ∈ init A'
      obtain pn where "p ∈ subterms (Γ pn)" by auto
      with 'wellformed Γ' show "q ∈ cterms Γ p" using "q∈sterms Γ p'
        by (rule subterms_sterms_in_cterms)
    next
      fix p σ a σ' q x
      assume "((σ, p) ∈ oreachable A S U"
      and IH: "∀ x. x ∈ cterms Γ p "
      and "((σ, p), a, (σ', q)) ∈ trans A"
      and "x ∈ cterms Γ q"
      from this(3) and 'trans A = oseqp sos Γ i'
      have step: "((σ, p), a, (σ', q)) ∈ oseqp sos Γ i" by simp
      from step 'wellformed Γ' obtain ps
      where ps: "ps ∈ cterms Γ p"
      and step': "((σ, ps), a, (σ', q)) ∈ oseqp sos Γ i" by simp
      from ps have "ps ∈ cterms Γ p" by (rule otrans_from_sterms [THEN bexE])
      moreover from step 'wellformed Γ' 'x ∈ cterms Γ q' have "x ∈ dterms Γ ps"
        by (rule otrans_to_dterms [rule_format])
      ultimately show "x ∈ cterms Γ p " by (rule ctermsDI)
      qed
    thus "qs ∈ cterms Γ " using ** .
    qed

corollary oseq_reachable_in_cterms:
assumes "wellformed Γ"
          and "control_within Γ (init A)"
          and "trans A = oseqp sos Γ i"
          and "((σ, p), a, (σ', q)) ∈ oreachable A S U"
          and "p' ∈ cterms Γ "
shows "p' ∈ cterms Γ "
using assms(1-3)
proof (rule cterms_includes_sterms_of_oseq_reachable [THEN set_mp])
  from assms(4-5) show "p' ∈ ∪ (sterms Γ ' snd ' oreachable A S U)"
    by (auto elim!: rev_bexI)
  qed

lemma oseq_invariant_ctermI:
assumes wf: "wellformed Γ"
          and cw: "control_within Γ (init A)"
          and sl: "simple_labels Γ"
          and sp: "trans A = oseqp sos Γ i"
          and init: "∀ σ p l. \[ σ p l. \] (σ, p) ∈ init A;
\[ l \in \text{labels } \Gamma \ p \]

and other: 
\[ \forall (\sigma, \sigma') p \in. [\]
\[ (\sigma, p) \in \text{oreachable } A S U; \]
\[ l \in \text{labels } \Gamma \ p; \]
\[ P (\sigma, l); \]
\[ U \sigma \sigma' ] \Rightarrow P (\sigma', 1)" \]

and local: 
\[ \forall \ p \ a \ q \ l' \sigma' pp. [\]
\[ p \in \text{cterms } \Gamma; \]
\[ l \in \text{labels } \Gamma \ p; \]
\[ P (\sigma, l); \]
\[ ((\sigma, p), a, (\sigma', q)) \in \text{oseqp_sos } \Gamma i; \]
\[ ((\sigma, p), a, (\sigma', q)) \in \text{trans } A; \]
\[ l' \in \text{labels } \Gamma \ q; \]
\[ (\sigma, pp) \in \text{oreachable } A S U; \]
\[ p \in \text{sterms } \Gamma \ pp; \]
\[ (\sigma', q) \in \text{oreachable } A S U; \]
\[ S \sigma \sigma' a \]
\[ ] \Rightarrow P (\sigma', l)'" \]

shows 
\[ A \models (S, U \rightarrow \text{onl } \Gamma \ p)" \]

proof

fix \( \sigma \ p \ l \)
assume \( "(\sigma, p) \in \text{init } A" \)
and \( *: l \in \text{labels } \Gamma \ p \)
with init show "P (\sigma, l)" by auto

next

fix \( \sigma \ a \ q \ l' \)
assume sr: "(\sigma, p) \in \text{oreachable } A S U"
and pl: "\( \forall l \in \text{labels } \Gamma \ p. P (\sigma, l)" \]
and tr: "((\sigma, p), a, (\sigma', q)) \in \text{trans } A"
and A6: "l' \in \text{labels } \Gamma \ q"
and "S \sigma \sigma' a"
thus "P (\sigma', l)"

proof -

from sr and tr and 'S \sigma \sigma' a have A7: "(\sigma', q) \in \text{oreachable } A S U" by (rule ooreachable_local')
from tr and sp have tr': "((\sigma, p), a, (\sigma', q)) \in \text{oseqp_sos } \Gamma i" by simp
then obtain p' where "p' \in \text{sterms } \Gamma p" by (blast dest: otrans_from_sterms [OF _ wf])
from wf cw sp sr this(1) have A1: "p' \in \text{cterms } \Gamma p" by (rule oseq_reachable_in_cterms)
from labels_not_empty [OF wf] obtain ll where A2: "ll \in \text{labels } \Gamma p'
by blast
with 'p' \in \text{sterms } \Gamma p' have "ll \in \text{labels } \Gamma p" by (rule labels_sterms_labels [OF wf])
with pl have A3: "P (\sigma, ll)" by simp
from sr 'p' \in \text{sterms } \Gamma p'
obtain pp where A7: "((\sigma, pp) \in \text{oreachable } A S U" 
and A8: "p' \in \text{sterms } \Gamma pp" by auto
from sr tr 'S \sigma \sigma' a have A9: "(\sigma', q) \in \text{oreachable } A S U" by (rule ooreachable_local')
from sp and '((\sigma, p'), a, (\sigma', q)) \in \text{oseqp_sos } \Gamma i' 
have A5: "((\sigma, p'), a, (\sigma', q)) \in \text{trans } A" by simp
from A1 A2 A3 A4 A5 A6 A7 A9 A9 'S \sigma \sigma' a' show ?thesis by (rule local)
qed

next

fix \( \sigma \ p \ l \)
assume sr: "(\sigma, p) \in \text{oreachable } A S U" 
and "\( \forall l \in \text{labels } \Gamma \ p. P (\sigma, l)" 
and "U \sigma \sigma'" 
show "\( \forall l \in \text{labels } \Gamma \ p. P (\sigma', l)" 
proof
lemma oseq_invariant_ctermI:
assumes wf: "wellformed Γ"
and cw: "control_within Γ (init A)"
and sl: "simple_labels Γ"
and sp: "trans A = oseqp_sos Γ i"
and init: "∀ σ p l. [ (σ, p) ∈ init A; l ∈ labels Γ p ] ⇒ P (σ, l)"
and other: "∀ σ σ' p l. [ wellformed Γ; (σ, p) ∈ ooreachable A S U; l ∈ labels Γ p; P (σ, l); U σ σ' ] ⇒ P (σ', l)"
and local: "∀ p l σ a q l' σ' pp pn. [ wellformed Γ; p ∈ ctermsl (Γ pn); not_call p; l ∈ labels Γ p; P (σ, l); ((σ, p), a, (σ', q)) ∈ oseqp_sos Γ i; ((σ, p), a, (σ', q)) ∈ trans A; l' ∈ labels Γ q; (σ, pp) ∈ ooreachable A S U; p ∈ stterms Γ pp; (σ', q) ∈ ooreachable A S U; S σ σ' a ] ⇒ P (σ', l')"
shows "A |= (S, U →) onl Γ P"
proof (rule oseq_invariant_ctermI [OF wf cw sl sp])
fix σ p l
assume "(σ, p) ∈ init A"
and "l ∈ labels Γ p"
thus "P (σ, l)" by (rule init)
next
fix σ σ' p l
assume "(σ, p) ∈ ooreachable A S U"
and "l ∈ labels Γ p"
and "P (σ, l)"
and "U σ σ'"
with wf show "P (σ', l)" by (rule other)
next
fix p l σ a q l' σ' pp
assume "p ∈ cterms Γ"
and otherassms: "l ∈ labels Γ p"
"P (σ, l)"
"((σ, p), a, (σ', q)) ∈ oseqp_sos Γ i"
"((σ, p), a, (σ', q)) ∈ trans A"
"l' ∈ labels Γ q"
"(σ, pp) ∈ ooreachable A S U"
"p ∈ stterms Γ pp"
"(σ', q) ∈ ooreachable A S U"
"S σ σ' a"
from this(1) obtain pn where "p ∈ ctermsl(Γ pn)"
and "not_call p"
unfolding cterms_def' [OF wf] by auto
with wf show "P (σ', l')"
using otherassms by (rule local)
qed

20.2 Open step invariants via labelled control terms

lemma onll_ostep_invariantI [intro]:
assumes "*: "∀σ p l a σ' p' l'. (σ, p) ∈ reachable A S U;
((σ, p), a, (σ', p')) ∈ trans A;
S σ σ' a;
l ∈ labels Γ p;
l' ∈ labels Γ p'
⇒ P ((σ, l), a, (σ', l'))"
shows "A |=A (S, U →) onll Γ P"
proof
  fix σ p σ' p' a
  assume "((σ, p), a, (σ', p')) ∈ trans A"
  hence "∀l ∈ labels Γ p. ∀l' ∈ labels Γ p'. P ((σ, l), a, (σ', l'))" by (auto elim!: *)
  thus "onll Γ P ((σ, p), a, (σ', p'))" ..
qed

lemma onll_ostep_invariantE [elim]:
assumes "A |=A (S, U →) onll Γ P"
and "((σ, p), a, (σ', p')) ∈ trans A"
and "S σ σ' a" and lp: "l ∈ labels Γ p"
and lp': "l' ∈ labels Γ p'"
shows "P ((σ, l), a, (σ', l'))"
proof -
  from assms(1-4) have "onll Γ P ((σ, p), a, (σ', p'))" ..
  with lp lp' show "P ((σ, l), a, (σ', l'))" by auto
qed

lemma onll_ostep_invariantD [dest]:
assumes "A |=A (S, U →) onll Γ P"
and "((σ, p), a, (σ', p')) ∈ trans A"
and "S σ σ' a"
shows "∀l ∈ labels Γ p. ∀l' ∈ labels Γ p'. P ((σ, l), a, (σ', l'))"
using assms by auto

lemma onll_ostep_invariant_weakenD [dest]:
assumes "A |=A (S', U' →) onll Γ P"
and "((σ, p), a, (σ', p')) ∈ trans A"
and "S' σ σ' a" and weakenS: "∀s s' a. S s s' a =⇒ S' s s' a"
and weakenU: "∀s s'. U s s' =⇒ U' s s'"
shows "∀l ∈ labels Γ p. ∀l' ∈ labels Γ p'. P ((σ, l), a, (σ', l'))"
proof -
  from "((σ, p), a, (σ', p')) ∈ trans A" and "S' σ σ' a'
  have "(σ, p) ∈ reachable A S' U'" by (rule reachable_weakenE)
  (erule weakenS, erule weakenU)
  with 'A |=A (S', U' →) onll Γ P' show ?thesis
  using '((σ, p), a, (σ', p')) ∈ trans A' and 'S' σ σ' a' ..
qed

lemma onll_ostep_to_invariantI [intro]:
assumes sinv: "A |=A (S, U →) onll Γ Q"
and wf: "wellformed Γ"
and init: "∀ σ. p ∈ init A; l ∈ labels Γ p |⇒ P (σ, l)"
and other: "∀ σ σ' p l.
   [ (σ, p) ∈ reachable A S U;
   l ∈ labels Γ p;
   P (σ, l);
   U σ σ'] |⇒ P (σ', l)"
and local: "∀ σ σ' p l.
   [ (σ, p) ∈ reachable A S U;
   l ∈ labels Γ p;
   P (σ, l);
   Q ((σ, l), a, (σ', l'));
   S σ σ' a] |⇒ P (σ', l')"

shows "A |⇒ (S, U) onl Γ P"

proof
fix σ p l
assume "(σ, p) ∈ init A" and "l ∈ labels Γ p"
thus "P (σ, l)" by (rule init)

next
fix σ σ' p l'
assume sr: "(σ, p) ∈ reachable A S U"
   and lp: "∀ l ∈ labels Γ p. P (σ, l)"
   and tr: "((σ, p), a, (σ', p')) ∈ trans A"
   and lp': "l' ∈ labels Γ p'"
   show "P (σ', l')"
   proof
   from lp obtain l where "l ∈ labels Γ p"
   using labels_not_empty [OF wf] by auto
   from sinv sr tr 'S σ σ' a' this(1) lp'
   have "P (σ', l')" ..
   with sr 'l ∈ labels Γ p'
   show "P (σ', l')" using 'S σ σ' a' by (rule local)
   qed

next
fix σ σ' p l
assume "(σ, p) ∈ reachable A S U"
   and "∀ l ∈ labels Γ p. P (σ, l)"
   and "U σ σ' a"
   and lp': "l' ∈ labels Γ p'"
   show "P (σ', l')"
   proof
   fix l
   assume "l ∈ labels Γ p"
   with '∀ l ∈ labels Γ p. P (σ, l)' have "P (σ, l)" ..
   with '{(σ, p), a, (σ', q)} ∈ trans A'
   show "P (σ', l')" using '{U σ σ'}' by (rule other)
   qed

qed

lemma onll_ostep_invariant_sterms:
assumes wf: "wellformed Γ"
   and si: "A |⇒ (S, U) onll Γ P"
   and sr: "(σ, p) ∈ reachable A S U"
   and sos: "((σ, p), a, (σ', q)) ∈ trans A"
   and "S σ σ' a"
   and "l' ∈ labels Γ p'
   and "p' ∈ terms Γ p"
   and "l ∈ labels Γ p"
   shows "P ((σ, l), a, (σ', l'))"
   proof
   from wf 'p' ∈ terms Γ p' 'l' ∈ labels Γ p' have "l ∈ labels Γ p"
   by (rule labels_sterms_labels)
   with si sr sos 'S σ σ' a' show "P ((σ, l), a, (σ', l'))" using '{l' ∈ labels Γ q' ..
   qed

lemma oseq_step_invariant_sterms:
assumes inv: "A |⇒ (S, U) onll Γ P"

and \( \text{wf: "wellformed } \Gamma \)"
and \( \text{sp: "trans } A = \text{oseq_sos } \Gamma \ i" \)
and \( \text{"} l' \in \text{labels } \Gamma \ q \)"
and \( \text{sr: "(} \sigma, p \text{) } \in \text{oreachable } \ A S U" \)
and \( \text{tr: "} ((\sigma, p'), a, (\sigma', q)) \in \text{trans } A " \)
and \( \text{"} S \sigma \sigma' a " \)
and \( \text{"} p' \in \text{terms } \Gamma \ p " \)
shows \( \forall l' \in \text{labels } \Gamma \ p'. \ P ((\sigma, l), a, (\sigma', l')) " \)

proof
from \( \text{assms}(3, 6) \) have "\((\sigma, p'), a, (\sigma', q)\) \in \text{oseq_sos } \Gamma \ i" by simp
hence 
\( ((\sigma, p), a, (\sigma', q)) \in \text{oseq_sos } \Gamma \ i " \)
using \( \text{wf } \ p' \in \text{terms } \Gamma \ p' " \)
with \( \text{assms}(3) \) have \( \text{trp: "} ((\sigma, p), a, (\sigma', q)) \in \text{trans } A " \)
fix \( l' \) assume 
\( l' \in \text{labels } \Gamma \ p'. \ P ((\sigma, l), a, (\sigma', l')) " 
by \( \text{(erule(? onll_ostep_invariant_sterms) } \)
qed

lemma \( \text{oseq_step_invariant_sterms_weaken} \):
assumes \( \text{inv: "} A \models_A (S, U \rightarrow) \ \text{onll } \Gamma \ P " \)
and \( \text{wf: "wellformed } \Gamma \)"
and \( \text{sp: "trans } A = \text{oseq_sos } \Gamma \ i" \)
and \( \text{"} l' \in \text{labels } \Gamma \ q \)"
and \( \text{sr: "} ((\sigma, p), a, (\sigma', q)) \in \text{trans } A " \)
and \( \text{"} S \sigma \sigma' a " \)
and \( \text{"} p' \in \text{terms } \Gamma \ p " \)
and \( \text{weakenS: "} \forall \sigma \sigma' a. S' \sigma \sigma' a \Longrightarrow S \sigma \sigma' a " \)
and \( \text{weakenU: "} \forall \sigma' \sigma. U' \sigma \sigma' \Longrightarrow U \sigma \sigma' " \)
shows \( \forall l' \in \text{labels } \Gamma \ p'. \ P ((\sigma, l), a, (\sigma', l')) " \)
by \( \text{(intro ballI) (rule onll_ostep_invariant_sterms [OF assms]) } \)
qed

lemma \( \text{onll_ostep_invariant_any_sterms} \):
assumes \( \text{wf: "wellformed } \Gamma " \)
and \( \text{si: "} A \models_A (S, U \rightarrow) \ \text{onll } \Gamma \ P " \)
and \( \text{sr: "} ((\sigma, p), a, (\sigma', q)) \in \text{trans } A " \)
and \( \text{"} l' \in \text{labels } \Gamma \ q \)"
shows \( \forall p' \in \text{terms } \Gamma \ p. \ P ((\sigma, l), a, (\sigma', l')) " \)
by \( \text{(intro ballI) (rule onll_ostep_invariant_sterms [OF assms]) } \)

lemma \( \text{oseq_step_invariant_ctermI } \) \( \text{[intro]} \):
assumes \( \text{wf: "wellformed } \Gamma " \)
and \( \text{cw: "control_within } \Gamma \ (\text{init } A) " \)
and \( \text{sl: "} \text{simple_labels } \Gamma " \)
and \( \text{sp: "trans } A = \text{oseq_sos } \Gamma \ i " \)
and \( \text{local: "} \forall p \ l \ a q l' \sigma' pp. \ [ p \in \text{terms } \Gamma; \ l \in \text{labels } \Gamma \ p; \ ((\sigma, p), a, (\sigma', q)) \in \text{oseq_sos } \Gamma \ i; \ ((\sigma, p), a, (\sigma', q)) \in \text{trans } A; \ l' \in \text{labels } \Gamma \ q; \ (\sigma, pp) \in \text{oreachable } A S U; \)
\[ p \in \text{terms } \Gamma \; pp; \]
\[ (\sigma', q) \in \text{oreachable } A \; S \; U; \]
\[ S \; \sigma \; \sigma' \; a \]
\[ ] \implies P ((\sigma, 1), a, (\sigma', 1')) \]

shows "\( A \mid_{\gamma} (S, U \rightarrow) \\text{onll } \Gamma \; p' \)"

proof

fix \( \sigma \; p \; l \; a \; \sigma' \; q \; l' \)
assume "\((\sigma, p) \in \text{oreachable } A \; S \; U\)"
and tr: "\(((\sigma, p), a, (\sigma', q)) \in \text{trans } A\)"
and "\( S \; \sigma \; \sigma' \; a \)"
and pl: "\( l \in \text{labels } \Gamma \; p \)"
and A5: "\( l' \in \text{labels } \Gamma \; q \)"

from this(2) and sp have "\(((\sigma, p), a, (\sigma', q)) \in \text{oseqp_sos } \Gamma \; i\) by simp
then obtain \( p' \) where "\( p' \in \text{terms } \Gamma \; p \)"
and A3: "\(((\sigma, p'), a, (\sigma', q)) \in \text{oseqp_sos } \Gamma \; i\)"
by (blast dest: otrans_from_terms [OF _ \_ wf])

from this(2) and sp have A4: "\(((\sigma, p'), a, (\sigma', q)) \in \text{trans } A\)"
by simp

from \( \sigma \; p' \in \text{terms } \Gamma \; p' \)
obtain \( pp \) where A6: "\((\sigma, pp) \in \text{oreachable } A \; S \; U\)"
and A7: "\( p' \in \text{terms } \Gamma \; pp \)"
by auto

from \( \sigma \; p \in \text{terms } \Gamma \; p \)
by (erule(2) ooreachable_local')

from \( \sigma \; p \in \text{terms } \Gamma \; p \)
by simp

lemma oseq_step_invariant_ctermI [intro]:
assumes wf: "\text{wellformed } \Gamma\"
and "\text{control_within } \Gamma \; (\text{init } A)\"
and "\text{simple_labels } \Gamma\"
and "\text{trans } A = \text{oseqp_sos } \Gamma \; i\"
and local: "\( \forall p \; l \; \sigma \; a \; q \; l' \; \sigma' \; pp \; pn. [ \]
\[ \begin{array}{l}
\text{wellformed } \Gamma; \\
p \in \text{terms } \Gamma \; (\Gamma \; pn); \\
\text{not_call } p; \\
l \in \text{labels } \Gamma \; p; \\
((\sigma, p), a, (\sigma', q)) \in \text{oseqp_sos } \Gamma \; i; \\
((\sigma, p), a, (\sigma', q)) \in \text{trans } A; \\
l' \in \text{labels } \Gamma \; q; \\
(\sigma, pp) \in \text{oreachable } A \; S \; U; \\
p \in \text{terms } \Gamma \; pp; \\
(\sigma', q) \in \text{oreachable } A \; S \; U; \\
S \; \sigma \; \sigma' \; a \\
\] \implies P ((\sigma, 1), a, (\sigma', 1'))\"
shows "\( A \mid_{\gamma} (S, U \rightarrow) \\text{onll } \Gamma \; p' \)"
using assms(1-4) proof (rule oseq_step_invariant_ctermI)
fix \( p \; l \; \sigma \; a \; q \; l' \; \sigma' \; pp \)
assume "\( p \in \text{terms } \Gamma\)"
and otherassms: "\( l \in \text{labels } \Gamma \; p\)"
"\((\sigma, p), a, (\sigma', q)) \in \text{oseqp_sos } \Gamma \; i\"
"\((\sigma, p), a, (\sigma', q)) \in \text{trans } A\"
"\( l' \in \text{labels } \Gamma \; q\)"
"\( (\sigma, pp) \in \text{oreachable } A \; S \; U\)"
"\( p \in \text{terms } \Gamma \; pp\)"
"\( (\sigma', q) \in \text{oreachable } A \; S \; U\)"
"\( S \; \sigma \; \sigma' \; a\)"
from this(1) obtain pn where "p ∈ ctermsl(Γ pn)"
and "not_call p"

unfolding cterms_def' [OF wf] by auto
with wf show "P (((σ, l), a, (σ', l')))"
using otherassms by (rule local)
qed

lemma open_seqp_action [elim]:
assumes "wellformed Γ"
and "((σ i, p), a, (σ' i, p')) ∈ seqp_sos Γ"
shows "((σ, p), a, (σ', p')) ∈ oseqp_sos Γ i"
proof -
from assms obtain ps where "ps ∈ sterms Γ p"
and "((σ i, ps), a, (σ' i, p')) ∈ seqp_sos Γ"
by - (drule trans_from_sterms, auto)
thus ?thesis
proof (induction p)
fix p1 p2
assume IH1: "[ ps ∈ sterms Γ p1; ((σ i, ps), a, (σ' i, p')) ∈ seqp_sos Γ ]
⇒ ((σ, p1), a, (σ', p')) ∈ oseqp_sos Γ i"
and IH2: "[ ps ∈ sterms Γ p2; ((σ i, ps), a, (σ' i, p')) ∈ seqp_sos Γ ]
⇒ ((σ, p2), a, (σ', p')) ∈ oseqp_sos Γ i"
and "ps ∈ sterms Γ (p1 ⊕ p2)"
and "((σ i, ps), a, (σ' i, p')) ∈ seqp_sos Γ"
with assms(1) show "((σ, p1 ⊕ p2), a, (σ', p')) ∈ oseqp_sos Γ i"
by simp (metis oseqp_sos.ochoiceT1 oseqp_sos.ochoiceT2)
next
fix l fip fmsg p1 p2
assume IH1: "[ ps ∈ sterms Γ p1; ((σ i, ps), a, (σ' i, p')) ∈ seqp_sos Γ ]
⇒ ((σ, p1), a, (σ', p')) ∈ oseqp_sos Γ i"
and IH2: "[ ps ∈ sterms Γ p2; ((σ i, ps), a, (σ' i, p')) ∈ seqp_sos Γ ]
⇒ ((σ, p2), a, (σ', p')) ∈ oseqp_sos Γ i"
and "ps ∈ sterms Γ ({l}unicast(fip, fmsg). p1 ⊕ p2)"
and "((σ i, ps), a, (σ' i, p')) ∈ seqp_sos Γ"
from this(3-4) have "((σ i, {l}unicast(fip, fmsg). p1 ⊕ p2), a, (σ' i, p')) ∈ seqp_sos Γ"
by simp
thus "((σ, {l}unicast(fip, fmsg). p1 ⊕ p2), a, (σ', p')) ∈ oseqp_sos Γ i"
proof (rule seqp_unicastTE)
assume "a = unicast (fip (σ i)) (fmsg (σ i))"
and "σ' i = σ i"
and "p' = p1"
thus ?thesis by auto
next
assume "a = ¬ unicast (fip (σ i))"
and "σ' i = σ i"
and "p' = p2"
thus ?thesis by auto
qed
next
fix p
assume "ps ∈ sterms Γ (call(p))"
and "((σ i, ps), a, (σ' i, p')) ∈ seqp_sos Γ"
with assms(1) have "((σ, ps), a, (σ', p')) ∈ oseqp_sos Γ i"
by (cases ps) auto
with assms(1) 'ps ∈ sterms Γ (call(p))' have "((σ, Γ p), a, (σ', p')) ∈ oseqp_sos Γ i"
by - (rule otrans_from_sterms', simp_all)
thus "((σ, call(p)), a, (σ', p')) ∈ oseqp_sos Γ i" by auto
qed auto
qed

end
theory DAWN_Convert
imports AWN_SOS_Labels AWN_Invariants
  OAWN_SOS OAWN_Invariants
begin

definition initiali :: "'i ⇒ ('g × 'l) set ⇒ ('g × 'l) set ⇒ bool"
  where "initiali i OI CI ≡ ({{(σ i, p)|σ p. (σ, p) ∈ OI} = CI}"

lemma initialiI [intro]:
  assumes OICI: "⋀σ p. (σ, p) ∈ OI =⇒ (σ i, p) ∈ CI"
  and CIOI: "⋀ξ p. ξ = σ i ∧ (ξ, p) ∈ CI =⇒ ∃σ. ξ = σ i ∧ (σ, p) ∈ OI"
  shows "initiali i OI CI"
  unfolding initiali_def by (intro set_eqI iffI) (auto elim!: OICI CIOI)

lemma open_from_initialiD [dest]:
  assumes "initiali i OI CI" and "(σ, p) ∈ OI"
  shows "∃ξ. σ i = ξ ∧ (ξ, p) ∈ CI"
  using assms unfolding initiali_def by auto

definition seql :: "'i ⇒ ('s × 'l) ⇒ bool" ⇒ (('i ⇒ 's) × 'l) ⇒ bool"
  where "seql i P ≡ (λ(σ i, p). P (σ i, p))"

lemma seqlI [intro]:
  "P (fst s i, snd s) =⇒ seql i P s"
  by (clarsimp simp: seql_def)

lemma same_seql [elim]:
  assumes "∀j ∈ {i}. σ' j = σ j"
  and "seql i P (σ', s)"
  shows "seql i P (σ, s)"
  using assms unfolding seql_def by (clarsimp)

lemma seqlsimp:
  "seql i P (σ, p) = P (σ i, p)"
  unfolding seql_def by simp

lemma other_steps_resp_local [intro!, simp]: "other_steps (other A I) I"
  by (clarsimp elim!: otherE)

lemma seql_onl_swap:
  "seql i (onl Γ P) = onl Γ (seql i P)"
  unfolding seql_def onl_def by simp

lemma oseqp_sos_resp_local_steps [intro!, simp]:
  fixes Γ :: "'p ⇒ ('s, 'm, 'p, 'l) seqp"
  shows "local_steps (oseqp_sos Γ i) {i}"
  proof
    fix σ σ' ζ ζ' :: "nat ⇒ 's" and s a s'
    assume tr: "((σ, s), a, σ', s') ∈ oseqp_sos Γ i"
    and ∀j∈{i}. ζ j = σ' j"
    thus "∃ζ'. ∀j∈{i}. ζ' j = σ' j ∧ ((ζ', s), a, (ζ', s')) ∈ oseqp_sos Γ i"
  proof induction
\[\sigma \cdot 1 \ms p\]

assume \"\sigma \cdot i = \sigma i\n
and \"\forall j \in i. \ \zeta j = \sigma j\n
hence \"\((\zeta, (1)\broadcast(ms).p), \broadcast (ms (\sigma i)), (\sigma', p)) \in \osq\sos \ \Gamma i\n
by (metis \obroadcastT \singletoniff)

with \'\forall j \in i. \ \zeta j = \sigma j\' show \"\exists \zeta'. \ ((\forall j \in i. \ \zeta j = \sigma j) \land

((\zeta, (1)\broadcast(ms).p), \broadcast (ms (\sigma i)), (\zeta', p)) \in \osq\sos \ \Gamma i\n
\] by blast

next

fix \sigma \cdot 1 \ p \ and \ \f : \ "'s set" \ and \ \sigma \cdot 1 \ p
assume \*: \ "\sigma \cdot i = \f (\sigma i)\n
and **: \ "\forall j \in i. \ \zeta j = \sigma j\n
hence \"\forall j \in i. \ \zeta (i := \f (\zeta i)) j = \sigma j\" by clarsimp

moreover from * **

have "((\zeta, (1)\receive(fmsg).p), \receive, \((\zeta := \f (\zeta i)), p)) \in \osq\sos \ \Gamma i\n
by (metis \fun_upd_same \osq\sos \ \Gamma i)

ultimately show \"\exists \zeta'. \ ((\forall j \in i. \ \zeta j = \sigma j) \land

((\zeta, (1)\receive(fmsg).p), \receive, (\zeta', p)) \in \osq\sos \ \Gamma i\n
\] by blast

next

fix g : \ "'s set" \ and \ \sigma \cdot 1 \ p
assume \*: \ "\sigma \cdot i \in g (\sigma i)\n
and **: \ "\forall j \in i. \ \zeta j = \sigma j\n
hence \"\forall j \in i. \ \zeta (i := g \zeta i) j = \sigma j\" by simp (metis \lifting \ \full_types \ \some_eq_ex)

moreover with * **

have "((\zeta, (1)\f p), \tau, (\zeta := \f (\zeta i)), p) \in \osq\sos \ \Gamma i\n
by (metis \guardT \ \step_seq\tau)

ultimately show \"\exists \zeta'. \ ((\forall j \in i. \ \zeta j = \sigma j) \land

((\zeta, (1)\f p), \tau, (\zeta', p)) \in \osq\sos \ \Gamma i\n
\] by blast

next

fix pn \ a \ \sigma \cdot p'
assume \"\((\sigma, \Gamma pn), a, (\sigma', p')) \in \osq\sos \ \Gamma i\n
and IH: \"\forall j \in i. \ \zeta j = \sigma j \ \Rightarrow \exists \zeta'. \ \ ((\forall j \in i. \ \zeta j = \sigma j) \land

((\zeta, \Gamma pn), a, (\zeta', p')) \in \osq\sos \ \Gamma i\n
\] and \"\forall j \in i. \ \zeta j = \sigma j\n
then obtain \zeta' where \"\forall j \in i. \ \zeta j = \sigma j\n
and "((\zeta, \Gamma pn), a, (\zeta', p')) \in \osq\sos \ \Gamma i\n
\] by blast

thus \"\exists \zeta'. \ ((\forall j \in i. \ \zeta j = \sigma j) \land

((\zeta, \call(pn)), a, (\zeta', p')) \in \osq\sos \ \Gamma i\n
\] by blast

next

fix q \ a \ \sigma \cdot p' \ q
assume \"\((\sigma, p), a, (\sigma', p')) \in \osq\sos \ \Gamma i\n
and \"\forall j \in i. \ \zeta j = \sigma j \ \Rightarrow \exists \zeta'. \ (\forall j \in i. \ \zeta j = \sigma j) \land

((\zeta, p), a, (\zeta', p')) \in \osq\sos \ \Gamma i\n
\] and \"\forall j \in i. \ \zeta j = \sigma j\n
then obtain \zeta' where \"\forall j \in i. \ \zeta j = \sigma j\n
and "((\zeta, p), a, (\zeta', p')) \in \osq\sos \ \Gamma i\n
\] by blast

thus \"\exists \zeta'. \ ((\forall j \in i. \ \zeta j = \sigma j) \land

((\zeta, p \oplus q), a, (\zeta', p')) \in \osq\sos \ \Gamma i\n
\] by blast

next

fix q \ a \ \sigma \cdot q' \ q'
assume \"\((\sigma, q), a, (\sigma', q')) \in \osq\sos \ \Gamma i\n
and \"\forall j \in i. \ \zeta j = \sigma j \ \Rightarrow \exists \zeta'. \ (\forall j \in i. \ \zeta j = \sigma j) \land

((\zeta, q), a, (\zeta', q')) \in \osq\sos \ \Gamma i\n
\]
∀j∈{i}. ζ j = σ j
then obtain ζ’ where ∀j∈{i}. ζ’ j = σ’ j
and (ζ, q), a, (ζ’, q’) ∈ oseqp_sos Γ i
by blast
thus "∃ζ’. (∀j∈{i}. ζ’ j = σ’ j) ∧ ((ζ, p ⊕ q), a, (ζ’, q’)) ∈ oseqp_sos Γ i"
by blast
qed (simp_all, (metis ogroupcastT ounicastT onotunicastT osendT odeliverT)+)

lemma oseqp_sos_subreachable [intro!, simp]:
assumes "trans OA = oseqp_sos Γ i"
shows "subreachable OA (other ANY {i}) {i}"
by rule (clarsimp simp add: assms(1))+

lemma oseq_step_is_seq_step:
fixes σ :: "ip ⇒ 's'
assumes "((σ, p), a :: 'm seq_action, (σ', p')) ∈ oseqp_sos Γ i" and "σ i = ξ"
shows "∃ξ'. σ' i = ξ' ∧ ((ξ, {l}broadcast(ms).p), broadcast (ms (σ i)), (ξ', p)) ∈ seqp_sos Γ" using assms
proof (induction)
fix σ σ' l ms p
assume "σ' i = σ i" and "σ i = ξ"
hence "σ' i = ξ" by simp
have "((ξ, {l}broadcast(ms).p), broadcast (ms ξ), (ξ, p)) ∈ seqp_sos Γ" by auto
with 'σ' i = ξ' and 'σ' i = ξ' show "∃ξ'. σ' i = ξ' ∧ ((ξ, (1)broadcast(ms).p), broadcast (ms (σ i)), (ξ', p)) ∈ seqp_sos Γ" by clarsimp
next
fix fmsg :: "'m ⇒ 's" and msg :: 'm and σ σ' p a
assume "σ' i = fmsg msg (σ i)" and "σ i = ξ"
have "((ξ, (1)receive(fmsg).p), receive msg, (fmsg msg ξ, p)) ∈ seqp_sos Γ" by auto
with 'σ' i = fmsg msg (σ i)' and 'σ' i = ξ'
and "∃ξ'. σ' i = ξ' ∧ ((ξ, (1)receive(fmsg).p), receive msg, (ξ, p)) ∈ seqp_sos Γ" by clarsimp
qed (simp_all, (metis assignT choiceT1 choiceT2 groupcastT guardT callT unicastT notunicastT sendT deliverT step_seq_tau)+)

lemma reachable_oseq_seqp_sos:
assumes "((σ, p), a :: 'm seq_action, (σ', p')) ∈ oseqp_sos Γ i" and "initiali i (init OA) (init A)
and spo: "trans OA = oseqp_sos Γ i" and sp: "trans A = seqp_sos Γ"
shows "∃ξ. σ i = ξ ∧ ((ξ, p), a, (ξ', p')) ∈ reachable A I"
using assms(1) proof (induction rule: reachable_pair_induct)
fix σ p
assume "((σ, p), a :: 'm seq_action, (σ', p')) ∈ reachable O A I" and "initiali i (init OA) (init A)"
and spo: "trans OA = oseqp_sos Γ i" and sp: "trans A = seqp_sos Γ"
and "((σ, p), a, (ξ', p')) ∈ reachable A I" and "ξ' ∈ reachable A I"
and "(ξ, p) ∈ reachable A I"
and "((σ, p), a, (ξ', p')) ∈ reachable A I"
and "ξ' ∈ reachable A I"
and "(ξ, p) ∈ reachable A I"
and "(ξ, p) ∈ reachable A I"
and "(ξ, p) ∈ reachable A I"
and "(ξ, p) ∈ reachable A I"
and "(ξ, p) ∈ reachable A I"
and "(ξ, p) ∈ reachable A I"
and "(ξ, p) ∈ reachable A I"
and "(ξ, p) ∈ reachable A I"
and "(ξ, p) ∈ reachable A I"
and "(ξ, p) ∈ reachable A I"
and "(ξ, p) ∈ reachable A I"
and "(ξ, p) ∈ reachable A I"
and "(ξ, p) ∈ reachable A I"
and "(ξ, p) ∈ reachable A I"
and "(ξ, p) ∈ reachable A I"
lemma reachable_oseq_seqp_sos':
assumes "s ∈ reachable OA I"
and "initial i (init OA) (init A)"
and "trans OA = oseqp_sos Γ i"
and "trans A = seqp_sos Γ"
shows "∃ξ. (fst s) i = ξ ∧ (ξ, snd s) ∈ reachable A I"
using assms
by (clarsimp)

Any invariant shown in the (simpler) closed semantics can be transferred to an invariant in the open semantics.

theorem open_seq_invariant [intro]:
assumes "A |= (I → P)"
and "initial i (init OA) (init A)"
and spo: "trans OA = oseqp_sos Γ i"
and sp: "trans A = seqp_sos Γ"
shows "OA |= (act I, other ANY {i} → (seql i P))"
proof -
  have "OA |= (I →) (seql i P)"
    proof (rule invariant_arbitraryI)
      fix s
      assume "s ∈ reachable OA I"
      with 'initial i (init OA) (init A)'
      obtain ξ where "(fst s) i = ξ"
        and "(ξ, snd s) ∈ reachable A I"
      by (auto dest: reachable_oseq_seqp_sos' [OF _ _ spo sp])
      with 'A |= (I →) P' have "P (ξ, snd s)" by auto
      with '(fst s) i = ξ' show "seql i P s" by auto
    qed
  moreover from spo have "subreachable OA (other ANY {i}) {i}" ..
  ultimately show ?thesis
  proof (rule open_closed_invariant)
    fix σ σ' s
    assume "∀j ∈ {i}. σ' j = σ j"
    and "seql i P (σ', s)"
    thus "seql i P (σ, s)" ..
  qed
  qed

definition
seql1 :: "'i ⇒ (((s × 'l) × 'a × ('s × 'l)) ⇒ bool) ⇒ (((i ⇒ 's) × 'l) × 'a × (((i ⇒ 's) × 'l)) ⇒ bool)" where 
"seql1 i P ≡ (λ((s, p), a, (σ', p')). P ((σ i, p), a, (σ' i, p')))"

lemma same_seqll [elim]:
assumes "∀j ∈ {i}. σ1' j = σ1 j"
and "∀j ∈ {i}. σ2' j = σ2 j"
and "seql1 i P ((σ1', s), a, (σ2', s'))"
shows "seql1 i P ((σ1, s), a, (σ2', s'))"
using assms unfolding seqll_def by (clarsimp)

lemma seqllI [intro]:
assumes "P ((σ i, p), a, (σ' i, p'))"
shows "seqll i P ((σ, p), a, (σ', p'))"
using assms unfolding seqll_def by simp

lemma seqllD [dest]:
assumes "seqll i P ((σ, p), a, (σ', p'))"
shows "P ((σ i, p), a, (σ' i, p'))"
using assms unfolding seqll_def by simp

lemma seqllsimp:
"seqll i P ((σ, p), a, (σ', p')) = P ((σ i, p), a, (σ' i, p'))"
unfolding seqll_def by simp

lemma seqll_onll_swap:
"seqll i (onll Γ P) = onll Γ (seqll i P)"
unfolding seqll_def onll_def by simp

theorem open_seq_step_invariant [intro]:
assumes "A |=A (I ->) P"
and "initiali i (init OA) (init A)"
and spo: "trans OA = oseqp_sos Γ i"
and sp: "trans A = seqp_sos Γ"
shows "OA |=A (act I, other ANY {i} ->) (seqll i P)"
proof -
have "OA |=A (I ->) (seqll i P)"
proof (rule step_invariant_arbitraryI)
  fix σ p a σ' p'
  assume or: "((σ, p) ∈ reachable OA I)"
  and ctr: "((σ, p), a, (σ', p')) ∈ trans OA"
  and "I a"
  from or 'initiali i (init OA) (init A)' spo sp obtain ξ where "σ i = ξ"
  and cr: "((ξ, p) ∈ reachable A I)"
    by - (drule(3) reachable_oseq_seqp_sos', auto)
  from ctr and spo have "((σ, p), a, (σ', p')) ∈ oseqp_sos Γ i" by simp
    with "σ i = ξ" obtain ξ' where "σ' i = ξ'"
    and cr: "((ξ, p), a, (ξ', p')) ∈ seqp_sos Γ"
    by (auto dest!: oseq_step_is_seq_step)
  with sp have "((ξ, p), a, (ξ', p')) ∈ trans A" by simp
    with 'A |=A (I ->) P' cr have "P ((ξ, p), a, (ξ', p'))" using 'I a' ..
    with 'σ i = ξ' and 'σ' i = ξ' have "P ((σ i, p), a, (σ' i, p'))" by simp
    thus "seqll i P ((σ, p), a, (σ', p'))" ..
  qed
moreover from spo have "local_steps (trans OA) {i}" by simp
moreover have "other_steps (other ANY {i}) {i}" ..
ultimately show ?thesis
proof (rule open_closed_step_invariant)
  fix σ ζ a σ' ζ' s s'
  assume "∀ j ∈ {i}. σ j = ζ j"
    and "∀ j ∈ {i}. σ' j = ζ' j"
    and "seqll i P ((σ, s), a, (σ', s'))"
    thus "seqll i P ((ζ, s), a, (ζ', s'))" ..
  qed
qed

end

22 Model the standard queuing model

theory Qmsg
imports AWN_SOS_Labels AWN_Invariants
begin

Define the queue process

fun Γ_QMSG :: "(('m::msg) list, 'm, unit, unit label) seqp_env"
where
"\( q_{\text{QMSG}} \) is labelled (receive(\( \lambda m \) msgs. msgs @ [msg]). \text{call}())
\[\oplus \{\text{msgs. msgs} \neq []\}
\text{send}(\( \lambda m \) msgs. hd msgs). \text{[msgs. tl msgs] call(())})
\text{receive}(\( \lambda m \) msgs. msgs @ [msg]). \text{call(())})"

**definition** \( \sigma_{\text{QMSG}} :: (\text{'m::msg} \text{ list} \times \text{('m list, 'm, unit, unit label) seqp}) \text{ set} \)

**where** \( \sigma_{\text{QMSG}} \equiv \{([], \text{\( \Gamma_{\text{QMSG}} \)})} \)

**abbreviation** \( q_{\text{msg}} :: (\text{'m::msg} \text{ list} \times \text{('m list, 'm, unit, unit label) seqp, 'm seq_action}) \text{ automaton} \)

**where** \( q_{\text{msg}} \equiv (\text{init} = \sigma_{\text{QMSG}}, \text{trans} = \text{seqp sos} \text{\( \Gamma_{\text{QMSG}} \)})" \)

**declare** \( \Gamma_{\text{QMSG}} \) \_\text{simps} [simp del, code del]

**lemmas** \( \Gamma_{\text{QMSG}} \) \_\text{simps} [simp, code] = \( \Gamma_{\text{QMSG}} \) \_\text{simps} [simplified]

**lemma** \( \sigma_{\text{QMSG}} \_\text{not_empty} [simp, intro]: \( \sigma_{\text{QMSG}} \neq \{\} \)

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

**lemma** \( \sigma_{\text{QMSG}} \_\text{exists} [simp]: \exists q_{\text{msg}} q. (q_{\text{msg}}, q) \in \sigma_{\text{QMSG}} \)

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

**lemma** \( q_{\text{msg}} \_\text{wf} [simp]: \text{wellformed} \text{\( \Gamma_{\text{QMSG}} \)}

**by** (rule \text{wf_no_direct_calls}) auto

**lemmas** \( q_{\text{msg}} \_\text{labels_not_empty} [simp] = \text{labels_not_empty} [\text{OF} \ q_{\text{msg}} \_\text{wf}] \)

**lemma** \( q_{\text{msg}} \_\text{control_within} [simp]: \text{control_within} \text{\( \Gamma_{\text{QMSG}} \) \text{\( \text{init} \ q_{\text{msg}} \)}}

**unfolding** \( \sigma_{\text{QMSG}} \) \_\text{def} by (rule \text{control_withinI}) (auto simp del: \( \Gamma_{\text{QMSG}} \) \_\text{simps})

**lemma** \( q_{\text{msg}} \_\text{simple_labels} [simp]: \text{simple_labels} \text{\( \Gamma_{\text{QMSG}} \)}

**unfolding** \( \text{simple_labels_def} \) by auto

**lemma** \( q_{\text{msg}} \_\text{trans} : \text{trans} \ q_{\text{msg}} = \text{seqp sos} \text{\( \Gamma_{\text{QMSG}} \)}

**by** simp

**lemma** \( \sigma_{\text{QMSG}} \_\text{labels} [simp]: (\xi, q) \in \sigma_{\text{QMSG}} \Rightarrow \text{labels} \text{\( \Gamma_{\text{QMSG}} \) q = \{()-:0\}}

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

**lemma** \( q_{\text{msg}} \_\text{proc_cases} [dest]:

**fixes** p n

**shows** "p \in \text{ctermsl} \text{\( \Gamma_{\text{QMSG}} \) pn} \Rightarrow p \in \text{ctermsl} \text{\( \Gamma_{\text{QMSG}} \)} (\))"

**using** \text{assms} by simp

**declare**

\( \Gamma_{\text{QMSG}} \) \_\text{simps} [cterms_env]

\( q_{\text{msg}} \_\text{proc_cases} [ctermsl_cases]

seq\_\text{invariant}\_\text{ctermsl} [\text{OF} \ q_{\text{msg}} \_\text{wf} \ q_{\text{msg}} \_\text{control_within} \ q_{\text{msg}} \_\text{simple_labels} \ q_{\text{msg}} \_\text{trans}, \text{cterms_intros}]

seq\_\text{step}\_\text{invariant}\_\text{ctermslI} [\text{OF} \ q_{\text{msg}} \_\text{wf} \ q_{\text{msg}} \_\text{control_within} \ q_{\text{msg}} \_\text{simple_labels} \ q_{\text{msg}} \_\text{trans}, \text{cterms_intros}]

end

23 Lifting rules for parallel compositions with QMSG
lemma qmsg_no_change_on_send_or_receive:
  fixes σ s a σ' s'
  assumes "((σ, s), a, (σ', s')) ∈ oparp_sos i (oseqp_sos Γ i) (seqp_sos ΓMSG)
  and "a ≠ τ"
  shows "σ' i = σ i"
proof -
  from assms(1) obtain p q p' q'
  where "((σ, (p, q)), a, (σ', (p', q'))) ∈ oparp_sos i (oseqp_sos Γ i) (seqp_sos ΓMSG)
  by (cases s, cases s', simp)
  thus ?thesis
  proof
    assume "((σ, p), a, (σ', p')) ∈ oseqp_sos Γ i"
    and "⋀m. a ≠ receive m"
    with 'a ≠ τ' show "σ' i = σ i"
    by - (drule oseq_no_change_on_send, cases a, auto)
  next
    assume "(q, a, q') ∈ seqp_sos Γ MSG"
    and "σ' i = σ i"
    thus "σ' i = σ i" by simp
  next
    assume "a = τ" with 'a ≠ τ'
    show ?thesis
    by auto
  qed
  qed

lemma qmsg_msgs_not_empty:
  "qmsg ||= onl ΓMSG (λ(msgs, l). l = ()-:1 → msgs ≠ [])"
by inv_cterms

lemma qmsg_send_from_queue:
  "qmsg ||= A (λ((msgs, q), a, _). sendmsg (λm. m∈set msgs) a)"
proof -
  have "qmsg ||= onl ΓMSG (λ((msgs, _), a, _). sendmsg (λm. m∈set msgs) a)"
by (inv_cterms inv add: onl_invariant_sterms [OF qmsg_wf qmsg_msgs_not_empty])
  thus ?thesis
  by (rule step_invariant_weakenE) (auto dest!: onllD)
  qed

lemma qmsg_queue_contents:
  "qmsg ||= A (λ((msgs, q), a, (msgs', q')). case a of receive m ⇒ msgs' = msgs @ [m] | _ ⇒ set msgs' ⊆ set msgs)"
proof -
  have "qmsg ||= onl ΓMSG (λ((msgs, q), a, (msgs', q')). case a of receive m ⇒ msgs' = msgs @ [m] | _ ⇒ set msgs' ⊆ set msgs)"
by (inv_cterms) (clarsimp elim!: in_set_tl)
  thus ?thesis
  by (rule step_invariant_weakenE) (auto dest!: onllD)
  qed

lemma qmsg_send_receive_or_tau:
  "qmsg ||= A (λ(_, a, _). ∃m. a = send m ∨ a = receive m ∨ a = τ)"
proof -
  have "qmsg ||= onl ΓMSG (λ(_, a, _). ∃m. a = send m ∨ a = receive m ∨ a = τ)"
  by inv_cterms
  thus ?thesis
  by rule (auto dest!: onllD)
  qed
lemma par_qmsg_oracleable:
assumes ""(σ, ζ) ∈ oracleable (A ⟨⟨ i qmsg ⟩⟩ (otherwith S {i} (orecvmsg R)) (other U {i}))"
(is ". ∈ oracleable . ?owS .")
and pив: "A |=A (otherwith S {i} (orecvmsg R), other U {i} →)
globala (λ(σ, ., σ'). U (σ i) (σ' i))"
and ustutter: "\A ξ. U ξ ξ"
and sgieus: "\A σ σ'. S ξ ξ' \⇒ U ξ ξ'"
and upreservesq: "\A σ σ m. [ \V j. U (σ j) (σ' j); R σ m ] \⇒ R σ' m"
shows 
"(σ, fst pq) ∈ oracleable A ?owS (other U {i})
∧ snd ζ ∈ reachable qmsg (recvmsg (R σ))
∧ (\V m∈set (fst (snd pq)). R σ m)"
using assms(1)
proof (induction rule: orachable_pair_induct)
fix σ pq
assume 
"(σ, pq) ∈ init (A ⟨⟨ i qmsg ⟩⟩)
then obtain p ms q p' ms' q'
where "pq = (p, (ms, q))"
and "(σ, p) ∈ init A"
and "(ms, q) ∈ init qmsg"
by (clarsimp simp del: Γ QM SG _simps)
from this(2) have "(σ, pq) ∈ orachable A ?owS (other U {i})" ..
moreover have '(ms, q) ∈ init qmsg' have "ms = []" unfolding σQM_def by simp
ultimately show "(σ, fst pq) ∈ oracleable A ?owS (other U {i})
∧ snd pq ∈ reachable qmsg (recvmsg (R σ))
∧ (\V m∈set (fst (snd pq)). R σ m)" using 'pq = (p, (ms, q))' by simp
next
note Γ QM SG _simps [simp del]
case (other σ pq σ')
herefore "(σ, pq) ∈ oracleable A ?owS (other U {i})" and "other U {i} σ σ'"
and qr: "snd pq ∈ reachable qmsg (recvmsg (R σ))" and "\V m∈set (fst (snd pq)). R σ m"
by simp_all
from 'other U {i} σ σ' and ustutter have "\V j. U (σ j) (σ' j)"
by (clarsimp elim!: otherE) metis
from 'other U {i} σ σ' have "(σ, fst pq) ∈ oracleable A ?owS (other U {i})" ..
moreover have "\V m∈set (fst (snd pq)). R σ m" proof
fix m assume "m ∈ set (fst (snd pq))" with '∀ m∈set (fst (snd pq)). R σ m' have "R σ m" ..
with '∀ j. U (σ j) (σ' j)' show "Rσ m" by (rule upreservesq) qed
moreover from qr have "snd pq ∈ reachable qmsg (recvmsg (R σ'))" proof
fix a assume "recvmsg (R σ) a"
thus "recvmsg (R σ') a" proof (rule recvmsgE [where R=R])
fix m assume "R σ m"
with '∀ j. U (σ j) (σ' j)' show "R σ m" by (rule upreservesq) qed
qed
ultimately show ?case using qr by simp
next
case (local σ pq σ' pq' a)
obtain p ms q p' ms' q' where "pq = (p, (ms, q))"
and "pq' = (p', (ms', q'))"
by (cases pq, cases pq') metis
with local.hyps local.IH
have pqtr: "((σ, (p, (ms, q))), a, (σ', (p', (ms', q'))))
138
\[ \in \text{oparp sos} \ i \ (\text{trans} \ A) \ (\text{seqp sos} \ \Gamma_{QMSG}) \]

and por: "(σ, p) \in \text{oreachable} \ A ?\text{owS} \ (\text{other} \ U \ \{i\})"

and qr: "(ms, q) \in \text{reachable} \ qmsg \ (\text{recvmsg} \ (R \ \sigma))"

and "\forall m \in \text{set} \ ms. \ R \ \sigma \ m"

and "?\text{owS} \ \sigma \ \sigma' \ a"

by (simp_all del: \Gamma_{QMSG}_{\text{simp}})

from "?\text{owS} \ \sigma \ \sigma' \ a' \ have \ "\forall j. j \neq i \rightarrow S (\sigma \ j) (\sigma' \ j)"

by (clarsimp dest!: otherwith_syncD)

with sgivesu have "\forall j. j \neq i \rightarrow U (\sigma \ j) (\sigma' \ j)" by simp

from "?\text{owS} \ \sigma \ \sigma' \ a' \ have \ "\text{orecvmsg} \ R \ \sigma \ a" by (rule otherwithE) hence "\text{recvmsg} (R \ \sigma) \ a" ..

from pqtr have "(\sigma', p') \in \text{oreachable} \ A ?\text{owS} \ (\text{other} \ U \ \{i\})

\land (ms', q') \in \text{reachable} \ qmsg \ (\text{recvmsg} \ (R \ \sigma'))

\land (\forall m \in \text{set} \ ms'. \ R \ \sigma' \ m)"

proof

assume "((\sigma, p), a, (\sigma', p')) \in \text{trans} \ A"

and "\forall m. \ a \neq \text{receive} \ m"

and "(ms', q') = (ms, q)"

from this(1) have ptr: "((\sigma, p), a, (\sigma', p')) \in \text{trans} \ A" by simp

with pinv por and "?\text{owS} \ \sigma \ \sigma' \ a' \ have \ "\U (\sigma \ i) (\sigma' \ i)"

by (auto dest!: ostep_invariantD)

with "\forall j. j \neq i \rightarrow \U (\sigma \ j) (\sigma' \ j)" have "\U (\sigma \ j) (\sigma' \ j)" by auto

hence \text{recvmsg}': "\forall a. \ \text{recvmsg} (R \ \sigma) \ a \Rightarrow \text{recvmsg} (R \ \sigma') \ a" by (auto elim!: recvmsgE [where R=R] upreservesq)

from por ptr '?\text{owS} \ \sigma \ \sigma' \ a' have "(\sigma', p') \in \text{oreachable} \ A ?\text{owS} \ (\text{other} \ U \ \{i\})"

by -(rule oreachable_local')

moreover have "(ms', q') \in \text{reachable} \ qmsg \ (\text{recvmsg} \ (R \ \sigma'))"

proof -

from qr and "(ms', q') = (ms, q)"'

have "(ms', q') \in \text{reachable} \ qmsg \ (\text{recvmsg} \ (R \ \sigma'))" by simp

thus \text{thesis} by (rule reachable_weakenE) (erule recvmsg')

qed

moreover have "\forall m \in \text{set} \ ms'. \ R \ \sigma' \ m"

proof

fix m

assume "m \in \text{set} \ ms'"

with "(ms', q') = (ms, q)" have "m \in \text{set} \ ms" by simp

with "\forall m \in \text{set} \ ms. \ R \ \sigma \ m" ..

with "\forall j. \ U (\sigma \ j) (\sigma' \ j)" show "R \ \sigma' \ m"

by (rule upreservesq)

qed

ultimately show "(\sigma', p') \in \text{oreachable} \ A ?\text{owS} \ (\text{other} \ U \ \{i\})

\land (ms', q') \in \text{reachable} \ qmsg \ (\text{recvmsg} \ (R \ \sigma'))

\land (\forall m \in \text{set} \ ms'. \ R \ \sigma' \ m)"

next

assume qtr: "((ms, q), a, (ms', q')) \in \text{seqp sos} \ \Gamma_{QMSG}"

and "\forall m. \ a \neq \text{send} \ m"

and "p' = p"

and "\sigma' \ i = \sigma \ i"

from this(4) and "\forall \xi. \ U \ \xi \ \xi'" have "\U (\sigma \ i) (\sigma' \ i)" by simp

with "\forall j. j \neq i \rightarrow U (\sigma \ j) (\sigma' \ j)" have "\forall j. U (\sigma \ j) (\sigma' \ j)" by auto

hence \text{recvmsg}': "\forall a. \ \text{recvmsg} (R \ \sigma) \ a \Rightarrow \text{recvmsg} (R \ \sigma') \ a" by (auto elim!: recvmsgE [where R=R] upreservesq)

139
from qt have tqtr: "((ms, q), a, (ms', q')) ∈ trans qmsg" by simp

from ∀ j. U (σ j) (σ j)' and 'σ' i = σ i' have "other U {i} σ σ'" by auto
with por and 'p' = p'
  have "(σ', p') ∈ or reachable A ?owS (other U {i})"
  by (auto dest: reachable_other)
moreover have "(ms', q') ∈ reachable qmsg (recvmsg (R σ'))"
proof (rule reachable_weakenE [where P="recvmsg (R σ')"])
  from qr tqtr 'recvmsg (R σ) a' show "(ms', q') ∈ reachable qmsg (recvmsg (R σ'))" ..
qed (rule recvmsg')

moreover have "∀ m∈set ms'. R σ m"
proof
  fix m
  assume "m ∈ set ms'"
  moreover have "case a of receive m ⇒ ms' = ms @ [m] | _ ⇒ set ms' ⊆ set ms"
  proof
    - from qr have "(ms, q) ∈ reachable qmsg TT" ..
    thus ?thesis using tqtr
      by (auto dest!: step_invariantD [OF qmsg_queue_contents])
  qed
ultimately have "R σ m" using '∀ m∈set ms. R σ m' and 'orecvmsg R σ a'
  by (cases a) auto
with '∀ j. j ≠ i −→ S (σ j) (σ j)'
  show "R σ' m"
  by (rule upreservesq)
qed

ultimately show "(σ', p') ∈ or reachable A ?owS (other U {i})
  ∧ (ms', q') ∈ reachable qmsg (recvmsg (R σ'))
  ∧ (∀ m∈set ms'. R σ m)" by simp

next
fix m
assume "a = τ"
and "((σ, p), receive m, (σ', p')) ∈ trans A"
and "((ms, q), send m, (ms', q')) ∈ seqp sos Γ QMSG"
from this(2-3) have ptr: "((σ, p), receive m, (σ', p')) ∈ trans A"
  and qtr: "((ms, q), send m, (ms', q')) ∈ trans qmsg" by simp_all
from qr have "(ms, q) ∈ reachable qmsg TT" ..
with qt have "m ∈ set ms"
  by (auto dest!: step_invariantD [OF qmsg_send_from_queue])
with '∀ m∈set ms. R σ m' have "R σ m" ..
hence "orecvmsg R σ (receive m)" by simp

with '∀ j. j ≠ i −→ S (σ j) (σ j)'
  have "?owS σ σ' (receive m)"
  by (auto intro!: otherwithI)
with pinv por ptr have "U (σ i) (σ i)"
  by (auto dest!: ostep_invariantD)
with '∀ j. j ≠ i −→ U (σ j) (σ j)'
  have "∀ j. U (σ j) (σ j)" by auto
hence recvmsg': "\(∀ a. recvmsg (R σ) a \Rightarrow recvmsg (R σ') a\)"
  by (auto elim!: recvmsgE [where R=R] upreservesq)
from por ptr have "(σ', p') ∈ or reachable A ?owS (other U {i})"
  using "?owS σ σ' (receive m)" by - (erule(i) orreachable_local, simp)
moreover have "(ms', q') ∈ reachable qmsg (recvmsg (R σ'))"
proof (rule reachable_weakenE [where P="recvmsg (R σ')"])
  have "recvmsg (R σ) (send m)" by simp
  with qr qtr show "(ms', q') ∈ reachable qmsg (recvmsg (R σ'))" ..
qed (rule recvmsg')
moreover have "∀m∈set ms'. R σ' m"
proof
  fix m
  assume "m ∈ set ms'"
  moreover have "set ms' ⊆ set ms"
  proof -
  from qr have "(ms, q) ∈ reachable qmsg TT" ..
  thus thesis using qtr
  by (auto dest!: step_invariantD [OF qmsg_queue_contents])
  qed
ultimately have "R σ m" using '∀m∈set ms. R σ m' by auto
with '∀j. U (σ j) (σ' j)' show "R σ' m"
  by (rule upreserves)
  qed

ultimately show "(σ', p') ∈ oreachable A ?owS (other U {i})
∧ (ms', q') ∈ reachable qmsg (recvmsg (R σ'))
∧ (∀m∈set ms'. R σ' m)" by simp
  qed
with 'pq = (p, (ms, q))' and 'pq' = (p', (ms', q'))' show ?case
  by (simp_all del: _QMSG_simps)
  qed

lemma par_qmsg_oreachable_statelessasm:
  assumes "(σ, ξ) ∈ oreachable (A (λ i qmsg)
  (λσ _ . orevcmsg (λ_. R) σ) (other (λ_ _. True) {i}))"
and u stutter: "∀ξ'. U ξ ξ'"
shows "(σ, fst ξ) ∈ oreachable A (λσ _ . orevcmsg (λ_. R) σ) (other (λ_ _. True) {i})
∧ snd ξ ∈ reachable qmsg (recvmsg R)
∧ (∀m∈set (fst (snd ξ))). R m)"
proof -
  from assms(1)
  have "(σ, ξ) ∈ oreachable (A (λ i qmsg)
  (otherwith (λ_ _. True) {i} (orevcmsg (λ_. R)))
  (other (λ_ _. True) {i})))" by auto
moreover
  have "A |=A (otherwith (λ_ _. True) {i} (orevcmsg (λ_. R)),
  other (λ_ _. True) {i} →) globala (λ(σ, _, σ'). True)"
    by auto
ultimately
  obtain "(σ, fst ξ) ∈ oreachable A
  (otherwith (λ_ _. True) {i} (orevcmsg (λ_. R))) (other (λ_ _. True) {i})"
    and *: "snd ξ ∈ reachable qmsg (recvmsg R)"
    and **: "(∀m∈set (fst (snd ξ))). R m)"
      by (auto dest!: par_qmsg_oreachable)
  from this(1)
  have "(σ, fst ξ) ∈ oreachable A (λσ _ . orevcmsg (λ_. R) σ) (other (λ_ _. True) {i})"
    by rule auto
  thus thesis using * ** by simp
  qed

lemma lift_into_qmsg:
  assumes "A |= (otherwith S {i} (orevcmsg R), other U {i} →) global P"
  and "∀ξ. U ξ ξ'"
  and "∀ξ'. S ξ ξ' → U ξ ξ'"
  and "∀σ σ' m. J j. U (σ j) (σ' j); R σ m → R σ' m"
  and "A |=A (otherwith S {i} (orevcmsg R), other U {i} →)
  globala (λ(σ, _, σ'). U (σ i) (σ' i))"
shows "A (λ i qmsg |= (otherwith S {i} (orevcmsg R), other U {i} →) global P"
proof (rule invariant_oreachable)
  fix σ ξ
  assume "(σ, ξ) ∈ oreachable (A (λ i qmsg) (otherwith S {i} (orevcmsg R)) (other U {i}))"
  then obtain s where "(σ, s) ∈ oreachable A (otherwith S {i} (orevcmsg R)) (other U {i})"
    by (auto dest!: par_qmsg_oreachable [OF _ assms(5,2-4)])
with \(\text{assms(1)}\) show "global \(P (\sigma, \zeta)\)"
by (auto dest: oinvariant_weakenD [OF \text{assms(1)}])

qed

lemma lift_step_into_qmsg:
assumes inv: "A \models_A \text{(otherwise with S \{i\} (orevmsg R), other U \{i\} \to) globala P}"
and ustutter: "\(\forall \xi. \ U \xi \xi\)"
and sgivesu: "\(\forall \xi \xi'. \ S \xi \xi' \to U \xi \xi'\)"
and upreserves: "\(\forall \sigma \sigma'. \ m. \ [\ \forall j. \ U (\sigma j) (\sigma' j); \ R \sigma m \] \to R \sigma' m\)"
and self_sync: "A \models_A \text{(otherwise with S \{i\} (orevmsg R), other U \{i\} \to) globala (\lambda(\sigma, _, \sigma'). \ U (\sigma i) (\sigma' i))}"
and recv_stutter: "\(\forall \sigma \sigma'. \ m. \ [\ \forall j. \ U (\sigma j) (\sigma' j); \ \sigma' i = \sigma i \] \to P (\sigma, receive m, \sigma')\)"
and recv_right: "\(\forall \sigma \sigma'. \ P (\sigma, receive m, \sigma') \to P (\sigma, \tau, \sigma')\)"

shows "A \models_A \text{(otherwise with S \{i\} (orevmsg R), other U \{i\} \to) globala P}"
(is "\_ \models_A \text{(?ows, ?U \to \_)}")

proof (rule ostep.InvariantI)
  fix \(\sigma \in a\) \(\zeta'\)
  assume or: "\((\sigma, \zeta) \in \text{oreachable (A \{i\} qmsg \?ows \?U)}\)"
  and \(\text{otr}': "\((\sigma, \zeta), a, (\sigma', \zeta')\) \in \text{trans (A \{i\} qmsg)}"
  and "\?ows \sigma \sigma' a"
  from this(2) have "\((\sigma, \zeta), a, (\sigma', \zeta')\) \in \text{oparp_sos i (trans A) (seqp_sos \Gamma_{QMSG})}"
  by simp
  then obtain \(s \text{ msgs q s' msgs' q'}\)
    where "\(\sigma = (s, (\text{msgs}, q'))\)" "\(\zeta' = (s', (\text{msgs'}, q'))\)"
    and "\((\sigma, (s, (\text{msgs}, q))), a, (\sigma', (s', (\text{msgs'}, q'))))\)
    \(\in \text{oparp_sos i (trans A) (seqp_sos \Gamma_{QMSG})}"
    by (metis prod_cases3)
  from this(1-2) and or
  obtain "\((\sigma, s) \in \text{oreachable A ?ows ?U}\)"
    "\((\text{msgs}, q) \in \text{reachable qmsg (recvmsg (R \sigma))}\)"
    "\(\forall m \in \text{set msgs. R} \sigma m\)"
  by (auto dest: par_qmsgoreachable [OF _ self_sync ustutter sgivesu] elim!: upreserves)
  from \(\text{otr}'\) "\(\zeta = (s, (\text{msgs}, q))\)" "\(\zeta' = (s', (\text{msgs'}, q'))\)"
  have "\((\sigma, (s, (\text{msgs}, q))), a, (\sigma', (s', (\text{msgs'}, q'))))\)
    \(\in \text{oparp_sos i (trans A) (seqp_sos \Gamma_{QMSG})}"
  by simp
  hence "globala P ((\sigma, s), a, (\sigma', s'))"
  proof
    assume "\((\sigma, s), a, (\sigma', s')\) \in \text{trans A}\)
  with "\((\sigma, s) \in \text{oreachable A ?ows ?U}\)"
    show "globala P ((\sigma, s), a, (\sigma', s'))"
      using "\?ows \sigma \sigma' a'\) by (rule ostep.InvariantD [OF inv])
  next
    assume "\((\text{msgs}, q), a, (\text{msgs'}, q')\) \in \text{seqp_sos \Gamma_{QMSG}}"
    and "\(\forall m. \ a \not\in \text{send m}\)"
    and "\(\sigma' i = \sigma i\)"
  from this(3) and ustutter have "U (\sigma i) (\sigma' i)" by simp
  with "\?ows \sigma \sigma' a' and sgivesu have "\(\forall j. \ U (\sigma j) (\sigma' j)\)"
  by (clarsimp dest!: otherwith_syncD) metis
  moreover have "\(\exists m. \ a = \text{receive m} \lor (a = \tau)\)"
  proof -
    from "\(\text{msgs}, q) \in \text{reachable qmsg (recvmsg (R \sigma))}\)"
    have "\(\text{msgs}, q) \in \text{reachable qmsg TT}\) ..
  moreover from "\((\text{msgs}, q), a, (\text{msgs'}, q')\) \in \text{seqp_sos \Gamma_{QMSG'}}\)"
    have "\((\text{msgs}, q), a, (\text{msgs'}, q')\) \in \text{trans qmsg}\) by simp
  ultimately show ?thesis
    using "\(\forall m. \ a \not\in \text{send m}\)"
  by (auto dest!: step.InvariantD [OF \text{qmsg_send_receive_or_tau}])
  qed
  ultimately show "globala P ((\sigma, s), a, (\sigma', s'))"
  using "\(\sigma' i = \sigma i\)"
  by simp (metis receive_right recv_stutter step_seq_tau)

142
next
  assume "a = τ"
  and "((σ, s), receive m, (σ', s')) ∈ trans A"
  and "((msgs, q), send m, (msgs', q')) ∈ seq_sos Γ_{MSG}"

from '(msgs, q) ∈ reachable qmsg (recvmsg (R σ))'
  have "(msgs, q) ∈ reachable qmsg TT" ..
moreover from '((msgs, q), send m, (msgs', q')) ∈ seq_sos Γ_{MSG}''
  have "((msgs, q), send m, (msgs', q')) ∈ trans qmsg" by simp
ultimately have "msg ∈ set msgs"
  by (auto dest!: step_invariantD [OF qmsg_send_from_queue])

with '∀ m ∈ set msgs. R σ m' have "R σ m" ..
with '?(ωs σ σ') a' have "?ωs σ σ' (receive m)"
  by (auto dest!: otherwith_syncD)

hence "P (σ, receive m, σ')" by (rule receive_right)
  with 'a = τ' show "globala P (σ, a, (σ', s'))" by simp
qed

with 'ζ = (s, (msgs, q))' and 'ζ' = (s', (msgs', q'))' show "globala P (σ, ζ, a, (σ', ζ'))" by simp
qed

lemma lift_step_into_qmsg_statelessassm:
  assumes A: "A |− A (λσ _. orecvmsg (λ _, R) σ, other (λ _ _. True) {i} →) globala P"
  and "∀ σ σ'. σ' i = σ i → P (σ, receive m, σ')"
  and "∀ σ σ'. P (σ, receive m, σ') → P (σ, τ, σ')"
  shows "A (i qmsg |− A (λσ _. orecvmsg (λ _, R) σ, other (λ _ _. True) {i} →) globala P"
proof -
  from assms(1) have "A |− A (otherwith (λ _ _. True) {i} (orecvmsg (λ _, R)),
    other (λ _ _. True) {i} →) globala P"
  by rule auto
  hence "A (i qmsg |− A (otherwith (λ _ _. True) {i} (orecvmsg (λ _, R)),
    other (λ _ _. True) {i} →) globala P"
  by (rule lift_step_into_qmsg)
  (auto elim!: assms(2-3) simp del: step_seq_tau)
  thus ?thesis by rule auto
qed

end

24 Transfer open results onto closed models

theory OClosed_Transfer
imports Closed OClosed_Lifting
begin

locale openproc =
  fixes np :: "ip ⇒ ('s, ('m::msg) seq_action) automaton" 
  and onp :: "ip ⇒ ((ip ⇒ 'g) × 'l, 'm seq_action) automaton" 
  and sr :: "'s ⇒ ('g × 'l)"
assumes init: "{ (σ, ζ) | σ ζ s. s ∈ init (np i) 
  ∧ (σ i, ζ) = sr s 
  ∧ (∀ j. j ≠ i → σ j ∈ (fst o sr) ' init (np j)) } ⊆ init (onp i)"
and init_notempty: "∀ j. init (np i) ≠ {}"
and trans: "∀ s a s' σ σ'. [ σ i = fst (sr s); 
  σ' i = fst (sr s'); 
  (s, a, s') ∈ trans (np i) ]"
begin

lemma init_pnet_p_NodeS:
  assumes "NodeS i s R ∈ init (pnet np p)"
  shows "p = ⟨i; R⟩"
  using assms by (cases p) (auto simp add: node_comps)

lemma init_pnet_p_SubnetS:
  assumes "SubnetS s1 s2 ∈ init (pnet np p)"
  obtains p1 p2 where "p = (p1 ∥ p2)"
    and "s1 ∈ init (pnet np p1)"
    and "s2 ∈ init (pnet np p2)"
  using assms by (cases p) (auto simp add: node_comps)

lemma init_pnet_fst_sr_netgmap:
  assumes "s ∈ init (pnet np p)"
    and "i ∈ net_ips s"
    and "wf_net_tree p"
  shows "the (fst (netgmap sr s) i) ∈ (fst ◦ sr) ' init (np i)"
  using assms proof (induction s arbitrary: p)
    fix ii s R i p
    assume "NodeS ii s R i ∈ init (pnet np p)"
      and "i ∈ net_ips (NodeS ii s R i)"
      and "wf_net_tree p"
    note this(1)
    moreover then have "p = ⟨ii; R⟩" by (rule init_pnet_p_NodeS)
    ultimately have "s ∈ init (np ii)" by (clarsimp simp: node_comps)
    with 'i ∈ net_ips (NodeS ii s R i)'
    show "the (fst (netgmap sr (NodeS ii s R i)) i) ∈ (fst ◦ sr) ' init (np i)" by clarsimp
  next
    fix s1 s2 p
    assume IH1: "∀p. s1 ∈ init (pnet np p)" 
      and "i ∈ net_ips s1"
      and "wf_net_tree p"
      and "the (fst (netgmap sr s1) i) ∈ (fst ◦ sr) ' init (np i)"
    and IH2: "∀p. s2 ∈ init (pnet np p)" 
      and "i ∈ net_ips s2"
      and "wf_net_tree p"
      and "the (fst (netgmap sr s2) i) ∈ (fst ◦ sr) ' init (np i)"
    and "SubnetS s1 s2 ∈ init (pnet np p)"
    and "i ∈ net_ips (SubnetS s1 s2)"
    and "wf_net_tree p"
    from this(3) obtain p1 p2 where "p = (p1 ∥ p2)"
      and "s1 ∈ init (pnet np p1)"
      and "s2 ∈ init (pnet np p2)"
    by (rule init_pnet_p_SubnetS)
    from this(1) and 'wf_net_tree p' have "wf_net_tree p1" 
      and "wf_net_tree p2"
      and "net_tree_ips p1 ∩ net_tree_ips p2 = {}" 
    by auto
    from 'i ∈ net_ips (SubnetS s1 s2)'
      have "i ∈ net_ips s1 ∨ i ∈ net_ips s2"
    by simp
    thus "the (fst (netgmap sr (SubnetS s1 s2)) i) ∈ (fst ◦ sr) ' init (np i)" proof
      assume "i ∈ net_ips s1"
      hence "i ∉ net_ips s2"
    proof -
      from 's1 ∈ init (pnet np p1)'
        and 'i ∈ net_ips s1'
        have "i ∈ net_tree_ips p1" ..
      with 'net_tree_ips p1 ∩ net_tree_ips p2 = {}'
        have "i ∉ net_tree_ips p2" by auto
    with 's2 ∈ init (pnet np p2)'
      show ?thesis ..

144
qed

moreover from ‘s1 ∈ init (pnet np p1)’ ‘i ∈ net_ips s1’ and ‘wf_net_tree p1’
  have "the (fst (netgmap sr s1) i) ∈ (fst o sr) ‘ init (np i)"
  by (rule IH1)
ultimately show ?thesis by simp

next
assume "i ∈ net_ips s2"
moreover from ‘s2 ∈ init (pnet np p2)’ have "the (fst (netgmap sr s2) i) ∈ (fst o sr) ‘ init (np i)"

using ‘wf_net_tree p2’ by (rule IH2)
moreover from ‘s2 ∈ init (pnet np p2)’ and ‘i ∈ net_ips s2’ have "i ∈ net_tree_ips p2" ..
ultimately show ?thesis by simp

qed

qed

lemma init_lifted:
  assumes "wf_net_tree p"
shows "{ (σ, snd (netgmap sr s)) | σ s. s ∈ init (pnet np p)
  ∧ (∀i. if i ∈ net_tree_ips p then σ i = the (fst (netgmap sr s) i)
  else σ i ∈ (fst o sr) ‘ init (np i)) } ⊆ init (opnet opn p)"
using assms proof (induction p)
next
fix p1 p2
assume IH1: "wf_net_tree (i; R)"
show "{(σ, snd (netgmap sr s)) | σ s. s ∈ init (pnet np (i; R))
  ∧ (∀j. if j ∈ net_tree_ips (i; R) then σ j = the (fst (netgmap sr s) j)
  else σ j ∈ (fst o sr) ‘ init (np j))} ⊆ init (opnet onp (i; R))"
  by (clarsimp simp add: node_comps onode_comps)
(rule set_mp [OF init], auto)

fix p1 p2
assume IH2: "wf_net_tree p2
  ⇒ { (σ, snd (netgmap sr s)) | σ s. s ∈ init (pnet np p2)
  ∧ (∀i. if i ∈ net_tree_ips p2 then σ i = the (fst (netgmap sr s) i)
  else σ i ∈ (fst o sr) ‘ init (np i)) } ⊆ init (opnet opn p2)"

from this(3) have "wf_net_tree p1" and "wf_net_tree p2"
  and "net_tree_ips p1 ∩ net_tree_ips p2 = {}" by auto
show "{ (σ, snd (netgmap sr s)) | σ s. s ∈ init (pnet np (p1 || p2))
  ∧ (∀i. if i ∈ net_tree_ips (p1 || p2) then σ i = the (fst (netgmap sr s) i)
  else σ i ∈ (fst o sr) ‘ init (np i))} ⊆ init (opnet onp (p1 || p2))"
proof (rule, clarsimp simp only: split_paired_all pnet.simps automaton.simps)
fix σ s1 s2
assume σ_desc: "∀i. if i ∈ net_tree_ips (p1 || p2)
  then σ i = the (fst (netgmap sr (SubnetS s1 s2)) i)
  else σ i ∈ (fst o sr) ‘ init (np i)"

and "s1 ∈ init (pnet np p1)"
and "s2 ∈ init (pnet np p2)"

from this(2-3) have "net_ips s1 = net_tree_ips p1"
  and "net_ips s2 = net_tree_ips p2" by auto
have "(σ, snd (netgmap sr s1)) ∈ ?S1" proof -
  { fix i
  assume "i ∈ net_tree_ips p1"
  with ‘net_tree_ips p1 ∩ net_tree_ips p2 = {}’ have "i ∉ net_tree_ips p2" by auto
  with ‘s2 ∈ init (pnet np p2)’ have "i ∉ net_ips s2" ..
  hence "the ((fst (netgmap sr s1) ++ fst (netgmap sr s2)) i) = the (fst (netgmap sr s1) i)"
  by simp

next
fix σ s1 s2
assume "i ∈ net_tree_ips s2"
moreover
  { fix i
    assume "i \notin \text{net_tree_ips p1}"
    have "\sigma i \in (\text{fst } \circ \text{sr}) ' \text{init (np i)}"
    proof (cases "i \in \text{net_tree_ips p2}"
      assume "i \notin \text{net_tree_ips p2}"
      with 'i \notin \text{net_tree_ips p1}' and \sigma\_desc show ?thesis
      by simp
    next
      assume "i \in \text{net_tree_ips p2}"
      with 's2 \in \text{init (pnet np p2)'} have "i \in \text{net_ips s2}"
      hence "(\text{fst (netmap sr s1)} ++ \text{fst (netmap sr s2)}) i = (\text{fst (netmap sr s2)} i)"
      by simp
    }
    ultimately show ?thesis
    using 's1 \in \text{init (pnet np p1)'} and \sigma\_desc by auto
  qed

  hence "(\sigma, \text{snd (netmap sr s1)}) \in \text{init (opnet onp p1)}"
  by (rule set_mp [OF IH1 [OF 'wf_net_tree p1']])

  have "(\sigma, \text{snd (netmap sr s2)}) \in ?S2"
  proof
    { fix i
      assume "i \notin \text{net_tree_ips p2}"
      with 's2 \in \text{init (pnet np p2)'} have "i \in \text{net_ips s2}"
      hence "\text{the ((fst (netmap sr s1)) ++ \text{netmap sr s2}) i} = \text{the (fst (netmap sr s2) i)}"
      by simp
    }
    moreover
    { fix i
      assume "i \notin \text{net_tree_ips p2}"
      with 's1 \in \text{init (pnet np p1)'} have "i \in \text{net_ips s1}"
      with 's1 \in \text{init (pnet np p1)'} have "\text{the (fst (netmap sr s1)) i} \in (\text{fst } \circ \text{sr}) ' \text{init (np i)}"
      using 'wf_net_tree p1' by (rule init_pnet_fst_sr_netgmap)
      moreover from 's2 \in \text{init (pnet np p2)'} and 'i \notin \text{net_tree_ips p2}' have "i \notin \text{net_ips s2}"
      ultimately show ?thesis
      using 'i \in \text{net_tree_ips p1}' 'i \in \text{net_ips s1}' and 'i \notin \text{net_tree_ips p2}' \sigma\_desc by simp
    }
    ultimately show ?thesis
    using 's2 \in \text{init (pnet np p2)'} and \sigma\_desc by auto
  qed

  hence "(\sigma, \text{snd (netmap sr s2)}) \in \text{init (opnet onp p2)}"
  by (rule set_mp [OF IH2 [OF 'wf_net_tree p2']])

  with 'the (\sigma, \text{snd (netmap sr s1)})' show "(\sigma, \text{snd (netmap sr (SubnetS s1 s2))}) \in \text{init (opnet onp (p1 || p2))}"
  using '\text{net_tree_ips p1} \cap \text{net_tree_ips p2} = {}'
  '\text{net_ips s1} = \text{net_tree_ips p1}'
  '\text{net_ips s2} = \text{net_tree_ips p2}' by simp

  qed

  qed
lemma init_pnet_opnet [elim]:
  assumes "wf_net_tree p"
  and "s ∈ init (pnet np p)"
  shows "netgmap sr s ∈ netmask (net_tree_ips p) \ init (opnet onp p)"
proof -
  from "wf_net_tree p"
  have "{ (σ, snd (netgmap sr s)) | σ s. s ∈ init (pnet np p)
          ∧ (∀i. if i∈net_tree_ips p then σ i = the (fst (netgmap sr s)) i
              else σ i ∈ (fst o sr) \ init (np i)) } ⊆ init (opnet onp p)"
  (is "?S ⊆ _")
  by (rule init_lifted)
  hence "netmask (net_tree_ips p) \ ?S ⊆ netmask (net_tree_ips p) \ init (opnet onp p)"
  by (rule image_mono)
  moreover have "netgmap sr s ∈ netmask (net_tree_ips p) \ ?S"
  proof -
    { fix i
      from init_notempty have "∃s. s ∈ (fst o sr) \ init (np i)" by auto
        hence "(SOME x. x ∈ (fst o sr) \ init (np i)) ∈ (fst o sr) \ init (np i)" ..
      }
    with "s ∈ init (pnet np p)" and init_notempty
    have "((λi. if i ∈ net_tree_ips p
          then the (fst (netgmap sr s)) i
              else SOME x. x ∈ (fst o sr) \ init (np i)), snd (netgmap sr s)) ∈ ?S"
    (is "?s ∈ ?S") by auto
    moreover have "netgmap sr s = netmask (net_tree_ips p) ?s"
    proof (intro prod_eqI ext)
      fix i
      show "fst (netgmap sr s) i = fst (netmask (net_tree_ips p) ?s) i"
      proof (cases "i ∈ net_tree_ips p")
        assume "i ∈ net_tree_ips p"
        with 's ∈ init (pnet np p)" have "i∈net_ips s" ..
        hence "Some (the (fst (netgmap sr s)) i)) = fst (netgmap sr s) i"
          by (rule some_the_fst_netgmap)
        with 'i∈net_tree_ips p' show ?thesis
          by simp
        next
        assume "i /∈ net_tree_ips p"
        moreover with 's ∈ init (pnet np p)" have "i∉net_ips s" ..
        ultimately show ?thesis
          by simp
      qed
      qed simp
    ultimately show ?thesis
      by (rule rev_image_eqI)
    qed
  qed
lemma transfer_connect:
  assumes "(s, connect(i, i'), s') ∈ trans (pnet np n)"
  and "s ∈ reachable (pnet np n) TT"
  and "netgmap sr s = netmask (net_tree_ips n) (σ, ζ)"
  and "wf_net_tree n"
  obtains σ' ζ' where "((σ, ζ), connect(i, i'), (σ', ζ')) ∈ trans (opnet onp n)"
  and "∀j. j∈net_ips ζ ↔ ζ' j = σ j"
  and "netgmap sr s' = netmask (net_tree_ips n) (σ', ζ')"
proof atomize_elim
  from assms have "((σ, snd (netgmap sr s)), connect(i, i'), (σ, snd (netgmap sr s'))) ∈ trans (opnet onp n)"
    ∧ netgmap sr s' = netmask (net_tree_ips n) (σ, snd (netgmap sr s'))"
  proof (induction n arbitrary: s s' ζ)
    fix ii R, ns ns' ζ
    assume "(ns, connect(i, i'), ns') ∈ trans (pnet np (ii; R))"
and "netgmap sr ns = netmask (net_tree_ips (ii; R_i)) (σ, ζ)"
from this(1) have "(ns, connect(i, i'), ns') ∈ node_sos (trans (np ii))"
by (simp add: node_comps)
moreover then obtain ni s s' R R'
where "ns = NodeS ni s R"
and "ns' = NodeS ni s' R'" ..
ultimately have "((NodeS ni s R, connect(i, i'), NODES ni s') ∈ node_sos (trans (np ii))"
by simp
moreover then have "s' = s" by auto
ultimately have "((σ, snd (netgmap sr s)), connect(i, i'), (σ, snd (netgmap sr s'))) ∈ trans (opnet onp (ii; R_i))"
∧ netgmap sr ns' = netmask (net_tree_ips (ii; R_i)) (σ, ζ)"
show "((σ, snd (netgmap sr ns)), connect(i, i'), (σ, snd (netgmap sr ns'))) ∈ trans (opnet onp (ii; R_i))"
∧ netgmap sr ns' = netmask (net_tree_ips (ii; R_i)) (σ, ζ)"
next
fix n1 n2 s s' ζ
assume IH1: "∀ s s' ζ. (s, connect(i, i'), s') ∈ trans (pnet np n1)
⇒ s ∈ reachable (pnet np n1) TT
⇒ netgmap sr s = netmask (net_tree_ips n1) (σ, ζ)
⇒ wf_net_tree n1
⇒ ((σ, snd (netgmap sr s)), connect(i, i'), (σ, snd (netgmap sr s'))) ∈ trans (opnet onp n1)
∧ netgmap sr s' = netmask (net_tree_ips n1) (σ, snd (netgmap sr s'))"
and IH2: "∀ s s' ζ. (s, connect(i, i'), s') ∈ trans (pnet np n2)
⇒ s ∈ reachable (pnet np n2) TT
⇒ netgmap sr s = netmask (net_tree_ips n2) (σ, ζ)
⇒ wf_net_tree n2
⇒ ((σ, snd (netgmap sr s)), connect(i, i'), (σ, snd (netgmap sr s'))) ∈ trans (opnet onp n2)
∧ netgmap sr s' = netmask (net_tree_ips n2) (σ, snd (netgmap sr s'))"
and tr: "(s, connect(i, i'), s') ∈ trans (pnet np (n1 || n2))"
and sr: "s ∈ reachable (pnet np (n1 || n2)) TT"
and nm: "netgmap sr s = netmask (net_tree_ips (n1 || n2)) (σ, ζ)"
and "wf_net_tree (n1 || n2)"
from this(3) have "(s, connect(i, i'), s') ∈ pnet_sos (trans (pnet np n1))
(trans (pnet np n2))"
by simp
then obtain s1 s1' s2 s2' where "s = SubnetS s1 s2"
and "s' = SubnetS s1' s2'"
and "(s1, connect(i, i'), s1') ∈ trans (pnet np n1)"
and "(s2, connect(i, i'), s2') ∈ trans (pnet np n2)"
by (rule partial_connectTE) auto
from this(1) and nm have "netgmap sr (SubnetS s1 s2) = netmask (net_tree_ips (n1 || n2)) (σ, ζ)"
by simp
from "wf_net_tree (n1 || n2)" have "wf_net_tree n1" and "wf_net_tree n2"
and "net_tree_ips n1 ∩ net_tree_ips n2 = {}" by auto
from sr 's = SubnetS s1 s2' have "s1 ∈ reachable (pnet np n1) TT" by (metis subnet_reachable(1))
hence "net_ips s1 = net_tree_ips n1" by (rule net_ips_net_tree_ips)
from sr 's = SubnetS s1 s2' have "s2 ∈ reachable (pnet np n2) TT" by (metis subnet_reachable(2))
hence "net_ips s2 = net_tree_ips n2" by (rule net_ips_net_tree_ips)
from nm 's = SubnetS s1 s2'
have "netgmap sr (SubnetS s1 s2) = netmask (net_tree_ips (n1 || n2)) (σ, ζ)" by simp
hence "netgmap sr s1 = netmask (net_tree_ips n1) (σ, snd (netgmap sr s1))"
using "net_tree_ips n1 ∩ net_tree_ips n2 = {}" 'net_ips s1 = net_tree_ips n1'
and "net_ips s2 = net_tree_ips n2" by (rule netgmap_subnet_split)
with "(s1, connect(i, i'), s1') ∈ trans (pnet np n1)"
and 's1 ∈ reachable (pnet np n1) TT"
have "((σ, snd (netgmap sr s1)), connect(i, i'), (σ, snd (netgmap sr s1')))) ∈ trans (opnet onp n1)"
and "netgmap sr s1' = netmask (net_tree_ips n1) (σ, snd (netgmap sr s1'))"
using 'wf_net_tree n1' unfolding atomize_conj by (rule IH1)

from 'netgmap sr (SubnetS s1 s2) = netmask (net_tree_ips (n1 || n2)) (σ, γ)'
  'net_ips s1 = net_tree_ips n1' and 'net_ips s2 = net_tree_ips n2'
have "netgmap sr s2 = netmask (net_tree_ips n2) (σ, snd (netgmap sr s2'))"
by (rule netgmap_subnet_split2)
with 's2, connect(i, i'), s2') ∈ trans (pnet np n2)' and 's2 ∈ reachable (pnet np n2) TT'
have "((σ, snd (netgmap sr s2)), connect(i, i'), (σ, snd (netgmap sr s2')))) ∈ trans (opnet onp n2)"
and "netgmap sr s2' = netmask (net_tree_ips n2) (σ, snd (netgmap sr s2'))"

proof -
  from 'netgmap sr s1' = netmask (net_tree_ips n1) (σ, snd (netgmap sr s1'))'
  'netgmap sr s2' = netmask (net_tree_ips n2) (σ, snd (netgmap sr s2'))'
  's' = SubnetS s1 s2''
  have "((σ, SubnetS (snd (netgmap sr s1)) (snd (netgmap sr s2))), connect(i, i'),
  (σ, SubnetS (snd (netgmap sr s1')) (snd (netgmap sr s2'))))
  ∈ opsos (trans (opnet onp n1)) (trans (opnet onp n2))" by (simp add: node_comps)

ultimately show "((σ, snd (netgmap sr s)), connect(i, i'), (σ, snd (netgmap sr s'))))
  ∈ trans (opnet onp (n1 || n2))" by auto

qed

moreover from 'netgmap sr s1' = netmask (net_tree_ips n1) (σ, snd (netgmap sr s1'))'
  'netgmap sr s2' = netmask (net_tree_ips n2) (σ, snd (netgmap sr s2'))'
  's' = SubnetS s1 s2''
have "netgmap sr s' = netmask (net_tree_ips (n1 || n2)) (σ, snd (netgmap sr s'))" ..

ultimately show "((σ, snd (netgmap sr s)), connect(i, i'), (σ, snd (netgmap sr s')))
  ∈ trans (opnet onp (n1 || n2)) ∧ netgmap sr s' = netmask (net_tree_ips (n1 || n2)) (σ, snd (netgmap sr s'))" ..

qed

moreover from 'netgmap sr s = netmask (net_tree_ips n) (σ, ρ)'
obtains "∃σ'. σ' = snd (netgmap sr s)" by simp
ultimately show "∃σ'. (∃j. j /∈ net_ips σ' −→ σ' j = σ j)
  ∧ netgmap sr s' = netmask (net_tree_ips n) (σ', ρ')" by auto

qed

lemma transfer_disconnect:
assumes "((s, disconnect(i, i'), s')) ∈ trans (pnet np n)"
and "s ∈ reachable (pnet np n) TT"
and "netgmap sr s = netmask (net_tree_ips n) (σ, ρ)"
and "wf_net_tree n" obtains σ' ρ' where "((σ, ρ), disconnect(i, i'), (σ', ρ')) ∈ trans (opnet onp n)"
  and "∀i. i /∈ net_ips σ −→ σ' i = σ i"
  and "netgmap sr s' = netmask (net_tree_ips n) (σ', ρ')"
proof atomize_elim
  from assms have "((σ, snd (netgmap sr s)), disconnect(i, i'), (σ, snd (netgmap sr s')))) ∈ trans (opnet onp n)"
    ∧ netgmap sr s' = netmask (net_tree_ips n) (σ, snd (netgmap sr s'))" by auto

proof (induction n arbitrary: s s' ρ)
fix i i', n1 s s' where "ns = NodeS ni s R"

have "((s, disconnect(i, i'), s')) ∈ trans (pnet np (i1; R))"
  and "netgmap sr ns = netmask (net_tree_ips (i1; R)) (σ, ρ)"
from this(l) have "((ns, disconnect(i, i'), ns')) ∈ node_sos (trans (np ii))" by (simp add: node_comps)
moreover then obtain ni s s' R' where "ns = NodeS ni s R"
ultimately have "((\sigma, \text{NodeS ni (snd (sr s)) R}), \text{disconnect}(i, i'), (\sigma, \text{NodeS ni (snd (sr s)) R})) \in \text{node_sos (trans (np ii))}"

by simp

moreover then have "s' = s" by auto

ultimately have "((\sigma, \text{NodeS ni s' R'}), \text{disconnect}(\sigma, i', i'), (\sigma, \text{NodeS ni (snd (sr s)) R'})) \in \text{node_sos (trans (onp ii))}"

by (rule node_disconnectTE', auto intro!: onode_sos.intros [simplified])

with 'ns = NodeS ni s R' 'ns' = NodeS ni s' R'' 's' = s'

and 'netgmap sr ns = netmask (net_tree_ips \langle ii; R \rangle) (\sigma, \zeta)'

show "((\sigma, \text{snd (netgmap sr ns)}), \text{disconnect}(i, i'), (\sigma, \text{snd (netgmap sr ns'))}) \in \text{trans (opnet onp \langle ii; R \rangle)}"

by simp add: onode_comps

next

fix n1 n2 s s' \zeta

assume IH1: "\(s, \text{disconnect}(i, i'), s'\) \in \text{trans (pnet np n1)}

\Rightarrow s \in \text{reachable (pnet np n1) TT}

\Rightarrow \text{netgmap sr s = netmask (net_tree_ips n1) (\sigma, \zeta)}

\Rightarrow \\text{wf_net_tree n1}

\Rightarrow ((\sigma, \text{snd (netgmap sr s)}), \text{disconnect}(i, i'), (\sigma, \text{snd (netgmap sr s'))}) \in \text{trans (opnet onp n1)}"

and IH2: "\(s, \text{disconnect}(i, i'), s'\) \in \text{trans (pnet np n2)}

\Rightarrow s \in \text{reachable (pnet np n2) TT}

\Rightarrow \text{netgmap sr s = netmask (net_tree_ips n2) (\sigma, \zeta)}

\Rightarrow \\text{wf_net_tree n2}

\Rightarrow ((\sigma, \text{snd (netgmap sr s)}), \text{disconnect}(i, i'), (\sigma, \text{snd (netgmap sr s'))}) \in \text{trans (opnet onp n2)}"

and tr: "\(s, \text{disconnect}(i, i'), s'\) \in \text{trans (pnet np (n1 \parallel n2))}"

and sr: "s \in \text{reachable (pnet np (n1 \parallel n2)) TT}"

and nm: "\text{netgmap sr s = netmask (net_tree_ips (n1 \parallel n2)) (\sigma, \zeta)}"

and "\text{wf_net_tree (n1 \parallel n2)}"

from this(3) have "\(s, \text{disconnect}(i, i'), s'\) \in \text{pnet_sos (trans (pnet np n1))}

(trans (pnet np n2))"

by simp

then obtain s1 s1' s2 s2' where "s = \text{SubnetS s1 s2}"

and "s' = \text{SubnetS s1' s2'}"

and "(s1, \text{disconnect}(i, i'), s1') \in \text{trans (pnet np n1)}"

and "(s2, \text{disconnect}(i, i'), s2') \in \text{trans (pnet np n2)}"

by (rule partial_disconnectTE) auto

from this(1) and nm have "\text{netgmap sr (SubnetS s1 s2) = netmask (net_tree_ips (n1 \parallel n2)) (\sigma, \zeta)}"

by simp

from 'wf_net_tree (n1 \parallel n2)' have "\text{wf_net_tree n1}" and "\text{wf_net_tree n2}"

and "\text{net_tree_ips n1 \cap net_tree_ips n2 = \{}\)" by auto

from sr 's = SubnetS s1 s2' have "s1 \in \text{reachable (pnet np n1) TT}" by (metis subnet_reachable(1))

hence "\text{net_ips s1 = net_tree_ips n1}" by (rule pnet_net_ips_net_tree_ips)

from sr 's = SubnetS s1 s2' have "s2 \in \text{reachable (pnet np n2) TT}" by (metis subnet_reachable(2))

hence "\text{net_ips s2 = net_tree_ips n2}" by (rule pnet_net_ips_net_tree_ips)

from nm 's = SubnetS s1 s2'

have "\text{netgmap sr (SubnetS s1 s2) = netmask (net_tree_ips (n1 \parallel n2)) (\sigma, \zeta)}" by simp

hence "\text{netgmap sr s1 = netmask (net_tree_ips n1) (\sigma, \text{snd (netgmap sr s1)})}"

using "\text{net_tree_ips n1 \cap net_tree_ips n2 = \{}\)" 'net_ips s1 = net_tree_ips n1'

and 'net_ips s2 = net_tree_ips n2' by (rule netgmap_subnet_split1)

with 's1, \text{disconnect}(i, i'), s1' \in \text{trans (pnet np n1)}'

and 's1 \in \text{reachable (pnet np n1) TT}'

have "((\sigma, \text{snd (netgmap sr s1)}), \text{disconnect}(i, i'), (\sigma, \text{snd (netgmap sr s1'))}) \in \text{trans (opnet onp n1)}"

and "\text{netgmap sr s1' = netmask (net_tree_ips n1) (\sigma, \text{snd (netgmap sr s1'))}"

using 'wf_net_tree n1' unfolding atomize_conj by (rule IH1)
from 'netgmap sr (SubnetS s1 s2) = netmask (net_tree_ips (n1 || n2)) (σ, ζ)'  
'net_ips s1 = net_tree_ips n1' and 'net_ips s2 = net_tree_ips n2'  
have "netgmap sr s2 = netmask (net_tree_ips n2) (σ, snd (netgmap sr s2))"  
by (rule netgmap_subnet_split2)  
with '(s2, disconnect(i, i'), s2') ∈ trans (pnet np n2)'  
and 's2 ∈ reachable (pnet np n2) TT'  
and 'net_ips s2 = net_tree_ips n2'  
have "((σ, snd (netgmap sr s2)), disconnect(i, i'), (σ, snd (netgmap sr s2')))) ∈ trans (opnet onp n2)"  
and "netgmap sr s2' = netmask (net_tree_ips n2) (σ, snd (netgmap sr s2'))"  
using 'wf_net_tree n2' unfolding atomize_conj by (rule IH2)  
proof -  
from 'netgmap sr s1' = netmask (net_tree_ips n1) (σ, snd (netgmap sr s1'))'  
'netgmap sr s2' = netmask (net_tree_ips n2) (σ, snd (netgmap sr s2'))'  
's' = SubnetS s1' s2''  
show ?thesis by simp  
qed  
moreover from 'netgmap sr s = netmask (net_tree_ips n) (σ, ζ)'  
have "ζ = snd (netgmap sr s)" by simp  
ultimately show "((σ, snd (netgmap sr s)), disconnect(i, i'), (σ, snd (netgmap sr s'))) ∈ trans (opnet onp n)"  
∧ netgmap sr s' = netmask (net_tree_ips (n1 || n2)) (σ, snd (netgmap sr s'))" by auto  
qed  
lemma transfer_tau:  
assumes "(s, τ, s') ∈ trans (pnet np n)"  
and "s ∈ reachable (pnet np n) TT"  
and "netgmap sr s = netmask (net_tree_ips n) (σ, ζ)"  
and "wf_net_tree n"  
obtains σ' ζ' where "((σ, ζ), τ, (σ', ζ')) ∈ trans (opnet onp n)"  
and "∀ j. j ≠ i −→ netmap s' j = netmap s j"  
and "netgmap sr s' = netmask (net_tree_ips n) (σ', ζ')"  
proof atomize_elim  
from assms(4,2,1) obtain i where "i ∈ net_ips s"  
and "∀ j. j ≠ i −→ netmap s' j = netmap s j"  
and "net_ip_action np τ i n s s'"  
by (metis pnet_tau_single_node)  
from this(2) have "∀ j. j ≠ i −→ fst (netgmap sr s') j = fst (netgmap sr s) j"  
by (clarsimp intro!: netmap_is_fst_netgmap')  
from '(s, τ, s') ∈ trans (pnet np n)' have "net_ips s' = net_ips s"  
by (rule pnet_maintains_dom [THEN sym])  
def σ' ≡ "λ j. if j = i then the (fst (netgmap sr s')) i else σ j"  
from '∀ j. j ≠ i −→ fst (netgmap sr s') j = fst (netgmap sr s) j'  
and 'netgmap sr s = netmask (net_tree_ips n) (σ, ζ)'  
have "∀ j. j ≠ i −→ σ' j = σ j"  
unfolding σ'_def by clarsimp
from assms(2) have "net_ips s = net_tree_ips n"
  by (rule pnet_net_ips_net_tree_ips)

from 'netgmap sr s = netmask (net_tree_ips n) (σ, ζ)'
  have "ζ = snd (netgmap sr s)" by simp

from '∀ j. j ≠ i → fst (netgmap sr s') j = fst (netgmap sr s) j' 'i ∈ net_ips s'
  'net_ips s = net_tree_ips n' 'net_ips s' = net_ips s'
  'netgmap sr s = netmask (net_tree_ips n) (σ, ζ)'
  have "fst (netgmap sr s') = fst (netmask (net_tree_ips n) (σ', snd (netgmap sr s')))"
  unfolding σ'_def by - (rule ext, clarsimp)

hence "netgmap sr s' = netmask (net_tree_ips n) (σ', snd (netgmap sr s'))" by (rule prod_eqI, simp)

with assms(1, 3)
  have "((σ, snd (netgmap sr s)), τ, (σ', snd (netgmap sr s'))) ∈ trans (opnet onp n)"
  using assms(2,4) 'i ∈ net_ips s' and 'net_ip_action np τ i n s s''
proof
  (induction n arbitrary: s s' ζ)
  fix ii R
  i ns ns' ζ
  assume "(ns, τ, ns') ∈ trans (pnet np ⟨ii; R⟩)
  and nsr: "ns ∈ reachable (pnet np ⟨ii; R⟩) TT"
  and "netgmap sr ns = netmask (net_tree_ips ⟨ii; R⟩) (σ, ζ)"
  and "netgmap sr ns' = netmask (net_tree_ips ⟨ii; R⟩) (σ', snd (netgmap sr ns'))"
  and "i ∈ net_ips ns"
  from this(1)
  have "(ns, τ, ns') ∈ node_sos (trans (np ii))" by (simp add: node_comps)
  moreover with nsr
  obtain s s' R R' where
    "ns = NodeS ii s R" and "ns' = NodeS ii s' R'
  by (metis net_node_reachable_is_node node_tauTE')
  moreover from 'i ∈ net_ips ns' and 'ns = NodeS ii s R' have "ii = i" by simp
  ultimately have ntr: "(NodeS i s R, τ, NodeS i s' R') ∈ node_sos (trans (np i))"
  by simp
  hence "R' = R" by (metis net_state.inject(1) node_tauTE')
  from ntr obtain a where
    "(s, a, s') ∈ trans (np i)"
    and "(∃ d. a = ¬unicast d ∧ d /∈ R) ∨ (a = τ)"
  by (rule node_tauTE') auto
  from 'netgmap sr ns = netmask (net_tree_ips ⟨ii; R⟩) (σ, ζ)'
  'ns = NodeS ii s R' and 'ii = i'
  have "σ i = fst (sr s)" by simp (metis map_upd_Some_unfold)
  moreover from 'netgmap sr ns' = netmask (net_tree_ips ⟨ii; R⟩) (σ', snd (netgmap sr ns'))'
  'ns' = NodeS ii s' R' and 'ii = i'
  have "σ' i = fst (sr s')"
  unfolding σ'_def by clarsimp (hypsubst_thin,
    metis (full_types, lifting) fun_upd_same option.sel)
  ultimately have "((σ, snd (sr s)), a, (σ', snd (sr s'))) ∈ trans (onp i)"
  using '(s, a, s') ∈ trans (np i)' by (rule trans)
  from '(∃ d. a = ¬unicast d ∧ d /∈ R) ∨ (a = τ)'
  and '∀ j. j ≠ i → σ j = σ j' 'R'=R'
  and '((((σ, snd (sr s)), a, (σ', snd (sr s'))) ∈ trans (onp i)'
  have "((σ, NodeS i (snd (sr s)) R), τ, (σ', NodeS i (snd (sr s')) R')) ∈ onode_sos (trans (onp i))"
  by (metis onode_sos.onode_notucast onode_sos.onode_tau)
  with 'ns = NodeS ii s R' 'ns' = NodeS ii s' R' 'ii = i'
  show "((σ, snd (netgmap sr ns)), τ, (σ', snd (netgmap sr ns'))) ∈ trans (opnet onp ⟨ii; R⟩)"
  by (simp add: onode_comps)
next
  fix n1 n2 s s' ζ
  assume IH1: "∀ s s' ζ. (s, τ, s') ∈ trans (pnet np n1)
  ⇒ netgmap sr s = netmask (net_tree_ips n1) (σ, ζ)
  ⇒ netgmap sr s' = netmask (net_tree_ips n1) (σ', snd (netgmap sr s'))"
\[ s \in \text{reachable (pnet np n1)} \quad \text{TT} \]
\[ \Rightarrow \text{wf_net_tree n1} \]
\[ i \in \text{net_ips s} \]
\[ \Rightarrow \text{net_ip_action np } \tau \ i \ n1 \ s \ s' \]
\[ \Rightarrow ((\sigma, \text{snd (netmask sr s)}), \tau, (\sigma', \text{snd (netmask sr s')})) \in \text{trans (opnet onp n1)} \]

and IH2: "\[ s' \in \text{reachable (pnet np n2)} \]
\[ \Rightarrow \text{netgmap sr s} = \text{netmask (net_tree_ips n2)} (\sigma, \zeta) \]
\[ \Rightarrow \text{netgmap sr s'} = \text{netmask (net_tree_ips n2)} (\sigma', \text{snd (netmask sr s')}) \]
\[ s \in \text{reachable (pnet np n2)} \quad \text{TT} \]
\[ \Rightarrow \text{wf_net_tree n2} \]
\[ i \in \text{net_ips s} \]
\[ \Rightarrow \text{net_ip_action np } \tau \ i \ n2 \ s \ s' \]
\[ \Rightarrow ((\sigma, \text{snd (netmask sr s)}), \tau, (\sigma', \text{snd (netmask sr s')})) \in \text{trans (opnet onp n2)} \]

and tr: "\[(s, \tau, s') \in \text{trans (pnet np (n1 || n2))}\]

and sr: "s \in \text{reachable (pnet np (n1 || n2)) TT}"

and nm: "netgmap sr s = netmask (net_tree_ips (n1 || n2)) (\sigma, \zeta)"

and nm': "netgmap sr s' = netmask (net_tree_ips (n1 || n2)) (\sigma', \text{snd (netmask sr s')})"

and "wf_net_tree (n1 || n2)"

and "i \in \text{net_ips s}"

and "net_ip_action np \tau \ i (n1 || n2) s s'" from tr have "\[(s, \tau, s') \in \text{pnet_sos (trans (pnet np n1)) (trans (pnet np n2))}\]" by simp

then obtain s1 s1' s2 s2' where "s = SubnetS s1 s2" and "s' = SubnetS s1' s2'" by (rule partial_tauTE) auto

from this(1) and nm have "netgmap sr (SubnetS s1 s2) = netmask (net_tree_ips (n1 || n2)) (\sigma, \zeta)" by simp

from 's' = SubnetS s1' s2' and nm'

have "netgmap sr (SubnetS s1' s2') = netmask (net_tree_ips (n1 || n2)) (\sigma', \text{snd (netmask sr s')})" by simp

from 'wf_net_tree (n1 || n2)' have "wf_net_tree n1"

and "wf_net_tree n2"

and "net_tree_ips n1 \cap net_tree_ips n2 = {}" by auto

from sr [simplified 's = SubnetS s1 s2'] have "s1 \in \text{reachable (pnet np n1)} \quad \text{TT}"

by (rule subnet_reachable(1))

hence "net_ips s1 = net_tree_ips n1" by (rule pnet_net_tree_ips)

from sr [simplified 's = SubnetS s1 s2'] have "s2 \in \text{reachable (pnet np n2)} \quad \text{TT}"

by (rule subnet_reachable(2))

hence "net_ips s2 = net_tree_ips n2" by (rule pnet_net_tree_ips)

from nm [simplified 's = SubnetS s1 s2']

'net_tree_ips n1 \cap net_tree_ips n2 = {}'

'net_ips s1 = net_tree_ips n1'

'net_ips s2 = net_tree_ips n2'

have "netgmap sr s1 = netmask (net_tree_ips n1) (\sigma, \text{snd (netmask sr s1)})" by (rule netgmap_subnet_split1)

from nm [simplified 's = SubnetS s1 s2']

'net_tree_ips s1 \cap net_tree_ips s2 = {}'

'net_ips s1 = net_tree_ips n1'

'net_ips s2 = net_tree_ips n2'

have "netgmap sr s2 = netmask (net_tree_ips n2) (\sigma, \text{snd (netmask sr s2)})" by (rule netgmap_subnet_split2)

from 'i \in net_ips s' and 's = SubnetS s1 s2' have "i \in net_ips s1 \lor i \in net_ips s2" by auto

thus "((\sigma, \text{snd (netmask sr s)}), \tau, (\sigma', \text{snd (netmask sr s')})) \in \text{trans (opnet onp (n1 || n2))}" proof

assume "i \in net_ips s1"

with 's = SubnetS s1 s2' 's' = SubnetS s1' s2' 'net_ip_action np \tau \ i (n1 || n2) s s'" have "(s1, \tau, s1') \in \text{trans (pnet np n1)}"

and "net_ip_action np \tau \ i \ n1 \ s1'" and "s2' = s2" by simp_all
from 'net_ips s1 = net_tree_ips n1' and '(s1, τ, s1') ∈ trans (pnet np n1)'
  have "net_ips s1' = net_tree_ips n1" by (metis pnet_maintains_dom)

from nm' [simplified 's' = SubnetS s1' s2' 's2' = s2']
  'net_tree_ips n1 ∩ net_tree_ips n2 = {}'
  'net_ips s1 = net_tree_ips n1'
  'net_ips s2 = net_tree_ips n2'
  have "netgmap sr s1 = netmask (net_tree_ips n1) (σ', snd (netgmap sr s1'))"
    by (rule netgmap_subnet_split1)

from '(s1, τ, s1') ∈ trans (pnet np n1)'
  'netgmap sr s1 = netmask (net_tree_ips n1) (σ, snd (netgmap sr s1'))'
  'wf_net_tree n1'
  'i ∈ net_ips s1'
  'net_ip_action np τ i n1 s1 s1''
  have "((σ, snd (netgmap sr s1)), τ, (σ', snd (netgmap sr s1')))) ∈ trans (opnet onp n1)"
    by (rule IH1)

with 's = SubnetS s1 s2' 's' = SubnetS s1' s2'' 's2' = s2' show ?thesis
  by (simp del: step_node_tau) (erule opnet_tau1)

next
assume "i ∈ net_ips s2'''
with 's = SubnetS s1 s2' 's' = SubnetS s1' s2'' 's2' = s2'
  'netgmap sr s2 = netmask (net_tree_ips n2) (σ, snd (netgmap sr s2'))'
  's2 ∈ reachable (pnet np n2) '
  'wf_net_tree n2'
  'i ∈ net_ips s2'
  'net_ip_action np τ i n2 s2 s2''
  have "((σ, snd (netgmap sr s2)), τ, (σ', snd (netgmap sr s2')))) ∈ trans (opnet onp n2)"
    by (rule IH2)

with 's = SubnetS s1 s2' 's' = SubnetS s1' s2'' 's2' = s2' show ?thesis
  by (simp del: step_node_tau) (erule opnet_tau2)

qed

with 'ζ = snd (netgmap sr s)' have "((σ, ζ), τ, (σ', snd (netgmap sr s'))) ∈ trans (opnet onp n)"
  by simp

moreover from '∀ j. j ≠ i ⟹ σ' j = σ j' 'i ∈ net_ips s' 'ζ = snd (netgmap sr s)'
  have "∀ j. j ∈ net_ips s ⟹ ζ = snd (netgmap sr s)" by (metis netgmap_netgmap)

ultimately have "((σ, ζ), τ, (σ', snd (netgmap sr s'))) ∈ trans (opnet onp n)
  ∧ (∀ j. j ∈ net_ips s ⟹ ζ = σ' j = σ j)
  ∧ netgmap sr s' = netmask (net_tree_ips n) (σ', snd (netgmap sr s'))"
  using 'netgmap sr s' = netmask (net_tree_ips n) (σ', snd (netgmap sr s'))'
  by simp

thus "∃ ζ'. ((σ, ζ), τ, (σ', ζ')) ∈ trans (opnet onp n)
  ∧ (∀ j. j ∈ net_ips s ⟹ ζ = σ' j = σ j)
  ∧ netgmap sr s' = netmask (net_tree_ips n) (σ', ζ')" by auto

qed
lemma transfer_deliver:
assumes "\((s, i:deliver(d), s')\) ∈ trans (pnet np n)"
and "s ∈ reachable (pnet np n) TT"
and "netmap sr s = netmask (net_tree_ips n) (\(\sigma, \zeta\))"
and "wf_net_tree n"
obtains \(\sigma' , \zeta'\) where "\(\((\sigma, \zeta), i:deliver(d), (\sigma', \zeta')\) ∈ trans (opnet onp n)\)"
and "\(\forall j. j \notin \text{net_ips} \zeta \rightarrow \sigma' j = \sigma j\)"
and "netmap sr s' = netmask (net_tree_ips n) (\(\sigma', \zeta'\))"
proof atomize_elim
from assms(4,2,1) obtain "i ∈ net_ips s"
and "\(\forall j. j \neq i \rightarrow \text{netmap} s' j = \text{netmap} s j\)"
and "\text{net_ip_action np (i:deliver(d)) i n s s'}"
by (metis delivered_to_net_ips pnet_deliver_single_node)
from this(2) have "\(\forall j. j \neq i \rightarrow \text{fst (netgmap sr s')} j = \text{fst (netgmap sr s)} j\)"
by (clarsimp intro!: netmap_is_fst_netgmap')
from '\((s, i:deliver(d), s')\) ∈ trans (pnet np n)'
have "net_ips s' = net_ips s"
by (rule pnet_maintains_dom [THEN sym])
def \(\sigma'\) ≡ "\(\lambda j. \text{if } j = i \text{ then } \text{the (fst (netgmap sr s') i) else } \sigma j\)"
from "\(\forall j. j \neq i \rightarrow \text{fst (netgmap sr s')} j = \text{fst (netgmap sr s)} j\)" and 'netgmap sr s = netmask (net_tree_ips n) (\(\sigma, \zeta\))'
have "\(\forall j. j \neq i \rightarrow \sigma' j = \sigma j\)"
unfolding \(\sigma'\)_def by clarsimp
from assms(2)

proof (induction n arbitrary: s s' \(\zeta\) \(i\) ns ns' \(\zeta\))
fix \(i\), \(R\), \(R'\)
assume "\((\\text{ns, i:deliver(d), ns'})\) ∈ trans (pnet np \(\langle ii; R_i \rangle\))"
and nsr: "\(\text{ns} \in \text{reachable (pnet np \(\langle ii; R_i \rangle\)) TT}\)"
and "netmap sr ns = netmask (net_tree_ips \(\langle ii; R_i \rangle\)) (\(\sigma, \zeta\))"
and "netmap sr ns' = netmask (net_tree_ips \(\langle ii; R_i \rangle\)) (\(\sigma', \text{snd (netmap sr ns')}\))"
and "i ∈ net_ips ns"
from this(1) have "\((\text{ns, i:deliver(d), ns'})\) ∈ node_sos (trans (np \(\langle ii \rangle\)))"
by (simp add: node_comp)
moreover with nsr obtain s s' \(\text{R'}\) where "\(\text{ns} = \text{NodeS ii s R}\)"
and "\(\text{ns'} = \text{NodeS ii s' R'}\)"
by (metis net_node_reachable_is_node_sos_dest)
moreover from 'i ∈ net_ips ns' and 'ns = NodeS ii s R' have 'ii = i' by simp
ultimately have ntr: "\((\text{NodeS i s R, i:deliver(d), NodeS i s' R'})\) ∈ node_sos (trans (np \(i\)))"
by simp
hence "\(\text{R'} = \text{R}\)" by (metis net_state.inject(1) node_deliverTE')
from ntr have "\((\text{s, deliver d, s'})\) ∈ trans (np \(i\))"
by (rule node_deliverTE')
simp
from 'netmap sr ns = netmask (net_tree_ips \(\langle ii; R_i \rangle\)) (\(\sigma, \zeta\))' \(\text{ns} = \text{NodeS ii s R}\) and 'ii = i'
have "\(\sigma \text{ i} = \text{fst (sr s)}\)" by simp (metis map_upd_Some_unfold)
moreover from ‘netgmap sr ns’ = netmask (net_tree_ips ⟨ii; R_i⟩) (σ’, snd (netgmap sr ns’))’ ‘ns’ = NodeS ii s’ R’’ and ‘ii = i’ have “σ’ i = fst (sr s)” unfolding σ’_def by clarsimp (hypsubst_thin, metis (lifting, full_types) fun_upd_same option.sel)
ultimately have “((σ, snd (sr s)), deliver d, (σ’, snd (sr s’))) ∈ trans (onp i)” using ‘(s, deliver d, s’) ∈ trans (np i)” by (rule trans)
with ‘∀ j. j ≠ i → σ’ j = σ j’ ‘R’=R’ have “((σ, NodeS i (snd (sr s)) R), i:deliver(d), (σ’, NodeS i (snd (sr s’)) R’)) ∈ onode_sos (trans (onp i))” by (metis onode_sos.onode_deliver)

next
fix n1 n2 s s’ ζ
assume IH1: “⋀ s s’ ζ. (s, i:deliver(d), s’) ∈ trans (pnet np n1) ⇒ netgmap sr s = netmask (net_tree_ips n1) (σ, ζ) ⇒ netgmap sr s’ = netmask (net_tree_ips n2) (σ’, snd (netgmap sr s’)) ⇒ s ∈ reachable (pnet np n1) TT ⇒ wf_net_tree n1 ⇒ i∈net_ips s ⇒ net_ip_action np (i:deliver(d)) i n1 s s’ ⇒ ((σ, snd (netgmap sr s)), i:deliver(d), (σ’, snd (netgmap sr s’))) ∈ trans (opnet onp n1)” and IH2: “⋀ s s’ ζ. (s, i:deliver(d), s’) ∈ trans (pnet np n2) ⇒ netgmap sr s = netmask (net_tree_ips (n1 ∥ n2)) (σ, ζ) ⇒ netgmap sr s’ = netmask (net_tree_ips (n1 ∥ n2)) (σ’, snd (netgmap sr s’)) ⇒ s ∈ reachable (pnet np n2) TT ⇒ wf_net_tree n2 ⇒ i∈net_ips s ⇒ net_ip_action np (i:deliver(d)) i n2 s s’ ⇒ ((σ, snd (netgmap sr s)), i:deliver(d), (σ’, snd (netgmap sr s’))) ∈ trans (opnet onp n2)” and tr: “(s, i:deliver(d), s’) ∈ trans (pnet np (n1 ∥ n2))” and sr: “s ∈ reachable (pnet np (n1 ∥ n2)) TT” and nm: “netgmap sr s = netmask (net_tree_ips (n1 ∥ n2)) (σ, ζ)” and nm’: “netgmap sr s’ = netmask (net_tree_ips (n1 ∥ n2)) (σ’, snd (netgmap sr s’))” and “wf_net_tree (n1 ∥ n2)” and “i∈net_ips s” and “net_ip_action np (i:deliver(d)) i (n1 ∥ n2) s s’” from tr have “(s, i:deliver(d), s’) ∈ pnet_sos (trans (pnet np n1)) (trans (pnet np n2))” by simp then obtain s1 s1’ s2 s2’ where “s = SubnetS s1 s2” and “s’ = SubnetS s1’ s2’” by (rule partial_deliverTE) auto
from this(1) and nm have “netgmap sr (SubnetS s1 s2) = netmask (net_tree_ips (n1 ∥ n2)) (σ, ζ)” by simp from ‘s’ = SubnetS s1’ s2’” and nm’ have “netgmap sr (SubnetS s1’ s2’) = netmask (net_tree_ips (n1 ∥ n2)) (σ’, snd (netgmap sr s’))” by simp from ‘wf_net_tree (n1 ∥ n2)” have “wf_net_tree n1” and “wf_net_tree n2” and “net_tree_ips n1 ∩ net_tree_ips n2 = {}” by auto
from sr [simplified ‘s = SubnetS s1 s2’] have “s1 ∈ reachable (pnet np n1) TT” by (rule subnet_reachable(1)) hence “net_ips s1 = net_tree_ips n1” by (rule pnet_net_ips_net_tree_ips)
from sr [simplified ‘s = SubnetS s1 s2’] have “s2 ∈ reachable (pnet np n2) TT”
by (rule subnet_reachable(2))

hence "net_ips s2 = net_tree_ips n2" by (rule pnet_net_ips_net_tree_ips)

from nm [simplified ‘s = SubnetS s1 s2’]
  'net_tree_ips n1 ∩ net_tree_ips n2 = {}'
  'net_ips s1 = net_tree_ips n1'
  'net_ips s2 = net_tree_ips n2'
have "netgmap sr s1 = netmask (net_tree_ips n1) (σ, snd (netgmap sr s1))"
  by (rule netgmap_subnet_split1)

from nm [simplified ‘s = SubnetS s1 s2’]
  'net_ips s1 = net_tree_ips n1'
  'net_ips s2 = net_tree_ips n2'
have "netgmap sr s2 = netmask (net_tree_ips n2) (σ, snd (netgmap sr s2))"
  by (rule netgmap_subnet_split2)

from ‘i ∈ net_ips s’ and ‘s = SubnetS s1 s2’ have "i ∈ net_ips s1 ∨ i ∈ net_ips s2" by auto
  thus "((σ, snd (netgmap sr s)), i:deliver(d), (σ’, snd (netgmap sr s’))) ∈ trans (opnet onp (n1 || n2))"

proof
  assume "i ∈ net_ips s1"
  with ‘s = SubnetS s1 s2’ ‘s’ = SubnetS s1’ s2’‘net_ip_action np (i:deliver(d)) i (n1 || n2) s’'
    have "((s1, i:deliver(d), s1’) ∈ trans (pnet np n1)"
      and "net_ip_action np (i:deliver(d)) i n1 s1 s1’”
      and "s2’ = s2” by simp_all

from ‘net_ips s1 = net_tree_ips n1’ and ‘(s1, i:deliver(d), s1’) ∈ trans (pnet np n1)’
  have "net_ips s1’ = net_tree_ips n1” by (metis pnet_maintains_dom)

from nm’ [simplified ‘s’ = SubnetS s1’ s2’’ ‘s2’ = s2’]
  'net_tree_ips n1 ∩ net_tree_ips n2 = {}'
  'net_ips s1’ = net_tree_ips n1’
  'net_ips s2 = net_tree_ips n2’
have "netgmap sr s1’ = netmask (net_tree_ips n1) (σ’, snd (netgmap sr s1’))"
  by (rule netgmap_subnet_split1)

from ‘(s1, i:deliver(d), s1’) ∈ trans (pnet np n1)”
  ‘netgmap sr s1 = netmask (net_tree_ips n1) (σ, snd (netgmap sr s1))’
  ‘netgmap sr s1’ = netmask (net_tree_ips n1) (σ’, snd (netgmap sr s1’))’
  ‘s1 ∈ reachable (pnet np n1) TT’
  ‘wf_net_tree n1’
  ‘i ∈ net_ips s1’
  ‘net_ip_action np (i:deliver(d)) i n1 s1 s1’;
  have "((σ, snd (netgmap sr s1)), i:deliver(d), (σ’, snd (netgmap sr s1’))) ∈ trans (opnet onp n1)"
      by (rule IH1)

with ‘s = SubnetS s1 s2’ ‘s’ = SubnetS s1’ s2’’ ‘s2’ = s2” show ?thesis
  by simp (erule opnet_deliver1)

next
  assume "i ∈ net_ips s2"
  with ‘s = SubnetS s1 s2’ ‘s’ = SubnetS s1’ s2’‘net_ip_action np (i:deliver(d)) i (n1 || n2) s’’
    have "((s2, i:deliver(d), s2’) ∈ trans (pnet np n2)"
      and "net_ip_action np (i:deliver(d)) i n2 s2 s2’”
      and "s1’ = s1” by simp_all

from ‘net_ips s2 = net_tree_ips n2’ and ‘(s2, i:deliver(d), s2’) ∈ trans (pnet np n2)’
  have "net_ips s2’ = net_tree_ips n2” by (metis pnet_maintains_dom)

from nm’ [simplified ‘s’ = SubnetS s1’ s2’’ ‘s1’ = s1’]
  ‘net_ips s1 = net_tree_ips n1’
  ‘net_ips s2’ = net_tree_ips n2’
have "netgmap sr s2’ = netmask (net_tree_ips n2) (σ’, snd (netgmap sr s2’))”
  by (rule netgmap_subnet_split2)
from '(s2, i:deliver(d), s2') ∈ trans (pnet np n2)'
'netmap sr s2 = netmask (net_tree_ips n2) (σ, snd (netmap sr s2))'
'netmap sr s2' = netmask (net_tree_ips n2) (σ', snd (netmap sr s2'))'
's2 ∈ reachable (pnet np n2) TT'
'wf_net_tree n2'
'i ∈ net_ips s2'

have "((σ, snd (netmap sr s2)), i:deliver(d), (σ', snd (netmap sr s2'))) ∈ trans (opnet onp n2)"

by (rule IH2)

with 's = SubnetS s1 s2' s' = SubnetS s1' s2'' s1' = s1'

show thesis

by simp (erule opnet_deliver2)

qed

with 'ζ = snd (netmap sr s)'

have "((σ, ζ), i:deliver(d), (σ', snd (netmap sr s'))) ∈ trans (opnet onp n)"

by simp

moreover from "∀ j. j ≠ i → σ j = σ j' i ∈ net_ips s, 'ζ = snd (netmap sr s)'

have "∀ j. j QModelIndex ζ = net_tree_ips n (σ', j = σ j)'

ultimately have "((σ, ζ), i:deliver(d), (σ', snd (netmap sr s'))) ∈ trans (opnet onp n)

∧ (∀ j. j QModelIndex ζ = net_tree_ips n (σ', j = σ j)'

∧ netmap sr s' = netmask (net_tree_ips n) (σ', snd (netmap sr s')))" by simp

thus "∃ σ', ζ'. (σ, ζ, i:deliver(d), (σ', ζ')) ∈ trans (opnet onp n)

∧ (∀ j. j QModelIndex ζ = net_tree_ips n (σ', j = σ j)'

∧ netmap sr s' = netmask (net_tree_ips n) (σ', ζ')" by auto

qed

lemma transfer_arrive':

assumes "(s, H ¬ K:arrive(m), s') ∈ trans (pnet np n)"

and "s ∈ reachable (pnet np n) TT" and "netmap sr s = netmask (net_tree_ips n) (σ, ζ)"

and "netmap sr s' = netmask (net_tree_ips n) (σ', ζ')" and "wf_net_tree n"

shows "((σ, ζ), H ¬ K:arrive(m), (σ', ζ')) ∈ trans (opnet onp n)"

proof -

from assms(2) have "net_ips s = net_tree_ips n" ..

with assms(1) have "net_ips s' = net_tree_ips n"

by (metis pnet_maintains_dom)

from 'netmap sr s = netmask (net_tree_ips n) (σ, ζ)'

have "ζ = snd (netmap sr s)" by simp

from 'netmap sr s' = netmask (net_tree_ips n) (σ', ζ')'

have "ζ' = snd (netmap sr s')" and "netmap sr s' = netmask (net_tree_ips n) (σ', snd (netmap sr s'))" by simp_all

from assms(1-3) 'netmap sr s' = netmask (net_tree_ips n) (σ', snd (netmap sr s'))' assms(5)

have "((σ, snd (netmap sr s), H ¬ K:arrive(m), (σ', snd (netmap sr s'))) ∈ trans (opnet onp n)"

proof (induction n arbitrary: s s' ζ H K)

fix ii R, ns ns' ζ H K

assume "(ns, H ¬ K:arrive(m), ns') ∈ trans (pnet np {ii; R})"

and nsr: "ns ∈ reachable (pnet np {ii; R}) TT" and "netmap sr ns = netmask (net_tree_ips {ii; R}) (σ, ζ)"

and "netmap sr ns' = netmask (net_tree_ips {ii; R}) (σ', snd (netmap sr ns'))"

from this(1) have "(ns, H ¬ K:arrive(m), ns') ∈ node_sos (trans (np ii))"

by (simp add: node_comps)

moreover with nsr obtain s s' R where "ns = NodeS ii s R" and "ns' = NodeS ii s' R"

by (metis net_node_reachable_is_node node_arriveTE')
ultimately have "(NodeS ii s R, H¬K: arrive(m), NodeS ii s' R) ∈ node_sos (trans (np ii))"
by simp
from this(1) have "((σ, NodeS ii (snd (sr s)) R), H¬K: arrive(m), (σ', NodeS ii (snd (sr s')) R)) ∈ onode_sos (trans (onp ii))"
proof (rule node_arriveTE)
  assume "(s, receive m, s') ∈ trans (np ii)"
  and "H = {ii}"
  and "K = {}"
  moreover from 'netgmap sr ns = netmask (net_tree_ips ⟨ii; R⟩) (σ, ζ)'
  'ns = NodeS ii s R'
  have "σ ii = fst (sr s)"
  by simp (metis map_upd_Some_unfold)
  moreover from 'netgmap sr ns' = netmask (net_tree_ips ⟨ii; R⟩) (σ, ζ)'
  'ns' = NodeS ii s' R'
  have "σ' ii = σ ii" by simp (metis option.sel)
  ultimately have "((σ, snd (sr s)), receive m, (σ', snd (sr s'))) ∈ trans (onp ii)"
  using (s, receive m, s') ∈ trans (np ii)' by (rule trans)
hence "((σ, NodeS ii (snd (sr s)) R), H¬K: arrive(m), (σ', NodeS ii (snd (sr s')) R)) ∈ onode_sos (trans (onp ii))"
  by (rule onode_arrive)
  qed

next
  assume "H = {}"
  and "s = s'" 
  and "K = {ii}" 
  moreover from 's = s'' 'netgmap sr ns' = netmask (net_tree_ips ⟨ii; R⟩) (σ, ζ)'
  'ns = NodeS ii s R' and 'ns' = NodeS ii s' R'
  have "σ' ii = σ ii" by simp (metis option.sel)
  hence "((σ, NodeS ii (snd (sr s)) R), {}¬{ii}: arrive(m), (σ', NodeS ii (snd (sr s')) R)) ∈ onode_sos (trans (onp ii))"
  by (rule onode_arrive)
  with 'H={ii}' 'K={ii}' and 's = s''
  show "((σ, NodeS ii (snd (sr s)) R), H¬K: arrive(m), (σ', NodeS ii (snd (sr s')) R)) ∈ onode_sos (trans (onp ii))"
  by simp

next
  fix n1 n2 s s' ∈ H K
  assume IH1: "\s s' ∈ H K. (s, H¬K: arrive(m), s') ∈ trans (pnet np n1) 
                ⇒ s ∈ reachable (pnet np n1) TT 
                ⇒ netmap sr s = netmask (net_tree_ips n1) (σ, ζ) 
                ⇒ netmap sr s' = netmask (net_tree_ips n1) (σ', snd (netmap sr s')) 
                ⇒ wf_net_tree n1 
                ⇒ ((σ, snd (netmap sr s)), H¬K: arrive(m), σ', snd (netmap sr s')) ∈ trans (opnet onp n1)" 
  and IH2: "\s s' ∈ H K. (s, H¬K: arrive(m), s') ∈ trans (pnet np n2) 
                ⇒ s ∈ reachable (pnet np n2) TT 
                ⇒ netmap sr s = netmask (net_tree_ips n2) (σ, ζ) 
                ⇒ netmap sr s' = netmask (net_tree_ips n2) (σ', snd (netmap sr s')) 
                ⇒ wf_net_tree n2 
                ⇒ ((σ, snd (netmap sr s)), H¬K: arrive(m), σ', snd (netmap sr s')) ∈ trans (opnet onp n2)" 
  and "(s, H¬K: arrive(m), s') ∈ trans (pnet np (n1 || n2))" 
  and sr: "s ∈ reachable (pnet np (n1 || n2)) TT" 
  and nm: "netmap sr s = netmask (net_tree_ips (n1 || n2)) (σ, ζ)" 
  and nm': "netmap sr s' = netmask (net_tree_ips (n1 || n2)) (σ', snd (netmap sr s'))" 
  and "wf_net_tree (n1 || n2)"
from this(3) have 

\[(s, H \cong K: \text{arrive}(m), s') \in \text{pnet_sos} (\text{trans (pnet np n1)}) \]

\[(\text{trans (pnet np n2)})\]

by simp 

thus ""((\sigma, \text{snd (netmap sr s))}, H \cong K: \text{arrive}(m), (\sigma', \text{snd (netmap sr s'))}) \in \text{trans (opnet onp (n1 || n2)})"" 

proof (rule partial_arriveTE)

fix s1 s1' s2 s2' H1 H2 K1 K2

assume "s = SubnetS s1 s2"

and "s' = SubnetS s1' s2'"

and tr1: "(s1, H1 \cong K1: \text{arrive}(m), s1') \in \text{trans (pnet np n1)}"

and tr2: "(s2, H2 \cong K2: \text{arrive}(m), s2') \in \text{trans (pnet np n2)}"

and "H = H1 \cup H2"

and "K = K1 \cup K2"

from 'wf_net_tree (n1 || n2)' have "wf_net_tree n1" 

and "wf_net_tree n2"

and "net_ips s1 \cap net_tree_ips n2 = {}" by auto

from sr [simplified 's = SubnetS s1 s2'] have "s1 \in \text{reachable (pnet np n1)} TT" 

by (rule subnet_reachable(1))

hence "net_ips s1 = net_tree_ips n1" by (rule pnet_net_ips_net_tree_ips)

with tr1 have "net_ips s1' = net_tree_ips n1" by (metis pnet_maintains_dom)

from sr [simplified 's = SubnetS s1 s2'] have "s2 \in \text{reachable (pnet np n2)} TT" 

by (rule subnet_reachable(2))

hence "net_ips s2 = net_tree_ips n2" by (rule pnet_net_ips_net_tree_ips)

with tr2 have "net_ips s2' = net_tree_ips n2" by (metis pnet_maintains_dom)

from '(s1, H1 \cong K1: \text{arrive}(m), s1') \in \text{trans (pnet np n1)}'

's1 \in \text{reachable (pnet np n1)} TT'

have "(((\sigma, \text{snd (netmap sr s1))}, H1 \cong K1: \text{arrive}(m), (\sigma', \text{snd (netmap sr s1'))}) \in \text{trans (opnet onp n1)}" 

proof (rule IH1 [OF _ _ _ _ 'wf_net_tree n1'])

from nm [simplified 's = SubnetS s1 s2']

'net_tree_ips n1 \cap net_tree_ips n2 = {}'

'net_ips s1 = net_tree_ips n1'

'net_ips s2 = net_tree_ips n2'

show "netmap sr s1 = netmask (net_tree_ips n1) (\sigma, \text{snd (netmap sr s1)})"

by (rule netmap_subnet_split1)

next

from nm' [simplified 's' = SubnetS s1' s2'']

'net_tree_ips n1 \cap net_tree_ips n2 = {}'

'net_ips s1' = net_tree_ips n1'

'net_ips s2' = net_tree_ips n2'

show "netmap sr s1' = netmask (net_tree_ips n1) (\sigma', \text{snd (netmap sr s1')})"

by (rule netmap_subnet_split1)

qed

moreover from '(s2, H2 \cong K2: \text{arrive}(m), s2') \in \text{trans (pnet np n2)}'

's2 \in \text{reachable (pnet np n2)} TT'

have "(((\sigma, \text{snd (netmap sr s2))}, H2 \cong K2: \text{arrive}(m), (\sigma', \text{snd (netmap sr s2'))}) \in \text{trans (opnet onp n2)}" 

proof (rule IH2 [OF _ _ _ _ 'wf_net_tree n2'])

from nm [simplified 's = SubnetS s1 s2']

'net_ips s1 = net_tree_ips n1'

'net_ips s2 = net_tree_ips n2'

show "netmap sr s2 = netmask (net_tree_ips n2) (\sigma, \text{snd (netmap sr s2)})"

by (rule netmap_subnet_split2)

next

from nm' [simplified 's' = SubnetS s1' s2'']

'net_ips s1' = net_tree_ips n1'

'net_ips s2' = net_tree_ips n2'

show "netmap sr s2' = netmask (net_tree_ips n2) (\sigma', \text{snd (netmap sr s2')})"

by (rule netmap_subnet_split2)
ultimately show "((σ, snd (netgmap sr s)), H¬K:arrive(m), (σ', snd (netgmap sr s')))
∈ trans (opnet onp (n1 || n2))" using 's = SubnetS s1 s2' 's' = SubnetS s1' s2'' 'H = H1 ∪ H2' 'K = K1 ∪ K2'
by simp (rule opnet_sos.opnet_arrive)

lemma transfer_arrive:
assumes "(s, H¬K:arrive(m), s') ∈ trans (pnet np n)"
and "s ∈ reachable (pnet np n) TT"
and "netgmap sr s = netmask (net_tree_ips n) (σ, ζ)"
and "wf_net_tree n"
obtains σ' ζ' where "((σ, ζ), H¬K:arrive(m), (σ', ζ')) ∈ trans (opnet onp n)"
and "∀ j. j /∈ net_ips ζ →→ σ' j = σ j"
and "netgmap sr s' = netmask (net_tree_ips n) (σ', ζ')"
proof atomize_elim
def σ' ≡ "λi. if i ∈ net_tree_ips n then the (fst (netgmap sr s') i) else σ i"
from assms(2) have "net_ips s = net_tree_ips n"
by (rule pnet_net_ips_net_tree_ips)
with assms(1) have "net_ips s' = net_tree_ips n"
by (metis pnet_maintains_dom)
have "netgmap sr s' = netmask (net_tree_ips n) (σ', snd (netgmap sr s'))"
proof (rule prod_eqI)
from 'net_ips s' = net_tree_ips n'
show "fst (netgmap sr s') = fst (netmask (net_tree_ips n) (σ', snd (netgmap sr s')))"
unfolding σ'_def by - (rule ext, clarsimp)
qed simp
moreover with assms(1-3)
have "((σ, ζ), H¬K:arrive(m), (σ', snd (netgmap sr s'))) ∈ trans (opnet onp n)"
using 'wf_net_tree n' by (rule transfer_arrive')

moreover have "∀ j. j /∈ net_ips ζ →→ σ' j = σ j"
proof -
have "∀ j. j /∈ net_tree_ips n →→ σ' j = σ j" unfolding σ'_def by simp
with assms(3) and 'net_ips s = net_tree_ips n'
show ?thesis
by clarsimp (metis (mono_tags) net_ips_netgmap snd_conv)
qed
ultimately show "∃ σ' ζ'. ((σ, ζ), H¬K:arrive(m), (σ', ζ')) ∈ trans (opnet onp n)
∧ (∀ j. j /∈ net_ips ζ →→ σ' j = σ j)
∧ netgmap sr s' = netmask (net_tree_ips n) (σ', ζ')" by auto

lemma transfer_cast:
assumes "(s, mR:*cast(m), s') ∈ trans (pnet np n)"
and "s ∈ reachable (pnet np n) TT"
and "netgmap sr s = netmask (net_tree_ips n) (σ, ζ)"
and "wf_net_tree n"
obtains σ' ζ' where "((σ, ζ), mR:*cast(m), (σ', ζ')) ∈ trans (opnet onp n)"
and "∀ j. j /∈ net_tree_ips ζ →→ σ' j = σ j"
and "netgmap sr s' = netmask (net_tree_ips n) (σ', ζ')"
proof atomize_elim
def σ' ≡ "λi. if i ∈ net_tree_ips n then the (fst (netgmap sr s') i) else σ i"
from assms(2) have "net_ips s = net_tree_ips n" ..
with assms(1) have "net_ips s' = net_tree_ips n"
  by (metis pnet_maintains_dom)
have "netgmap sr s' = netmask (net_tree_ips n) (σ', snd (netgmap sr s'))"
proof (rule prod_eqI)
  from 'net_ips s' = net_tree_ips n'
  show "∀ j. j ∈ net_ips (snd (netgmap sr s)) → σ' j = σ j"
  unfolding σ'_def
  by (rule ext, clarsimp simp add: some_the_fst_netgmap)
qed simp

from 'net_ips s' = net_tree_ips n' and 'net_ips s = net_tree_ips n'
have "∀ j. j ∈ net_ips (snd (netgmap sr s)) → σ j = σ' j" by simp

from assms(1-3) 'netgmap sr s' = netmask (net_tree_ips n) (σ', ζ)
have "((σ, ζ), rmask (net_tree_ips n) (σ, ζ)) ∈ trans (opnet onp n)" by simp
proof (rule node_castTE)
  assume "((s, broadcast m, s'), R) ∈ trans (np ii)"
  and "R = mR" and "mR = R ∩ D"
  from 'σ ii = fst (sr s)' 'σ' ii = fst (sr s')' and this(1)
  have "((σ, snd (sr s)), broadcast m, (σ', snd (sr s'))) ∈ trans (np ii)"
  by (rule trans)
  hence "((σ, Nodes ii (snd (sr s)) R), (R ∩ D):*cast(m), (σ', Nodes ii (snd (sr s')) R))
     ∈ onode_sos (trans (onp ii))" by simp
with 'R=mR' show "((σ, Nodes ii (snd (sr s)) R), mR:*cast(m), (σ', Nodes ii (snd (sr s')) R))
     ∈ onode_sos (trans (onp ii))" by simp

next
  fix D
  assume "(s, groupcast D m, s') ∈ trans (np ii)"
  and "mR = R ∩ D"
  from 'σ ii = fst (sr s)' 'σ' ii = fst (sr s')' and this(1)
  have "((σ, snd (sr s)), groupcast D m, (σ', snd (sr s'))) ∈ trans (onp ii)"
  by (rule trans)
  hence "((σ, Nodes ii (snd (sr s)) R), (R ∩ D):*cast(m), (σ', Nodes ii (snd (sr s')) R))
     ∈ onode_sos (trans (onp ii))" by simp
by (rule node_gcast)
with \( mR = R \cap D \) show 
\[
((\sigma, \text{NodeS ii (snd (sr s)) R}), mR:*\text{cast}(m), (\sigma', \text{NodeS ii (snd (sr s')) R})) \in \text{onode_sos (trans (onp ii))}
\]

by simp

next
fix \( d \)
assume 
\[
(s, \text{unicast d m, s'}) \in \text{trans (np ii)}
\]
and "\( d \in R \)"
and "\( mR = \{d\} \)"
from \( \sigma ii = \text{fst (sr s)} \) ' \( \sigma' ii = \text{fst (sr s')} \) ' and this(1)
have 
\[
(((\sigma, \text{snd (sr s)}), \text{unicast d m, (\sigma', \text{snd (sr s'))})) \in \text{trans (onp ii)}
\]
by (rule trans)

hence 
\[
(((\sigma, \text{NodeS ii (snd (sr s)) R}), \{d\}:*\text{cast}(m), (\sigma', \text{NodeS ii (snd (sr s')) R})) \in \text{onode_sos (trans (onp ii))}
\]
using "\( d \in R \)" by (rule onode_ucast)

show 
\[
((\sigma, \text{NodeS ii (snd (sr s)) R}), mR:*\text{cast}(m), (\sigma', \text{NodeS ii (snd (sr s')) R})) \in \text{onode_sos (trans (onp ii))}
\]
by simp

qed

with 'ns = NodeS ii s R' 'ns' = NodeS ii s' R'

show 
\[
((\sigma, \text{snd (netgmap sr ns)}), mR:*\text{cast}(m), (\sigma', \text{snd (netgmap sr ns')})) \in \text{trans (opnet onp (ii; R))}
\]
by (simp add: onode_comps)

next
fix \( n1 \) \( n2 \) \( s \) \( s' \) \( \zeta \) 
assume IH1: "\( \forall s s' \zeta mR. (s, mR:*\text{cast}(m), s') \in \text{trans (pnet np n1)} \Rightarrow s \in \text{reachable (pnet np n1) TT} \)
\( \Rightarrow \) netgmap sr s = netmask (net_tree_ips n1) (\( \sigma, \zeta \))
\( \Rightarrow \) netgmap sr s' = netmask (net_tree_ips n1) (\( \sigma', \text{snd (netgmap sr s')} \))
\( \Rightarrow \) wf_net_tree n1
\( \Rightarrow \) \(((\sigma, \text{snd (netgmap sr s)}), mR:*\text{cast}(m), (\sigma', \text{snd (netgmap sr s')})) \in \text{trans (opnet onp n1)}" 

and IH2: "\( \forall s s' \zeta mR. (s, mR:*\text{cast}(m), s') \in \text{trans (pnet np n2)} \Rightarrow s \in \text{reachable (pnet np n2) TT} \)
\( \Rightarrow \) netgmap sr s = netmask (net_tree_ips n2) (\( \sigma, \zeta \))
\( \Rightarrow \) netgmap sr s' = netmask (net_tree_ips n2) (\( \sigma', \text{snd (netgmap sr s')} \))
\( \Rightarrow \) wf_net_tree n2
\( \Rightarrow \) \(((\sigma, \text{snd (netgmap sr s)}), mR:*\text{cast}(m), (\sigma', \text{snd (netgmap sr s')})) \in \text{trans (opnet onp n2)}" 

and "\( (s, mR:*\text{cast}(m), s') \in \text{trans (pnet np (n1 || n2))} \)"
and sr: "\( s \in \text{reachable (pnet np (n1 || n2)) TT} \)"
and nm: "netgmap sr s = netmask (net_tree_ips (n1 || n2)) (\( \sigma, \zeta \))"
and nm': "netgmap sr s' = netmask (net_tree_ips (n1 || n2)) (\( \sigma', \text{snd (netgmap sr s')} \))"
and "\( \text{wf_net_tree (n1 || n2)} \)"

from this(3) have "\( (s, mR:*\text{cast}(m), s') \in \text{pnet_sos (trans (pnet np n1)) (trans (pnet np n2))} \)" by simp

then obtain \( s1 \) \( s1' \) \( s2 \) \( s2' \) \( \text{H K} \) 
where "\( s = \text{SubnetS s1 s2} \)"
and "\( s' = \text{SubnetS s1' s2'} \)"
and "\( \text{H} \subseteq mR \)"
and "\( \text{K} \cap mR = \{\} \)"
and trtr: "\( ((s1, mR:*\text{cast}(m), s1')) \in \text{trans (pnet np n1)} \)
\( \wedge \) \( (s2, \text{H-K: arrive(m), s2'}) \in \text{trans (pnet np n2)} \) 
\( \vee \) \( ((s1, \text{H-K: arrive(m), s1'}) \in \text{trans (pnet np n1)} \)
\( \wedge \) \( (s2, mR:*\text{cast}(m), s2') \in \text{trans (pnet np n2)} \)"
by (rule partial_castTE) metis+

from 'wf_net_tree (n1 || n2)' have "\( \text{wf_net_tree n1} \)"
and "\( \text{wf_net_tree n2} \)"
and "\( \text{net_tree_ips n1 \cap net_tree_ips n2 = \{\}} \)" by auto

from sr [simplified 's = SubnetS s1 s2'] have "\( s1 \in \text{reachable (pnet np n1) TT} \)"
by (rule subnet_reachable(1))

hence "\( \text{net_ips s1 = net_tree_ips n1} \)" by (rule pnet_net_ips_net_tree_ips)
with \( \text{trtr} \) have "\( \text{net_ips } s1' = \text{net_tree_ips } n1 \)" by (metis \( \text{pnet_maintains_dom} \))

from \( \text{sr} \) [simplified \( 's = \text{SubnetS } s1 \ s2' \)] have "\( \text{reachable } (\text{pnet } n2) \ TT'\)" by (rule \( \text{subnet_reachable(2)} \))

hence "\( \text{net_ips } s2' = \text{net_tree_ips } n2 \)" by (rule \( \text{pnet_net_ips_net_tree_ips} \))

with \( \text{trtr} \) have "\( \text{net_ips } s2' = \text{net_tree_ips } n2 \)" by (metis \( \text{pnet_maintains_dom} \))

from \( \text{nm} \) [simplified \( 's = \text{SubnetS } s1 \ s2' \)]

\[
\begin{align*}
\text{net_tree_ips } n1 \cap \text{net_tree_ips } n2 &= \emptyset' \\
\text{net_ips } s1 &= \text{net_tree_ips } n1' \\
\text{net_ips } s2 &= \text{net_tree_ips } n2'
\end{align*}
\]

have "\( \text{netgmap } sr \ s1 = \text{netmask } (\text{net_tree_ips } n1) \ (\sigma', \text{snd } (\text{netgmap } sr \ s1'))\)"

by (rule \( \text{netgmap_subnet_split1} \))

from \( \text{nm'} \) [simplified \( 's = \text{SubnetS } s1' \ s2'' \)]

\[
\begin{align*}
\text{net_tree_ips } n1 \cap \text{net_tree_ips } n2 &= \emptyset' \\
\text{net_ips } s1' &= \text{net_tree_ips } n1' \\
\text{net_ips } s2' &= \text{net_tree_ips } n2'
\end{align*}
\]

have "\( \text{netgmap } sr \ s1' = \text{netmask } (\text{net_tree_ips } n1) \ (\sigma', \text{snd } (\text{netgmap } sr \ s1'))\)"

by (rule \( \text{netgmap_subnet_split1} \))

from \( \text{nm} \) [simplified \( 's = \text{SubnetS } s1 \ s2' \)]

\[
\begin{align*}
\text{net_ips } s1 &= \text{net_tree_ips } n1' \\
\text{net_ips } s2 &= \text{net_tree_ips } n2'
\end{align*}
\]

have "\( \text{netgmap } sr \ s2 = \text{netmask } (\text{net_tree_ips } n2) \ (\sigma', \text{snd } (\text{netgmap } sr \ s2'))\)"

by (rule \( \text{netgmap_subnet_split2} \))

from \( \text{nm'} \) [simplified \( 's = \text{SubnetS } s1' \ s2'' \)]

\[
\begin{align*}
\text{net_ips } s1' &= \text{net_tree_ips } n1' \\
\text{net_ips } s2' &= \text{net_tree_ips } n2'
\end{align*}
\]

have "\( \text{netgmap } sr \ s2' = \text{netmask } (\text{net_tree_ips } n2) \ (\sigma', \text{snd } (\text{netgmap } sr \ s2'))\)"

by (rule \( \text{netgmap_subnet_split2} \))

from \( \text{trtr} \) show "\( (\sigma, \text{snd } (\text{netgmap } sr \ s')) \), \( \text{mR:*cast(m)}, (\sigma', \text{snd } (\text{netgmap } sr \ s')) \)" in trans \( (\text{opnet onp } n1 \parallel n2) \)"

proof (elim disjE conjE)

assume "\( (s1, \text{mR:*cast(m)}, s1') \in \text{trans } (\text{pnet } np n1) \)"

and "\( (s2, \neg K: \text{arrive(m)}, s2') \in \text{trans } (\text{pnet } np n2) \)"

from "\( (s1, \text{mR:*cast(m)}, s1') \in \text{trans } (\text{pnet } np n1) \)"

\[
\begin{align*}
s1 &\in \text{reachable } (\text{pnet } np n1) \ TT' \\
\text{netgmap } sr \ s1 &= \text{netmask } (\text{net_tree_ips } n1) \ (\sigma, \text{snd } (\text{netgmap } sr \ s1'))' \\
\text{netgmap } sr \ s1' &= \text{netmask } (\text{net_tree_ips } n1) \ (\sigma', \text{snd } (\text{netgmap } sr \ s1'))' \\
\text{wf}_n't = \text{net_tree } n1'
\end{align*}
\]

have "\( (((\sigma', \text{snd } (\text{netgmap } sr \ s1')), \text{mR:*cast(m)}, (\sigma', \text{snd } (\text{netgmap } sr \ s1')))) \in \text{trans } (\text{opnet onp } n1) \)"

by (rule \( \text{IH1} \))

moreover from "\( (s2, \neg K: \text{arrive(m)}, s2') \in \text{trans } (\text{pnet } np n2) \)"

\[
\begin{align*}
s2 &\in \text{reachable } (\text{pnet } np n2) \ TT' \\
\text{netgmap } sr \ s2 &= \text{netmask } (\text{net_tree_ips } n2) \ (\sigma, \text{snd } (\text{netgmap } sr \ s2'))' \\
\text{netgmap } sr \ s2' &= \text{netmask } (\text{net_tree_ips } n2) \ (\sigma', \text{snd } (\text{netgmap } sr \ s2'))' \\
\text{wf}_n't = \text{net_tree } n2'
\end{align*}
\]

have "\( (((\sigma, \text{snd } (\text{netgmap } sr \ s2)), \text{H:*cast(m)}, (\sigma', \text{snd } (\text{netgmap } sr \ s2')))) \in \text{trans } (\text{opnet onp } n2) \)"

by (rule \( \text{transfer_arrive'} \))

ultimately have "\( (((\sigma, \text{SubnetS } (\text{snd } (\text{netgmap } sr \ s1)) \ (\text{snd } (\text{netgmap } sr \ s2))), \text{mR:*cast(m)}, (\sigma', \text{SubnetS } (\text{snd } (\text{netgmap } sr \ s1'))) \ (\text{snd } (\text{netgmap } sr \ s2'))) \)"

in \( \text{opnet sos } (\text{trans } (\text{opnet onp } n1)) \) \( (\text{trans } (\text{opnet onp } n2)) \)"

using "\( H \subseteq \text{mR} \)" and "\( K \cap \text{mR} = \{\} \)" by (rule \( \text{opnet sos.intros(1)} \))

with "\( s = \text{SubnetS } s1 \ s2' \)" \( (s' = \text{SubnetS } s1' \ s2'') \) show \( ? \text{thesis by simp} \)

next

assume "\( (s1, \neg K: \text{arrive(m)}, s1') \in \text{trans } (\text{pnet np n1}) \)"

and "\( (s2, \text{mR:*cast(m)}, s2') \in \text{trans } (\text{pnet np n2}) \)"
from \((s_1, H \land K : \text{arrive}(m), s_1') \in \text{trans}(\text{pnet np n1})\)
's_1 \in \text{reachable}(\text{pnet np n1}) \land T\text{T}'
'\text{netmap sr } s_1 = \text{netmask}(\text{net_tree_ips n1}) (\sigma, \text{snd}(\text{netmap sr } s_1))'
'\text{netmap sr } s_1' = \text{netmask}(\text{net_tree_ips n1}) (\sigma', \text{snd}(\text{netmap sr } s_1'))'
'\text{wf_net_tree n1}'
'\text{have } (((\sigma, \text{snd}(\text{netmap sr } s_1)), H \land K : \text{arrive}(m), (\sigma', \text{snd}(\text{netmap sr } s_1')))) \in \text{trans}(\text{opnet onp n1})''
by \text{(rule transfer_arrive)'
moreover from \((s_2, mR : \ast \text{cast}(m), s_2') \in \text{trans}(\text{pnet np n2})\)
's_2 \in \text{reachable}(\text{pnet np n2}) \land T\text{T}'
'\text{netmap sr } s_2 = \text{netmask}(\text{net_tree_ips n2}) (\sigma, \text{snd}(\text{netmap sr } s_2))'
'\text{netmap sr } s_2' = \text{netmask}(\text{net_tree_ips n2}) (\sigma', \text{snd}(\text{netmap sr } s_2'))'
'\text{wf_net_tree n2}'
'\text{have } (((\sigma, \text{snd}(\text{netmap sr } s_2)), mR : \ast \text{cast}(m), (\sigma', \text{snd}(\text{netmap sr } s_2'))) \in \text{trans}(\text{opnet onp n2})''
by \text{(rule IH2)'
ultimately have \(((\sigma, \text{SubnetS } (\text{snd}(\text{netmap sr } s_1)) (\text{snd}(\text{netmap sr } s_2))), mR : \ast \text{cast}(m), (\sigma', \text{SubnetS } (\text{snd}(\text{netmap sr } s_1')) (\text{snd}(\text{netmap sr } s_2')))) \in \text{opnet_sos} (\text{trans}(\text{opnet onp n1})) (\text{trans}(\text{opnet onp n2}))''
using \('H \subseteq mR' \land 'K \cap mR = \emptyset' \text{ by \text{(rule opnet_sos.intros(2))' with } s = \text{SubnetS } s_1 s_2' 's' = \text{SubnetS } s_1' s_2'' \text{ show } \text{thesis by simp' qd' qd' with } '\zeta = \text{snd}(\text{netmap sr } s)' \text{ have } (((\sigma, \zeta), mR : \ast \text{cast}(m), (\sigma', \text{snd}(\text{netmap sr } s')))) \in \text{trans}(\text{opnet onp n})''
by simp'
moreover from \('\forall j \in \text{net_ips } (\text{snd}(\text{netmap sr } s)) \rightarrow \sigma' j = \sigma j' \zeta = \text{snd}(\text{netmap sr } s)'\'
'\text{have } '\forall j \in \text{net_ips } \zeta \rightarrow \sigma' j = \sigma j' \text{ by simp' qd' qd' note } \text{netmap sr } s' = \text{netmask}(\text{net_tree_ips n}) (\sigma', \text{snd}(\text{netmap sr } s'))''
ultimately show \(((\sigma', \zeta'), ((\sigma, \zeta), mR : \ast \text{cast}(m), (\sigma', \zeta'))) \in \text{trans}(\text{opnet onp n})''
\land '\forall j \in \text{net_ips } \zeta \rightarrow \sigma' j = \sigma j' \land \text{netmap sr } s' = \text{netmask}(\text{net_tree_ips n}) (\sigma', \zeta')''
by \text{auto' qd'
lemma transfer_pnet_action:
assumes 's \in \text{reachable}(\text{pnet np n}) \land T\text{T}''
and '\text{netmap sr } s = \text{netmask}(\text{net_tree_ips n}) (\sigma, \zeta)''
and '\text{wf_net_tree n}''
obtains '\sigma' \zeta' where '(((\sigma, \zeta), a, (\sigma', \zeta'))) \in \text{trans}(\text{opnet onp n})''
\land '\forall j \in \text{net_ips } \zeta \rightarrow \sigma' j = \sigma j' ''
and '\text{netmap sr } s' = \text{netmask}(\text{net_tree_ips n}) (\sigma', \zeta')''
proof \text{atomize_elim' show } '((\sigma, \zeta), a, (\sigma', \zeta')) \in \text{trans}(\text{opnet onp n})''
\land '\forall j \in \text{net_ips } \zeta \rightarrow \sigma' j = \sigma j' ''
\land \text{netmap sr } s' = \text{netmask}(\text{net_tree_ips n}) (\sigma', \zeta')''
proof \text{(cases a)'
case node_cast'
with assms(4) show \text{thesis by \text{(auto elim!}: transfer_cast \text[OF assms(1-3)])'}
next'
case node_deliver'
with assms(4) show \text{thesis by \text{(auto elim!}: transfer_deliver \text[OF assms(1-3)])'}
next'
case node_arrive'
with assms(4) show \text{thesis by \text{(auto elim!}: transfer_arrive \text[OF assms(1-3)])'}
next'
case node_connect'
with assms(4) show \text{thesis
by (auto elim!: transfer_connect [OF _ assms(1-3)])

next
  case node_disconnect
  with assms(4) show ?thesis
    by (auto elim!: transfer_disconnect [OF _ assms(1-3)])

next
  case node_newpkt
  with assms(4) have False by (metis pnet_never_newpkt)
  thus ?thesis ..

next
  case node_tau
  with assms(4) show ?thesis
    by (auto elim!: transfer_tau [OF _ assms(1-3), simplified])
qed

lemma transfer_action_pnet_closed:
  assumes "(s, a, s') \in trans (closed (pnet np n))"
  obtains a' where "((s, a), a', (s', \zeta')) \in trans (pnet np n);
  (\forall j. j \notin net_ips \zeta \rightarrow \sigma' j = \sigma j) \rightarrow ((\sigma, \zeta), a, (\sigma', \zeta')) \in trans (oclosed (opennet onp n))"
proof (atomize_elim)
  from assms have "(s, a, s') \in cnet_sos (trans (pnet np n))" by simp
  thus \( \exists a'. (s, a', s') \in trans (pnet np n)
  \land (\forall \sigma \zeta \sigma' \zeta'. ((\sigma, \zeta), a', (\sigma', \zeta')) \in trans (pnet np n);
  (\forall j. j \notin net_ips \zeta \rightarrow \sigma' j = \sigma j) \rightarrow ((\sigma, \zeta), a, (\sigma', \zeta')) \in trans (oclosed (opennet onp n)))"
proof cases
  case (cnet_cast R m)
  thus ?thesis
    by (auto intro!: exI [where x="R:*cast(m)" dest!: ocnet_cast])
qed (auto intro!: ocnet_sos.intros [simplified])

qed

lemma transfer_action:
  assumes "s \in reachable (closed (pnet np n)) TT" and "netgmap sr s = netmask (net_tree_ips n) (\sigma, \zeta)" and "wf_net_tree n"
  and "(s, a, s') \in trans (closed (pnet np n))"
  obtains \( \sigma' \zeta' \) where "((\sigma, \zeta), a, (\sigma', \zeta')) \in trans (oclosed (opennet onp n))"
    and "netgmap sr s' = netmask (net_tree_ips n) (\sigma', \zeta')"
proof atomize_elim
  from assms(1) have "s \in reachable (pnet np n) TT" ..
  from assms(4)
  show "\exists \sigma' \zeta'. ((\sigma, \zeta), a, (\sigma', \zeta')) \in trans (oclosed (opennet onp n))
    \land netgmap sr s' = netmask (net_tree_ips n) (\sigma', \zeta')"
  by (cases a)
    (elim transfer_action_pnet_closed transfer_action [OF 's \in reachable (pnet np n) TT' assms(2-3)]?,
    (auto intro!: exI)[1]+)
qed

lemma pnet_reachable_transfer':
  assumes "wf_net_tree n" and "s \in reachable (closed (pnet np n)) TT"
  shows "netgmap sr s \in netmask (net_tree_ips n) \ord reachable (oclosed (opennet onp n)) (\lambda_ _ _. True)
  \is " _ \in ?f \ord reachable n"
using assms(2) proof induction
  fix s
  assume "s \in init (closed (pnet np n))"
  hence "s \in init (pnet np n)" by simp
  with "wf_net_tree n" have "netgmap sr s \in netmask (net_tree_ips n) \ord init (opennet onp n)"
  by (rule init_pnet_opnet)
hence \( \text{"netgmap sr s} \in \text{netmask (net_tree_ips n) } \text{'} \text{init (oclosed (opnet onp n))\"} \)
by simp

moreover have \( \text{"netmask (net_tree_ips n) } \text{'} \text{init (oclosed (opnet onp n))} \subseteq \text{netmask (net_tree_ips n) } \text{'} \text{?oreachable n\"} \)
by (intro image_mono subsetI) (rule oreachable_init)

ultimately show \( \text{"netgmap sr s} \in \text{netmask (net_tree_ips n) } \text{'} \text{?oreachable n\"} \)
by (rule set_rev_mp)

next
fix \( s, a, s' \)
assume \( \text{"s} \in \text{reachable (closed (pnet np n)) TT}\"
and \( \text{"netgmap sr s} \in \text{netmask (net_tree_ips n) } \text{'} \text{?oreachable n}\"
and \( \text{"(s, a, s')} \in \text{trans (closed (pnet np n))}\"
from this(2) obtain \( \sigma, \zeta \) where \( \text{"netgmap sr s} = \text{netmask (net_tree_ips n) (\sigma, \zeta)}\"
and \( \text{"(\sigma, \zeta)} \in \text{?oreachable n}\"
by clarsimp
from \( \text{\textit{s} \in \text{reachable (closed (pnet np n)) TT this(1) \text{'} \text{\textit{wf_net_tree n}}} \)
and \( \text{"(s, a, s')} \in \text{trans (closed (pnet np n))}\)
obtain \( \sigma', \zeta' \) where \( \text{"((\sigma, \zeta), a, (\sigma', \zeta')}\in \text{trans (oclosed (opnet onp n))}\"
and \( \text{"netgmap sr s'} \in \text{netmask (net_tree_ips n) } \text{'} \text{?oreachable n}\"
by (rule oreachable_local) simp
with \( \text{"netgmap sr s'} \in \text{netmask (net_tree_ips n) } \text{'} \text{?oreachable n\"} \)
by (rule image_eqI) qed

definition
someinit :: \( \text{nat } \Rightarrow \text{'g}\)
where
"\text{someinit i} \equiv \text{SOME x. x} \in \text{(fst o sr) } \text{'} \text{init (np i)}\"

definition
initmissing :: \( \text{((nat } \Rightarrow \text{'g option) } \times \text{'a)} \Rightarrow \text{(nat } \Rightarrow \text{'g)} \times \text{'a}\)
where
"\text{initmissing \sigma} = \text{(\lambda i. case (fst \sigma) i of None } \Rightarrow \text{someinit i | Some s } \Rightarrow \text{s, snd \sigma)}\"

lemma initmissing_def':
"\text{initmissing = apfst (default someinit)}"
by (auto simp add: initmissing_def default_def)

lemma netmask_initmissing_netgmap:
"\text{netmask (net_ips s) (initmissing (netgmap sr s))} = \text{netgmap sr s}\"
proof (intro prod_eql ext)
fix \( i \)
show \( \text{\"\text{fst (netmask (net_ips s) (initmissing (netgmap sr s))} i = \text{fst (netgmap sr s) i\"}} \)
unfolding initmissing_def by (clarsimp split: option.split)
qed (simp add: initmissing_def)

lemma snd_initmissing [simp]:
"\text{snd (initmissing x)} = \text{snd x}\"
using assms unfolding initmissing_def by simp

lemma initmissing_snd_netgmap [simp]:
assumes \( \text{"initmissing (netgmap sr s) = (\sigma, \zeta)\"} \)
shows \( \text{"snd (netgmap sr s) = \zeta\"} \)
using assms unfolding initmissing_def by simp

lemma in_net_ips_fst_init_missing [simp]:
assumes \( \text{"i} \in \text{net_ips s}\"
shows \( \text{"\text{fst (initmissing (netgmap sr s)) i = the (fst (netgmap sr s) i)\"}} \)
using assms unfolding initmissing_def by (clarsimp split: option.split)

lemma not_in_net_ips_fst_init_missing [simp]:
assumes "i ∉ net_ips s"
  shows "fst (initmissing (netgmap sr s)) i = someinit i"
using assms unfolding initmissing_def by (clarsimp split: option.split)

lemma initmissing_orreachable_netmask [elim]:
  assumes "initmissing (netgmap sr s) ∈ orreachable (oclosed (opnet onp n)) (λ_. __. True) U"
  (is "_; _ ∈ ?orreachable n")
  and "net_ips s = net_tree_ips n"
  shows "netgmap sr s ∈ netmask (net_tree_ips n) ' ?orreachable n"
proof -
  obtain σ ζ where "initmissing (netgmap sr s) = (σ, ζ)" by (metis surj_pair)
  with assms(1) have "(σ, ζ) ∈ ?orreachable n" by simp

have "netgmap sr s = netmask (net_ips s) (σ, ζ)"
proof (intro prod_eqI ext)
  fix i
  show "fst (netgmap sr s) i = fst (netmask (net_ips s) (σ, ζ)) i"
  proof (cases "i ∈ net_ips s")
    assume "i ∈ net_ips s"
    hence "fst (initmissing (netgmap sr s)) i = the (fst (netgmap sr s) i)"
    by (rule in_net_ips_fst_init_missing)
    moreover from 'i ∈ net_ips s' have "Some (the (fst (netgmap sr s) i)) = fst (netgmap sr s) i"
    by (rule some_the_fst_netgmap)
    ultimately show "fst (netgmap sr s) i = fst (netmask (net_ips s) (σ, ζ)) i"
  qed

  next
    from 'initmissing (netgmap sr s) = (σ, ζ)'
    show "snd (netgmap sr s) = snd (netmask (net_ips s) (σ, ζ))"
    by simp

  qed

  with assms(2) have "netgmap sr s = netmask (net_tree_ips n) (σ, ζ)" by simp
  moreover from "(σ, ζ) ∈ ?orreachable n'
  have "netmask (net_ips s) (σ, ζ) ∈ netmask (net_ips s) ' ?orreachable n" by (rule imageI)
  ultimately show "true"
  by (simp only: assms(2))
qed

lemma pnet_reachable_transfer:
  assumes "wf_net_tree n"
  and "s ∈ reachable (closed (pnet np n)) TT"
  shows "initmissing (netgmap sr s) ∈ orreachable (oclosed (opnet onp n)) (λ_. __. True) U"
  (is "_; _ ∈ ?orreachable n")
using assms(2) proof induction
  fix s
  assume "s ∈ init (closed (pnet np n))"
  hence "s ∈ init (pnet np n)" by simp

  from 'wf_net_tree n' have "initmissing (netgmap sr s) ∈ init (opnet onp n)"
  proof (rule init_lifted [THEN set_mp], intro CollectI exI conjI allI)
    show "initmissing (netgmap sr s) = (fst (initmissing (netgmap sr s)), snd (netgmap sr s))"
    by (metis snd_initmissing surjective_pairing)
  next
    from 's ∈ init (pnet np n)' show "s ∈ init (pnet np n)" ..
  next
    fix i
    show "if i ∈ net_tree_ips n
    then (fst (initmissing (netgmap sr s))) i = the (fst (netgmap sr s) i)
    else (fst (initmissing (netgmap sr s))) i ∈ (fst o sr) ' init (np i)"
    proof (cases "i ∈ net_tree_ips n", simp_all only: if_True if_False)
      assume "i ∈ net_tree_ips n"
      with 's ∈ init (pnet np n)' have "i ∈ net_ips s" ..
      thus "fst (initmissing (netgmap sr s)) i = the (fst (netgmap sr s) i)" by simp
  qed
next
assume "i \not\in net_tree_ips n"
with 's \in init (pnet np n)' have "i \not\in net_ips s" ..

hence "fst (initmissing (netgmap sr s)) i = someinit i" by simp

moreover have "someinit i \in (fst \circ sr) ' init (np i)"

unfolding someinit_def proof (rule someI_ex)
  from init_notempty show "\exists x. x \in (fst \circ sr) ' init (np i)" by auto

ultimately show "fst (initmissing (netgmap sr s)) i \in (fst \circ sr) ' init (np i)"
  by simp

qed

next
fix s a s'
assume "s \in reachable (closed (pnet np n)) TT"
  and "(s, a, s') \in trans (closed (pnet np n))"
  and "initmissing (netgmap sr s) \in ?oreachable n"

from this(1) have "s \in reachable (pnet np n) TT" ..

hence "net_ips s = net_tree_ips n" by (rule pnet_net_ips_net_tree_ips)
with 'initmissing (netgmap sr s) \in ?oreachable n'
have "netgmap sr s \in netmask (net_tree_ips n) ' ?oreachable n"
  by (rule initmissing_oreachable_netmask)

obtain \(\sigma, \zeta\) where "\((\sigma, \zeta) = initmissing (netgmap sr s)\)" by (metis surj_pair)
with 'initmissing (netgmap sr s) \in ?oreachable n'
have "(\sigma, \zeta) \in ?oreachable n" by simp
from '(\sigma, \zeta) = initmissing (netgmap sr s)' and 'net_ips s = net_tree_ips n' [symmetric]
  have "netgmap sr s = netmask (net_tree_ips n) (\sigma, \zeta)"
    by (clarsimp simp add: netmask_initmissing_netgmap)

with 's \in reachable (closed (pnet np n)) TT'
  obtain \(\sigma', \zeta'\) where "\((\sigma', \zeta') = initmissing (netgmap sr s')\)" by (metis surj_pair)
    and "netgmap sr s' = netmask (net_tree_ips n) (\sigma', \zeta')"
    using 'wf_net_tree n' and '(s, a, s') \in trans (closed (pnet np n))'
    by (rule transfer_action)

from '(\sigma, \zeta) \in ?oreachable n' have "net_ips \zeta = net_tree_ips n"
  by (rule pnet_net_ips_net_tree_ips [OF oclosed_oreachable_oreachable])
with '\((\sigma, \zeta), a, (\sigma', \zeta')\) \in trans (oclosed (opnet onp n))'
  have "\forall j. j \notin net_tree_ips n \rightarrow \sigma' j = \sigma j"
    by (clarsimp elim!: ocomplete_no_change)

have "initmissing (netgmap sr s') = (\sigma', \zeta')" (intro prod_eqI)
proof (intro prod_eql ext)
  fix i
  from 'netgmap sr s' = netmask (net_tree_ips n) (\sigma', \zeta')'
    '\forall j. j \notin net_tree_ips n \rightarrow \sigma' j = \sigma j'
    '(\sigma, \zeta) = initmissing (netgmap sr s)'
    'net_ips s = net_tree_ips n'
  show "fst (initmissing (netgmap sr s')) i = fst (\sigma', \zeta') i"
    unfolding initmissing_def by simp

next
from 'netgmap sr s' = netmask (net_tree_ips n) (\sigma', \zeta')'
  show "snd (initmissing (netgmap sr s')) = snd (\sigma', \zeta')" by simp

qed

moreover from '(\sigma, \zeta) \in ?oreachable n' '(\sigma, \zeta), a, (\sigma', \zeta') \in trans (oclosed (opnet onp n))'
  have "(\sigma', \zeta') \in ?oreachable n"
    by (rule oreachable_local) (rule TrueI)

ultimately show "initmissing (netgmap sr s') \in ?oreachable n"
  by simp

qed
definition

\[
\text{netglobal} :: ((\text{nat} \to 'g) \to \text{bool}) \to 's \text{ net_state} \to \text{bool}
\]

where

\[
\text{netglobal } P \equiv (\lambda s. P (\text{fst} \text{ (initmissing (netgmap sr s)))))
\]

lemma netglobalsimp [simp]:

"netglobal P s = P (fst (initmissing (netgmap sr s)))"

unfolding netglobal_def by simp

lemma netglobalE [elim]:

\[
\begin{array}{l}
\text{assumes } "\text{netglobal } P s" \\
\text{and } "\forall \sigma. [ P \sigma; \text{fst} \text{ (initmissing (netgmap sr s))} = \sigma ] \implies Q \sigma"
\end{array}
\]

shows "netglobal Q s"

using assms by simp

lemma netglobal_weakenE [elim]:

\[
\begin{array}{l}
\text{assumes } "p \parallel= \text{netglobal } P" \\
\text{and } "\forall \sigma. P \sigma \implies Q \sigma"
\end{array}
\]

shows "p \parallel= \text{netglobal } Q"

using assms(1)

proof (rule invariant_weakenE)

fix s

assume "netglobal P s"

thus "netglobal Q s" by (rule netglobalE) (erule assms(2))

qed

lemma close_opnet:

\[
\begin{array}{l}
\text{assumes } "\text{wf_net_tree } n" \\
\text{and } "\text{oclosed (opnet onp n)} \parallel= (\lambda _ _ _. \text{True}, U \to) \text{ global } P"
\end{array}
\]

shows "closed (pnet np n) \parallel= \text{netglobal } P"

unfolding invariant_def

proof

fix s

assume "s \in \text{reachable (closed (pnet np n)) } \text{TT}"

with assms(1)

have "\text{initmissing (netgmap sr s)} \in \text{oreachable (oclosed (opnet onp n)) } (\lambda _ _ _. \text{True}) U"

by (rule pnet_reachable_transfer)

with assms(2)

have "\text{global } P (\text{initmissing (netgmap sr s)})" ..

thus "netglobal P s" by simp

qed

 locales openproc_parq =

\[
\begin{array}{l}
\text{op: openproc np onp sr for np :: "ip } \to ('s, ('m::msg) \text{ seq_action}) \text{ automaton" and onp sr} \\
\text{+ fixes qp :: "('t, 'm seq_action) \text{ automaton"}
\end{array}
\]

assumes init_qp_notempty: "init qp \neq \{}"

sublocale openproc_parq \subseteq openproc

\[
\begin{array}{l}
\ "\lambda i. \text{np i } \parallel= (\text{qp)"} \\
\ "\lambda i. \text{np i } \parallel= (i \text{ qp)"} \\
\ "\lambda(p, q). (\text{fst (sr p), (snd (sr p), q)})"
\end{array}
\]

proof unfold_locales

fix i

show "\{( \sigma, \zeta ); \zeta \in \text{init (np i } \parallel= (\text{qp)"} \\
\wedge (\sigma i, \zeta) = (\langle \lambda(p, q). (\text{fst (sr p), (snd (sr p), q)}) \rangle s) \\
\wedge (\forall j. j \neq i \Rightarrow \sigma j \in \text{init o } \langle \lambda(p, q). (\text{fst (sr p), (snd (sr p), q)}) \rangle) \\
\text{'} init (np j } \parallel= (\text{qp)'} \} \subseteq \text{init (onp i } \parallel= (i \text{ qp)"}
\]

(is "?S \subseteq \_")

proof

fix s

assume "s \in ?S"

then obtain \sigma p lq

where "s = (\sigma, (\text{snd (sr p), lq}))" \\
and "lq \in \text{init qp}"

and "p \in \text{init (np i)"

170
and "σ i = fst (sr p)" 
and "∀ j. j ≠ i → σ j ∈ (fst ○ (λ(p, q). (fst (sr p), snd (sr p), q)))'(init (np j) × init qp)"

by auto

from this(5) have "∀ j. j ≠ i → σ j ∈ (fst ○ sr)' (init (np j))"

by auto

with 'p ∈ init (np i)' and 'σ i = fst (sr p)' have "(σ, snd (sr p)) ∈ init (onp i)"

by (rule init [THEN set_mp], auto)

with 'lq ∈ init qp' have "((σ, (snd (sr p)), lq)) ∈ init (onp i) × init qp"

by simp

hence "((σ, (snd (sr p)), lq)) ∈ extg' (init (onp i) × init qp)"

by (rule rev_image_eqI) simp

with 's = (σ, (snd (sr p), lq))' show "s ∈ init (onp i ⟨⟨i⟩⟩ qp)"

by simp

qed

next

fix i s a s' σ σ'

assume "σ i = fst ((λ(p, q). (fst (sr p), (snd (sr p), q))) s)"
and "σ' i = fst ((λ(p, q). (fst (sr p), (snd (sr p), q))) s')"
and "(s, a, s') ∈ trans (np i ⟨⟨i⟩⟩ qp)"
then obtain p q p' q' where "s = (p, q)"
and "s' = (p', q')"
and "σ i = fst (sr p)"
and "σ' i = fst (sr p')"

by (clarsimp split: split_split_asm)

from this(1-2) and '(s, a, s') ∈ trans (np i ⟨⟨i⟩⟩ qp)'
have "((p, q), a, (p', q')) ∈ parp sos (trans (np i)) (trans qp)" by simp

hence "((σ, (snd (sr p), q)), a, (σ', (snd (sr p'), q'))) ∈ trans (onp i ⟨⟨i⟩⟩ qp)"

proof cases

assume "q' = q"

and "(p, a, p') ∈ trans (np i)"

and "∀ m. a ≠ receive m"

from 'σ i = fst (sr p)' and 'σ' i = fst (sr p)'
this(2)

have "((σ, (snd (sr p)), a, (σ', (snd (sr p')))) ∈ trans (onp i)" by (rule trans)

with 'q' = q' and '∀ m. a ≠ receive m'

show "((σ, (snd (sr p), q)), a, (σ', (snd (sr p'), q'))) ∈ trans (onp i ⟨⟨i⟩⟩ qp)" by (auto elim!: oparleft)

next

assume "p' = p"

and "(q, a, q') ∈ trans qp"

and "∀ m. a ≠ send m"

with 'σ i = fst (sr p)' and 'σ' i = fst (sr p)'

show "((σ, (snd (sr p), q)), a, (σ', (snd (sr p'), q'))) ∈ trans (onp i ⟨⟨i⟩⟩ qp)"

by (auto elim!: oparrright)

next

fix m

assume "a = τ"

and "(p, receive m, p') ∈ trans (np i)"

and "(q, send m, q') ∈ trans qp"

from 'σ i = fst (sr p)' and 'σ' i = fst (sr p)'
this(2)

have "((σ, (snd (sr p)), receive m, (σ', (snd (sr p')))) ∈ trans (onp i)" by (rule trans)

with '(q, send m, q') ∈ trans qp' and 'a = τ'

show "((σ, (snd (sr p), q)), a, (σ', (snd (sr p'), q'))) ∈ trans (onp i ⟨⟨i⟩⟩ qp)"

by (simp del: step_seq_tau) (rule oparboth)

qed

with 's = (p, q)' 's' = (p', q)'

show "((σ, (snd ((λ(p, q). (fst (sr p), (snd (sr p), q))) s)), a, (σ', (snd ((λ(p, q). (fst (sr p), (snd (sr p), q))) s')))) ∈ trans (onp i ⟨⟨i⟩⟩ qp)"

by simp

next

show "∀ j. init (np j ⟨⟨j⟩⟩ qp) ≠ {}" by (clarsimp simp set_notempty)

by (clarsimp simp add: init_notempty init_qp_notempty)

qed
25 Import all AWN-related theories

theory AWN_Main
imports AWN_SOS AWN_SOS_Labels OAWN_SOS_Labels AWN_Invariants
            OAWN_Convert OClosed_Transfer
begin
end

26 Simple toy example

theory Toy
imports Main AWN_Main Qmsg_Lifting
begin

26.1 Messages used in the protocol

datatype msg =
    Pkt data ip
  | Newpkt data ip
instantiation msg :: msg
begin
definition newpkt_def [simp]: "newpkt ≡ λ(d,dip). Newpkt d dip"
definition eq_newpkt_def: "eq_newpkt m ≡ case m of Newpkt d dip ⇒ True | _ ⇒ False"
instance by intro_classes (simp add: eq_newpkt_def)
end

definition pkt :: "nat × nat ⇒ msg"
where "pkt ≡ λ(no, sip). Pkt no sip"
lemma pkt_simp [simp]:
  "pkt(no, sip) = Pkt no sip"
unfolding pkt_def by simp
lemma not_eq_newpkt_pkt [simp]: "¬eq_newpkt (Pkt no sip)"
  unfolding eq_newpkt_def by simp

26.2 Protocol model

record state =
    ip :: "nat"
  no :: "nat"
  nhip :: "nat"
  msg :: "msg"
  num :: "nat"
  sip :: "nat"

abbreviation toy_init :: "ip ⇒ state"
where "toy_init i ≡ (
    ip = i,
    no = 0,
    nhip = i,
    msg = (SOME x. True),
    num = (SOME x. True),
    sip = (SOME x. True)
)
"
lemma some_neq_not_eq [simp]: "\((\text{SOME } x :: \text{nat}. \ x \neq i) = i\)"
by (subst some_eq_ex) (metis zero_neq_numeral)

definition clear_locals :: "state \Rightarrow state"
where "clear_locals \(\xi\) = \(\xi\) (| msg := \(\text{SOME } x. \text{True} \),
num := \(\text{SOME } x. \text{True} \),
sip := \(\text{SOME } x. \text{True} \) | )"

lemma clear_locals_but_not_globals [simp]:
"ip (clear_locals \(\xi\)) = ip \(\xi\)"
"no (clear_locals \(\xi\)) = no \(\xi\)"
"nhip (clear_locals \(\xi\)) = nhip \(\xi\)"
unfolding clear_locals_def by auto

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

definition is_pkt
where "is_pkt \(\xi\) \equiv \text{case } msg \(\xi\) of
Pkt num' sip' \Rightarrow \{ \(\xi\)[num := num', sip := sip'] \}
| _ \Rightarrow \{\}"

lemmas is_msg_defs =
is_pkt_def is_newpkt_def

lemma is_msg_inv_ip [simp]:
"\(\xi'\) \in is_pkt \(\xi\) \Rightarrow \text{ip } \(\xi'\) = \text{ip } \(\xi\)"
"\(\xi'\) \in is_newpkt \(\xi\) \Rightarrow \text{ip } \(\xi'\) = \text{ip } \(\xi\)"
unfolding is_msg_defs by (cases "msg \(\xi\)", clarsimp)+

lemma is_msg_inv_sip [simp]:
"\(\xi'\) \in is_newpkt \(\xi\) \Rightarrow \text{sip } \(\xi'\) = \text{sip } \(\xi\)"
unfolding is_msg_defs by (cases "msg \(\xi\)", clarsimp)+

lemma is_msg_inv_no [simp]:
"\(\xi'\) \in is_pkt \(\xi\) \Rightarrow \text{no } \(\xi'\) = \text{no } \(\xi\)"
"\(\xi'\) \in is_newpkt \(\xi\) \Rightarrow \text{no } \(\xi'\) = \text{no } \(\xi\)"
unfolding is_msg_defs by (cases "msg \(\xi\)", clarsimp)+

lemma is_msg_inv_nhip [simp]:
"\(\xi'\) \in is_pkt \(\xi\) \Rightarrow \text{nhip } \(\xi'\) = \text{nhip } \(\xi\)"
"\(\xi'\) \in is_newpkt \(\xi\) \Rightarrow \text{nhip } \(\xi'\) = \text{nhip } \(\xi\)"
unfolding is_msg_defs by (cases "msg \(\xi\)", clarsimp)+

lemma is_msg_inv_msg [simp]:
"\(\xi'\) \in is_pkt \(\xi\) \Rightarrow \text{msg } \(\xi'\) = \text{msg } \(\xi\)"
"\(\xi'\) \in is_newpkt \(\xi\) \Rightarrow \text{msg } \(\xi'\) = \text{msg } \(\xi\)"
unfolding is_msg_defs by (cases "msg \(\xi\)", clarsimp)+

datatype pseqp =
PToy

fun nat_of_seqp :: "pseqp \Rightarrow \text{nat}"
where
"nat_of_seqp PToy = 1"

instantiation "pseqp" :: ord
begin

  definition less_eq_seqp [iff]: "11 ≤ 12 = (nat_of_seqp 11 ≤ nat_of_seqp 12)"

  definition less_seqp [iff]: "11 < 12 = (nat_of_seqp 11 < nat_of_seqp 12)"

instance ..
end

abbreviation Toy
where
  "Toy ≡ λ_. [clear_locals] call(PToy)"

fun Γ_{TOY} :: "(state, msg, pseqp, pseqp label) seqp_env"
where
  "Γ_{TOY} PToy = labelled PToy (  
  receive(λmsg’ ξ. ξ ( msg := msg’ )),  
  [ξ. ξ [nhip := ip ξ]]  
  (  
    (is_newpkt)  
    (  
      [ξ. ξ [no := max (no ξ) (num ξ)]]  
      broadcast(λξ. pkt(no ξ, ip ξ)). Toy()  
    )  
  )
  )

  ⊕ (is_pkt)  
  (  
    (ξ. num ξ ≥ no ξ)  
    [ξ. ξ [no := num ξ]]  
    [ξ. ξ [nhip := sip ξ]]  
    broadcast(λξ. pkt(no ξ, ip ξ)). Toy()  
  )  
  ⊕ (ξ. num ξ < no ξ)  
  )

)"

declare Γ_{TOY}..simps [simp del, code del]
lemmas Γ_{TOY}_simps [simp, code] = Γ_{TOY}..simps [simplified]

fun Γ_{TOY}_skeleton
where "Γ_{TOY}_skeleton PToy = seqp_skeleton (Γ_{TOY} PToy)"

lemma Γ_{TOY}_skeleton_wf [simp]:  
  "wellformed Γ_{TOY}_skeleton"
proof (rule, intro allI)
  fix pn pn'
  show "call(pn') ∉ stermSL (Γ_{TOY}_skeleton pn)"
  by (cases pn) simp_all
qed

declare Γ_{TOY}_skeleton.simps [simp del, code del]
lemmas Γ_{TOY}_skeleton_simps [simp, code] = Γ_{TOY}_skeleton.simps [simplified Γ_{TOY}_simps seqp_skeleton.simps]

lemma toy_proc_cases [dest]:
  fixes p pn
  assumes "p ∈ ctermSL (Γ_{TOY} pn)"
  shows "p ∈ ctermSL (Γ_{TOY} PToy)"
  using assms
  by (cases pn) simp_all

definition σ_{TOY} :: "ip ⇒ (state × (state, msg, pseqp, pseqp label) seqp) set"
where "σ_{TOY} i ≡ {(toy_init i, Γ_{TOY} PToy)}"

abbreviation ptoy
  :: "ip ⇒ (state × (state, msg, pseqp, pseqp label) seqp, msg seq_action) automaton"
where
"ptoy i ≡ (init = σ_T O Y i, trans = seqp_sos Γ_T O Y i)"

lemma toy_trans: "trans (ptoy i) = seqp_sos Γ_T O Y i"
by simp

lemma toy_control_within [simp]: "control_within Γ_T O Y (init (ptoy i))"
unfolding σ_T O Y_def by (rule control_withinI) (auto simp del: Γ_T O Y_simps)

lemma toy_wf [simp]:
"wellformed Γ_T O Y"
proof (rule, intro allI)
fix pn pn'
show "call(pn') ∉ stermsl (Γ_T O Y pn)"
by (cases pn) simp_all
qed

lemmas toy_labels_not_empty [simp] = labels_not_empty [OF toy_wf]

lemma toy_ex_label [intro]: "∃ l. l ∈ labels Γ_T O Y p"
by (metis toy_labels_not_empty all_not_in_conv)

lemma toy_ex_labelE [elim]:
assumes "∀ l ∈ labels Γ_T O Y p. P l p"
and "∃ p l. P l p =⇒ Q"
shows "Q"
using assms by (metis toy_ex_label)

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

lemma σ_T O Y_labels [simp]: "((ξ, p) ∈ σ_T O Y i =⇒ labels Γ_T O Y p = {PToy-:0})"
unfolding σ_T O Y_def by simp

By default, we no longer let the simplifier descend into process terms.
declare seqp_congs [cong]

declare Γ_T O Y_simps [cterms_env]
toy_proc_cases [ctermsl_cases]
seq_invariant_ctermsI [OF toy_wf toy_control_within toy_simple_labels toy_trans, cterms_intros]
seq_step_invariant_ctermsI [OF toy_wf toy_control_within toy_simple_labels toy_trans, cterms_intros]

26.3 Define an open version of the protocol
definition σ_T O Y :: "((ip ⇒ state) × ((state, msg, pseqp, pseqp label) seqp)) set"
where "σ_T O Y ≡ (toy_init, Γ_T O Y PToy)"

abbreviation optoy
:: "ip ⇒ ((ip ⇒ state) × (state, msg, pseqp, pseqp label) seqp, msg seq_action) automaton"
where "optoy i ≡ ( init = σ_T O Y, trans = oseqp_sos Γ_T O Y i )"

lemma initiali_toy [intro!, simp]: "initiali i (init (optoy i)) (init (ptoy i))"
unfolding σ_T O Y_def σ_T O Y_def by rule simp_all

lemma caodv_control_within [simp]: "control_within Γ_T O Y (init (optoy i))"
unfolding σ_T O Y_def by (rule control_withinI) (auto simp del: Γ_T O Y_simps)
lemma \(\sigma_{\text{OTOY}}\)-labels [simp]: \((\sigma, p) \in \sigma_{\text{OTOY}} \Rightarrow \text{labels } \Gamma_{\text{TOY}}\ p = \{\text{PToy-:0}\}\)

unfolding \(\sigma_{\text{OTOY}}\)-def by simp

lemma otoy_trans: "\(\text{trans } \text{optoy } i = \text{oseqp_sos } \Gamma_{\text{TOY}}\ i\)"
by simp

declare
oseq_invariant_ctermsI [OF toy_wf oadov_control_within toy_simple_labels otoy_trans, cterms_intros]
oseq_step_invariant_ctermsI [OF toy_wf oadov_control_within toy_simple_labels otoy_trans, cterms_intros]

26.4 Predicates

definition msg_sender :: "msg \Rightarrow ip"
where "msg_sender \(m \equiv \text{case } m \text{ of Pkt } \_ \_ \Rightarrow \_\_\)"

lemma msg_sender_simps [simp]:
"\(\forall d dip sip. \text{msg_sender } (\text{Pkt } d sip) = sip\)"
unfolding msg_sender_def by simp_all

abbreviation not_Pkt :: "msg \Rightarrow bool"
where "not_Pkt \(m \equiv \text{case } m \text{ of Pkt } \_ \_ \Rightarrow \text{False } | \_ \Rightarrow \text{True}\)"

definition nos_increase :: "state \Rightarrow state \Rightarrow bool"
where "nos_increase \(\xi \xi' \equiv (\text{no } \xi \leq \text{no } \xi')\)"

definition msg_num_ok :: "\(\text{ip } \Rightarrow \text{state}\) \Rightarrow msg \Rightarrow bool"
where "msg_num_ok \(\sigma m \equiv \text{case } m \text{ of Pkt } \text{num'} \_ \Rightarrow \text{num'} \leq \text{no } (\sigma \text{sip'}) \_ \Rightarrow \text{True}\)"

lemma msg_num_okI [intro]:
assumes "\(\forall \text{num'} \_ \Rightarrow \_ \Rightarrow \text{msg_num_ok } \sigma m\)"
shows "msg_num_ok \(\sigma m\)"
using assms unfolding msg_num_ok_def
by (auto split: msg.split)

lemma msg_num_ok_Pkt [simp]:
"msg_num_ok \(\sigma m \equiv \text{case } m \text{ of Pkt } \text{data } \text{src} \Rightarrow \text{data } \leq \text{no } (\sigma \text{src})\)"
unfolding msg_num_ok_def by simp

lemma msg_num_ok_pkt [simp]:
"msg_num_ok \(\sigma m \equiv \text{case } m \text{ of Pkt } \text{pkt(data, src)} \Rightarrow \text{data } \leq \text{no } (\sigma \text{src})\)"
unfolding msg_num_ok_def by simp

lemma msg_num_ok_Newpkt [simp]:
"msg_num_ok \(\sigma m \equiv \text{case } m \text{ of Newpkt } \text{data } \text{dst} \Rightarrow \text{no}\)"
unfolding msg_num_ok_def by simp

lemma msg_num_ok_newpkt [simp]:
"msg_num_ok \(\sigma m \equiv \text{case } m \text{ of Newpkt } \text{data } \text{dst} \Rightarrow \text{no}\)"
unfolding msg_num_ok_def by simp

26.5 Sequential Invariants

lemma seq_no_leq_num:
"\(\text{ptoy } i \models \text{onl } \Gamma_{\text{TOY}} (\lambda (\xi, _). \_ \in \{\text{PToy-:7..PToy-:9}\} \Rightarrow \text{no } \xi \leq \text{num } \xi)\)"
by inv_cterms

lemma seqNos_increases:
"\(\text{ptoy } i \models \text{onll } \Gamma_{\text{TOY}} (\lambda ((\xi, _), _, (\xi', _)). \text{nos_increase } \xi \xi')\)"
unfolding nos_increase_def
proof -
  show "\(\text{ptoy } i \models \text{onll } \Gamma_{\text{TOY}} (\lambda ((\xi, _), _, (\xi', _)). \text{no } \xi \leq \text{no } \xi')\)"
  by (inv_cterms inv add: onl_invariant_sterms [OF toy_wf seq_no_leq_num])
qed
lemma seq_nos_increases':
  "ptoy i ⪯ A (λ((ξ, _), _, (ξ', _)). nos_increase ξ ξ')"
by (rule step_invariant_weakenE [OF seq_nos_increases]) (auto dest!: onllD)

lemma sender_ip_valid:
  "ptoy i ⪯ A (λ((ξ, _), _, _). anycast (λm. msg_sender m = ip ξ) a)"
by inv_cterms

lemma ip_constant:
  "ptoy i ⪯ A (λ((ξ, _), _). ip ξ = i)"
by inv_cterms (simp add: σTOY_def)

lemma nhip_eq_ip:
  "ptoy i ⪯ A (λ((ξ, _), _). l ∈ {PToy-:2..PToy-:8} → nhip ξ = ip ξ)"
by inv_cterms

lemma seq_msg_num_ok:
  "ptoy i ⪯ A (λ((ξ, _), a, _). anycast (λm. case m of Pkt num' sip' ⇒ num' = no ξ ∧ sip' = i | _ ⇒ True) a)"
by (inv_cterms inv add: onl_invariant_sterms [OF toy_wf ip_constant])

lemma nhip_eq_i:
  "ptoy i ⪯ A (λ((ξ, _), _). l ∈ {PToy-:2..PToy-:8} −→ nhip ξ = i)"
proof (rule invariant_arbitraryI, clarify intro!: onlI impI)
  fix ξ p l n
  assume "((ξ, p) ∈ reachable (ptoy i) TT" and "1 ∈ labels Γ TOY p" and "l ∈ {PToy-:2..PToy-:8}"
  from this(1-3) have "nhip ξ = ip ξ" by (drule invariantD [OF nhip_eq_ip], auto)
  moreover with "((ξ, p) ∈ reachable (ptoy i) TT'" and "l ∈ labels Γ TOY p'" have "ip ξ = i"
  by (auto dest: invariantD [OF ip_constant])
  ultimately show "nhip ξ = i" by simp
  qed

26.6 Global Invariants

lemma nos_increaseD [dest]:
  assumes "nos_increase ξ ξ'"
  shows "no ξ ≤ no ξ'"
  using assms unfolding nos_increase_def.

lemma nos_increase_simp [simp]:
  "nos_increase ξ ξ' = (no ξ ≤ no ξ')"
  using assms unfolding nos_increase_def..

lemmas oseq_nos_increases =
  open_seq_step_invariant [OF seq_nos_increases initiali_toy otoy_trans toy_trans, simplified seql_onl_swap]

lemmas oseq_no_leq_num =
  open_seq_invariant [OF seq_no_leq_num initiali_toy otoy_trans toy_trans, simplified seql_onll_swap]

lemma all_nos_increase:
  shows "optoy i ⪯ A (otherwith nos_increase {i} S, other nos_increase {i} →)
    onll Γ TOY (λ((σ, _), a, (σ', _)). (∀ j. nos_increase (σ j) (σ' j))))"
proof -
  have *: "∀ σ σ' a. [ otherwith nos_increase {i} S σ σ' a; no (σ i) ≤ no (σ' i) ]
    →∀ j. no (σ j) ≤ no (σ' j)"
  by (auto dest!: otherwith_syncD)
show thesis 
by (inv_cterms 
  inv add: oseq_step_invariant_sterms [OF oseq_nos_increases [THEN oinvariant_step_anyact] 
  toy_wf otoy_trans] 
  simp add: seqllsimp) (auto elim!: *)

qed

lemma oreceived_msg_inv: 
assumes other: "∀ σ σ' m. [ [ P σ m; other Q {i} ] ] =⇒ P σ' m" 
and local: "∀ σ m. P σ m =⇒ P (σ(i := σ i(msg := m))(msg := m)) m" 
shows "optoy i = (otherwith Q {i} (orecvmsg P), other Q {i} →) 
  onl ΓTOY (λ(σ, l). l ∈ {PToy-:1} −→ P σ(msg (σ i)))" 

proof (inv_cterms, intro impI) 
fix σ σ' l 
assume "l = PToy-:1 −→ P σ(msg (σ i))" 
and "l = PToy-: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' 
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 otherE) 
  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

lemma msg_num_ok_other_nos_increase [elim]: 
assumes "msg_num_ok σ m" 
and "other nos_increase {i} σ σ'" 
shows "msg_num_ok σ' m" 

proof (cases m) 
fix num sip 
assume "m = Pkt num sip" 
with 'msg_num_ok σ m' have "num ≤ no (σ sip)" by simp 
also from 'other nos_increase {i} σ σ' have "no (σ sip) ≤ no (σ' sip)" 
  by (rule otherE) (metis eq_iff nos_increaseD) 
finally have "num ≤ no (σ' sip)" .
with 'm = Pkt num sip' show ?thesis 
  by simp

qed

lemma msg_num_ok_no_leq_no [simp, elim]: 
assumes "msg_num_ok σ m" 
and "∀ j. no (σ j) ≤ no (σ' j)" 
shows "msg_num_ok σ' m" 

using assms(1) proof (cases m) 
fix num sip 
assume "m = Pkt num sip" 
with 'msg_num_ok σ m' have "num ≤ no (σ sip)" by simp 
also from '∀ j. no (σ j) ≤ no (σ' j)' have "no (σ sip) ≤ no (σ' sip)" 
  by simp 
finally have "num ≤ no (σ' sip)" .
with 'm = Pkt num sip' show ?thesis

178
lemma oreceived_msg_num_ok:
"optoy i ⇝ (otherwith nos_increase {i} (orecvmsg msg_num_ok),
other nos_increase {i} →)
onl ΓTOY (λ(σ, l). l ∈ {PToy::1..} → msg_num_ok σ (msg (σ i)))"
(is "=" (?(S, ?U) → "")
proof (inv_cterms inv add: oseq_step_invariant_sterms [OF all_nos_increase toy_wf otoy_trans],
intro impI, elim impE)
  fix σ σ'
  assume "msg_num_ok σ (msg (σ i))"
  and "other nos_increase {i} σ σ'"
  moreover from this(2) have "msg (σ' i) = msg (σ i)"
    by (clarsimp elim!: otherE)
  ultimately show "msg_num_ok σ' (msg (σ' i))"
    by (auto)
next
  fix p l σ a q l' σ' pp p' m
  assume a1: "(σ', p') ∈ oreachable (optoy i) ?S ?U"
    and a2: "PToy::1 ∈ labels ΓTOY p'"
    and a3: "σ' i = σ i (msg := m)"
  have inv: "optoy i ⇝ (otherwith nos_increase {i} (orecvmsg msg_num_ok),
other nos_increase {i} →)
onl ΓTOY (λ(σ, l). l ∈ {PToy::1} −→ msg_num_ok σ (msg (σ i)))"
    proof (rule oreceived_msg_inv)
      fix σ σ' m
      assume "msg_num_ok σ m"
      and "other nos_increase {i} σ σ'"
      thus "msg_num_ok σ' m" ..
    qed
next
  fix σ m
  assume "msg_num_ok σ m"
  thus "msg_num_ok (σ (i := σ i (msg := m))) m"
    by (cases m) auto
qed
from a1 a2 a3 show "msg_num_ok σ' m"
  by (clarsimp dest!: oinvariantD [OF inv] onlD)
qed simp

lemma is_pkt_handler_num_leq_no:
  shows "optoy i ⇝ (otherwith nos_increase {i} (orecvmsg msg_num_ok),
other nos_increase {i} →)
onl ΓTOY (λ(σ, l). l ∈ {PToy::6..PToy::10} −→ num (σ i) ≤ no (σ (sip (σ i))))"
proof -
  { fix σ σ'
    assume "∀ j. no (σ j) ≤ no (σ' j)"
    and "num (σ i) ≤ no (σ (sip (σ i)))"
    have "num (σ i) ≤ no (σ' (sip (σ i)))"
    proof -
      note 'num (σ i) ≤ no (σ (sip (σ i)))'
      also from "∀ j. no (σ j) ≤ no (σ' j)" have "no (σ (sip (σ i))) ≤ no (σ' (sip (σ i)))"
        by auto
      finally show ?thesis .
    qed
  } note solve_step = this
show ?thesis
proof (inv_cterms inv add: oseq_step_invariant_sterms [OF all_nos_increase toy_wf otoy_trans]
onl oinvariant_sterms [OF toy_wf oreceived_msg_num_ok]
solve: solve_step, intro impI, elim impE)
  fix σ σ'
  assume *: "num (σ i) ≤ no (σ (sip (σ i)))"
    and "other nos_increase {i} σ σ'"
from this(2) obtain "∀ i∈{i}. σ' i = σ i"
  and "∀ j. j ∉ {i} −→ nos_increase (σ j) (σ' j)" ..
  show "num (σ' i) ≤ no (σ' (sip (σ' i)))"
proof (cases "sip (σ i) = i")
  assume "sip (σ i) = i"
  with * \( ∀ i \in \{i\}. \ σ' i = σ i' \)
  show ?thesis by simp
next
  assume "sip (σ i) \( \neq \) i"
  with \( ∀ j. j \notin \{i\} \rightarrow \) nos_increase (σ j) (σ' j)'
    have "no (σ (sip (σ i))) \leq no (σ' (sip (σ i)))" by simp
  with * \( ∀ i \in \{i\}. \ σ' i = σ i' \)
  show ?thesis by simp
qed

next
  fix p l σ a q l' σ' pp p'
  assume "msg_num_ok σ (msg (σ i))"
  and "∀ j. no (σ j) \leq no (σ' j)"
  and "σ' i ∈ is_pkt (σ i)"
  show "num (σ' i) \leq no (σ (sip (σ i)))"
  proof (cases "msg (σ i)"
    fix num' sip'
    assume "msg (σ i) = Pkt num' sip'"
    with "σ' i ∈ is_pkt (σ i)" obtain "num (σ' i) = num'"
    and "sip (σ' i) = sip'"
    unfolding is_pkt_def by auto
    with "msg (σ i) = Pkt num' sip'" and "msg_num_ok σ (msg (σ i))"'
      have "num (σ' i) \leq no (σ (sip (σ' i)))" by simp
    also from "∀ j. no (σ j) \leq no (σ' j)" have "no (σ (sip (σ' i))) \leq no (σ' (sip (σ' i)))" ..
    finally show ?thesis .
  next
    fix num' sip'
    assume "msg (σ i) = Newpkt num' sip'"
    with "σ' i ∈ is_pkt (σ i)" have False
    unfolding is_pkt_def by simp
    thus ?thesis ..
  qed
qed

lemmas oseq_ip_constant =
  open_seq_invariant [OF ip_constant initiali_toy otoy_trans toy_trans,
                      simplified seql_onl_swap]
lemmas oseq_nhip_eq_i =
  open_seq_invariant [OF nhip_eq_i initiali_toy otoy_trans toy_trans,
                      simplified seql_onl_swap]
lemmas oseq_nhip_eq_ip =
  open_seq_invariant [OF nhip_eq_ip initiali_toy otoy_trans toy_trans,
                      simplified seql_onl_swap]

lemma oseq_bigger_than_next:
  shows "optoy i \( \models \) (otherwith nos_increase \{i\} (orecvmsg msg_num_ok),
  other nos_increase \{i\} \rightarrow)\) global (λσ. no (σ i) \leq no (σ (nhip (σ i))))"
  (is ".\ (\ ?S, ?U \rightarrow) ?P")
proof -
  have nhipinv: "optoy i \( \models \) (?S, ?U \rightarrow)
  onl \( Γ_{TOY} (λ(σ, l). l∈\{PToy-:2..PToy-:8\}
  \rightarrow\) nhip (σ i) = ip (σ i)"
    by (rule oinvariant_weakenE [OF oseq_nhip_eq_ip]) (auto simp: seqlsimp)
  have ipinv: "optoy i \( \models \) (?S, ?U \rightarrow) onl \( Γ_{TOY} (λ(σ, l). ip (σ i) = i)"
    by (rule oinvariant_weakenE [OF oseq_ip_constant]) (auto simp: seqlsimp)
  { fix σ σ' a
    assume "no (σ i) \leq no (σ (nhip (σ i)))"
    and "∀ j. nos_increase (σ j) (σ' j)"
note this(1)
also from '∀ j. nos_increase (σ j) (σ' j)' have "no (σ (nhip (σ i))) ≤ no (σ' (nhip (σ i)))"
by auto
finally have "no (σ i) ≤ no (σ' (nhip (σ i)))" ..
}

have "optoy i | (otherwith nos_increase {i} (orecmsg msg_num_ok),
other nos_increase {i} →)
onl ΓTOY (λ(σ, l). no (σ i) ≤ no (σ' (nhip (σ i))))"
proof (inv_cterms
  inv add: onl_o invariant_sterms [OF toy_wf oseq_no_leq_num [THEN o invariant anyact]]
  oseq.step_invariant_sterms [OF all_nos_increase toy_wf otoy.trans]
  onl_o invariant_sterms [OF toy_wf is_pkt_handler_num_leq_no]
  onl_o invariant_sterms [OF toy_wf nhipInv]
  onl_o invariant_sterms [OF toy_wf ipInv]
  simp add: seq_lsimp seq_lsimp
  simp del: nos_increase_simp
  solve: *)

  fix σ p l
  assume "((σ, p) ∈ σ _O T O Y" 
  thus "no (σ i) ≤ no (σ' (nhip (σ i)))"
  by (simp add: σ _O T O Y_def)

next

  fix σ σ' p l
  assume or: "((σ, p) ∈ oreachable (optoy i) ?S ?U"
      and "l ∈ labels ΓTOY p"
      and "no (σ i) ≤ no (σ' (nhip (σ i)))"
  show "no (σ' i) ≤ no (σ' (nhip (σ' i)))"
  proof (cases "nhip (σ' i) = i")
    assume "nhip (σ' i) = i"
    with 'no (σ i) ≤ no (σ' (nhip (σ i)))' show ?thesis
    by simp
  next
    assume "nhip (σ' i) ≠ i"
    moreover from 'other nos_increase {i} σ σ' [THEN other_localD] have "σ' i = σ i"
    by simp
    ultimately have "no (σ (nhip (σ i))) ≤ no (σ' (nhip (σ' i)))"
      using 'other nos_increase {i} σ σ' and 'σ' i = σ i' by (auto)
    with 'no (σ i) ≤ no (σ' (nhip (σ i)))' and 'σ' i = σ i' show ?thesis
      by simp
  qed

next

  fix p l a q l' σ' pp p'
  assume "no (σ i) ≤ num (σ i)"
      and "num (σ i) ≤ no (σ (sip (σ i)))"
      and "∀ j. nos_increase (σ j) (σ' j)"
  from this(1-2) have "no (σ i) ≤ no (σ (sip (σ i)))"
    by (rule le_trans)
  also from '∀ j. nos_increase (σ j) (σ' j)'
    have "no (σ (sip (σ i))) ≤ no (σ' (sip (σ i)))"
    by auto
  finally show "no (σ i) ≤ no (σ' (sip (σ i)))" ..
  qed

thus ?thesis
  by (rule o invariant weakenE)
  (auto simp: onl_def)

qed

lemma anycast weakenE [elim]:
  assumes "anycast P a"
  and "∀m. P m ⇒ Q m"
  shows "anycast Q a"
  using assms unfolding anycast_def
  by (auto split: seq_action.split)
lemma oseq_msg_num_ok:
"optoy i \vdash_A (\text{act} \; TT, \; \text{others} \; \{i\} \rightarrow) \text{globala} \; (\lambda(\sigma, \; a, \; _{\_}). \; \text{anycast} \; (\text{msg_num} \; \text{ok} \; \sigma) \; a)"
by (rule ostep_invariant_weakene [OF open_seq_step_invariant
[OF seq_msg_num_ok initial_toy otoy_trans toy_trans, simplified seql_onl_swap])
(auto simp: seqllsimp dest!: onllD elim!: anycast_weakenE intro!: msg_num_okI)

26.7 Lifting

lemma opar_bigger_than_next:
shows "optoy i \vdash_A (\text{otherwith} \; \text{nos_increase} \; \{i\} \; (\text{orecvmsg} \; \text{msg_num} \; \text{ok}), \; \text{other} \; \text{nos_increase} \; \{i\} \rightarrow) \text{globala} \; (\lambda(\sigma \; . \; \text{no} \; (\sigma \; i) \leq \text{no} \; (\sigma \; (\text{nhip} \; (\sigma \; i))))"
proof (rule lift_into_qmsg [OF oseq_bigger_than_next])
fix \sigma \; \sigma' \; m
assume "\forall \; j. \; \text{nos_increase} \; (\sigma \; j) \; (\sigma' \; j)"
also from this(2) have "\forall \; j. \; \text{nos_increase} \; (\sigma \; j) \; (\sigma' \; j)" have "\text{no} \; (\sigma \; (\text{nhip} \; (\sigma \; i))) \leq \text{no} \; (\sigma' \; (\text{nhip} \; (\sigma \; i)))"
by simp
finally show "\text{num'} \leq \text{no} \; (\sigma' \; (\text{nhip} \; (\sigma \; i)))".
qed simp

next
show "optoy i \vdash_A (\text{otherwith} \; \text{nos_increase} \; \{i\} \; (\text{orecvmsg} \; \text{msg_num} \; \text{ok}), \; \text{other} \; \text{nos_increase} \; \{i\} \rightarrow) \text{globala} \; (\lambda(\sigma, \; a, \; _{\_}). \; \text{anycast} \; (\text{msg_num} \; \text{ok} \; \sigma) \; a)"
by (rule ostep_invariant_weakene [OF open_seq_step_invariant
[OF seq_nos_increases initial_toy otoy_trans toy_trans])
(auto simp: seqllsimp dest!: onllD)

lemma onode_bigger_than_next:
"\langle\langle i : \text{optoy} \; i \rangle\rangle_\vdash_A (\text{otherwith} \; \text{nos_increase} \; \{i\} \; (\text{oarrivemsg} \; \text{msg_num} \; \text{ok}), \; \text{other} \; \text{nos_increase} \; \{i\} \rightarrow) \text{globala} \; (\lambda(\sigma \; . \; \text{no} \; (\sigma \; i) \leq \text{no} \; (\sigma \; (\text{nhip} \; (\sigma \; i))))"
by (rule node_lift [OF opar_bigger_than_next])

lemma node_local_nos_increase:
"\langle\langle i : \text{optoy} \; i \rangle\rangle_\vdash_A (\lambda(\sigma \; . \; \text{orearrivemsg} \; (\lambda \; _{\_} \; . \; \text{True}) \; \sigma, \; \text{other} \; (\lambda \; _{\_} \; . \; \text{True}) \; \{i\} \rightarrow) \text{globala} \; (\lambda(\sigma, \; a, \; _{\_}). \; \text{anycast} \; (\text{msg_num} \; \text{ok} \; \sigma) \; a))"
proof (rule node_lift_step_statelessassm)
have "optoy i \vdash_A (\text{otherwith} \; \text{nos_increase} \; \{i\} \; (\text{orecvmsg} \; \text{msg_num} \; \text{ok}), \; \text{other} \; \text{nos_increase} \; \{i\} \rightarrow) \text{globala} \; (\lambda(\sigma, \; a, \; _{\_}). \; \text{anycast} \; (\text{msg_num} \; \text{ok} \; \sigma) \; a))"
by (rule ostep_invariant_weakene [OF oseq_nos_increases])
(auto simp: seqllsimp dest!: onllD)
thus "optoy i \vdash_A (\lambda(\sigma \; . \; \text{orecvmsg} \; (\lambda \; _{\_} \; . \; \text{True}) \; \sigma, \; \text{other} \; (\lambda \; _{\_} \; . \; \text{True}) \; \{i\} \rightarrow) \text{globala} \; (\lambda(\sigma, \; a, \; _{\_}). \; \text{anycast} \; (\text{msg_num} \; \text{ok} \; \sigma) \; a))"
by (rule lift_step_into_qmsg_statelessassm)
qed simp

lemma opnet_bigger_than_next:
"\text{opnet} \; (\lambda \; i. \; \text{optoy} \; i \; (\langle\langle i \; \text{qmsg} \rangle\rangle \; \vdash_A \; (\text{otherwith} \; \text{nos_increase} \; \{i\} \; (\text{orecvmsg} \; \text{msg_num} \; \text{ok}), \; \text{other} \; \text{nos_increase} \; \{i\} \rightarrow) \text{globala} \; (\lambda(\sigma, \; a, \; _{\_}). \; \text{anycast} \; (\text{msg_num} \; \text{ok} \; \sigma) \; a))"
proof (rule pnet_lift [OF onode_bigger_than_next])
fix \; i \; R
have "\langle\langle i : \text{optoy} \; i \rangle\rangle_\vdash_A (\lambda(\sigma \; . \; \text{orecvmsg} \; (\lambda \; _{\_} \; . \; \text{True}) \; \sigma, \; \text{other} \; (\lambda \; _{\_} \; . \; \text{True}) \; \{i\} \rightarrow) \text{globala} \; (\lambda(\sigma, \; a, \; _{\_}). \; \text{anycast} \; (\text{msg_num} \; \text{ok} \; \sigma) \; a))"
proof (rule node_lift_anycast_statelessassm)
have "optoy i \vdash_A (\lambda(\sigma \; . \; \text{orecvmsg} \; (\lambda \; a \; . \; \text{True}) \; \sigma, \; \text{other} \; (\lambda \; a \; . \; \text{True}) \; \{i\} \rightarrow) \text{globala} \; (\lambda(\sigma, \; a, \; _{\_}). \; \text{anycast} \; (\text{msg_num} \; \text{ok} \; \sigma) \; a))"
by (rule ostep_invariant_weakene [OF oseq_msg_num_ok])
(auto simp: seqllsimp dest!: onllD)
182
hence "optoy i \(\langle i \rangle \ qmsg \models_{A} (\lambda \sigma \_\_. \text{orecvmsg} \ (\lambda \_\_. \text{True}) \, \sigma, \text{other} \ (\lambda \_\_. \text{True}) \{i\} \rightarrow\) \)
\[\text{globala} \ (\lambda (\sigma, a, \_\_). \text{anycast} \ (\text{msg_num_ok} \ \sigma) \ a)\] 
by (rule lift_step_into_qmsg_statelessassm) auto
thus "optoy i \(\langle i \rangle \ qmsg \models_{A} (\lambda \sigma \_\_. \text{orecvmsg} \ \text{msg_num_ok} \ \sigma, \text{other} \ (\lambda \_\_. \text{True}) \{i\} \rightarrow\) \)
\[\text{globala} \ (\lambda (\sigma, a, \_\_). \text{anycast} \ (\text{msg_num_ok} \ \sigma) \ a)\] 
by (rule ostep_invariant_weakenE) auto
qed

thus "\(\langle i \rangle \ qmsg \models_{A} (\lambda (\sigma, a, \_\_. \text{orecvmsg} \ \text{msg_num_ok} \ \sigma), \text{other} \ (\lambda \_\_. \text{True}) \{i\} \rightarrow\) \)
\[\text{globala} \ (\lambda (\sigma, a, \_\_. \text{anycast} \ (\text{msg_num_ok} \ \sigma) \ a)\] 
by (rule lift_step_into_qmsg_statelessassm) auto
thus "\(\langle i \rangle \ qmsg \models_{A} (\lambda (\sigma, a, \_\_. \text{orecvmsg} \ \text{msg_num_ok} \ \sigma), \text{other} \ (\lambda \_\_. \text{True}) \{i\} \rightarrow\) \)
\[\text{globala} \ (\lambda (\sigma, a, \_\_. \text{anycast} \ (\text{msg_num_ok} \ \sigma) \ a)\] 
by (rule ostep_invariant_weakenE) auto
qed

26.8 Transfer

definition
\[\text{initmissing} :: (\text{nat} \Rightarrow \text{state option}) \times 'a \Rightarrow (\text{nat} \Rightarrow \text{state}) \times 'a\] 
where
\[\text{initmissing} \ \sigma = (\lambda i. \text{case} \ (\text{fst} \ \sigma) \ i \ \text{of} \ \text{None} \Rightarrow \text{toy_init} \ i \ | \ \text{Some} \ \sigma \Rightarrow \text{state} \ \sigma)\] 

lemma \text{not_in_net_ips_fst_init_missing} [simp]:
assumes "i \notin \text{net_ips} \ \sigma"
shows "\text{fst} \ (\text{initmissing} \ (\text{netmap} \ \text{fst} \ \sigma)) \ i = \text{toy_init} \ i"
using assms unfolding \text{initmissing_def} by simp

lemma \text{fst_initmissing_netmap_pair_fst} [simp]:
"\text{fst} \ (\text{initmissing} \ (\text{netmap} \ (\lambda (p, q). \ (\text{fst} \ (id \ p), \ \text{snd} \ (id \ p), \ q)) \ s)) = \text{fst} \ (\text{initmissing} \ (\text{netmap} \ \text{fst} \ s))"
unfolding \text{initmissing_def} by auto

interpretation \text{toy_openproc}: openproc pttoy optoy id
where "toy_openproc.initmissing = initmissing"
proof -
  show "openproc ptoy optoy id"
  proof unfold_locales
    fix i :: ip
    have "{(σ, ζ). (σ i, ζ) ∈ σTOY i ∧ (∀j. j ≠ i ⟹ σ j ∈ fst ' σTOY j)} ⊆ σOT OY"
    unfolding σTOY_def σOT OY_def
    proof (rule equalityD1)
      show "⋀ f p. {(σ, ζ). (σ i, ζ) ∈ {(f i, p)} ∧ (∀j. j ≠ i ⟹ σ j ∈ fst ' {(f j, p)})} = {(f, p)}"
      by (rule set_eqI) auto
      qed
    thus "{ (σ, ζ) | σ ζ s. s ∈ init (ptoy i) ∧ (σ i, ζ) = id s ∧ (∀j. j ≠ i ⟹ σ j ∈ (fst o id) ' init (ptoy j)) } ⊆ init (optoy i)"
    by simp
  next
    show "∀ j. init (ptoy j) ≠ {}"
    unfolding σTOY_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 (ptoy i)"
    then obtain q q' where "s = (σ i, q)" and "s' = (σ' i, q')" and "⟨(σ i, q), a, (σ' i, q')⟩ ∈ trans (ptoy i)"
    by (cases s, cases s') auto
    from this(3) have "⟨(σ, q), a, (σ', q')⟩ ∈ trans (optoy i)"
    by simp (rule open_seqp_action [OF toy_wf])
    with 's = (σ i, q)' and 's' = (σ' i, q')'
    show "⟨(σ, snd (id s)), a, (σ', snd (id s'))⟩ ∈ trans (optoy i)"
    by simp
  qed
  then interpret op: openproc ptoy optoy id .
  have [simp]: "⋀ i. (SOME x. x ∈ (fst o id) ' init (ptoy i)) = toy_init i"
  unfolding σTOY_def by simp
  hence "⋀ i. openproc.initmissing ptoy id i = initmissing i"
  unfolding op.initmissing_def op.someinit_def initmissing_def
  by (auto split: option.split)
  thus "openproc.initmissing ptoy id = initmissing" ..
qed

lemma fst_initmissing_netgmap_default_toy_init_netlift:
  "fst (initmissing (netgmap fst s)) = default toy_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 toy_init (netlift fst s)))"

interpretation toy_openproc_par_qmsg: openproc_parq ptoy optoy id qmsg
where "toy_openproc_par_qmsg.netglobal = netglobal"
  and "toy_openproc_par_qmsg.initmissing = initmissing"
proof -
  show "openproc_parq ptoy optoy id qmsg"
  by (unfold_locales) simp
  then interpret opq: openproc_parq ptoy optoy id qmsg .
  have im: "⋀ σ. openproc.initmissing (λi. ptoy i ⟨⟨ qmsg⟩⟩ (λ(p, q). (fst (id p), snd (id p), q)) σ = initmissing σ"
unfolding \text{opq.initmissing}_\text{def} \text{opq.someinit}_\text{def} \text{initmissing}_\text{def}

unfolding \sigma_{\text{TOY}}_\text{def} \sigma_{\text{QMSG}}_\text{def} \text{by} \text{(clarsimp cong: option.case_cong)}

thus "\text{openproc.initmissing}(\lambda i. \text{ptoy}\ i\ (\langle qmsg \rangle\ (\lambda (p, q). (\text{fst} (id\ p),\ \text{snd} (id\ p),\ q))) = \text{initmissing}"

by (rule ext)

have "\\sigma. \text{openproc.netglobal}(\lambda i. \text{ptoy}\ i\ (\langle qmsg \rangle\ (\lambda (p, q). (\text{fst} (id\ p),\ \text{snd} (id\ p),\ q))) P \sigma = \text{netglobal}\ P \sigma"

unfolding \text{opq.netglobal}_\text{def} \text{netglobal}_\text{def} \text{opq.initmissing}_\text{def} \text{initmissing}_\text{def} \text{opq.someinit}_\text{def}

unfolding \sigma_{\text{TOY}}_\text{def} \sigma_{\text{QMSG}}_\text{def} \text{by (clarsimp cong: option.case_cong}

simp del: One_nat_def

simp add: \text{fst_initmissing_netgmap_default_toy_init_netlift}

[symmetric, unfolded \text{initmissing}_\text{def}]

thus "\text{openproc.netglobal}(\lambda i. \text{ptoy}\ i\ (\langle qmsg \rangle\ (\lambda (p, q). (\text{fst} (id\ p),\ \text{snd} (id\ p),\ q))) = \text{netglobal}"

by auto

qed

26.9 Final result

lemma \textit{bigger_than_next}:

assumes "\text{wf_net_tree}\ n"

shows "\text{closed}\ (\text{pnet}\ (\lambda i. \text{ptoy}\ i\ (\langle qmsg \rangle\ n)))\ \models\ \text{netglobal}\ (\lambda s. \forall i. \text{no}\ (s\ i) \leq \text{no}\ (s\ (\text{nhip}\ (s\ i))))"

(is "\_ \models \text{netglobal}\ (\lambda s. \forall i. \text{?inv}\ s\ i)"

proof -

from "\text{wf_net_tree}\ n"

have proto: "\text{closed}\ (\text{pnet}\ (\lambda i. \text{ptoy}\ i\ (\langle qmsg \rangle\ n)))\ \models\ \text{netglobal}\ (\lambda s. \forall i\in\text{net_tree_ips}\ n. \text{no}\ (s\ i) \leq \text{no}\ (s\ (\text{nhip}\ (s\ i))))"

by (rule \text{toy_openproc_par_qmsg.close_opnet [OF _ ocnet_bigger_than_next])

show \?thesis

unfolding \text{invariant}_\text{def} \text{opnet_sos.opnet_tau1}

proof (rule, simp only: \text{toy_openproc_par_qmsg.netglobalsimp}

\text{fst_initmissing.netgmap.pair_fst}, \text{rule allI})

fix \sigma\ i

assume sr: "\sigma\in\text{reachable}\ (\text{closed}\ (\text{pnet}\ (\lambda i. \text{ptoy}\ i\ (\langle qmsg \rangle\ n)))\ TT"

hence "\forall i\in\text{net_tree_ips}\ n. \text{?inv}\ (\text{fst}\ (\text{initmissing}\ (\text{netgmap}\ \text{fst}\ s)))\ i"

by - (drule \text{invariantD [OF proto],}

\text{simp only: \text{toy_openproc_par_qmsg.netglobalsimp}

\text{fst_initmissing.netgmap.pair_fst})

thus \"\text{?inv}\ (\text{fst}\ (\text{initmissing}\ (\text{netgmap}\ \text{fst}\ s)))\ i\"

proof (cases "i\in\text{net_tree_ips}\ n")

assume "i\notin\text{net_tree_ips}\ n"

from sr have "\sigma\in\text{reachable}\ (\text{pnet}\ (\lambda i. \text{ptoy}\ i\ (\langle qmsg \rangle\ n)))\ TT" ..

hence "\text{net_ips}\ \sigma = \text{net_tree_ips}\ n" ..

with \"i\notin\text{net_tree_ips}\ n\" have "i\notin\text{net_ips}\ \sigma" by simp

hence "(\text{fst}\ (\text{initmissing}\ (\text{netgmap}\ \text{fst}\ s)))\ i = \text{toy_init}\ i"

by simp

thus \?thesis by simp

qed metis

qed

end

27 Acknowledgements

We thank Peter Höfner for agreeing to the inclusion of the simple ‘Toy’ example model.

References