

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://hal.inria.fr/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*

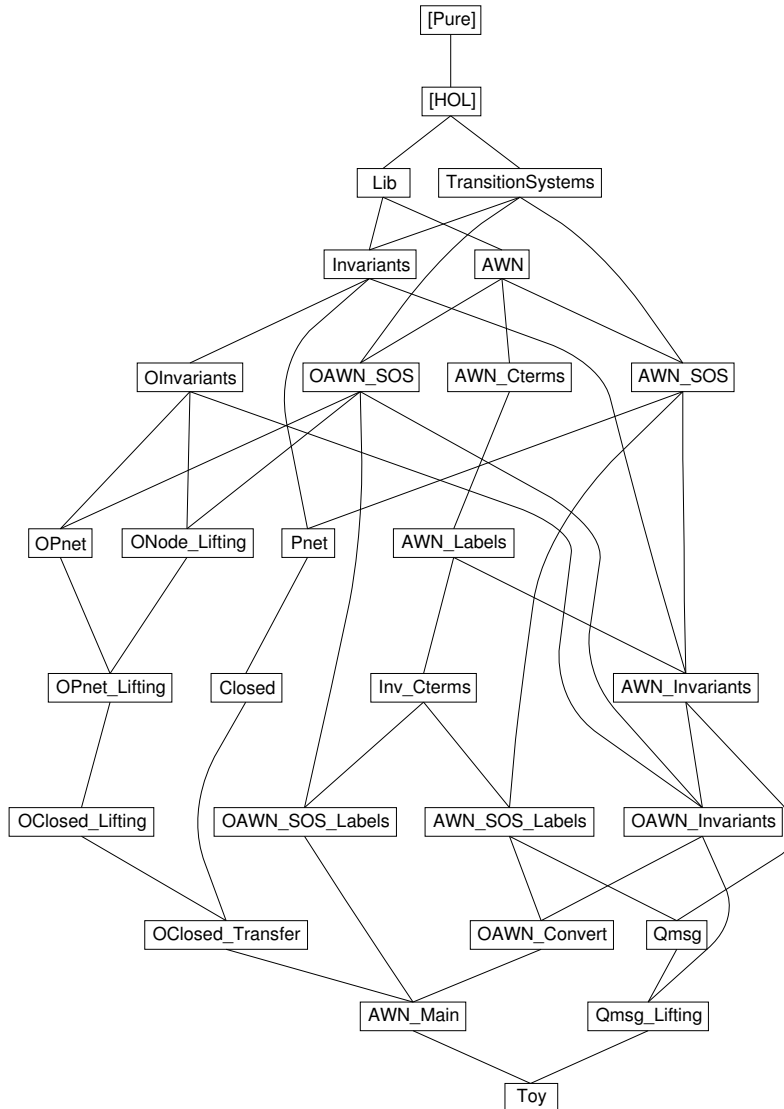
August 28, 2014

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].



*Inria, École normale supérieure, and NICTA

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 (sterms)	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-cterm 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-cterm tactic for open sequential processes	84

15	Lemmas for open partial networks	86
16	Lifting rules for (open) nodes	87
17	Lifting rules for (open) partial networks	95
18	Lifting rules for (open) closed networks	108
19	Generic invariants on sequential AWN processes	109
19.1	Invariants via labelled control terms	109
19.2	Step invariants via labelled control terms	115
20	Generic open invariants on sequential AWN processes	119
20.1	Open invariants via labelled control terms	119
20.2	Open step invariants via labelled control terms	126
21	Transfer standard invariants into open invariants	131
22	Model the standard queuing model	135
23	Lifting rules for parallel compositions with QMSG	136
24	Transfer open results onto closed models	143
25	Import all AWN-related theories	172
26	Simple toy example	172
26.1	Messages used in the protocol	172
26.2	Protocol model	172
26.3	Define an open version of the protocol	175
26.4	Predicates	176
26.5	Sequential Invariants	176
26.6	Global Invariants	177
26.7	Lifting	182
26.8	Transfer	183
26.9	Final result	185
27	Acknowledgements	185

1 Generic functions and lemmas

```

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"

```

```

    shows "P x y"
using assms unfolding nat_less_le by auto

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

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

end

```

2 Transition systems (automata)

```

theory TransitionSystems
imports Main
begin

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

record ('s, 'a) automaton =
  init :: "'s set"
  trans :: "('s, 'a) transition set"

end

```

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 ∈ init A ⇒ s ∈ reachable A I"
  | reachable_step: "[[ s ∈ reachable A I; (s, a, s') ∈ trans A; I a ]] ⇒ s' ∈ reachable A I"

inductive_cases reachable_icas: "s ∈ reachable A I"

lemma reachable_pair_induct [consumes, case_names init step]:
  assumes "(ξ, p) ∈ reachable A I"
  and "∧ξ p. (ξ, p) ∈ init A ⇒ P ξ p"
  and "(∧ξ p ξ' p' a. [[ (ξ, p) ∈ reachable A I; P ξ p;
    ((ξ, p), a, (ξ', p')) ∈ trans 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) ∈ init 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')) ∈ trans A"
  and "I a"

```

```

    and IH: " $\bigwedge \xi p. s = (\xi, p) \implies P \xi p$ "
  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: " $\bigwedge a. P \ a \implies Q \ a$ "
  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
qed



### 3.2 Invariance



definition invariant
  :: "('s, 'a) automaton  $\implies$  ('a  $\implies$  bool)  $\implies$  ('s  $\implies$  bool)  $\implies$  bool"
  ("_  $\models$  (1'(_  $\rightarrow$ '))/_" [100, 0, 9] 8)
where
  "(A  $\models$  (I  $\rightarrow$ ) P) = ( $\forall s \in \text{reachable } A \ I. P \ s$ )"

abbreviation
  any_invariant
  :: "('s, 'a) automaton  $\implies$  ('s  $\implies$  bool)  $\implies$  bool"
  ("_  $\models$  _" [100, 9] 8)
where
  "(A  $\models$  P)  $\equiv$  (A  $\models$  (TT  $\rightarrow$ ) P)"

lemma invariantI [intro]:

```

```

assumes init: " $\bigwedge s. s \in \text{init } A \implies P s$ "
  and step: " $\bigwedge s a s'. \llbracket s \in \text{reachable } A I; P s; (s, a, s') \in \text{trans } A; I a \rrbracket \implies P s'$ "
shows " $A \models (I \rightarrow) P$ "
unfolding invariant_def
proof
  fix s
  assume " $s \in \text{reachable } A I$ "
  thus " $P s$ "
proof induction
  fix s assume " $s \in \text{init } A$ "
  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
qed

```

```

lemma invariant_pairI [intro]:
  assumes init: " $\bigwedge \xi p. (\xi, p) \in \text{init } A \implies P (\xi, p)$ "
  and step: " $\bigwedge \xi p \xi' p' a. \llbracket (\xi, p) \in \text{reachable } A I; P (\xi, p); ((\xi, p), a, (\xi', p')) \in \text{trans } A; I a \rrbracket \implies P (\xi', p')$ "
  shows " $A \models (I \rightarrow) P$ "
using assms by auto

```

```

lemma invariant_arbitraryI:
  assumes " $\bigwedge s. s \in \text{reachable } A I \implies P s$ "
  shows " $A \models (I \rightarrow) P$ "
using assms unfolding invariant_def by simp

```

```

lemma invariantD [dest]:
  assumes " $A \models (I \rightarrow) P$ "
  and " $s \in \text{reachable } A I$ "
  shows " $P s$ "
using assms unfolding invariant_def by blast

```

```

lemma invariant_initE [elim]:
  assumes invP: " $A \models (I \rightarrow) P$ "
  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  $\sigma$  P Q
  assumes invP: " $A \models (PI \rightarrow) P$ "
  and PQ: " $\bigwedge s. P s \implies Q s$ "
  and QUIPI: " $\bigwedge a. QI a \implies PI a$ "
  shows " $A \models (QI \rightarrow) Q$ "
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) ⇒ bool"
 ("_ \models_A (1'(_ →'))/_" [100, 0, 0] 8)

where

"(A \models_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 \models (TT →) P"
 shows "A \models (QI →) P"
 using assms by auto

abbreviation

any_step_invariant
 :: "('s, 'a) automaton ⇒ (('s, 'a) transition ⇒ bool) ⇒ bool"
 ("_ \models_A _" [100, 9] 8)

where

"(A \models_A P) ≡ (A \models_A (TT →) P)"

lemma step_invariant_true:

"p \models_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 \models_A (I →) P"
 unfolding step_invariant_def
 using assms by auto

lemma step_invariantD [dest]:

assumes "A \models_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 \models_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 ξ' p' a.
 [(ξ, p) ∈ reachable A I; ((ξ, p), a, (ξ', p')) ∈ trans A; I a]
 ⇒ P ((ξ, p), a, (ξ', p'))"
 shows "A \models_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]


```

     $\implies P ((\xi, p), a, (\xi', p'))$ "
  shows "A  $\models_A (I \rightarrow) P$ "
  using assms by auto

```

lemma step_invariant_weakenE [elim!]:

```

fixes T  $\sigma$  P Q
assumes invP: "A  $\models_A (PI \rightarrow) P$ "
  and PQ: " $\bigwedge t. P t \implies Q t$ "
  and QIPI: " $\bigwedge a. QI a \implies PI a$ "
shows "A  $\models_A (QI \rightarrow) Q$ "
proof
  fix s a s'
  assume "s  $\in$  reachable A QI"
  and "(s, a, s')  $\in$  trans A"
  and "QI a"
  from 'QI a' have "PI a" by (rule QIPI)
  from 's  $\in$  reachable A QI' have "s  $\in$  reachable A PI" using QIPI ..
  with invP have "P (s, a, s')" using '(s, a, s')  $\in$  trans A' 'PI a' ..
  thus "Q (s, a, s')" by (rule PQ)
qed

```

lemma step_invariant_weaken_with_invariantE [elim]:

```

assumes pinv: "A  $\models (I \rightarrow) P$ "
  and qinv: "A  $\models_A (I \rightarrow) Q$ "
  and wr: " $\bigwedge s a s'. \llbracket P s; P s'; Q (s, a, s'); I a \rrbracket \implies R (s, a, s')$ "
shows "A  $\models_A (I \rightarrow) R$ "
proof
  fix s a s'
  assume sr: "s  $\in$  reachable A I"
  and tr: "(s, a, s')  $\in$  trans A"
  and "I a"
  hence "s'  $\in$  reachable A I" ..
  with pinv have "P s'" ..
  from pinv and sr have "P s" ..
  from qinv sr 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 sinv: "A  $\models_A (I \rightarrow) Q$ "
  and init: " $\bigwedge s. s \in$  init A  $\implies P s$ "
  and step: " $\bigwedge s s' a. \llbracket s \in$  reachable A I;
  P s;
  Q (s, a, s');
  I a  $\rrbracket \implies P s'$ "
shows "A  $\models (I \rightarrow) P$ "
proof
  fix s assume "s  $\in$  init A" thus "P s" by (rule init)
next
  fix s s' a
  assume "s  $\in$  reachable A I"
  and "P s"
  and "(s, a, s')  $\in$  trans A"
  and "I a"
  show "P s'"
proof -
  from sinv and 's  $\in$  reachable A I' and '(s, a, s')  $\in$  trans A' and 'I a' have "Q (s, a, s')" ..
  with 's  $\in$  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 .

```
inductive_set oreachable
  :: "('g × 'l, 'a) automaton
     ⇒ ('g ⇒ 'g ⇒ 'a ⇒ bool)
     ⇒ ('g ⇒ 'g ⇒ bool)
     ⇒ ('g × 'l) set"
for A :: "('g × 'l, 'a) automaton"
and S :: "'g ⇒ 'g ⇒ 'a ⇒ bool"
and U :: "'g ⇒ 'g ⇒ bool"
where
  oreachable_init: "s ∈ init A ⇒ s ∈ oreachable A S U"
| oreachable_local: "[[ s ∈ oreachable A S U; (s, a, s') ∈ trans A; S (fst s) (fst s') a ]]
                    ⇒ s' ∈ oreachable A S U"
| oreachable_other: "[[ s ∈ oreachable A S U; U (fst s) σ' ]]
                    ⇒ (σ', snd s) ∈ oreachable A S U"
```

```
lemma oreachable_local' [elim]:
  assumes "(σ, p) ∈ oreachable A S U"
  and "((σ, p), a, (σ', p')) ∈ trans A"
  and "S σ σ' a"
  shows "(σ', p') ∈ oreachable A S U"
using assms by (metis fst_conv oreachable.oreachable_local)
```

```
lemma oreachable_other' [elim]:
  assumes "(σ, p) ∈ oreachable A S U"
  and "U σ σ'"
  shows "(σ', p) ∈ oreachable A S U"
proof -
  from 'U σ σ'' have "U (fst (σ, p)) σ'" by simp
  with '(σ, p) ∈ oreachable A S U' have "(σ', snd (σ, p)) ∈ oreachable A S U"
  by (rule oreachable_other)
  thus "(σ', p) ∈ oreachable A S U" by simp
qed
```

```
lemma oreachable_pair_induct [consumes, case_names init other local]:
  assumes "(σ, p) ∈ oreachable A S U"
  and "∧σ p. (σ, p) ∈ init A ⇒ P σ p"
  and "(∧σ p σ'. [[ (σ, p) ∈ oreachable A S U; P σ p; U σ σ' ]] ⇒ P σ' p)"
  and "(∧σ p σ' p' a. [[ (σ, p) ∈ oreachable A S U; P σ p;
                        ((σ, p), a, (σ', p')) ∈ trans A; S σ σ' a ]] ⇒ P σ' p)"
  shows "P σ p"
using assms (1) proof (induction "(σ, p)" arbitrary: σ p)
  fix σ p
  assume "(σ, p) ∈ init A"
  with assms(2) show "P σ p" .
next
  fix s σ'
  assume "s ∈ oreachable A S U"
  and "U (fst s) σ'"
  and IH: "∧σ p. s = (σ, p) ⇒ P σ p"
  from this(1) obtain σ p where "s = (σ, p)"
```

```

        and "(σ, p) ∈ oreachable A S U"
    by (metis surjective_pairing)
note this(2)
moreover from IH and 's = (σ, p)' have "P σ p" .
moreover from 'U (fst s) σ'' and 's = (σ, p)' have "U σ σ'" by simp
ultimately have "P σ' p" by (rule assms(3))
with 's = (σ, p)' show "P σ' (snd s)" by simp
next
fix s a σ' p'
assume "s ∈ oreachable A S U"
    and tr: "(s, a, (σ', p')) ∈ trans A"
    and "S (fst s) (fst (σ', p')) a"
    and IH: "∧σ p. s = (σ, p) ⇒ P σ p"
from this(1) obtain σ p where "s = (σ, p)"
    and "(σ, p) ∈ oreachable A S U"
    by (metis surjective_pairing)
note this(2)
moreover from IH 's = (σ, p)' have "P σ p" .
moreover from tr and 's = (σ, p)' have "((σ, p), a, (σ', p')) ∈ trans A" by simp
moreover from 'S (fst s) (fst (σ', p')) a' and 's = (σ, p)' have "S σ σ' a" by simp
ultimately show "P σ' p'" by (rule assms(4))
qed

lemma oreachable_weakenE [elim]:
  assumes "s ∈ oreachable A PS PU"
    and PSQS: "∧s s' a. PS s s' a ⇒ QS s s' a"
    and PUQU: "∧s s'. PU s s' ⇒ QU s s'"
  shows "s ∈ oreachable A QS QU"
using assms(1)
proof (induction)
  fix s assume "s ∈ init A"
  thus "s ∈ oreachable A QS QU" ..
next
  fix s a s'
  assume "s ∈ oreachable A QS QU"
    and "(s, a, s') ∈ trans A"
    and "PS (fst s) (fst s') a"
  from 'PS (fst s) (fst s') a' have "QS (fst s) (fst s') a" by (rule PSQS)
  with 's ∈ oreachable A QS QU' and '(s, a, s') ∈ trans A' show "s' ∈ oreachable A QS QU" ..
next
  fix s g'
  assume "s ∈ oreachable A QS QU"
    and "PU (fst s) g'"
  from 'PU (fst s) g'' have "QU (fst s) g'" by (rule PUQU)
  with 's ∈ oreachable A QS QU' show "(g', snd s) ∈ oreachable A QS QU" ..
qed

definition
  act :: "('a ⇒ bool) ⇒ 's ⇒ 's ⇒ 'a ⇒ bool"
where
  "act I ≡ (λ_ _ . I)"

lemma act_simp [iff]: "act I s s' a = I a"
  unfolding act_def ..

lemma reachable_in_oreachable [elim]:
  fixes s
  assumes "s ∈ reachable A I"
  shows "s ∈ oreachable A (act I) U"
  unfolding act_def using assms proof induction
  fix s
  assume "s ∈ init A"
  thus "s ∈ oreachable A (λ_ _ . I) U" ..
next

```

```

fix s a s'
assume "s ∈ oreachable A (λ_ _ . I) U"
  and "(s, a, s') ∈ trans A"
  and "I a"
thus "s' ∈ oreachable A (λ_ _ . I) U"
  by (rule oreachable_local)
qed

```

4.2 Open Invariance

definition oinvariant

```

:: "('g × 'l, 'a) automaton
⇒ ('g ⇒ 'g ⇒ 'a ⇒ bool) ⇒ ('g ⇒ 'g ⇒ bool)
⇒ (('g × 'l) ⇒ bool) ⇒ bool"
("_ ⊨ (1'((1_)/ (1_ →')/ _)" [100, 0, 0, 9] 8)

```

where

```

"(A ⊨ (S, U →) P) = (∀s∈oreachable A S U. P s)"

```

lemma oinvariantI [intro]:

```

fixes T TI S U P
assumes init: "∧s. s ∈ init A ⇒ P s"
  and other: "∧g g' l.
  [ (g, l) ∈ oreachable A S U; P (g, l); U g g' ] ⇒ P (g', l)"
  and local: "∧s a s'.
  [ s ∈ oreachable A S U; P s; (s, a, s') ∈ trans A; S (fst s) (fst s') a ] ⇒ P s'"
shows "A ⊨ (S, U →) P"

```

unfolding oinvariant_def

proof

```

  fix s
  assume "s ∈ oreachable A S U"
  thus "P s"
proof induction
  fix s assume "s ∈ init A"
  thus "P s" by (rule init)
next
  fix s a s'
  assume "s ∈ oreachable A S U"
  and "P s"
  and "(s, a, s') ∈ trans A"
  and "S (fst s) (fst s') a"
  thus "P s'" by (rule local)
next
  fix s g'
  assume "s ∈ oreachable A S U"
  and "P s"
  and "U (fst s) g'"
  thus "P (g', snd s)"
  by - (rule other [where g="fst s"], simp_all)
qed
qed

```

lemma oinvariant_oreachableI:

```

assumes "∧σ s. (σ, s) ∈ oreachable A S U ⇒ P (σ, s)"
shows "A ⊨ (S, U →) P"
using assms unfolding oinvariant_def by auto

```

lemma oinvariant_pairI [intro]:

```

assumes init: "∧σ p. (σ, p) ∈ init A ⇒ P (σ, p)"
  and local: "∧σ p σ' p' a.
  [ (σ, p) ∈ oreachable A S U; P (σ, p); ((σ, p), a, (σ', p')) ∈ trans A;
  S σ σ' a ] ⇒ P (σ', p'"
  and other: "∧σ σ' p.
  [ (σ, p) ∈ oreachable A S U; P (σ, p); U σ σ' ] ⇒ P (σ', p)"
shows "A ⊨ (S, U →) P"

```

by (rule oinvariantI)
 (clarsimp | erule init | erule(3) local | erule(2) other)+

lemma oinvariantD [dest]:
 assumes "A \models (S, U \rightarrow) P"
 and "s \in oreachable A S U"
 shows "P s"
 using assms unfolding oinvariant_def
 by clarsimp

lemma oinvariant_initD [dest, elim]:
 assumes invP: "A \models (S, U \rightarrow) P"
 and init: "s \in init A"
 shows "P s"
 proof -
 from init have "s \in oreachable A S U" ..
 with invP show ?thesis ..
 qed

lemma oinvariant_weakenE [elim!]:
 assumes invP: "A \models (PS, PU \rightarrow) P"
 and PQ: " $\bigwedge s. P s \implies Q s$ "
 and QSPS: " $\bigwedge s s' a. QS s s' a \implies PS s s' a$ "
 and QUPU: " $\bigwedge s s'. QU s s' \implies PU s s'$ "
 shows "A \models (QS, QU \rightarrow) Q"
 proof
 fix s
 assume "s \in init A"
 with invP have "P s" ..
 thus "Q s" by (rule PQ)
 next
 fix $\sigma p \sigma' p' a$
 assume " $(\sigma, p) \in$ oreachable A QS QU"
 and " $((\sigma, p), a, (\sigma', p')) \in$ trans A"
 and "QS $\sigma \sigma' a$ "
 from this(3) have "PS $\sigma \sigma' a$ " by (rule QSPS)
 from ' $(\sigma, p) \in$ oreachable A QS QU' and QSPS QUPU have " $(\sigma, p) \in$ oreachable A PS PU" ..
 hence " $(\sigma', p') \in$ oreachable A PS PU" using ' $((\sigma, p), a, (\sigma', p')) \in$ trans A' and ' $PS \sigma \sigma' a$ ' ..
 with invP have "P (σ', p')" ..
 thus "Q (σ', p')" by (rule PQ)
 next
 fix $\sigma \sigma' p$
 assume " $(\sigma, p) \in$ oreachable A QS QU"
 and "Q (σ, p)"
 and "QU $\sigma \sigma'$ "
 from 'QU $\sigma \sigma'$ ' have "PU $\sigma \sigma'$ " by (rule QUPU)
 from ' $(\sigma, p) \in$ oreachable A QS QU' and QSPS QUPU have " $(\sigma, p) \in$ oreachable A PS PU" ..
 hence " $(\sigma', p) \in$ oreachable A PS PU" using 'PU $\sigma \sigma'$ ' ..
 with invP have "P (σ', p)" ..
 thus "Q (σ', p)" by (rule PQ)
 qed

lemma oinvariant_weakenD [dest]:
 assumes "A \models (S', U' \rightarrow) P"
 and " $(\sigma, p) \in$ oreachable A S U"
 and weakenS: " $\bigwedge s s' a. S s s' a \implies S' s s' a$ "
 and weakenU: " $\bigwedge s s'. U s s' \implies U' s s'$ "
 shows "P (σ, p)"
 proof -
 from ' $(\sigma, p) \in$ oreachable A S U' have " $(\sigma, p) \in$ oreachable A S' U'"
 by (rule oreachable_weakenE)
 (erule weakenS, erule weakenU)
 with 'A \models (S', U' \rightarrow) 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) ∈ reachable A I"
  and step: "((ξ, p), a, (ξ', p')) ∈ trans A"
  and "I a"
  hence "(ξ', p') ∈ reachable A I" ..
  hence "(ξ', p') ∈ oreachable 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' 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) ⇒ 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 ∈ oreachable 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 ∈ oreachable 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) ∈ reachable A I" 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"
  and "((ζ, snd s), a, (ζ', snd s')) ∈ trans A"
  using '∀j∈J. ζ j = (fst s) j' by - (drule(2) local_stepsE, clarsimp)
from '(ζ, snd s) ∈ reachable A I'
  and '((ζ, snd s), a, (ζ', snd s')) ∈ trans A'
  and 'I a'
  have "(ζ', snd s') ∈ reachable A I" ..

with '∀j∈J. ζ' j = (fst s') j'
  show "∃σ. (∀j∈J. σ j = (fst s') j) ∧ (σ, snd s') ∈ reachable A I" by auto
next
fix s σ'
assume "∃σ. (∀j∈J. σ j = (fst s) j) ∧ (σ, snd s) ∈ reachable A I"
  and "U (fst s) σ'"
then obtain σ where "∀j∈J. σ j = (fst s) j"
  and "(σ, snd s) ∈ reachable A I" by auto
from 'other_steps U J' and 'U (fst s) σ'' have "∀j∈J. σ' j = (fst s) j"
  by - (erule(1) other_stepsE)
with '∀j∈J. σ j = (fst s) j' have "∀j∈J. σ j = σ' j"
  by clarsimp
with '(σ, snd s) ∈ reachable A I'
  show "∃σ. (∀j∈J. σ j = fst (σ', snd s) j) ∧ (σ, snd (σ', snd s)) ∈ reachable A I"
  by auto
qed
qed

```

```

lemma subreachableE [elim]:
  assumes "subreachable A U J"
  and "s ∈ oreachable 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) ∈ oreachable 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, l) ∈ oreachable A (λs s'. I) U"
  and "U σ σ'"
  shows "∃ζ'. (∀j∈J. ζ' j = σ' j) ∧ (ζ', l) ∈ reachable A I"
proof -
  from '(σ, l) ∈ oreachable A (λs s'. I) U' and 'U σ σ''
  have "(σ', l) ∈ oreachable A (λs s'. I) U"

```

```

    by - (rule oreachable_other')
  with 'subreachable A U J' show ?thesis
    by auto
qed

```

lemma open_closed_invariant:

```

  fixes J
  assumes "A  $\models$  (I  $\rightarrow$ ) P"
    and "subreachable A U J"
    and localp: " $\bigwedge \sigma \sigma' s. [\forall j \in J. \sigma' j = \sigma j; P(\sigma', s)] \implies P(\sigma, s)$ "
  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 " $\forall j \in J. \sigma' j = (\text{fst } s') j$ "
    and " $(\sigma', \text{snd } s') \in \text{reachable A I}$ "
    by (metis subreachableE)
  from 'A  $\models$  (I  $\rightarrow$ ) P' and ' $(\sigma', \text{snd } s') \in \text{reachable A I}$ ' have "P ( $\sigma', \text{snd } s'$ )" ..
  with ' $\forall j \in J. \sigma' j = (\text{fst } s') j$ ' 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 ' $(gg', l) \in \text{reachable A I}$ '
    have "P (gg', l)" ..
  with ' $\forall j \in J. gg' j = g' j$ ' show "P (g', l)"
    by (rule localp)
qed

```

lemma oinvariant_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
     $\Rightarrow$  ('g  $\Rightarrow$  'g  $\Rightarrow$  'a  $\Rightarrow$  bool)  $\Rightarrow$  ('g  $\Rightarrow$  'g  $\Rightarrow$  bool)
     $\Rightarrow$  (('g  $\times$  'l, 'a) transition  $\Rightarrow$  bool)  $\Rightarrow$  bool"
  ("_  $\models_A$  (1'((1_)/ (1_  $\rightarrow$ ))/ _)" [100, 0, 0, 9] 8)

```

where

```

  "(A  $\models_A$  (S, U  $\rightarrow$ ) P) =
    ( $\forall s \in \text{oreachable A S U}. (\forall a s'. (s, a, s') \in \text{trans A} \wedge S (\text{fst } s) (\text{fst } s') a \longrightarrow P (s, a, s'))$ )"

```

lemma ostep_invariant_def':

```

  "(A  $\models_A$  (S, U  $\rightarrow$ ) P) = ( $\forall s \in \text{oreachable A S U}.
    (\forall a s'. (s, a, s') \in \text{trans A} \wedge S (\text{fst } s) (\text{fst } s') a \longrightarrow P (s, a, s'))$ )"
  unfolding ostep_invariant_def by auto

```


lemma ostep_invariantI [intro]:
 assumes *: " $\bigwedge \sigma s a \sigma' s'. \llbracket (\sigma, s) \in \text{oreachable } A \ S \ U; ((\sigma, s), a, (\sigma', s')) \in \text{trans } A; S \ \sigma \ \sigma' \ a \rrbracket$
 $\implies P ((\sigma, s), a, (\sigma', s'))$ "
 shows " $A \models_A (S, U \rightarrow) P$ "
 unfolding ostep_invariant_def
 using assms by auto

lemma ostep_invariantD [dest]:
 assumes " $A \models_A (S, U \rightarrow) P$ "
 and " $(\sigma, s) \in \text{oreachable } A \ S \ U$ "
 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 " $A \models_A (S, U \rightarrow) P$ "
 and " $(\sigma, s) \in \text{oreachable } A \ S \ U$ "
 and " $((\sigma, s), a, (\sigma', s')) \in \text{trans } A$ "
 and " $S \ \sigma \ \sigma' \ a$ "
 and " $P ((\sigma, s), a, (\sigma', s')) \implies Q$ "
 shows " Q "
 using assms by auto

lemma ostep_invariant_weakenE [elim!]:
 assumes invP: " $A \models_A (PS, PU \rightarrow) P$ "
 and PQ: " $\bigwedge t. P \ t \implies Q \ t$ "
 and QSPS: " $\bigwedge \sigma \ \sigma' \ a. QS \ \sigma \ \sigma' \ a \implies PS \ \sigma \ \sigma' \ a$ "
 and QUPU: " $\bigwedge \sigma \ \sigma'. \quad QU \ \sigma \ \sigma' \implies PU \ \sigma \ \sigma'$ "
 shows " $A \models_A (QS, QU \rightarrow) Q$ "
 proof
 fix $\sigma \ s \ \sigma' \ s' \ a$
 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: " $A \models (S, U \rightarrow) P$ "
 and qinv: " $A \models_A (S, U \rightarrow) Q$ "
 and wr: " $\bigwedge \sigma \ s \ a \ \sigma' \ s'. \llbracket P (\sigma, s); P (\sigma', s'); Q ((\sigma, s), a, (\sigma', s')); S \ \sigma \ \sigma' \ a \rrbracket$
 $\implies R ((\sigma, s), a, (\sigma', s'))$ "
 shows " $A \models_A (S, U \rightarrow) R$ "
 proof
 fix $\sigma \ s \ a \ \sigma' \ s'$
 assume sr: " $(\sigma, s) \in \text{oreachable } A \ S \ U$ "
 and tr: " $((\sigma, s), a, (\sigma', s')) \in \text{trans } A$ "
 and " $S \ \sigma \ \sigma' \ a$ "
 hence " $(\sigma', s') \in \text{oreachable } A \ S \ U$ " ..
 with pinv have " $P (\sigma', s')$ " ..
 from pinv and sr have " $P (\sigma, s)$ " ..
 from qinv sr tr ' $S \ \sigma \ \sigma' \ a$ ' 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: " $A \models_A (S, U \rightarrow) Q$ "
 and init: " $\bigwedge \sigma \ s. (\sigma, s) \in \text{init } A \implies P (\sigma, s)$ "
 and local: " $\bigwedge \sigma \ s \ \sigma' \ s' \ a.$ "

```

      [ (σ, s) ∈ oreachable A S U;
        P (σ, s);
        Q ((σ, s), a, (σ', s'));
        S σ σ' a ] ⇒ P (σ', s)"
    and other: "∧σ σ' s. [ (σ, s) ∈ oreachable A S U; U σ σ'; P (σ, s) ] ⇒ P (σ', s)"
  shows "A ⊨ (S, U →) P"
proof
  fix σ s assume "(σ, s) ∈ init A" thus "P (σ, s)" by (rule init)
next
  fix σ s σ' s' a
  assume "(σ, s) ∈ oreachable A S U"
    and "P (σ, s)"
    and "((σ, s), a, (σ', s')) ∈ trans A"
    and "S σ σ' a"
  show "P (σ', s)"
proof -
  from sinv and '(σ, s) ∈ oreachable A S U' and '((σ, s), a, (σ', s')) ∈ trans A' and 'S σ σ' a'
  have "Q ((σ, s), a, (σ', s'))" ..
  with '(σ, s) ∈ oreachable A S U' and 'P (σ, s)' show "P (σ', s)"
  using 'S σ σ' a' by (rule local)
qed
next
  fix σ σ' l
  assume "(σ, l) ∈ oreachable A S U"
    and "U σ σ'"
    and "P (σ, l)"
  thus "P (σ', l)" by (rule other)
qed

lemma open_closed_step_invariant:
  assumes "A ⊨A (I →) P"
    and "local_steps (trans A) J"
    and "other_steps U J"
    and localp: "∧σ ζ a σ' ζ' s s'.
      [ ∃j∈J. σ j = ζ j; ∃j∈J. σ' j = ζ' j; P ((σ, s), a, (σ', s')) ]
      ⇒ P ((ζ, s), a, (ζ', s'))"
  shows "A ⊨A (act I, U →) P"
proof
  fix σ s a σ' s'
  assume or: "(σ, s) ∈ oreachable A (act I) U"
    and tr: "((σ, s), a, (σ', s')) ∈ trans A"
    and "act I σ σ' a"
  from 'act I σ σ' a' have "I a" ..
  from 'local_steps (trans A) J' and 'other_steps U J' have "subreachable A U J" ..
  then obtain ζ where "∃j∈J. ζ j = σ j"
    and "(ζ, s) ∈ reachable A I"
  using or unfolding act_def
  by (auto dest!: subreachableE_pair)

  from 'local_steps (trans A) J' and tr and '∃j∈J. ζ j = σ j'
  obtain ζ' where "∃j∈J. ζ' j = σ' j"
    and "((ζ, s), a, (ζ', s')) ∈ trans A"
  by auto

  from 'A ⊨A (I →) P' and '(ζ, s) ∈ reachable A I'
    and '((ζ, s), a, (ζ', s')) ∈ trans A'
    and 'I a'
  have "P ((ζ, s), a, (ζ', s'))" ..
  with '∃j∈J. ζ j = σ j' and '∃j∈J. ζ' j = σ' j' show "P ((σ, s), a, (σ', s'))"
  by (rule localp)
qed

lemma oinvariant_step_anyact:
  assumes "p ⊨A (act TT, U →) 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 bool)
 \Rightarrow 'i set
 \Rightarrow (('i \Rightarrow 's) \Rightarrow 'a \Rightarrow bool)
 \Rightarrow ('i \Rightarrow 's) \Rightarrow ('i \Rightarrow 's) \Rightarrow 'a \Rightarrow bool"
where "otherwith Q I P σ σ' a \equiv ($\forall i. i \notin I \rightarrow Q (\sigma i) (\sigma' i)$) \wedge P σ a"

lemma *otherwithI* [*intro*]:
assumes *other*: " $\bigwedge j. j \notin I \implies Q (\sigma j) (\sigma' j)$ "
and *sync*: "P σ a"
shows "otherwith Q I P σ σ' a"
unfolding *otherwith_def* **using** *assms* by *simp*

lemma *otherwithE* [*elim*]:
assumes "otherwith Q I P σ σ' a"
and " $\llbracket P \sigma a; \forall j. j \notin I \rightarrow Q (\sigma j) (\sigma' j) \rrbracket \implies R \sigma \sigma' a$ "
shows "R $\sigma \sigma' a$ "
using *assms* **unfolding** *otherwith_def* by *simp*

lemma *otherwith_actionD* [*dest*]:
assumes "otherwith Q I P σ σ' a"
shows "P σ a"
using *assms* by *auto*

lemma *otherwith_syncD* [*dest*]:
assumes "otherwith Q I P σ σ' a"
shows " $\forall j. j \notin I \rightarrow Q (\sigma j) (\sigma' j)$ "
using *assms* by *auto*

lemma *otherwithEI* [*elim*]:
assumes "otherwith P I P0 σ σ' a"
and " $\bigwedge \sigma a. P0 \sigma a \implies Q0 \sigma a$ "
shows "otherwith P I Q0 σ σ' a"
using *assms*(1) **unfolding** *otherwith_def*
by (*clarsimp* *elim!*: *assms*(2))

lemma *all_but*:
assumes " $\bigwedge \xi. S \xi \xi$ "
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 *metis*

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 - (*rule* *ext*, *metis*)

other

definition *other* :: "('s \Rightarrow 's \Rightarrow bool) \Rightarrow 'i set \Rightarrow ('i \Rightarrow 's) \Rightarrow ('i \Rightarrow 's) \Rightarrow bool"
where "other P I σ σ' \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*: " $\bigwedge i. i \in I \implies \sigma' i = \sigma i$ "
and *other*: " $\bigwedge j. j \notin I \implies P (\sigma j) (\sigma' j)$ "
shows "other P I σ σ' "
using *assms* **unfolding** *other_def* by *clarsimp*

```

lemma otherE [elim]:
  assumes "other P I  $\sigma$   $\sigma'$ "
    and "[ $\forall i \in I. \sigma' i = \sigma i; \forall j. j \notin I \rightarrow P (\sigma j) (\sigma' j)$ ]"  $\implies$  R  $\sigma$   $\sigma'$ "
  shows "R  $\sigma$   $\sigma'$ "
  using assms unfolding other_def by simp

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

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

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

lemma weaken_local [elim]:
  assumes "other P I  $\sigma$   $\sigma'$ "
    and PQ: " $\bigwedge \xi \xi'. P \xi \xi' \implies Q \xi \xi'$ "
  shows "other Q I  $\sigma$   $\sigma'$ "
  using assms unfolding other_def by auto

definition global :: "(nat  $\Rightarrow$  's)  $\Rightarrow$  bool"  $\Rightarrow$  (nat  $\Rightarrow$  's)  $\times$  'local  $\Rightarrow$  bool"
where "global P  $\equiv$  ( $\lambda(\sigma, _). P \sigma$ )"

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

definition globala :: "(nat  $\Rightarrow$  's, 'action) transition  $\Rightarrow$  bool"
   $\Rightarrow$  ((nat  $\Rightarrow$  's)  $\times$  'local, 'action) transition  $\Rightarrow$  bool"
where "globala P  $\equiv$  ( $\lambda((\sigma, _), a, (\sigma', _)). P (\sigma, a, \sigma')$ )"

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  $\times$  ip  $\Rightarrow$  'a"
  and eq_newpkt :: "'a  $\Rightarrow$  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 "'1" "'s ⇒ 's set" "( 's, 'm, 'p, 'l) seqp"
| ASSIGN "'1" "'s ⇒ 's" "( 's, 'm, 'p, 'l) seqp"
| CHOICE "( 's, 'm, 'p, 'l) seqp" "( 's, 'm, 'p, 'l) seqp"
| UCAST "'1" "'s ⇒ ip" "'s ⇒ 'm" "( 's, 'm, 'p, 'l) seqp" "( 's, 'm, 'p, 'l) seqp"
| BCAST "'1" "'s ⇒ 'm" "( 's, 'm, 'p, 'l) seqp"
| GCAST "'1" "'s ⇒ ip set" "'s ⇒ 'm" "( 's, 'm, 'p, 'l) seqp"
| SEND "'1" "'s ⇒ 'm" "( 's, 'm, 'p, 'l) seqp"
| DELIVER "'1" "'s ⇒ data" "( 's, 'm, 'p, 'l) seqp"
| RECEIVE "'1" "'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, 0, 60] 60)
"_ifguard"    :: "[ ptrrn, bool, ( 's, 'm, 'p, unit) seqp ] ⇒ ( 's, 'm, 'p, unit) seqp"
               ("(00<_ . _>)//_" [0, 0, 60] 60)

"_bassign"    :: "[ ptrrn, 'a, ( 's, 'm, 'p, unit) seqp ] ⇒ ( 's, 'm, 'p, unit) seqp"
               ("(00[_ . _])//_" [0, 0, 60] 60)
"_lbassign"   :: "[ 'a, ptrrn, 'a, ( 's, 'm, 'p, 'a) seqp ] ⇒ ( 's, 'm, 'p, 'a) seqp"
               ("{ } (00[_ . _])//_" [0, 0, 0, 60] 60)

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

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

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

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

```

```

"_assign f p"      ⇒ "CONST ASSIGN () f p"
"_lassign l f p"  ⇒ "CONST ASSIGN l f p"

"_bassign ξ e p"   ⇒ "CONST ASSIGN () (λξ. e) p"
"_lbassign l ξ e p" ⇒ "CONST ASSIGN l (λξ. e) p"

"_unicast fip fmsg p q" ⇒ "CONST UCAST () fip fmsg p q"
"_lunicast l fip fmsg p q" ⇒ "CONST UCAST l fip fmsg p q"

"_bcast fmsg p"    ⇒ "CONST BCAST () fmsg p"
"_lbcast l fmsg p" ⇒ "CONST BCAST l fmsg p"

"_gcast fipset fmsg p" ⇒ "CONST GCAST () fipset fmsg p"
"_lgcast l fipset fmsg p" ⇒ "CONST GCAST l fipset fmsg p"

"_send fmsg p"     ⇒ "CONST SEND () fmsg p"
"_lsend l fmsg p" ⇒ "CONST SEND l fmsg p"

"_deliver fdata p" ⇒ "CONST DELIVER () fdata p"
"_ldeliver l fdata p" ⇒ "CONST DELIVER l fdata p"

"_receive fmsg p"  ⇒ "CONST RECEIVE () fmsg p"
"_lreceive l fmsg p" ⇒ "CONST RECEIVE l fmsg p"

```

```

notation "CHOICE" ("((_)//⊕//(_)" [56, 55] 55)
and "CALL" ("(3call'((3_))'" [0] 60)

```

```

definition not_call :: "('s, 'm, 'p, 'l) seqp ⇒ bool"
where "not_call p ≡ ∀pn. p ≠ call(pn)"

```

```

lemma not_call_simps [simp]:

```

```

"∧l fg p.      not_call (f1}⟨fg⟩ p)"
"∧l fa p.      not_call (f1}[fa] p)"
"∧p1 p2.      not_call (p1 ⊕ p2)"
"∧l fip fmsg p q. not_call (f1}unicast(fip, fmsg).p ▷ q)"
"∧l fmsg p.    not_call (f1}broadcast(fmsg).p)"
"∧l fips fmsg p. not_call (f1}groupcast(fips, fmsg).p)"
"∧l fmsg p.    not_call (f1}send(fmsg).p)"
"∧l fdata p.   not_call (f1}deliver(fdata).p)"
"∧l fmsg p.    not_call (f1}receive(fmsg).p)"
"∧l pn.       ¬(not_call (call(pn)))"

```

```

unfolding not_call_def by auto

```

```

definition not_choice :: "('s, 'm, 'p, 'l) seqp ⇒ bool"
where "not_choice p ≡ ∀p1 p2. p ≠ p1 ⊕ p2"

```

```

lemma not_choice_simps [simp]:

```

```

"∧l fg p.      not_choice (f1}⟨fg⟩ p)"
"∧l fa p.      not_choice (f1}[fa] p)"
"∧p1 p2.      ¬(not_choice (p1 ⊕ p2))"
"∧l fip fmsg p q. not_choice (f1}unicast(fip, fmsg).p ▷ q)"
"∧l fmsg p.    not_choice (f1}broadcast(fmsg).p)"
"∧l fips fmsg p. not_choice (f1}groupcast(fips, fmsg).p)"
"∧l fmsg p.    not_choice (f1}send(fmsg).p)"
"∧l fdata p.   not_choice (f1}deliver(fdata).p)"
"∧l fmsg p.    not_choice (f1}receive(fmsg).p)"
"∧l pn.       not_choice (call(pn))"

```

```

unfolding not_choice_def by auto

```

```

lemma seqp_congs:

```

```

"∧l fg p. {l}⟨fg⟩ p = {l}⟨fg⟩ p"
"∧l fa p. {l}[fa] p = {l}[fa] p"
"∧p1 p2. p1 ⊕ p2 = p1 ⊕ p2"
"∧l fip fmsg p q. {l}unicast(fip, fmsg).p ▷ q = {l}unicast(fip, fmsg).p ▷ q"

```

```

" $\wedge$ l fmsg p. {l}broadcast(fmsg).p = {l}broadcast(fmsg).p"
" $\wedge$ l fips fmsg p. {l}groupcast(fips, fmsg).p = {l}groupcast(fips, fmsg).p"
" $\wedge$ l fmsg p. {l}send(fmsg).p = {l}send(fmsg).p"
" $\wedge$ l fdata p. {l}deliver(fdata).p = {l}deliver(fdata).p"
" $\wedge$ l fmsg p. {l}receive(fmsg).p = {l}receive(fmsg).p"
" $\wedge$ l pn. call(pn) = 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 ({l}<_> p) = {l}< $\lambda$ _. {}> (seqp_skeleton p)"
  | "seqp_skeleton ({l}[_] p) = {l}[ $\lambda$ _. ()] (seqp_skeleton p)"
  | "seqp_skeleton (p  $\oplus$  q) = (seqp_skeleton p)  $\oplus$  (seqp_skeleton q)"
  | "seqp_skeleton ({l}unicast(_, _). p  $\triangleright$  q) = {l}unicast( $\lambda$ _. 0,  $\lambda$ _. (). (seqp_skeleton p)  $\triangleright$  (seqp_skeleton q))"
  | "seqp_skeleton ({l}broadcast(_). p) = {l}broadcast( $\lambda$ _. (). (seqp_skeleton p))"
  | "seqp_skeleton ({l}groupcast(_, _). p) = {l}groupcast( $\lambda$ _. {},  $\lambda$ _. (). (seqp_skeleton p))"
  | "seqp_skeleton ({l}send(_). p) = {l}send( $\lambda$ _. (). (seqp_skeleton p))"
  | "seqp_skeleton ({l}deliver(_). p) = {l}deliver( $\lambda$ _. 0). (seqp_skeleton p)"
  | "seqp_skeleton ({l}receive(_). p) = {l}receive( $\lambda$  _ . (). (seqp_skeleton p))"
  | "seqp_skeleton (call(pn)) = call(pn)"

```

Calculate the subterms of a term.

```

fun subterms :: "('s, 'm, 'p, 'l) seqp  $\Rightarrow$  ('s, 'm, 'p, 'l) seqp set"
where
  "subterms ({l}<fg> p) = {{l}<fg> p}  $\cup$  subterms p"
  | "subterms ({l}[fa] p) = {{l}[fa] p}  $\cup$  subterms p"
  | "subterms (p1  $\oplus$  p2) = {p1  $\oplus$  p2}  $\cup$  subterms p1  $\cup$  subterms p2"
  | "subterms ({l}unicast(fip, fmsg). p  $\triangleright$  q) =
    {{l}unicast(fip, fmsg). p  $\triangleright$  q}  $\cup$  subterms p  $\cup$  subterms q"
  | "subterms ({l}broadcast(fmsg). p) = {{l}broadcast(fmsg). p}  $\cup$  subterms p"
  | "subterms ({l}groupcast(fips, fmsg). p) = {{l}groupcast(fips, fmsg). p}  $\cup$  subterms p"
  | "subterms ({l}send(fmsg). p) = {{l}send(fmsg).p}  $\cup$  subterms p"
  | "subterms ({l}deliver(fdata). p) = {{l}deliver(fdata).p}  $\cup$  subterms p"
  | "subterms ({l}receive(fmsg). p) = {{l}receive(fmsg).p}  $\cup$  subterms p"
  | "subterms (call(pn)) = {call(pn)}"

```

```

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

```

```

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

```

```

lemma root_in_subterms [simp]:
  " $\wedge$  $\Gamma$  pn.  $\exists$ pn'.  $\Gamma$  pn  $\in$  subterms ( $\Gamma$  pn')"
by (rule_tac x=pn in exI) simp

```

```

lemma deriv_in_subterms [elim, dest]:
  " $\wedge$ l f p q. {l}<f> q  $\in$  subterms p  $\Rightarrow$  q  $\in$  subterms p"
  " $\wedge$ l fa p q. {l}[fa] q  $\in$  subterms p  $\Rightarrow$  q  $\in$  subterms p"
  " $\wedge$ p1 p2 p. p1  $\oplus$  p2  $\in$  subterms p  $\Rightarrow$  p1  $\in$  subterms p"
  " $\wedge$ p1 p2 p. p1  $\oplus$  p2  $\in$  subterms p  $\Rightarrow$  p2  $\in$  subterms p"
  " $\wedge$ l fip fmsg p q r. {l}unicast(fip, fmsg). q  $\triangleright$  r  $\in$  subterms p  $\Rightarrow$  q  $\in$  subterms p"
  " $\wedge$ l fip fmsg p q r. {l}unicast(fip, fmsg). q  $\triangleright$  r  $\in$  subterms p  $\Rightarrow$  r  $\in$  subterms p"
  " $\wedge$ l fmsg p q. {l}broadcast(fmsg). q  $\in$  subterms p  $\Rightarrow$  q  $\in$  subterms p"
  " $\wedge$ l fips fmsg p q. {l}groupcast(fips, fmsg). q  $\in$  subterms p  $\Rightarrow$  q  $\in$  subterms p"
  " $\wedge$ l fmsg p q. {l}send(fmsg). q  $\in$  subterms p  $\Rightarrow$  q  $\in$  subterms p"
  " $\wedge$ l fdata p q. {l}deliver(fdata). q  $\in$  subterms p  $\Rightarrow$  q  $\in$  subterms p"
  " $\wedge$ l fmsg p q. {l}receive(fmsg). q  $\in$  subterms p  $\Rightarrow$  q  $\in$  subterms p"
by auto

```

5.2 Actions

There are two sorts of τ 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.

```
class tau =
  fixes tau :: "'a" ("τ")
```

5.2.1 Sequential Actions (and related predicates)

```
datatype 'm seq_action =
  broadcast 'm
| groupcast "ip set" 'm
| unicast ip 'm
| notunicast ip          ("¬unicast _" [1000] 60)
| send 'm
| deliver data
| receive 'm
| seq_tau                ("τs")
```

```
instantiation "seq_action" :: (type) tau
begin
definition step_seq_tau [simp]: "τ ≡ τs"
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]:
  "∧m. recvmsg P (broadcast m) = True"
  "∧ips m. recvmsg P (groupcast ips m) = True"
  "∧ip m. recvmsg P (unicast ip m) = True"
  "∧ip. recvmsg P (notunicast ip) = True"
  "∧m. recvmsg P (send m) = True"
  "∧d. recvmsg P (deliver d) = True"
  "∧m. recvmsg P (receive m) = P m"
  " recvmsg P τs = True"
unfolding recvmsg_def by simp_all
```

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

```
lemma recvmsgE [elim]:
  assumes "recvmsg (R σ) a"
  and "∧m. R σ m ⇒ R σ' m"
  shows "recvmsg (R σ') 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]:
  "∧m. anycast P (broadcast m) = P m"
  "∧ips m. anycast P (groupcast ips m) = P m"
  "∧ip m. anycast P (unicast ip m) = P m"
  "∧ip. anycast P (notunicast ip) = True"
  "∧m. anycast P (send m) = True"
  "∧d. anycast P (deliver d) = True"
  "∧m. anycast P (receive m) = True"
  " anycast P τs = True"
```


unfolding anycast_def by simp_all

```
definition orecvmsg :: "(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 orecvmsg_simps [simp]:
"∧m. orecvmsg P σ (broadcast m) = True"
"∧ips m. orecvmsg P σ (groupcast ips m) = True"
"∧ip m. orecvmsg P σ (unicast ip m) = True"
"∧ip. orecvmsg P σ (notunicast ip) = True"
"∧m. orecvmsg P σ (send m) = True"
"∧d. orecvmsg P σ (deliver d) = True"
"∧m. orecvmsg P σ (receive m) = P σ m"
" orecvmsg P σ τs = True"
unfolding orecvmsg_def by simp_all
```

```
lemma orecvmsgEI [elim]:
"[[ orecvmsg P σ a; ∧σ a. P σ a ⇒ Q σ a ]] ⇒ orecvmsg Q σ a"
by (cases a) simp_all
```

```
lemma orecvmsg_stateless_recvmsg [elim]:
"orecvmsg (λ_. P) σ a ⇒ recvmsg P a"
by (cases a) simp_all
```

```
lemma orecvmsg_recv_weaken [elim]:
"[[ orecvmsg P σ a; ∧σ a. P σ a ⇒ Q a ]] ⇒ recvmsg Q a"
by (cases a) simp_all
```

```
lemma orecvmsg_recvmsg [elim]:
"orecvmsg P σ a ⇒ recvmsg (P σ) 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 τs = True"
unfolding sendmsg_def by simp_all
```

```
type_synonym ('s, 'm, 'p, 'l) seqp_env = "'p ⇒ ('s, 'm, 'p, 'l) seqp"
```

5.2.2 Node Actions (and related predicates)

```
datatype 'm node_action =
node_cast "ip set" 'm ("_*cast'(_)") [200, 200] 200)
| node_deliver ip data ("_:_deliver'(_)") [200, 200] 200)
| node_arrive "ip set" "ip set" 'm ("_:_arrive'(_)") [200, 200, 200] 200)
| node_connect ip ip ("connect'(_, _)") [200, 200] 200)
| node_disconnect ip ip ("disconnect'(_, _)") [200, 200] 200)
| node_newpkt ip data ip ("_:_newpkt'(_, _)") [200, 200, 200] 200)
| node_tau ("τn")
```

```
instantiation "node_action" :: (type) tau
begin
definition step_node_tau [simp]: "τ ≡ τn"
instance ..
```

```

end

definition arrivemsg :: "'ip ⇒ ('m ⇒ bool) ⇒ '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)"
  "∧i1 i2.    arrivemsg i P (connect(i1, i2))  = True"
  "∧i1 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 m | _ ⇒ 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"
  "∧i1 i2.    oarrivemsg P σ (connect(i1, i2))  = True"
  "∧i1 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"
  "∧i1 i2.    castmsg P (connect(i1, i2))  = True"
  "∧i1 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 s 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 Ri) = [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 (induction ns) auto

lemma net_ips_is_dom_netmap: "net_ips s = dom(netmap s)"
proof (induction s)
  fix i Ri and p :: 's
  show "net_ips (NodeS i p Ri) = dom (netmap (NodeS i p Ri))"
    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 "∀i∈net_ips s1 ∪ net_ips s2. the ((netmap s1 ++ netmap s2) i) = σ i"
    and "net_ips s1 ∩ net_ips s2 = {}"
  shows "∀i∈net_ips s1. the (netmap s1 i) = σ i"
proof
  fix i
  assume "i ∈ net_ips s1"
  hence "i ∈ dom(netmap s1)" by (simp add: net_ips_is_dom_netmap)
  moreover with assms(2) have "i ∉ dom(netmap s2)" by (auto simp add: net_ips_is_dom_netmap)
  ultimately have "the (netmap s1 i) = the ((netmap s1 ++ netmap s2) i)"
    by (simp add: map_add_dom_app_simps)
  with assms(1) and 'i∈net_ips s1' show "the (netmap s1 i) = σ i" by simp
qed

lemma netmap_add_disjoint2 [elim]:
  assumes "∀i∈net_ips s1 ∪ net_ips s2. the ((netmap s1 ++ netmap s2) i) = σ i"
  shows "∀i∈net_ips s2. the (netmap s2 i) = σ 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 ∩ net_ips s2 = {}"
    and "∀i∈net_ips (SubnetS s1 s2). the (netmap (SubnetS s1 s2) i) = σ i"
  shows "∀i∈net_ips s1. the (netmap s1 i) = σ i"
    and "∀i∈net_ips s2. the (netmap s2 i) = σ i"
proof -
  from assms(2) have "∀i∈net_ips s1 ∪ net_ips s2. the ((netmap s1 ++ netmap s2) i) = σ i" by auto
  with assms(1) show "∀i∈net_ips s1. the (netmap s1 i) = σ i"
    by - (erule(1) netmap_add_disjoint)
next
  from assms(2) have "∀i∈net_ips s1 ∪ net_ips s2. the ((netmap s1 ++ netmap s2) i) = σ i" by auto
  thus "∀i∈net_ips s2. the (netmap s2 i) = σ i"
    by - (erule netmap_add_disjoint2)
qed

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

lemma inclosed_simps [simp]:
  "∧σ ii ni. inclosed σ (ii~ni:arrive(m)) = eq_newpkt m"
  "∧σ d di. inclosed σ (i:newpkt(d, di)) = False"
  "∧σ R m. inclosed σ (R:*cast(m)) = True"
  "∧σ i d. inclosed σ (i:deliver(d)) = True"
  "∧σ i i'. inclosed σ (connect(i, i')) = True"
  "∧σ i i'. inclosed σ (disconnect(i, i')) = True"
  "∧σ. inclosed σ (τ) = True"
by auto

definition
  netmask :: "ip set ⇒ ((ip ⇒ 's) × 'l) ⇒ ((ip ⇒ 's option) × 'l)"
where
  "netmask I s ≡ (λi. if i∈I then Some (fst s i) else None, snd s)"

lemma netmask_def' [simp]:
  "netmask I (σ, ζ) = (λi. if i∈I then Some (σ i) else None, ζ)"
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 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 (netgmap sr s1) = fst (netmask (net_tree_ips n1) ( $\sigma$ , snd (netgmap sr s1)))"
  proof (rule ext, simp, intro conjI impI)
    fix i
    assume "i $\in$ net_tree_ips n1"
    with 'net_tree_ips n1  $\cap$  net_tree_ips n2 = {}' have "i $\notin$ net_tree_ips n2"
    by auto
    from assms(1) [simplified prod_eq_iff]
    have "(fst (netgmap sr s1) ++ fst (netgmap sr s2)) i =
      (if i  $\in$  net_tree_ips n1  $\vee$  i  $\in$  net_tree_ips n2 then Some ( $\sigma$  i) else None)"
    by simp
    also from 'i $\notin$ net_tree_ips n2' and 'net_ips s2 = net_tree_ips n2'
    have "(fst (netgmap sr s1) ++ fst (netgmap sr s2)) i = fst (netgmap sr s1) i"
    by (metis dom_fst_netgmap map_add_dom_app_simps(3))
    finally show "fst (netgmap sr s1) i = Some ( $\sigma$  i)"
    using 'i $\in$ net_tree_ips n1' by simp
  next
    fix i
    assume "i  $\notin$  net_tree_ips n1"
    with 'net_ips s1 = net_tree_ips n1' have "i  $\notin$  net_ips s1" by simp
    thus "fst (netgmap 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)) ( $\sigma$ ,  $\zeta$ )"
  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) ( $\sigma$ , snd (netgmap sr s2))"
proof (rule prod_eqI)
  show "fst (netgmap sr s2) = fst (netmask (net_tree_ips n2) ( $\sigma$ , snd (netgmap sr s2)))"
  proof (rule ext, simp, intro conjI impI)
    fix i
    assume "i $\in$ net_tree_ips n2"
    from assms(1) [simplified prod_eq_iff]
    have "(fst (netgmap sr s1) ++ fst (netgmap sr s2)) i =
      (if i  $\in$  net_tree_ips n1  $\vee$  i  $\in$  net_tree_ips n2 then Some ( $\sigma$  i) else None)"
    by simp
    also from 'i $\in$ net_tree_ips n2' and 'net_ips s2 = net_tree_ips n2'
    have "(fst (netgmap sr s1) ++ fst (netgmap sr s2)) i = fst (netgmap sr s2) i"
    by (metis dom_fst_netgmap map_add_dom_app_simps(1))
    finally show "fst (netgmap sr s2) i = Some ( $\sigma$  i)"
    using 'i $\in$ net_tree_ips n2' by simp
  next
    fix i
    assume "i  $\notin$  net_tree_ips n2"
    with 'net_ips s2 = net_tree_ips n2' have "i  $\notin$  net_ips s2" by simp
    thus "fst (netgmap sr s2) i = None" by simp
  qed
qed simp

```

lemma netmap_fst_netgmap_rel:

```

shows "( $\lambda$ i. map_option (fst  $\circ$  sr) (netmap s i)) = fst (netgmap sr s)"
proof (induction s)
  fix ii s R
  show "( $\lambda$ i. map_option (fst  $\circ$  sr) (netmap (NodeS ii s R) i)) = fst (netgmap sr (NodeS ii s R))"
  by auto
next
  fix s1 s2
  assume a1: "( $\lambda$ i. map_option (fst  $\circ$  sr) (netmap s1 i)) = fst (netgmap sr s1)"
    and a2: "( $\lambda$ i. map_option (fst  $\circ$  sr) (netmap s2 i)) = fst (netgmap sr s2)"
  show "( $\lambda$ i. map_option (fst  $\circ$  sr) (netmap (SubnetS s1 s2) i)) = fst (netgmap sr (SubnetS s1 s2))"
  proof (rule ext)
    fix i

```

```

from a1 a2 have "map_option (fst ∘ 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 ∘ sr) (netmap (SubnetS s1 s2) i) = fst (netgmap sr (SubnetS s1 s2)) i"
  by simp
qed
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 ⇒ (nat ⇒ '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 Γ :: "('s, 'm, 'p, 'l) seqp_env"
where
  broadcastT: "((ξ, {l}broadcast(s_msg).p), broadcast (s_msg ξ), (ξ, p)) ∈ seqp_sos Γ"
  | groupcastT: "((ξ, {l}groupcast(s_ips, s_msg).p), groupcast (s_ips ξ) (s_msg ξ), (ξ, p)) ∈ seqp_sos Γ"

```


$$\begin{aligned}
&/ \text{unicastT: } "((\xi, \{1\}\text{unicast}(s_{ip}, s_{msg}).p \triangleright q), \text{unicast } (s_{ip} \xi) (s_{msg} \xi), (\xi, p)) \in \text{seqp_sos } \Gamma" \\
&/ \text{notunicastT: } "((\xi, \{1\}\text{unicast}(s_{ip}, s_{msg}).p \triangleright q), \neg\text{unicast } (s_{ip} \xi), (\xi, q)) \in \text{seqp_sos } \Gamma" \\
&/ \text{sendT: } "((\xi, \{1\}\text{send}(s_{msg}).p), \text{send } (s_{msg} \xi), (\xi, p)) \in \text{seqp_sos } \Gamma" \\
&/ \text{deliverT: } "((\xi, \{1\}\text{deliver}(s_{data}).p), \text{deliver } (s_{data} \xi), (\xi, p)) \in \text{seqp_sos } \Gamma" \\
&/ \text{receiveT: } "((\xi, \{1\}\text{receive}(u_{msg}).p), \text{receive } \text{msg}, (u_{msg} \text{msg } \xi, p)) \in \text{seqp_sos } \Gamma" \\
&/ \text{assignT: } "((\xi, \{1\}\llbracket u \rrbracket p), \tau, (u \xi, p)) \in \text{seqp_sos } \Gamma" \\
\\
&/ \text{callT: } " \llbracket ((\xi, \Gamma \text{pn}), a, (\xi', p')) \in \text{seqp_sos } \Gamma \rrbracket \implies \\
&\quad ((\xi, \text{call}(\text{pn})), a, (\xi', p')) \in \text{seqp_sos } \Gamma" \\
\\
&/ \text{choiceT1: } "((\xi, p), a, (\xi', p')) \in \text{seqp_sos } \Gamma \implies ((\xi, p \oplus q), a, (\xi', p')) \in \text{seqp_sos } \Gamma" \\
&/ \text{choiceT2: } "((\xi, q), a, (\xi', q')) \in \text{seqp_sos } \Gamma \implies ((\xi, p \oplus q), a, (\xi', q')) \in \text{seqp_sos } \Gamma" \\
\\
&/ \text{guardT: } "\xi' \in g \xi \implies ((\xi, \{1\}\langle g \rangle p), \tau, (\xi', p)) \in \text{seqp_sos } \Gamma"
\end{aligned}$$

inductive_cases

$$\begin{aligned}
&\text{seqp_callTE [elim]: } "((\xi, \text{call}(\text{pn})), a, (\xi', q)) \in \text{seqp_sos } \Gamma" \\
&\text{and seqp_choiceTE [elim]: } "((\xi, p1 \oplus p2), a, (\xi', q)) \in \text{seqp_sos } \Gamma"
\end{aligned}$$

lemma seqp_broadcastTE [elim]:

$$\begin{aligned}
&" \llbracket ((\xi, \{1\}\text{broadcast}(s_{msg}).p), a, (\xi', q)) \in \text{seqp_sos } \Gamma; \\
&\quad \llbracket a = \text{broadcast } (s_{msg} \xi); \xi' = \xi; q = p \rrbracket \implies P \rrbracket \implies P" \\
&\text{by (ind_cases "((\xi, \{1\}\text{broadcast}(s_{msg}).p), a, (\xi', q)) \in \text{seqp_sos } \Gamma") simp}
\end{aligned}$$

lemma seqp_groupcastTE [elim]:

$$\begin{aligned}
&" \llbracket ((\xi, \{1\}\text{groupcast}(s_{ips}, s_{msg}).p), a, (\xi', q)) \in \text{seqp_sos } \Gamma; \\
&\quad \llbracket a = \text{groupcast } (s_{ips} \xi) (s_{msg} \xi); \xi' = \xi; q = p \rrbracket \implies P \rrbracket \implies P" \\
&\text{by (ind_cases "((\xi, \{1\}\text{groupcast}(s_{ips}, s_{msg}).p), a, (\xi', q)) \in \text{seqp_sos } \Gamma") simp}
\end{aligned}$$

lemma seqp_unicastTE [elim]:

$$\begin{aligned}
&" \llbracket ((\xi, \{1\}\text{unicast}(s_{ip}, s_{msg}).p \triangleright q), a, (\xi', r)) \in \text{seqp_sos } \Gamma; \\
&\quad \llbracket a = \text{unicast } (s_{ip} \xi) (s_{msg} \xi); \xi' = \xi; r = p \rrbracket \implies P; \\
&\quad \llbracket a = \neg\text{unicast } (s_{ip} \xi); \xi' = \xi; r = q \rrbracket \implies P \rrbracket \implies P" \\
&\text{by (ind_cases "((\xi, \{1\}\text{unicast}(s_{ip}, s_{msg}).p \triangleright q), a, (\xi', r)) \in \text{seqp_sos } \Gamma") simp_all}
\end{aligned}$$

lemma seqp_sendTE [elim]:

$$\begin{aligned}
&" \llbracket ((\xi, \{1\}\text{send}(s_{msg}).p), a, (\xi', q)) \in \text{seqp_sos } \Gamma; \\
&\quad \llbracket a = \text{send } (s_{msg} \xi); \xi' = \xi; q = p \rrbracket \implies P \rrbracket \implies P" \\
&\text{by (ind_cases "((\xi, \{1\}\text{send}(s_{msg}).p), a, (\xi', q)) \in \text{seqp_sos } \Gamma") simp}
\end{aligned}$$

lemma seqp_deliverTE [elim]:

$$\begin{aligned}
&" \llbracket ((\xi, \{1\}\text{deliver}(s_{data}).p), a, (\xi', q)) \in \text{seqp_sos } \Gamma; \\
&\quad \llbracket a = \text{deliver } (s_{data} \xi); \xi' = \xi; q = p \rrbracket \implies P \rrbracket \implies P" \\
&\text{by (ind_cases "((\xi, \{1\}\text{deliver}(s_{data}).p), a, (\xi', q)) \in \text{seqp_sos } \Gamma") simp}
\end{aligned}$$

lemma seqp_receiveTE [elim]:

$$\begin{aligned}
&" \llbracket ((\xi, \{1\}\text{receive}(u_{msg}).p), a, (\xi', q)) \in \text{seqp_sos } \Gamma; \\
&\quad \llbracket \bigwedge \text{msg. } \llbracket a = \text{receive } \text{msg}; \xi' = u_{msg} \text{msg } \xi; q = p \rrbracket \implies P \rrbracket \implies P" \\
&\text{by (ind_cases "((\xi, \{1\}\text{receive}(u_{msg}).p), a, (\xi', q)) \in \text{seqp_sos } \Gamma") simp}
\end{aligned}$$

lemma seqp_assignTE [elim]:

$$\begin{aligned}
&" \llbracket ((\xi, \{1\}\llbracket u \rrbracket p), a, (\xi', q)) \in \text{seqp_sos } \Gamma; \llbracket a = \tau; \xi' = u \xi; q = p \rrbracket \implies P \rrbracket \implies P" \\
&\text{by (ind_cases "((\xi, \{1\}\llbracket u \rrbracket p), a, (\xi', q)) \in \text{seqp_sos } \Gamma") simp}
\end{aligned}$$

lemma seqp_guardTE [elim]:

$$\begin{aligned}
&" \llbracket ((\xi, \{1\}\langle g \rangle p), a, (\xi', q)) \in \text{seqp_sos } \Gamma; \llbracket a = \tau; \xi' \in g \xi; q = p \rrbracket \implies P \rrbracket \implies P" \\
&\text{by (ind_cases "((\xi, \{1\}\langle g \rangle p), a, (\xi', q)) \in \text{seqp_sos } \Gamma") simp}
\end{aligned}$$

lemmas seqpTEs =

$$\begin{aligned}
&\text{seqp_broadcastTE} \\
&\text{seqp_groupcastTE} \\
&\text{seqp_unicastTE} \\
&\text{seqp_sendTE} \\
&\text{seqp_deliverTE}
\end{aligned}$$

```

seqp_receiveTE
seqp_assignTE
seqp_callTE
seqp_choiceTE
seqp_guardTE

```

```
declare seqp_sos.intros [intro]
```

6.2 Table 2: Structural operational semantics for parallel process expressions

inductive_set

```

parp_sos :: "('s1, 'm seq_action) transition set
           => ('s2, 'm seq_action) transition set
           => ('s1 × 's2, 'm seq_action) transition set"

```

```
for S :: "('s1, 'm seq_action) transition set"
```

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

where

```

parleft: "[ (s, a, s') ∈ S; ∧m. a ≠ receive m ] => ((s, t), a, (s', t)) ∈ parp_sos S T"
| parright: "[ (t, a, t') ∈ T; ∧m. a ≠ send m ] => ((s, t), a, (s, t')) ∈ parp_sos S T"
| parboth: "[ (s, receive m, s') ∈ S; (t, send m, t') ∈ T ]
            => ((s, t), τ, (s', t')) ∈ parp_sos S T"

```

lemma par_broadcastTE [elim]:

```

"[(s, t), broadcast m, (s', t')] ∈ parp_sos S T;
 [(s, broadcast m, s') ∈ S; t' = t] => P;
 [(t, broadcast m, t') ∈ T; s' = s] => P] => P"
by (ind_cases "(s, t), broadcast m, (s', t') ∈ parp_sos S T") simp_all

```

lemma par_groupcastTE [elim]:

```

"[(s, t), groupcast ips m, (s', t')] ∈ parp_sos S T;
 [(s, groupcast ips m, s') ∈ S; t' = t] => P;
 [(t, groupcast ips m, t') ∈ T; s' = s] => P] => P"
by (ind_cases "(s, t), groupcast ips m, (s', t') ∈ parp_sos S T") simp_all

```

lemma par_unicastTE [elim]:

```

"[(s, t), unicast i m, (s', t')] ∈ parp_sos S T;
 [(s, unicast i m, s') ∈ S; t' = t] => P;
 [(t, unicast i m, t') ∈ T; s' = s] => P] => P"
by (ind_cases "(s, t), unicast i m, (s', t') ∈ parp_sos S T") simp_all

```

lemma par_notunicastTE [elim]:

```

"[(s, t), notunicast i, (s', t')] ∈ parp_sos S T;
 [(s, notunicast i, s') ∈ S; t' = t] => P;
 [(t, notunicast i, t') ∈ T; s' = s] => P] => P"
by (ind_cases "(s, t), notunicast i, (s', t') ∈ parp_sos S T") simp_all

```

lemma par_sendTE [elim]:

```

"[(s, t), send m, (s', t')] ∈ parp_sos S T;
 [(s, send m, s') ∈ S; t' = t] => P] => P"
by (ind_cases "(s, t), send m, (s', t') ∈ parp_sos S T") auto

```

lemma par_deliverTE [elim]:

```

"[(s, t), deliver d, (s', t')] ∈ parp_sos S T;
 [(s, deliver d, s') ∈ S; t' = t] => P;
 [(t, deliver d, t') ∈ T; s' = s] => P] => P"
by (ind_cases "(s, t), deliver d, (s', t') ∈ parp_sos S T") simp_all

```

lemma par_receiveTE [elim]:

```

"[(s, t), receive m, (s', t')] ∈ parp_sos S T;
 [(t, receive m, t') ∈ T; s' = s] => P] => P"
by (ind_cases "(s, t), receive m, (s', t') ∈ parp_sos S T") auto

```

inductive_cases par_tauTE: "(s, t), τ, (s', t') ∈ 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; ∧m. a ≠ receive m; t' = t ] ⇒ P"
    and "[ (t, a, t') ∈ T; ∧m. a ≠ send m; s' = s ] ⇒ P"
    and "∧m. [ (s, receive m, s') ∈ S; (t, send m, t') ∈ T ] ⇒ 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"
("(_ << _)" [102, 103] 102)

```

where

```

"s << t ≡ (| init = init s × init t, trans = parp_sos (trans s) (trans t) |)"

```

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 ⇒ (NodeS i s R, R:*cast(m), NodeS i s' R) ∈ node_sos S"
/ node_gcast:
  "(s, groupcast D m, s') ∈ S ⇒ (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 ] ⇒ (NodeS i s R, {d}:*cast(m), NodeS i s' R) ∈ node_sos S"
/ node_notucast:
  "[ (s, ¬unicast d, s') ∈ S; d∉R ] ⇒ (NodeS i s R, τ, NodeS i s' R) ∈ node_sos S"
/ node_deliver:
  "(s, deliver d, s') ∈ S ⇒ (NodeS i s R, i:deliver(d), NodeS i s' R) ∈ node_sos S"
/ node_receive:
  "(s, receive m, s') ∈ S ⇒ (NodeS i s R, {i}¬{i}:arrive(m), NodeS i s' R) ∈ node_sos S"
/ node_tau:
  "(s, τ, s') ∈ S ⇒ (NodeS i s R, τ, NodeS i s' R) ∈ node_sos S"
/ node_arrive:
  "(NodeS i s R, {i}¬{i}:arrive(m), 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"
/ node_connect_other:

```

"[$i \neq i'$; $i \neq i''$] \implies (NodeS i s R , connect(i' , i''), NodeS i s R) \in node_sos S "
 / node_disconnect_other:
 "[$i \neq i'$; $i \neq i''$] \implies (NodeS i s R , disconnect(i' , i''), NodeS i s R) \in node_sos S "

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

lemma node_sos_never_newpkt [simp]:

assumes "(s , a , s') \in node_sos S "
 shows " $a \neq i$:newpkt(d , di)"
 using assms by cases auto

lemma arrives_or_not:

assumes "(NodeS i s R , $ii \neg ni$:arrive(m), NodeS i' s' R') \in node_sos S "
 shows "($ii = \{i\} \wedge ni = \{\}$) \vee ($ii = \{\} \wedge ni = \{i\}$)"
 using assms by rule simp_all

definition

node_comp :: "ip \implies ('s, 'm seq_action) automaton \implies ip set
 \implies ('s net_state, 'm node_action) automaton"
 ("($\langle _ : _ \rangle$)" [0, 0, 0] 104)

where

" $\langle i : np : R_i \rangle \equiv$ ($\{ \text{init} = \{\text{NodeS } i \text{ } s \text{ } R_i \mid s. s \in \text{init } np\}, \text{trans} = \text{node_sos } (\text{trans } np) \}$)"

lemma trans_node_comp:

"trans ($\langle i : np : R_i \rangle$) = node_sos (trans np)"
 unfolding node_comp_def by simp

lemma init_node_comp:

"init ($\langle i : np : R_i \rangle$) = {NodeS i s $R_i \mid s. s \in \text{init } np}$ "
 unfolding node_comp_def by simp

lemmas node_comps = trans_node_comp init_node_comp

lemma trans_par_node_comp [simp]:

"trans ($\langle i : s \langle t : R \rangle \rangle$) = node_sos (parp_sos (trans s) (trans t))"
 unfolding node_comp_def by simp

lemma snd_par_node_comp [simp]:

"init ($\langle i : s \langle t : R \rangle \rangle$) = {NodeS i st $R \mid st. st \in \text{init } s \times \text{init } t}$ "
 unfolding node_comp_def by simp

lemma node_sos_dest_is_net_state:

assumes "(s , a , s') \in node_sos S "
 shows " $\exists i' P' R'. s' = \text{NodeS } i' P' R'$ "
 using assms by induct auto

lemma node_sos_dest:

assumes "(NodeS i p R , a , s') \in node_sos S "
 shows " $\exists P' R'. s' = \text{NodeS } i P' R'$ "
 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 R, a, NodeS i p' R') ∈ node_sos S ⇒
  (∧ m .      [ a = R:*cast(m);          R' = R; (p, broadcast m, p') ∈ S ] ⇒ P) ⇒
  (∧ m D.     [ a = (R ∩ D):*cast(m);    R' = R; (p, groupcast D m, p') ∈ S ] ⇒ P) ⇒
  (∧ d m.     [ a = {d}:*cast(m);        R' = R; (p, unicast d m, p') ∈ S; d ∈ R ] ⇒ P) ⇒
  (∧ d.       [ a = τ;                   R' = R; (p, ¬unicast d, p') ∈ S; d ∉ R ] ⇒ P) ⇒
  (∧ d.       [ a = i:deliver(d);         R' = R; (p, deliver d, p') ∈ S ] ⇒ P) ⇒
  (∧ m.       [ a = {i}¬{j}:arrive(m);    R' = R; (p, receive m, p') ∈ S ] ⇒ P) ⇒
  (          [ a = τ;                     R' = R; (p, τ, p') ∈ S ] ⇒ P) ⇒
  (∧ m.       [ a = {j}¬{i}:arrive(m);    R' = R; p = p' ] ⇒ P) ⇒
  (∧ i i'.    [ a = connect(i, i');       R' = R ∪ {i'}; p = p' ] ⇒ P) ⇒
  (∧ i i'.    [ a = connect(i', i);       R' = R ∪ {i'}; p = p' ] ⇒ P) ⇒
  (∧ i i'.    [ a = disconnect(i, i');    R' = R - {i'}; p = p' ] ⇒ P) ⇒
  (∧ i i'.    [ a = disconnect(i', i);    R' = R - {i'}; p = p' ] ⇒ P) ⇒
  (∧ i i' i''. [ a = connect(i', i'');    R' = R; p = p'; i ≠ i'; i ≠ i'' ] ⇒ P) ⇒
  (∧ i i' i''. [ a = disconnect(i', i''); R' = R; p = p'; i ≠ i'; i ≠ i'' ] ⇒ P) ⇒
  P"
  by (erule 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: "[ (s, R:*cast(m), s') ∈ S; (t, H¬K:arrive(m), t') ∈ T; H ⊆ R; K ∩ R = {} ]
              ⇒ (SubnetS s t, R:*cast(m), SubnetS s' t') ∈ pnet_sos S T"

  / pnet_cast2: "[ (s, H¬K:arrive(m), s') ∈ S; (t, R:*cast(m), t') ∈ T; H ⊆ R; K ∩ R = {} ]
               ⇒ (SubnetS s t, R:*cast(m), SubnetS s' t') ∈ pnet_sos S T"

  / pnet_arrive: "[ (s, H¬K:arrive(m), s') ∈ S; (t, H'¬K':arrive(m), t') ∈ T ]
                 ⇒ (SubnetS s t, (H ∪ H')¬(K ∪ K'):arrive(m), SubnetS s' t') ∈ pnet_sos S T"

  / pnet_deliver1: "(s, i:deliver(d), s') ∈ S
                   ⇒ (SubnetS s t, i:deliver(d), SubnetS s' t) ∈ pnet_sos S T"
  / pnet_deliver2: "[ (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; (t, connect(i, i'), t') ∈ T ]
                  ⇒ (SubnetS s t, connect(i, i'), SubnetS s' t') ∈ pnet_sos S T"

  / pnet_disconnect: "[ (s, disconnect(i, i'), s') ∈ S; (t, disconnect(i, i'), t') ∈ T ]
                     ⇒ (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-K: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; Ri>) = <i : np i : Ri>"
| "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

```

qed

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

newpkt_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 AWW_Cterms

imports AWW

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) seq  
           => bool"
```

```
for  $\Gamma$  ::>('s, 'm, 'p, 'l) seqp_env"
```

where

```
microstep_choiceI1 [intro, simp]: "microstep  $\Gamma$  (p1  $\oplus$  p2) p1"  
| microstep_choiceI2 [intro, simp]: "microstep  $\Gamma$  (p1  $\oplus$  p2) p2"  
| microstep_callI [intro, simp]: "microstep  $\Gamma$  (call(pn)) ( $\Gamma$  pn)"
```

abbreviation microstep_rtcl

```
where "microstep_rtcl  $\Gamma$  p q  $\equiv$  (microstep  $\Gamma$ )** p q"
```

abbreviation microstep_tcl

```
where "microstep_tcl  $\Gamma$  p q  $\equiv$  (microstep  $\Gamma$ )++ p q"
```

syntax

```
"_microstep"  
  :: [('s, 'm, 'p, 'l) seqp, ('s, 'm, 'p, 'l) seqp_env, ('s, 'm, 'p, 'l) seqp] => bool"  
  ("(_  $\rightsquigarrow$  _)" [61, 0, 61] 50)  
"_microstep_rtcl"  
  :: [('s, 'm, 'p, 'l) seqp, ('s, 'm, 'p, 'l) seqp_env, ('s, 'm, 'p, 'l) seqp] => bool"  
  ("(_  $\rightsquigarrow$ * _)" [61, 0, 61] 50)  
"_microstep_tcl"  
  :: [('s, 'm, 'p, 'l) seqp, ('s, 'm, 'p, 'l) seqp_env, ('s, 'm, 'p, 'l) seqp] => bool"  
  ("(_  $\rightsquigarrow$ + _)" [61, 0, 61] 50)
```

translations

```
"p1  $\rightsquigarrow$  $\Gamma$  p2"  $\equiv$  "CONST microstep  $\Gamma$  p1 p2"  
"p1  $\rightsquigarrow$  $\Gamma$ * p2"  $\equiv$  "CONST microstep_rtcl  $\Gamma$  p1 p2"  
"p1  $\rightsquigarrow$  $\Gamma$ + p2"  $\equiv$  "CONST microstep_tcl  $\Gamma$  p1 p2"
```

lemma microstep_choiceD [dest]:

```
"(p1  $\oplus$  p2)  $\rightsquigarrow$  $\Gamma$  p  $\implies$  p = p1  $\vee$  p = p2"  
by (ind_cases "(p1  $\oplus$  p2)  $\rightsquigarrow$  $\Gamma$  p") auto
```

lemma microstep_choiceE [elim]:

```
"[ (p1  $\oplus$  p2)  $\rightsquigarrow$  $\Gamma$  p;  
  (p1  $\oplus$  p2)  $\rightsquigarrow$  $\Gamma$  p1  $\implies$  P;  
  (p1  $\oplus$  p2)  $\rightsquigarrow$  $\Gamma$  p2  $\implies$  P ]  $\implies$  P"  
by (blast)
```

lemma microstep_callD [dest]:

```
"(call(pn))  $\rightsquigarrow$  $\Gamma$  p  $\implies$  p =  $\Gamma$  pn"  
by (ind_cases "(call(pn))  $\rightsquigarrow$  $\Gamma$  p")
```

lemma microstep_callE [elim]:

```
"[ (call(pn))  $\rightsquigarrow$  $\Gamma$  p; p =  $\Gamma$ (pn)  $\implies$  P ]  $\implies$  P"  
by auto
```

lemma no_microstep_guard: " \neg (({l}<g> p) \rightsquigarrow Γ q)"

```
by (rule notI) (ind_cases "{l}<g> p)  $\rightsquigarrow$  $\Gamma$  q")
```

lemma no_microstep_assign: " \neg ({l}[f] p) \rightsquigarrow Γ q"

```
by (rule notI) (ind_cases "{l}[f] p)  $\rightsquigarrow$  $\Gamma$  q")
```

lemma no_microstep_unicast: " \neg (({l}unicast(s_{ip}, s_{msg}).p \triangleright q) \rightsquigarrow Γ r)"

```
by (rule notI) (ind_cases "{l}unicast(sip, smsg).p  $\triangleright$  q)  $\rightsquigarrow$  $\Gamma$  r")
```

lemma no_microstep_broadcast: " \neg (({l}broadcast(s_{msg}).p) \rightsquigarrow Γ q)"

```
by (rule notI) (ind_cases "{l}broadcast(smsg).p)  $\rightsquigarrow$  $\Gamma$  q")
```

lemma no_microstep_groupcast: " \neg (({l}groupcast(s_{ips}, s_{msg}).p) \rightsquigarrow Γ q)"

by (rule notI) (ind_cases "{1}groupcast(s_{ips}, s_{msg}).p) $\rightsquigarrow_{\Gamma}$ q")

lemma no_microstep_send: " \neg (({1}send(s_{msg}).p) $\rightsquigarrow_{\Gamma}$ q)"
 by (rule notI) (ind_cases "{1}send(s_{msg}).p) $\rightsquigarrow_{\Gamma}$ q")

lemma no_microstep_deliver: " \neg (({1}deliver(s_{data}).p) $\rightsquigarrow_{\Gamma}$ q)"
 by (rule notI) (ind_cases "{1}deliver(s_{data}).p) $\rightsquigarrow_{\Gamma}$ q")

lemma no_microstep_receive: " \neg (({1}receive(u_{msg}).p) $\rightsquigarrow_{\Gamma}$ q)"
 by (rule notI) (ind_cases "{1}receive(u_{msg}).p) $\rightsquigarrow_{\Gamma}$ q")

lemma microstep_call_or_choice [dest]:
 assumes "p $\rightsquigarrow_{\Gamma}$ q"
 shows "(\exists pn. p = call(pn)) \vee (\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 Γ is wellformed if its *microstep* Γ relation is free of loops and infinite chains.

For example, these specifications are not wellformed: Γ_1 p1 = call(p1)

Γ_2 p1 = send(msg) . call(p1) \oplus call(p1)

Γ_3 p1 = send(msg) . call(p2) Γ_3 p2 = call(p3) Γ_3 p3 = call(p4) Γ_3 p4 = call(p5) ...

definition

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

where

"wellformed Γ = wf {(q, p). p $\rightsquigarrow_{\Gamma}$ q}"

lemma wellformed_defP: "wellformed Γ = wfP (λ q p. p $\rightsquigarrow_{\Gamma}$ q)"
 unfolding wellformed_def wfP_def by simp

The induction rule for wellformed Γ is stronger than $\llbracket \bigwedge 1$ fun seqp. ?P seqp \Longrightarrow ?P ({1}<fun> seqp); $\bigwedge 1$ fun seqp. ?P seqp \Longrightarrow ?P ({1}[[fun]] seqp); \bigwedge seqp1 seqp2. \llbracket ?P seqp1; ?P seqp2 $\rrbracket \Longrightarrow$?P (seqp1 \oplus seqp2); $\bigwedge 1$ fun1 fun2 seqp1 seqp2. \llbracket ?P seqp1; ?P seqp2 $\rrbracket \Longrightarrow$?P ({1}unicast(fun1, fun2) . seqp1 \triangleright seqp2); $\bigwedge 1$ fun seqp. ?P seqp \Longrightarrow ?P ({1}broadcast(fun) . seqp); $\bigwedge 1$ fun1 fun2 seqp. ?P seqp \Longrightarrow ?P ({1}groupcast(fun1, fun2) . seqp); $\bigwedge 1$ fun seqp. ?P seqp \Longrightarrow ?P ({1}send(fun) . seqp); $\bigwedge 1$ fun seqp. ?P seqp \Longrightarrow ?P ({1}deliver(fun) . seqp); $\bigwedge 1$ fun seqp. ?P seqp \Longrightarrow ?P ({1}receive(fun) . seqp); \bigwedge p. ?P (call(p)) $\rrbracket \Longrightarrow$?P ?seqp because the case for call(pn) can be shown with the assumption on Γ pn.

lemma wellformed_induct

[consumes 1, case_names ASSIGN CHOICE CALL GUARD UCAST BCAST GCAST SEND DELIVER RECEIVE,
 induct set: wellformed]:

assumes "wellformed Γ "

and ASSIGN:	" $\bigwedge 1$ f p.	wellformed $\Gamma \Longrightarrow P$ ({1}[[f]] p)"
and GUARD:	" $\bigwedge 1$ f p.	wellformed $\Gamma \Longrightarrow P$ ({1}<f> p)"
and UCAST:	" $\bigwedge 1$ fip fmsg p q.	wellformed $\Gamma \Longrightarrow P$ ({1}unicast(fip, fmsg). p \triangleright q)"
and BCAST:	" $\bigwedge 1$ fmsg p.	wellformed $\Gamma \Longrightarrow P$ ({1}broadcast(fmsg). p)"
and GCAST:	" $\bigwedge 1$ fips fmsg p.	wellformed $\Gamma \Longrightarrow P$ ({1}groupcast(fips, fmsg). p)"
and SEND:	" $\bigwedge 1$ fmsg p.	wellformed $\Gamma \Longrightarrow P$ ({1}send(fmsg). p)"
and DELIVER:	" $\bigwedge 1$ fdata p.	wellformed $\Gamma \Longrightarrow P$ ({1}deliver(fdata). p)"
and RECEIVE:	" $\bigwedge 1$ fmsg p.	wellformed $\Gamma \Longrightarrow P$ ({1}receive(fmsg). p)"
and CHOICE:	" \bigwedge p1 p2.	\llbracket wellformed Γ ; P p1; P p2 $\rrbracket \Longrightarrow P$ (p1 \oplus p2)"
and CALL:	" \bigwedge pn.	\llbracket wellformed Γ ; P (Γ pn) $\rrbracket \Longrightarrow P$ (call(pn))"

shows "P a"

```

using assms(1) unfolding wellformed_defP
proof (rule wfP_induct_rule, case_tac x, simp_all)
  fix p1 p2
  assume " $\bigwedge q. (p1 \oplus p2) \rightsquigarrow_{\Gamma} q \implies P q$ "
  then obtain "P p1" and "P p2" by (auto intro!: microstep.intros)
  thus "P (p1  $\oplus$  p2)" by (rule CHOICE [OF 'wellformed  $\Gamma$ '])
next
  fix pn
  assume " $\bigwedge q. (call(pn)) \rightsquigarrow_{\Gamma} q \implies P q$ "
  hence "P ( $\Gamma$  pn)" by (auto intro!: microstep.intros)
  thus "P (call(pn))" by (rule CALL [OF 'wellformed  $\Gamma$ '])
qed (auto intro: assms)

```

7.3 Start terms (sterms)

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

```

fun stermsl :: "('s, 'm, 'p, 'l) seqp  $\implies$  ('s, 'm, 'p, 'l) seqp set"
where

```

```

  "stermsl (p1  $\oplus$  p2) = stermsl p1  $\cup$  stermsl p2"
  | "stermsl p = {p}"

```

```

lemma stermsl_nobigger: "q  $\in$  stermsl p  $\implies$  size q  $\leq$  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  $\in$  stermsl (p1  $\oplus$  p2) = (p  $\in$  stermsl p1  $\vee$  p  $\in$  stermsl p2)"
  by simp

```

```

lemma stermsl_in_branch[elim]:
  " $\llbracket p \in \text{stermsl } (p1 \oplus p2); p \in \text{stermsl } p1 \implies P; p \in \text{stermsl } p2 \implies P \rrbracket \implies P$ "
  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]:
  " $(\bigcup q \in \text{stermsl } p. \text{stermsl } q) = \text{stermsl } p$ "
  by (induct p) simp_all

```

```

lemma stermsl_in_wfpf:
  assumes AA: "A  $\subseteq$  {(q, p). p  $\rightsquigarrow_{\Gamma}$  q} " " A"
  and *: "p  $\in$  A"
  shows " $\exists r \in \text{stermsl } p. r \in A$ "
  using *
  proof (induction p)
    fix p1 p2
    assume IH1: "p1  $\in$  A  $\implies$   $\exists r \in \text{stermsl } p1. r \in A$ "
      and IH2: "p2  $\in$  A  $\implies$   $\exists r \in \text{stermsl } p2. r \in A$ "
      and *: "p1  $\oplus$  p2  $\in$  A"
    from * and AA have "p1  $\oplus$  p2  $\in$  {(q, p). p  $\rightsquigarrow_{\Gamma}$  q} " " A" by auto
    hence "p1  $\in$  A  $\vee$  p2  $\in$  A" by auto
    hence " $(\exists r \in \text{stermsl } p1. r \in A) \vee (\exists r \in \text{stermsl } p2. r \in A)$ "
  proof
    assume "p1  $\in$  A" hence " $\exists r \in \text{stermsl } p1. r \in A$ " by (rule IH1) thus ?thesis ..

```

```

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.prem show ?case by auto
qed auto

```

```

lemma nocall_stermsl_max:
  assumes "r ∈ stermsl p"
    and "not_call r"
  shows "¬ (r ↘Γ q)"
using assms
by (induction p) auto

```

```

theorem wf_no_direct_calls[intro]:
  fixes Γ :: "('s, 'm, 'p, 'l) seqp_env"
  assumes no_calls: "∧pn. ∀pn'. call(pn') ∉ stermsl(Γ(pn))"
  shows "wellformed Γ"
unfolding wellformed_def wfP_def
proof (rule wfI_pf)
  fix A
  assume ARA: "A ⊆ {(q, p). p ↘Γ q} ‘‘ A"
  hence hasnext: "∧p. p ∈ A ⇒ ∃q. p ↘Γ q ∧ q ∈ A" by auto
  show "A = {}"
proof (rule Set.equalsOI)
  fix p assume "p ∈ A" thus "False"
proof (induction p)
  fix l f p'
  assume *: "{l}<f> p' ∈ A"
  from hasnext [OF *] have "∃q. (l}<f> p') ↘Γ q" by simp
  thus "False" by simp
next
  fix p1 p2
  assume *: "p1 ⊕ p2 ∈ A"
  and IH1: "p1 ∈ A ⇒ False"
  and IH2: "p2 ∈ A ⇒ False"
  have "∃q. (p1 ⊕ p2) ↘Γ q ∧ q ∈ A" by (rule hasnext [OF *])
  hence "p1 ∈ A ∨ p2 ∈ A" by auto
  thus "False" by (auto dest: IH1 IH2)
next
  fix pn
  assume "call(pn) ∈ A"
  hence "∃q. (call(pn)) ↘Γ q ∧ q ∈ A" by (rule hasnext)
  hence "Γ(pn) ∈ A" by auto

  with ARA [THEN stermsl_in_wfpf] obtain q where "q∈stermsl (Γ pn)" and "q ∈ A" by metis
  hence "not_call q" using no_calls [of pn]
  unfolding not_call_def by auto

  from hasnext [OF 'q ∈ A'] obtain q' where "q ↘Γ q'" by auto
  moreover from 'q ∈ stermsl (Γ pn)' 'not_call q' have "¬ (q ↘Γ q)"
  by (rule nocall_stermsl_max)
  ultimately show "False" by simp
qed (auto dest: hasnext)
qed
qed

```

7.4 Start terms

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

```

function (domintros, sequential) sterms

```

```

:: "('s, 'm, 'p, 'l) seqp_env  $\Rightarrow$  ('s, 'm, 'p, 'l) seqp  $\Rightarrow$  ('s, 'm, 'p, 'l) seqp set"
where
  sterms_choice: "sterms  $\Gamma$  (p1  $\oplus$  p2) = sterms  $\Gamma$  p1  $\cup$  sterms  $\Gamma$  p2"
/ sterms_call: "sterms  $\Gamma$  (call(pn)) = sterms  $\Gamma$  ( $\Gamma$  pn)"
/ sterms_other: "sterms  $\Gamma$  p = {p}"
by pat_completeness auto

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

lemma sterms_termination:
  assumes "wellformed  $\Gamma$ "
  shows "sterms_dom ( $\Gamma$ , p)"
proof -
  have sterms_rel':
    "sterms_rel = ( $\lambda$ gq gp. (gq, gp)  $\in$  {(( $\Gamma$ , q), ( $\Gamma'$ , p)).  $\Gamma$  =  $\Gamma'$   $\wedge$  p  $\rightsquigarrow_{\Gamma}$  q})"
  by (rule ext)+ (auto simp: sterms_rel.simps elim: microstep.cases)

  from assms have " $\forall$ x. x  $\in$  Wellfounded.acc {(q, p). p  $\rightsquigarrow_{\Gamma}$  q}"
  unfolding wellformed_def by (simp add: wf_acc_iff)
  hence "p  $\in$  Wellfounded.acc {(q, p). p  $\rightsquigarrow_{\Gamma}$  q}" ..

  hence "( $\Gamma$ , p)  $\in$  Wellfounded.acc {( $\Gamma$ , q), ( $\Gamma'$ , p)}.  $\Gamma$  =  $\Gamma'$   $\wedge$  p  $\rightsquigarrow_{\Gamma}$  q}"
  by (rule acc_induct) (auto intro: accI)

  thus "sterms_dom ( $\Gamma$ , p)" unfolding sterms_rel' accp_acc_eq .
qed

declare sterms.psimps [simp]

lemmas sterms_psimps[simp] = sterms.psimps [OF sterms_termination]
and sterms_pinduct = sterms.pinduct [OF sterms_termination]

lemma sterms_reflD [dest]:
  assumes "q  $\in$  sterms  $\Gamma$  p"
    and "not_choice p" "not_call p"
  shows "q = p"
using assms by (cases p) auto

lemma sterms_choice_disj [simp]:
  assumes "wellformed  $\Gamma$ "
  shows "p  $\in$  sterms  $\Gamma$  (p1  $\oplus$  p2) = (p  $\in$  sterms  $\Gamma$  p1  $\vee$  p  $\in$  sterms  $\Gamma$  p2)"
using assms by (simp)

lemma sterms_no_choice [simp]:
  assumes "wellformed  $\Gamma$ "
  shows "p1  $\oplus$  p2  $\notin$  sterms  $\Gamma$  p"
using assms by induction auto

lemma sterms_not_choice [simp]:
  assumes "wellformed  $\Gamma$ "
    and "q  $\in$  sterms  $\Gamma$  p"
  shows "not_choice q"
using assms unfolding not_choice_def
by (auto dest: sterms_no_choice)

```

```

lemma sterms_no_call [simp]:
  assumes "wellformed  $\Gamma$ "
  shows "call(pn)  $\notin$  sterms  $\Gamma$  p"
  using assms by induction auto

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

lemma sterms_in_branch:
  assumes "wellformed  $\Gamma$ "
  and "p  $\in$  sterms  $\Gamma$  (p1  $\oplus$  p2)"
  and "p  $\in$  sterms  $\Gamma$  p1  $\implies$  P"
  and "p  $\in$  sterms  $\Gamma$  p2  $\implies$  P"
  shows "P"
  using assms by auto

lemma sterms_commute:
  assumes "wellformed  $\Gamma$ "
  shows "sterms  $\Gamma$  (p1  $\oplus$  p2) = sterms  $\Gamma$  (p2  $\oplus$  p1)"
  using assms by simp (rule Un_commute)

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

lemma sterms_sterms [simp]:
  assumes "wellformed  $\Gamma$ "
  shows " $(\bigcup_{x \in \text{sterms } \Gamma p. \text{sterms } \Gamma x}) = \text{sterms } \Gamma p$ "
  using assms by induction simp_all

lemma sterms_stermsl:
  assumes "ps  $\in$  sterms  $\Gamma$  p"
  and "wellformed  $\Gamma$ "
  shows "ps  $\in$  stermsl p  $\vee$  ( $\exists$ pn. ps  $\in$  stermsl ( $\Gamma$  pn))"
  using assms by (induction p rule: sterms_pinduct [OF 'wellformed  $\Gamma$ ']) auto

lemma stermsl_sterms [elim]:
  assumes "q  $\in$  stermsl p"
  and "not_call q"
  and "wellformed  $\Gamma$ "
  shows "q  $\in$  sterms  $\Gamma$  p"
  using assms by (induct p) auto

lemma sterms_stermsl_heads:
  assumes "ps  $\in$  sterms  $\Gamma$  ( $\Gamma$  pn)"
  and "wellformed  $\Gamma$ "
  shows " $\exists$ pn. ps  $\in$  stermsl ( $\Gamma$  pn)"
  proof -
    from assms have "ps  $\in$  stermsl ( $\Gamma$  pn)  $\vee$  ( $\exists$ pn'. ps  $\in$  stermsl ( $\Gamma$  pn'))"
    by (rule sterms_stermsl)
    thus ?thesis by auto
  qed

lemma sterms_subterms [dest]:
  assumes "wellformed  $\Gamma$ "
  and " $\exists$ pn. p  $\in$  subterms ( $\Gamma$  pn)"
  and "q  $\in$  sterms  $\Gamma$  p"
  shows " $\exists$ pn. q  $\in$  subterms ( $\Gamma$  pn)"

```

```

using assms by (induct p) auto

lemma no_microsteps_sterms_refl:
  assumes "wellformed  $\Gamma$ "
  shows " $(\neg(\exists q. p \rightsquigarrow_{\Gamma} q)) = (\text{sterms } \Gamma p = \{p\})$ "
  proof (cases p)
    fix p1 p2
    assume "p = p1  $\oplus$  p2"
    from 'wellformed  $\Gamma$ ' have "p1  $\oplus$  p2  $\notin$  sterms  $\Gamma$  (p1  $\oplus$  p2)" by simp
    hence "sterms  $\Gamma$  (p1  $\oplus$  p2)  $\neq$  {p1  $\oplus$  p2}" by auto
    moreover have " $\exists q. (p1 \oplus p2) \rightsquigarrow_{\Gamma} q$ " by auto
    ultimately show ?thesis
      using 'p = p1  $\oplus$  p2' by simp
  next
    fix pn
    assume "p = call(pn)"
    from 'wellformed  $\Gamma$ ' have "call(pn)  $\notin$  sterms  $\Gamma$  (call(pn))" by simp
    hence "sterms  $\Gamma$  (call(pn))  $\neq$  {call(pn)}" by auto
    moreover have " $\exists q. (\text{call}(pn)) \rightsquigarrow_{\Gamma} q$ " by auto
    ultimately show ?thesis
      using 'p = call(pn)' by simp
  qed simp_all

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

lemma microstep_rtranscl_equal:
  assumes "not_call p"
  and "not_choice p"
  and "p  $\rightsquigarrow_{\Gamma}^* q$ "
  shows "q = p"
  using assms(3) proof (rule converse_rtranclpE)
    fix p'
    assume "p  $\rightsquigarrow_{\Gamma} p'$ "
    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  $\rightsquigarrow_{\Gamma}^* q \wedge$  sterms  $\Gamma$  q = {q}} = {p}"
  proof (rule set_eqI)
    fix p'
    show "(p'  $\in$  {q. p  $\rightsquigarrow_{\Gamma}^* q \wedge$  sterms  $\Gamma$  q = {q}}) = (p'  $\in$  {p})"
    proof
      assume "p'  $\in$  {q. p  $\rightsquigarrow_{\Gamma}^* q \wedge$  sterms  $\Gamma$  q = {q}}"
      hence "(microstep  $\Gamma$ )** p p'" and "sterms  $\Gamma$  p' = {p'}" by auto
      from this(1) have "p' = p"
      proof (rule converse_rtranclpE)
        fix q assume "p  $\rightsquigarrow_{\Gamma} q$ "
        with 'not_call p' and 'not_choice p' have False
          by (cases p) auto
        thus "p' = p" ..
      qed simp
      thus "p'  $\in$  {p}" by simp
    next
      assume "p'  $\in$  {p}"
      hence "p' = p" ..
      with 'not_call p' and 'not_choice p' show "p'  $\in$  {q. p  $\rightsquigarrow_{\Gamma}^* q \wedge$  sterms  $\Gamma$  q = {q}}"
        by (cases p) simp_all
    qed
  qed

```

qed
qed

theorem *sterms_maximal_microstep*:

assumes "wellformed Γ "

shows "*sterms* Γ $p = \{q. p \rightsquigarrow_{\Gamma}^* q \wedge \neg(\exists q'. q \rightsquigarrow_{\Gamma} q')\}$ "

proof

from 'wellformed Γ ' have "*sterms* Γ $p \subseteq \{q. p \rightsquigarrow_{\Gamma}^* q \wedge \text{sterms } \Gamma q = \{q\}\}$ "

proof induction

fix $p1\ p2$

assume *IH1*: "*sterms* Γ $p1 \subseteq \{q. p1 \rightsquigarrow_{\Gamma}^* q \wedge \text{sterms } \Gamma q = \{q\}\}$ "

and *IH2*: "*sterms* Γ $p2 \subseteq \{q. p2 \rightsquigarrow_{\Gamma}^* q \wedge \text{sterms } \Gamma q = \{q\}\}$ "

have "*sterms* Γ $p1 \subseteq \{q. (p1 \oplus p2) \rightsquigarrow_{\Gamma}^* q \wedge \text{sterms } \Gamma q = \{q\}\}$ "

proof

fix p'

assume " $p' \in \text{sterms } \Gamma p1$ "

with *IH1* have " $p1 \rightsquigarrow_{\Gamma}^* p'$ " by auto

moreover have " $(p1 \oplus p2) \rightsquigarrow_{\Gamma} p1$ " ..

ultimately have " $(p1 \oplus p2) \rightsquigarrow_{\Gamma}^* p'$ "

by - (rule *converse_rtranclp_into_rtranclp*)

moreover from 'wellformed Γ ' and ' $p' \in \text{sterms } \Gamma p1$ ' have "*sterms* Γ $p' = \{p'\}$ " ..

ultimately show " $p' \in \{q. (p1 \oplus p2) \rightsquigarrow_{\Gamma}^* q \wedge \text{sterms } \Gamma q = \{q\}\}$ "

by *simp*

qed

moreover have "*sterms* Γ $p2 \subseteq \{q. (p1 \oplus p2) \rightsquigarrow_{\Gamma}^* q \wedge \text{sterms } \Gamma q = \{q\}\}$ "

proof

fix p'

assume " $p' \in \text{sterms } \Gamma p2$ "

with *IH2* have " $p2 \rightsquigarrow_{\Gamma}^* p'$ " and "*sterms* Γ $p' = \{p'\}$ " by auto

moreover have " $(p1 \oplus p2) \rightsquigarrow_{\Gamma} p2$ " ..

ultimately have " $(p1 \oplus p2) \rightsquigarrow_{\Gamma}^* p'$ "

by - (rule *converse_rtranclp_into_rtranclp*)

with '*sterms* Γ $p' = \{p'\}$ ' show " $p' \in \{q. (p1 \oplus p2) \rightsquigarrow_{\Gamma}^* q \wedge \text{sterms } \Gamma q = \{q\}\}$ "

by *simp*

qed

ultimately show "*sterms* Γ $(p1 \oplus p2) \subseteq \{q. (p1 \oplus p2) \rightsquigarrow_{\Gamma}^* q \wedge \text{sterms } \Gamma q = \{q\}\}$ "

using 'wellformed Γ ' by *simp*

next

fix pn

assume *IH*: "*sterms* Γ $(\Gamma pn) \subseteq \{q. \Gamma pn \rightsquigarrow_{\Gamma}^* q \wedge \text{sterms } \Gamma q = \{q\}\}$ "

show "*sterms* Γ $(\text{call}(pn)) \subseteq \{q. (\text{call}(pn)) \rightsquigarrow_{\Gamma}^* q \wedge \text{sterms } \Gamma q = \{q\}\}$ "

proof

fix p'

assume " $p' \in \text{sterms } \Gamma (\text{call}(pn))$ "

with 'wellformed Γ ' have " $p' \in \text{sterms } \Gamma (\Gamma pn)$ " by *simp*

with *IH* have " $\Gamma pn \rightsquigarrow_{\Gamma}^* p'$ " and "*sterms* Γ $p' = \{p'\}$ " by auto

note *this(1)*

moreover have " $(\text{call}(pn)) \rightsquigarrow_{\Gamma} \Gamma pn$ " by *simp*

ultimately have " $(\text{call}(pn)) \rightsquigarrow_{\Gamma}^* p'$ "

by - (rule *converse_rtranclp_into_rtranclp*)

with '*sterms* Γ $p' = \{p'\}$ ' show " $p' \in \{q. (\text{call}(pn)) \rightsquigarrow_{\Gamma}^* q \wedge \text{sterms } \Gamma q = \{q\}\}$ "

by *simp*

qed

qed *simp_all*

with 'wellformed Γ ' show "*sterms* Γ $p \subseteq \{q. p \rightsquigarrow_{\Gamma}^* q \wedge \neg(\exists q'. q \rightsquigarrow_{\Gamma} q')\}$ "

by (*simp only: no_microsteps_sterms_refl*)

next

from 'wellformed Γ ' have " $\{q. p \rightsquigarrow_{\Gamma}^* q \wedge \text{sterms } \Gamma q = \{q\}\} \subseteq \text{sterms } \Gamma p$ "

proof (induction)

fix $p1\ p2$

assume *IH1*: " $\{q. p1 \rightsquigarrow_{\Gamma}^* q \wedge \text{sterms } \Gamma q = \{q\}\} \subseteq \text{sterms } \Gamma p1$ "

and *IH2*: " $\{q. p2 \rightsquigarrow_{\Gamma}^* q \wedge \text{sterms } \Gamma q = \{q\}\} \subseteq \text{sterms } \Gamma p2$ "

show " $\{q. (p1 \oplus p2) \rightsquigarrow_{\Gamma}^* q \wedge \text{sterms } \Gamma q = \{q\}\} \subseteq \text{sterms } \Gamma (p1 \oplus p2)$ "

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!: rtrancldD sterms_no_choice)
hence "p1 ↘Γ* q' ∨ p2 ↘Γ* q'"
  by (auto dest: trancldD)
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
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!: rtrancldD sterms_no_call)
  moreover have "(call(pn)) ↘Γ Γ pn" ..
  ultimately have "Γ pn ↘Γ* q'"
  by (auto dest!: trancldD)
  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
qed simp_all
with 'wellformed Γ' show "{q. p ↘Γ* q ∧ ¬(∃q'. q ↘Γ q')}" ⊆ sterms Γ p"
by (simp only: no_microsteps_sterms_refl)
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 terms, 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(sip, smsg).p ▷ q) = sterms Γ p ∪ sterms Γ q"
  | "dterms Γ ({l}broadcast(smsg). p) = sterms Γ p"
  | "dterms Γ ({l}groupcast(sips, smsg). p) = sterms Γ p"
  | "dterms Γ ({l}send(smsg).p) = sterms Γ p"
  | "dterms Γ ({l}deliver(sdata).p) = sterms Γ p"
  | "dterms Γ ({l}receive(umsg).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 ( $\Gamma$ , p)"
  with assms show "dterms_dom y"
  by (cases p) (auto simp: dterms_rel.simps)
qed

lemma dterms_termination:
  assumes "wellformed  $\Gamma$ "
  shows "dterms_dom ( $\Gamma$ , p)"
proof -
  have dterms_rel': "dterms_rel = ( $\lambda$ gq gp. (gq, gp)  $\in$  {(( $\Gamma$ , q), ( $\Gamma'$ , p)).  $\Gamma = \Gamma' \wedge p \rightsquigarrow_{\Gamma} q$ })"
  by (rule ext)+ (auto simp: dterms_rel.simps elim: microstep.cases)
  from 'wellformed( $\Gamma$ )' have " $\forall x. x \in \text{Wellfounded.acc } \{(q, p). p \rightsquigarrow_{\Gamma} q\}$ "
  unfolding wellformed_def by (simp add: wf_acc_iff)
  hence "p  $\in$  Wellfounded.acc {( $\Gamma$ , p). p  $\rightsquigarrow_{\Gamma}$  q}" ..
  hence "( $\Gamma$ , p)  $\in$  Wellfounded.acc {(( $\Gamma$ , q), ( $\Gamma'$ , p)).  $\Gamma = \Gamma' \wedge p \rightsquigarrow_{\Gamma} q$ }"
  by (rule acc_induct) (auto intro: accI)
  thus "dterms_dom ( $\Gamma$ , p)"
  unfolding dterms_rel' by (subst accp_acc_eq)
qed

lemmas dterms_psimps [simp] = dterms.psimps [OF dterms_termination]
and dterms_pinduct = dterms.pinduct [OF dterms_termination]

lemma sterms_after_dterms [simp]:
  assumes "wellformed  $\Gamma$ "
  shows "( $\bigcup_{x \in \text{dterms } \Gamma} p. \text{sterms } \Gamma x$ ) = dterms  $\Gamma p$ "
  using assms by (induction p) simp_all

lemma sterms_before_dterms [simp]:
  assumes "wellformed  $\Gamma$ "
  shows "( $\bigcup_{x \in \text{sterms } \Gamma} p. \text{dterms } \Gamma x$ ) = dterms  $\Gamma p$ "
  using assms by (induction p) simp_all

lemma dterms_choice_disj [simp]:
  assumes "wellformed  $\Gamma$ "
  shows "p  $\in$  dterms  $\Gamma$  (p1  $\oplus$  p2) = (p  $\in$  dterms  $\Gamma$  p1  $\vee$  p  $\in$  dterms  $\Gamma$  p2)"
  using assms by (simp)

lemma dterms_in_branch:
  assumes "wellformed  $\Gamma$ "
  and "p  $\in$  dterms  $\Gamma$  (p1  $\oplus$  p2)"
  and "p  $\in$  dterms  $\Gamma$  p1  $\implies P$ "
  and "p  $\in$  dterms  $\Gamma$  p2  $\implies P$ "
  shows "P"
  using assms by auto

lemma dterms_no_choice:
  assumes "wellformed  $\Gamma$ "
  shows "p1  $\oplus$  p2  $\notin$  dterms  $\Gamma$  p"
  using assms by induction simp_all

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

lemma dterms_no_call:
  assumes "wellformed  $\Gamma$ "
  shows "call(pn)  $\notin$  dterms  $\Gamma$  p"
  using assms by induction simp_all

```

```
lemma dterms_not_call [simp]:
```

```
  assumes "wellformed  $\Gamma$ "
    and "q  $\in$  dterms  $\Gamma$  p"
  shows "not_call q"
using assms unfolding not_call_def
by (auto dest: dterms_no_call)
```

```
lemma dterms_subterms:
```

```
  assumes wf: "wellformed  $\Gamma$ "
    and "∃pn. p  $\in$  subterms ( $\Gamma$  pn)"
    and "q  $\in$  dterms  $\Gamma$  p"
  shows "∃pn. q  $\in$  subterms ( $\Gamma$  pn)"
using assms
proof (induct p)
  fix p1 p2
  assume IH1: "∃pn. p1  $\in$  subterms ( $\Gamma$  pn)  $\implies$  q  $\in$  dterms  $\Gamma$  p1  $\implies$  ∃pn. q  $\in$  subterms ( $\Gamma$  pn)"
    and IH2: "∃pn. p2  $\in$  subterms ( $\Gamma$  pn)  $\implies$  q  $\in$  dterms  $\Gamma$  p2  $\implies$  ∃pn. q  $\in$  subterms ( $\Gamma$  pn)"
    and *: "∃pn. p1  $\oplus$  p2  $\in$  subterms ( $\Gamma$  pn)"
    and "q  $\in$  dterms  $\Gamma$  (p1  $\oplus$  p2)"
  from * obtain pn where "p1  $\oplus$  p2  $\in$  subterms ( $\Gamma$  pn)"
  by auto
  hence "p1  $\in$  subterms ( $\Gamma$  pn)" and "p2  $\in$  subterms ( $\Gamma$  pn)"
  by auto
  from 'q  $\in$  dterms  $\Gamma$  (p1  $\oplus$  p2)' wf have "q  $\in$  dterms  $\Gamma$  p1  $\vee$  q  $\in$  dterms  $\Gamma$  p2"
  by auto
  thus "∃pn. q  $\in$  subterms ( $\Gamma$  pn)"
  proof
    assume "q  $\in$  dterms  $\Gamma$  p1"
    with 'p1  $\in$  subterms ( $\Gamma$  pn)' show ?thesis
    by (auto intro: IH1)
  next
    assume "q  $\in$  dterms  $\Gamma$  p2"
    with 'p2  $\in$  subterms ( $\Gamma$  pn)' show ?thesis
    by (auto intro: IH2)
  qed
qed auto
```

Note that the converse of $\llbracket \text{wellformed } ?\Gamma; \exists pn. ?p \in \text{subterms } (?\Gamma \text{ pn}); ?q \in \text{dterms } ?\Gamma \text{ ?p} \rrbracket \implies \exists pn. ?q \in \text{subterms } (?\Gamma \text{ pn})$ 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 $p1 \oplus p2$ 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  $\Rightarrow$  ('s, 'm, 'p, 'l) seqp set"
  for  $\Gamma$  :: "('s, 'm, 'p, 'l) seqp_env"
```

```
where
```

```
  ctermsSI[intro]: "p  $\in$  sterm  $\Gamma$  ( $\Gamma$  pn)  $\implies$  p  $\in$  cterms  $\Gamma$ "
  | ctermsDI[intro]: " $\llbracket pp \in \text{cterm } \Gamma; p \in \text{dterm } \Gamma \text{ pp} \rrbracket \implies p \in \text{cterm } \Gamma$ "
```

```
lemma cterms_not_choice [simp]:
```

```
  assumes "wellformed  $\Gamma$ "
    and "p  $\in$  cterms  $\Gamma$ "
  shows "not_choice p"
using assms
proof (cases p)
  case CHOICE from 'p  $\in$  cterms  $\Gamma$ ' show ?thesis
  using 'wellformed  $\Gamma$ ' by cases simp_all
```

```

qed simp_all

lemma cterms_no_choice [simp]:
  assumes "wellformed  $\Gamma$ "
  shows " $p_1 \oplus p_2 \notin cterms \Gamma$ "
  using assms by (auto dest: cterms_not_choice)

lemma cterms_not_call [simp]:
  assumes "wellformed  $\Gamma$ "
  and "p  $\in cterms \Gamma$ "
  shows "not_call p"
  using assms
  proof (cases p)
    case CALL from 'p  $\in cterms \Gamma$ ' show ?thesis
      using 'wellformed  $\Gamma$ ' by cases simp_all
  qed simp_all

lemma cterms_no_call [simp]:
  assumes "wellformed  $\Gamma$ "
  shows "call(pn)  $\notin cterms \Gamma$ "
  using assms by (auto dest: cterms_not_call)

lemma sterms_cterms [elim]:
  assumes "p  $\in cterms \Gamma$ "
  and "q  $\in sterms \Gamma p$ "
  and "wellformed  $\Gamma$ "
  shows "q  $\in cterms \Gamma$ "
  using assms by - (cases p, auto)

lemma dterms_cterms [elim]:
  assumes "p  $\in cterms \Gamma$ "
  and "q  $\in dterms \Gamma p$ "
  and "wellformed  $\Gamma$ "
  shows "q  $\in cterms \Gamma$ "
  using assms by (cases p) auto

lemma derivs_in_cterms [simp]:
  " $\bigwedge f p. \{1\}\langle f \rangle p \in cterms \Gamma \implies sterms \Gamma p \subseteq cterms \Gamma$ "
  " $\bigwedge f p. \{1\}[f] p \in cterms \Gamma \implies sterms \Gamma p \subseteq cterms \Gamma$ "
  " $\bigwedge fip fmsg q p. \{1\}unicast(fip, fmsg). p \triangleright q \in cterms \Gamma \implies sterms \Gamma p \subseteq cterms \Gamma \wedge sterms \Gamma q \subseteq cterms \Gamma$ "
  " $\bigwedge fmsg p. \{1\}broadcast(fmsg).p \in cterms \Gamma \implies sterms \Gamma p \subseteq cterms \Gamma$ "
  " $\bigwedge fips fmsg p. \{1\}groupcast(fips, fmsg).p \in cterms \Gamma \implies sterms \Gamma p \subseteq cterms \Gamma$ "
  " $\bigwedge fmsg p. \{1\}send(fmsg).p \in cterms \Gamma \implies sterms \Gamma p \subseteq cterms \Gamma$ "
  " $\bigwedge fdata p. \{1\}deliver(fdata).p \in cterms \Gamma \implies sterms \Gamma p \subseteq cterms \Gamma$ "
  " $\bigwedge fmsg p. \{1\}receive(fmsg).p \in cterms \Gamma \implies sterms \Gamma p \subseteq cterms \Gamma$ "
  by (auto simp: dterms.psimps)

```

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 *sterms1*, even if they must be filtered out for the desired characterisation.

function

cterms1 :: "(*s*, *m*, *p*, *l*) seqp \Rightarrow (*s*, *m*, *p*, *l*) seqp set"

where

```

"cterms1 ( $\{1\}\langle g \rangle p$ ) = insert ( $\{1\}\langle g \rangle p$ ) (cterms1 p)"
| "cterms1 ( $\{1\}[u] p$ ) = insert ( $\{1\}[u] p$ ) (cterms1 p)"
| "cterms1 ( $\{1\}unicast(s_{ip}, s_{msg}). p \triangleright q$ ) = insert ( $\{1\}unicast(s_{ip}, s_{msg}). p \triangleright q$ )
  (cterms1 p  $\cup$  cterms1 q)"
| "cterms1 ( $\{1\}broadcast(s_{msg}). p$ ) = insert ( $\{1\}broadcast(s_{msg}). p$ ) (cterms1 p)"
| "cterms1 ( $\{1\}groupcast(s_{ips}, s_{msg}). p$ ) = insert ( $\{1\}groupcast(s_{ips}, s_{msg}). p$ ) (cterms1 p)"

```

```

| "ctermsl ( $\{l\}$ send( $s_{msg}$ ). p)          = insert ( $\{l\}$ send( $s_{msg}$ ). p)          (ctermsl p)"
| "ctermsl ( $\{l\}$ deliver( $s_{data}$ ). p)      = insert ( $\{l\}$ deliver( $s_{data}$ ). p)      (ctermsl p)"
| "ctermsl ( $\{l\}$ receive( $u_{msg}$ ). p)      = insert ( $\{l\}$ receive( $u_{msg}$ ). p)      (ctermsl p)"
| "ctermsl ( $p_1 \oplus p_2$ )                = ctermsl  $p_1 \cup$  ctermsl  $p_2$ "
| "ctermsl (call( $pn$ ))                    = {call( $pn$ )}"
by 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  $\in$  subterms p  $\wedge$  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)  $\vee$  (q  $\in$  ctermsl p2)"
    using CHOICE.prem1 by simp
  hence "r  $\in$  ctermsl p1  $\vee$  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' ; not_call p ]]  $\implies$  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"

```

```

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  $\Gamma$  p  $\subseteq$  cterms  $\Gamma$ "
    and "wellformed  $\Gamma$ "
  shows "q ∈ cterms  $\Gamma$ "
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  $\Rightarrow$  ('s, 'm, 'p, 'l) seqp set"
  where
    "dtermsl ({}<fg> p)           = stermsl p"
  | "dtermsl ({}[[fa]] p)        = stermsl p"
  | "dtermsl (p1  $\oplus$  p2)        = dtermsl p1  $\cup$  dtermsl p2"
  | "dtermsl ({}unicast(fip, fmsg).p  $\triangleright$  q) = stermsl p  $\cup$  stermsl q"
  | "dtermsl ({}broadcast(fmsg). p)      = stermsl p"
  | "dtermsl ({}groupcast(fips, fmsg). p) = stermsl p"
  | "dtermsl ({}send(fmsg).p)           = stermsl p"
  | "dtermsl ({}deliver(fdata).p)       = stermsl p"
  | "dtermsl ({}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 "( $\bigcup x \in \text{dtermsl } p. \text{stermsl } x$ ) = dtermsl p"
using assms by (induct p) simp_all

```

```

lemma stermsl_before_dtermsl [simp]:
  "( $\bigcup x \in \text{stermsl } p. \text{dtermsl } x$ ) = dtermsl p"
using assms by (induct p) simp_all

```

```

lemma dtermsl_no_choice [simp]: "p1  $\oplus$  p2  $\notin$  dtermsl p"
  by (induct p) simp_all

```

```

lemma dtermsl_choice_disj [simp]:
  "p ∈ dtermsl (p1  $\oplus$  p2) = (p ∈ dtermsl p1  $\vee$  p ∈ dtermsl p2)"
  by simp

```

```

lemma dtermsl_in_branch [elim]:
  "[[p ∈ dtermsl (p1  $\oplus$  p2); p ∈ dtermsl p1  $\implies$  P; p ∈ dtermsl p2  $\implies$  P]]  $\implies$  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_dterms [elim]:
  assumes "q ∈ dtermsl p"
    and "not_call q"
    and "wellformed  $\Gamma$ "
  shows "q ∈ dterms  $\Gamma$  p"
using assms
using assms by (induct p) (simp_all, (metis stermsl_sterms)+)

```

```

lemma ctermsl_stermsl_or_dtermsl:
  assumes "q ∈ ctermsl p"
  shows "q ∈ stermsl p  $\vee$  ( $\exists p' \in \text{dtermsl } p. q \in \text{ctermsl } p'$ )"
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_reachable`, 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 ∈ stermsl (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 'p ∈ 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)
  thus "p ∈ cterms Γ"
  proof
    assume "p ∈ stermsl (Γ pn)"
    hence "p ∈ sterms Γ (Γ pn)" using 'not_call p' wfg ..
  qed

```

```

    thus "p ∈ cterms Γ" ..
next
assume "∃p'∈dtermsl (Γ pn). p ∈ ctermsl p'"
then obtain p' where p'1: "p' ∈ dtermsl (Γ pn)"
      and p'2: "p ∈ ctermsl p'" ..
from p'2 and 'not_call p' have "not_call p'" ..
from p'1 obtain ps where ps1: "ps ∈ stermsl (Γ pn)"
      and ps2: "p' ∈ dtermsl ps"
  by (blast dest: dtermsl_add_stermsl_beforeD)
from ps2 have "not_call ps" ..
with ps1 have "ps ∈ cterms Γ" using wfg by auto
with 'p' ∈ dtermsl ps' and 'not_call p'' have "p' ∈ cterms Γ" using wfg by auto
hence "sterms Γ p' ⊆ cterms Γ" using wfg by auto
with 'p ∈ ctermsl p'' 'not_call p' show "p ∈ cterms Γ" using wfg ..
qed
qed

lemma ctermsE [elim]:
  assumes "wellformed Γ"
    and "p ∈ cterms Γ"
  obtains pn where "p ∈ ctermsl (Γ pn)"
    and "not_call p"
  using assms(2) unfolding cterms_def' [OF assms(1)] by auto

corollary cterms_subterms:
  assumes "wellformed Γ"
  shows "cterm Γ = {p | p pn. p ∈ subterms (Γ pn) ∧ not_call p ∧ not_choice p}"
  by (subst cterms_def' [OF assms(1)], subst ctermsl_subterms) auto

lemma subterms_in_cterms [elim]:
  assumes "wellformed Γ"
    and "p ∈ subterms (Γ pn)"
    and "not_call p"
    and "not_choice p"
  shows "p ∈ cterms Γ"
  using assms unfolding cterms_subterms [OF 'wellformed Γ'] by auto

lemma subterms_stermsl_ctermsl:
  assumes "q ∈ subterms p"
    and "r ∈ stermsl q"
  shows "r ∈ ctermsl p"
  using assms
proof (induct p)
  fix p1 p2
  assume IH1: "q ∈ subterms p1 ⇒ r ∈ stermsl q ⇒ r ∈ ctermsl p1"
    and IH2: "q ∈ subterms p2 ⇒ r ∈ stermsl q ⇒ r ∈ ctermsl p2"
    and *: "q ∈ subterms (p1 ⊕ p2)"
    and "r ∈ stermsl q"
  from * have "q ∈ {p1 ⊕ p2} ∪ subterms p1 ∪ subterms p2" by simp
  thus "r ∈ ctermsl (p1 ⊕ p2)"
proof (elim UnE)
  assume "q ∈ {p1 ⊕ p2}" with 'r ∈ stermsl q' show ?thesis
  by simp (metis stermsl_ctermsl)
next
  assume "q ∈ subterms p1" hence "r ∈ ctermsl p1" using 'r ∈ stermsl q' by (rule IH1)
  thus ?thesis by simp
next
  assume "q ∈ subterms p2" hence "r ∈ ctermsl p2" using 'r ∈ stermsl q' by (rule IH2)
  thus ?thesis by simp
qed
qed auto

lemma subterms_sterms_cterms:
  assumes wf: "wellformed Γ"

```



```

    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"
    and *: "p1 ⊕ p2 ∈ subterms (Γ pn)"
  from * have "p1 ∈ subterms (Γ pn)" by auto
  hence "sterms Γ p1 ⊆ cterms Γ" by (rule IH1)
  moreover from * have "p2 ∈ subterms (Γ pn)" by auto
  hence "sterms Γ p2 ⊆ cterms Γ" by (rule IH2)
  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 ({1}⟨fg⟩ p) = {1}"
| "labels Γ ({1}[[fa]] p) = {1}"
| "labels Γ (p1 ⊕ p2) = labels Γ p1 ∪ labels Γ p2"
| "labels Γ ({1}unicast(fip, fmsg).p ▷ q) = {1}"
| "labels Γ ({1}broadcast(fmsg). p) = {1}"
| "labels Γ ({1}groupcast(fips, fmsg). p) = {1}"
| "labels Γ ({1}send(fmsg).p) = {1}"
| "labels Γ ({1}deliver(fdata).p) = {1}"
| "labels Γ ({1}receive(fmsg).p) = {1}"
| "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"

```

```

    by (cases p) (auto simp: labels_rel.simps)
qed

lemma labels_termination:
  fixes  $\Gamma$  p
  assumes "wellformed( $\Gamma$ )"
  shows "labels_dom ( $\Gamma$ , p)"
proof -
  have labels_rel': "labels_rel = ( $\lambda$ gq gp. (gq, gp)  $\in$  {(( $\Gamma$ , q), ( $\Gamma'$ , p)).  $\Gamma = \Gamma' \wedge p \rightsquigarrow_{\Gamma} q$ })"
  by (rule ext)+ (auto simp: labels_rel.simps intro: microstep.intros elim: microstep.cases)
  from 'wellformed( $\Gamma$ )' have " $\forall x. x \in \text{Wellfounded.acc } \{(q, p). p \rightsquigarrow_{\Gamma} q\}$ "
  unfolding wellformed_def by (simp add: wf_acc_iff)
  hence "p  $\in$  Wellfounded.acc {( $q$ , p). p  $\rightsquigarrow_{\Gamma}$  q}" ..
  hence "(( $\Gamma$ , p)  $\in$  Wellfounded.acc {(( $\Gamma$ , q), ( $\Gamma'$ , p)).  $\Gamma = \Gamma' \wedge p \rightsquigarrow_{\Gamma} q$ })"
  by (rule acc_induct) (auto intro: accI)
  thus "labels_dom ( $\Gamma$ , 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  $\Gamma$  p
  assumes "wellformed  $\Gamma$ "
  shows "labels  $\Gamma$  p  $\neq$  {}"
  by (induct p rule: labels_pinduct [OF 'wellformed  $\Gamma$ ']) simp_all

lemma has_label [dest]:
  fixes  $\Gamma$  p
  assumes "wellformed  $\Gamma$ "
  shows " $\exists l. l \in \text{labels } \Gamma$  p"
  using labels_not_empty [OF assms] by auto

lemma singleton_labels [simp]:
  " $\bigwedge \Gamma l l' f p. l \in \text{labels } \Gamma \{l'\}\langle f \rangle p \quad = (l = l')$ "
  " $\bigwedge \Gamma l l' f p. l \in \text{labels } \Gamma \{l'\}\llbracket f \rrbracket p \quad = (l = l')$ "
  " $\bigwedge \Gamma l l' fip fmsg p q. l \in \text{labels } \Gamma \{l'\}\text{unicast}(fip, fmsg).p \triangleright q \quad = (l = l')$ "
  " $\bigwedge \Gamma l l' fmsg p. l \in \text{labels } \Gamma \{l'\}\text{broadcast}(fmsg). p \quad = (l = l')$ "
  " $\bigwedge \Gamma l l' fips fmsg p. l \in \text{labels } \Gamma \{l'\}\text{groupcast}(fips, fmsg). p \quad = (l = l')$ "
  " $\bigwedge \Gamma l l' fmsg p. l \in \text{labels } \Gamma \{l'\}\text{send}(fmsg).p \quad = (l = l')$ "
  " $\bigwedge \Gamma l l' fdata p. l \in \text{labels } \Gamma \{l'\}\text{deliver}(fdata).p \quad = (l = l')$ "
  " $\bigwedge \Gamma l l' fmsg p. l \in \text{labels } \Gamma \{l'\}\text{receive}(fmsg).p \quad = (l = l')$ "
  by auto

lemma in_labels_singletons [dest!]:
  " $\bigwedge \Gamma l l' f p. l \in \text{labels } \Gamma \{l'\}\langle f \rangle p \quad \implies l = l'$ "
  " $\bigwedge \Gamma l l' f p. l \in \text{labels } \Gamma \{l'\}\llbracket f \rrbracket p \quad \implies l = l'$ "
  " $\bigwedge \Gamma l l' fip fmsg p q. l \in \text{labels } \Gamma \{l'\}\text{unicast}(fip, fmsg).p \triangleright q \quad \implies l = l'$ "
  " $\bigwedge \Gamma l l' fmsg p. l \in \text{labels } \Gamma \{l'\}\text{broadcast}(fmsg). p \quad \implies l = l'$ "
  " $\bigwedge \Gamma l l' fips fmsg p. l \in \text{labels } \Gamma \{l'\}\text{groupcast}(fips, fmsg). p \quad \implies l = l'$ "
  " $\bigwedge \Gamma l l' fmsg p. l \in \text{labels } \Gamma \{l'\}\text{send}(fmsg).p \quad \implies l = l'$ "
  " $\bigwedge \Gamma l l' fdata p. l \in \text{labels } \Gamma \{l'\}\text{deliver}(fdata).p \quad \implies l = l'$ "
  " $\bigwedge \Gamma l l' fmsg p. l \in \text{labels } \Gamma \{l'\}\text{receive}(fmsg).p \quad \implies l = l'$ "
  by auto

definition
  simple_labels :: "( $s$ ,  $m$ ,  $p$ ,  $l$ ) seqp_env  $\Rightarrow$  bool"
where
  "simple_labels  $\Gamma \equiv \forall pn. \forall p \in \text{subterms } (\Gamma pn). (\exists ! l. \text{labels } \Gamma p = \{l\})"$ "

lemma simple_labelsI [intro]:

```

```

assumes "\pn p. p ∈ subterms (Γ pn) ⇒ ∃!l. labels Γ p = {l}"
shows "simple_labels Γ"
using assms unfolding simple_labels_def by auto

```

The *simple_labels* Γ property is necessary to transfer results shown over the *cterms* of a process specification Γ to the reachable actions of that process.

Consider the process $\{l_1\}\text{send}(m1) . p1 \oplus \{l_2\}\text{send}(m2) . p2$. The iteration over *cterms* Γ will cover the two transitions $(l_1, \text{send } m1, p1)$ and $(l_2, \text{send } m2, p2)$, but reachability requires the four transitions $(l_1, \text{send } m1, p1)$, $(l_1, \text{send } m2, p2)$, $(l_2, \text{send } m1, p1)$, and $(l_2, \text{send } m2, p2)$.

In a simply labelled process, the former is sufficient to show the latter, since $l_1 = l_2$.

This requirement seems really only to be restrictive for processes where a *call*(*pn*) occurs as a direct subterm of a choice operator. Consider, for instance, $\{l_1\}[\text{e}] p \oplus \text{call}(pn)$. Here l_1 must equal the label of Γpn , which can then not be distinguished from any other subterm that calls *pn* in any other process.

This limitation stems from the fact that the "call points" of a process are effectively treated as the root of the called process. This is by design; we try to treat call sites as "syntactic pastings" of process terms, giving rise, conceptually, to an infinite tree structure. But this prejudices the alternative view that process calls are used as "join points" of "process threads", in complement to the "fork points" of the $p1 \oplus p2$ operator.

```

lemma simple_labels_in_sterms:
  fixes Γ l p
  assumes "simple_labels Γ"
    and "wellformed Γ"
    and "\pn. p ∈ subterms (Γ pn)"
    and "l ∈ labels Γ p"
  shows "\p' ∈ sterms Γ p. l ∈ labels Γ p'"
using assms
proof (induct p rule: labels_pinduct [OF 'wellformed Γ'])
  fix Γ p1 p2
  assume s1: "simple_labels Γ"
    and wf: "wellformed Γ"
    and IH1: "[[ simple_labels Γ; wellformed Γ;
      ∃pn. p1 ∈ subterms (Γ pn); l ∈ labels Γ p1 ]]
      ⇒ ∀p' ∈ sterms Γ p1. l ∈ labels Γ p'"
    and IH2: "[[ simple_labels Γ; wellformed Γ;
      ∃pn. p2 ∈ subterms (Γ pn); l ∈ labels Γ p2 ]]
      ⇒ ∀p' ∈ sterms Γ p2. l ∈ labels Γ p'"
    and ein: "\pn. p1 ⊕ p2 ∈ subterms (Γ pn)"
    and l12: "l ∈ labels Γ (p1 ⊕ p2)"
  from s1 ein l12 have "labels Γ (p1 ⊕ p2) = {l}"
    unfolding simple_labels_def by (metis empty_iff insert_iff)
  with wf have "labels Γ p1 ∪ labels Γ p2 = {l}" by simp
  moreover have "labels Γ p1 ≠ {}" and "labels Γ p2 ≠ {}"
    using wf by (metis labels_not_empty)+
  ultimately have "l ∈ labels Γ p1" and "l ∈ labels Γ p2"
    by (metis Un_iff empty_iff insert_iff set_eqI)+
  moreover from ein have "\pn. p1 ∈ subterms (Γ pn)"
    and "\pn. p2 ∈ subterms (Γ pn)"
    by auto
  ultimately show "\p' ∈ sterms Γ (p1 ⊕ p2). l ∈ labels Γ p'"
    using wf IH1 [OF s1 wf] IH2 [OF s1 wf] by auto
qed auto

```

```

lemma labels_in_sterms:
  fixes Γ l p
  assumes "wellformed Γ"
    and "l ∈ labels Γ p"
  shows "\p' ∈ sterms Γ p. l ∈ labels Γ p'"
using assms
by (induct p rule: labels_pinduct [OF 'wellformed Γ']) (auto intro: Un_iff)

```

```

lemma labels_sterms_labels:
  fixes Γ p p' l
  assumes "wellformed Γ"

```

```

    and "p' ∈ sterms 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} p'))"
| "labelfrom n nn (f) p =
  (let (nn', p') = labelfrom nn (nn + 1) p in
   (nn', {n} 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 ▷ q) =
  (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 ▷ q)
   = {(lf f)unicast(fip, fmsg). (labelmap lf p) ▷ (labelmap lf q)}"
| "labelmap lf (broadcast(fmsg). p) = {lf f}broadcast(fmsg). (labelmap lf p)"
| "labelmap lf (groupcast(fipset, fmsg). p) = {lf f}groupcast(fipset, fmsg). (labelmap lf p)"
| "labelmap lf (send(fmsg). p) = {lf f}send(fmsg). (labelmap lf p)"
| "labelmap lf (deliver(fdata). p) = {lf f}deliver(fdata). (labelmap lf p)"
| "labelmap lf (receive(fmsg). p) = {lf f}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 "(l1:-:n1) ≤ (l2:-:n2) = (l1 = l2 ∧ n1 ≤ n2)"

```

definition `less_label`: "`(l1::'a label) < l2 ↔ l1 ≤ l2 ∧ ¬ (l1 ≤ l2)`"

```
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
imports AWN_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 ctermssl_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: lemmanames”), to add forward rules on assumptions (to introduce previously proved invariants; “inv add: lemmanames”), or to add elimination rules that solve any remaining subgoals (“solve: lemmanames”).

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 aadv_proc_cases [ctermssl_cases]
3. add invariant intros declare seq_invariant_ctermsI [OF aadv_wf aadv_control_within aadv_simple_labels, cterms_intro]
 seq_step_invariant_ctermsI [OF aadv_wf aadv_control_within aadv_simple_labels, cterms_intros]

lemma has_ctermssl: " $p \in \text{ctermssl } \Gamma \implies p \in \text{ctermssl } \Gamma$ " .

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 CtermsslCases = Named_Thms
  (val name = @{binding "ctermssl_cases"})
  (val description = "Destruction rules for case splitting ctermssl")
```

```

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 "cterm_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 "cterm_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 "cterm_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_cterm_sl} 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' process_vcs 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."
*)
attribute_setup cterms_seqte = {* Attrib.add_del cterms_seqte_add cterms_seqte_del *}
attribute_setup cterms_elimders = {* Attrib.add_del cterms_elimders_add cterms_elimders_del *}
attribute_setup cterms_env = {* Attrib.add_del cterms_env_add cterms_env_del *}
attribute_setup ctermsl_cases = {* Attrib.add_del ctermsl_cases_add ctermsl_cases_del *}
attribute_setup cterms_intros = {* Attrib.add_del cterms_intros_add cterms_intros_del *}
setup {* inv_cterms_setup *}

declare
  insert_iff [cterms_env]
  Un_insert_right [cterms_env]
  sup_bot_right [cterms_env]

```

```

Product_Type.prod_cases [cterms_env]
ctermssl.simps [cterms_env]

```

end

10 Configure the inv-ctermns 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' 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

```

```

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

```

```

lemmas elimders =
  elimder_guard
  elimder_assign
  elimder_ucast
  elimder_bcast
  elimder_gcast
  elimder_send
  elimder_deliver
  elimder_receive

```

```

declare
  seqpTEs [cterm_ssqte]
  elimders [cterm_elimders]

```

```
end
```

11 Lemmas for partial networks

```

theory Pnet
  imports AWN_SOS Invariants
begin

```

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

```

lemma pnet_maintains_dom:
  assumes "(s, a, s') ∈ trans (pnet np p)"
  shows "net_ips s = net_ips s'"
  using assms proof (induction p arbitrary: s a s')
    fix i R σ s a s'
    assume "(s, a, s') ∈ trans (pnet np ⟨i; R⟩)"
    hence "(s, a, s') ∈ node_sos (trans (np i))" ..
    thus "net_ips s = net_ips s'"
      by (rule node_sos.cases) simp_all
  next
    fix p1 p2 s a s'
    assume "∧ s a s'. (s, a, s') ∈ trans (pnet np p1) ⇒ net_ips s = net_ips s'"
      and "∧ s a s'. (s, a, s') ∈ trans (pnet np p2) ⇒ net_ips s = net_ips s'"
      and "(s, a, s') ∈ trans (pnet np (p1 || p2))"
    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 ∈ reachable (pnet np p) I"
  shows "net_ips s = net_tree_ips p"
  using assms proof induction
    fix s
    assume "s ∈ init (pnet np p)"
    thus "net_ips s = net_tree_ips p"
  proof (induction p arbitrary: s)
    fix i R s
    assume "s ∈ init (pnet np ⟨i; R⟩)"
    then obtain ns where "s = 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⟩) I"
  shows "∃ns R. st = NodeS ii ns R"
  using assms proof induct
  fix s
  assume "s ∈ init (pnet np ⟨ii; R_i⟩)"
  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⟩) I"
    and "∃ns R. s = NodeS ii ns R"

```

```

    and "(s, a, s') ∈ trans (pnet np ⟨ii; Ri⟩)"
    and "I a"
  thus "∃ 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') ∈ pnet_sos (trans (pnet np p1)) (trans (pnet np p2))"
  shows "∃ s' t'. st' = SubnetS s' t'"
  using assms by cases simp_all

lemma net_par_reachable_is_subnet:
  assumes "st ∈ reachable (pnet np (p1 || p2)) I"
  shows "∃ 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 ∈ reachable (pnet np (p1 || p2)) I"
  and init: "∧ s t. SubnetS s t ∈ init (pnet np (p1 || p2)) ⇒ P s t"
  and step: "∧ s t s' t' a. [
    SubnetS s t ∈ reachable (pnet np (p1 || p2)) I;
    P s t; (SubnetS s t, a, SubnetS s' t') ∈ (trans (pnet np (p1 || p2))); I a ]
    ⇒ 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 ∈ init (pnet np (p1 || p2))"
    with init show "P s t" .
  next
    fix st a s' t'
    assume "st ∈ reachable (pnet np (p1 || p2)) I"
    and tr: "(st, a, SubnetS s' t') ∈ trans (pnet np (p1 || p2))"
    and "I a"
    and IH: "∧ s t. st = SubnetS s t ⇒ P s t"
    from this(1) obtain s t where "st = SubnetS s t"
    and str: "SubnetS s t ∈ reachable (pnet np (p1 || 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') ∈ trans (pnet np (p1 || p2))" by simp
    ultimately show "P s' t'"
    using 'I a' by (rule step)
  qed

lemma subnet_reachable:
  assumes "SubnetS s1 s2 ∈ reachable (pnet np (p1 || p2)) TT"
  shows "s1 ∈ reachable (pnet np p1) TT"
  "s2 ∈ reachable (pnet np p2) TT"
  proof -
    from assms have "s1 ∈ reachable (pnet np p1) TT"
    ∧ s2 ∈ reachable (pnet np p2) TT"
  proof (induction rule: reachable_par_subnet_induct)
    fix s1 s2
    assume "SubnetS s1 s2 ∈ init (pnet np (p1 || p2))"
    thus "s1 ∈ reachable (pnet np p1) TT"
    ∧ s2 ∈ reachable (pnet np p2) TT"
    by (auto dest: reachable_init)
  next
    case (step s1 s2 s1' s2' a)
    hence "SubnetS s1 s2 ∈ reachable (pnet np (p1 || p2)) TT"
    and sr1: "s1 ∈ reachable (pnet np p1) TT"
    and sr2: "s2 ∈ reachable (pnet np p2) TT"
    and "(SubnetS s1 s2, a, SubnetS s1' s2') ∈ trans (pnet np (p1 || p2))" by auto
  qed

```

```

from this(4)
  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)
qed
thus "s1 ∈ reachable (pnet np p1) TT"
  "s2 ∈ reachable (pnet np p2) TT" by auto
qed

lemma delivered_to_node [elim]:
  assumes "s ∈ reachable (pnet np ⟨ii; Ri⟩) TT"
    and "(s, i:deliver(d), s') ∈ trans (pnet np ⟨ii; Ri⟩)"
  shows "i = ii"
proof -
  from assms(1) obtain P R where "s = NodeS ii P R"
    by (metis net_node_reachable_is_node)
  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 Ri s s'
  assume sr: "s ∈ reachable (pnet np ⟨ii; Ri⟩) TT"
    and "(s, i:deliver(d), s') ∈ trans (pnet np ⟨ii; Ri⟩)"
  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 simp (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"
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 s1" using tr 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 ∈ net_ips s2" using tr by (rule IH2)
    thus "i ∈ net_ips s" by (simp add: 's = SubnetS s1 s2')
qed
qed

lemma wf_net_tree_net_ips_disjoint [elim]:
  assumes "wf_net_tree (p1 || p2)"
    and "s1 ∈ reachable (pnet np p1) S"
    and "s2 ∈ reachable (pnet np p2) S"
  shows "net_ips s1 ∩ net_ips s2 = {}"
proof -
  from 'wf_net_tree (p1 || p2)' have "net_tree_ips p1 ∩ net_tree_ips p2 = {}" by auto
  moreover from assms(2) have "net_ips s1 = net_tree_ips p1" ..
  moreover from assms(3) have "net_ips s2 = net_tree_ips p2" ..
  ultimately show ?thesis by simp
qed

lemma init_mapstate_Some_aadv_init [elim]:
  assumes "s ∈ init (pnet np p)"
    and "netmap s i = Some v"
  shows "v ∈ init (np i)"
using assms proof (induction p arbitrary: s)
  fix ii R s
  assume "s ∈ init (pnet np ⟨ii; R⟩)"
    and "netmap s i = Some v"
  from this(1) obtain ns where "s = NodeS ii ns R"
    and "ns ∈ init (np ii)" ..
  moreover from this(1) and 'netmap s i = Some v' have "i = ii"
    by simp (metis domI domIff)
  ultimately show "v ∈ init (np i)"
    using 'netmap s i = Some v' by simp
next
  fix p1 p2 s
  assume IH1: "∧s. s ∈ init (pnet np p1) ⇒ netmap s i = Some v ⇒ v ∈ init (np i)"
    and IH2: "∧s. s ∈ init (pnet np p2) ⇒ netmap s i = Some v ⇒ v ∈ init (np i)"
    and "s ∈ init (pnet np (p1 || p2))"
    and "netmap s i = Some v"
  from this(3) obtain s1 s2 where "s = SubnetS s1 s2"
    and "s1 ∈ init (pnet np p1)"
    and "s2 ∈ init (pnet np p2)" by auto
  from this(1) and 'netmap s i = Some v'
  have "netmap s1 i = Some v ∨ netmap s2 i = Some v" by auto
  thus "v ∈ init (np i)"
proof
  assume "netmap s1 i = Some v"
  with 's1 ∈ init (pnet np p1)' show ?thesis by (rule IH1)
next
  assume "netmap s2 i = Some v"
  with 's2 ∈ init (pnet np p2)' show ?thesis by (rule IH2)
qed
qed

lemma reachable_connect_netmap [elim]:
  assumes "s ∈ reachable (pnet np n) TT"
    and "(s, connect(i, i'), s') ∈ trans (pnet np n)"
  shows "netmap s' = netmap s"
using assms proof (induction n arbitrary: s s')
  fix ii Ri s s'
  assume sr: "s ∈ reachable (pnet np ⟨ii; Ri⟩) TT"
    and "(s, connect(i, i'), s') ∈ trans (pnet np ⟨ii; Ri⟩)"
  from this(2) have tr: "(s, connect(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: " $\bigwedge s s'.$   $\llbracket s \in \text{reachable (pnet np p1)} \text{ TT};$ 
    (s, connect(i, i'), s')  $\in$  trans (pnet np p1)  $\rrbracket \implies \text{netmap } s' = \text{netmap } s$ "
    and IH2: " $\bigwedge s s'.$   $\llbracket s \in \text{reachable (pnet np p2)} \text{ TT};$ 
    (s, connect(i, i'), s')  $\in$  trans (pnet np p2)  $\rrbracket \implies \text{netmap } s' = \text{netmap } s$ "
    and sr: " $s \in \text{reachable (pnet np (p1 \parallel p2)) TT}$ "
    and tr: "(s, connect(i, i'), s')  $\in$  trans (pnet np (p1 \parallel p2))"
from tr have "(s, connect(i, i'), s')  $\in$  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')  $\in$  trans (pnet np p1)"
    and tr2: "(s2, connect(i, i'), s2')  $\in$  trans (pnet np p2)"
from this(1) and sr
    have "SubnetS s1 s2  $\in$  reachable (pnet np (p1 \parallel p2)) TT" by simp
hence sr1: "s1  $\in$  reachable (pnet np p1) TT"
    and sr2: "s2  $\in$  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  $\in$  reachable (pnet np n) TT"
    and "(s, disconnect(i, i'), s')  $\in$  trans (pnet np n)"
shows "netmap s' = netmap s"
using assms proof (induction n arbitrary: s s')
fix ii Ri s s'
assume sr: "s  $\in$  reachable (pnet np ⟨ii; Ri⟩) TT"
    and "(s, disconnect(i, i'), s')  $\in$  trans (pnet np ⟨ii; Ri⟩)"
from this(2) have tr: "(s, disconnect(i, i'), s')  $\in$  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: " $\bigwedge s s'.$   $\llbracket s \in \text{reachable (pnet np p1)} \text{ TT};$ 
    (s, disconnect(i, i'), s')  $\in$  trans (pnet np p1)  $\rrbracket \implies \text{netmap } s' = \text{netmap } s$ "
    and IH2: " $\bigwedge s s'.$   $\llbracket s \in \text{reachable (pnet np p2)} \text{ TT};$ 
    (s, disconnect(i, i'), s')  $\in$  trans (pnet np p2)  $\rrbracket \implies \text{netmap } s' = \text{netmap } s$ "
    and sr: " $s \in \text{reachable (pnet np (p1 \parallel p2)) TT}$ "
    and tr: "(s, disconnect(i, i'), s')  $\in$  trans (pnet np (p1 \parallel p2))"
from tr have "(s, disconnect(i, i'), s')  $\in$  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')  $\in$  trans (pnet np p1)"
    and tr2: "(s2, disconnect(i, i'), s2')  $\in$  trans (pnet np p2)"
from this(1) and sr
    have "SubnetS s1 s2  $\in$  reachable (pnet np (p1 \parallel p2)) TT" by simp
hence sr1: "s1  $\in$  reachable (pnet np p1) TT"
    and sr2: "s2  $\in$  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

```

```

fun net_ip_action :: "(ip  $\Rightarrow$  ('s, 'm seq_action) automaton)
   $\Rightarrow$  'm node_action  $\Rightarrow$  ip  $\Rightarrow$  net_tree  $\Rightarrow$  's net_state  $\Rightarrow$  's net_state  $\Rightarrow$  bool"

```

where

```

"net_ip_action np a i (p1 || p2) (SubnetS s1 s2) (SubnetS s1' s2') =
  ((i  $\in$  net_ips s1  $\longrightarrow$  ((s1, a, s1')  $\in$  trans (pnet np p1)
     $\wedge$  s2' = s2  $\wedge$  net_ip_action np a i p1 s1 s1'))
   $\wedge$  (i  $\in$  net_ips s2  $\longrightarrow$  ((s2, a, s2')  $\in$  trans (pnet np p2))
     $\wedge$  s1' = s1  $\wedge$  net_ip_action np a i p2 s2 s2'))"
| "net_ip_action np a i p s s' = True"

```

lemma pnet_tau_single_node [elim]:

```

assumes "wf_net_tree p"
  and "s  $\in$  reachable (pnet np p) TT"
  and "(s,  $\tau$ , s')  $\in$  trans (pnet np p)"
shows " $\exists i \in$  net_ips s. (( $\forall j. j \neq i \longrightarrow$  netmap s' j = netmap s j)
   $\wedge$  net_ip_action np  $\tau$  i p s s')"
```

using assms proof (induction p arbitrary: s s')

```

fix ii Ri s s'
assume "s  $\in$  reachable (pnet np (ii; Ri)) TT"
  and "(s,  $\tau$ , s')  $\in$  trans (pnet np (ii; Ri))"
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 " $\exists i \in$  dom (netmap s).  $\forall j. j \neq i \longrightarrow$  netmap s' j = netmap s j"
  by (simp add: net_ips_is_dom_netmap)
thus " $\exists i \in$  net_ips s. ( $\forall j. j \neq i \longrightarrow$  netmap s' j = netmap s j)
   $\wedge$  net_ip_action np  $\tau$  i ((ii; Ri) s s'"
  by (simp add: net_ips_is_dom_netmap)

```

next

```

fix p1 p2 s s'
assume IH1: " $\wedge s s'. \llbracket$  wf_net_tree p1;
  s  $\in$  reachable (pnet np p1) TT;
  (s,  $\tau$ , s')  $\in$  trans (pnet np p1)  $\rrbracket$ 
 $\implies \exists i \in$  net_ips s. ( $\forall j. j \neq i \longrightarrow$  netmap s' j = netmap s j)
   $\wedge$  net_ip_action np  $\tau$  i p1 s s'"

```

```

and IH2: " $\wedge s s'. \llbracket$  wf_net_tree p2;
  s  $\in$  reachable (pnet np p2) TT;
  (s,  $\tau$ , s')  $\in$  trans (pnet np p2)  $\rrbracket$ 
 $\implies \exists i \in$  net_ips s. ( $\forall j. j \neq i \longrightarrow$  netmap s' j = netmap s j)
   $\wedge$  net_ip_action np  $\tau$  i p2 s s'"

```

```

and sr: "s  $\in$  reachable (pnet np (p1 || p2)) TT"
and "wf_net_tree (p1 || p2)"
and tr: "(s,  $\tau$ , s')  $\in$  trans (pnet np (p1 || p2))"

```

```

from 'wf_net_tree (p1 || p2)' have "net_tree_ips p1  $\cap$  net_tree_ips p2 = {}"
  and "wf_net_tree p1"
  and "wf_net_tree p2" by auto

```

```

from tr have "(s,  $\tau$ , s')  $\in$  pnet_sos (trans (pnet np p1)) (trans (pnet np p2))" by simp
thus " $\exists i \in$  net_ips s. ( $\forall j. j \neq i \longrightarrow$  netmap s' j = netmap s j)
   $\wedge$  net_ip_action np  $\tau$  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,  $\tau$ , s1')  $\in$  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)
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 impI)
  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_simps(1) map_add_dom_app_simps(3))
qed
ultimately show ?thesis using 'i ∈ dom(netmap s1)' subs subs'
  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' j = netmap s2 j"
  and "net_ip_action np τ i p2 s2 s2'"
  by (auto simp add: net_ips_is_dom_netmap dest!: IH2)
from this(1) and 'dom(netmap s1) ∩ dom(netmap s2) = {}' have "i ∉ dom(netmap s1)"
  by auto
with subs subs' tr2 'net_ip_action np τ i p2 s2 s2'' 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 impI)
  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)' subs subs'
  by (clarsimp simp add: net_ips_is_dom_netmap)
  (metis domI dom_map_add map_add_find_right)
qed simp_all
qed

```

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'"
  (is "?P p s s'")
using assms proof (induction p arbitrary: s s')
  fix ii Ri s s'
  assume sr: "s ∈ reachable (pnet np ⟨ii; Ri⟩) TT"

```



```

    and tr: "(s, i:deliver(d), s') ∈ trans (pnet np ⟨ii; Ri⟩)"
  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; Ri⟩) 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'"
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 add: net_ips_is_dom_netmap)
moreover have "∀j. j ≠ i → (netmap s1' ++ netmap s2) j = (netmap s1 ++ netmap s2) j"
proof (intro allI impI)
  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_simps(1) map_add_dom_app_simps(3))
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 subs': "s' = SubnetS s1 s2'"
    and tr2: "(s2, i:deliver(d), 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)
moreover from sr2 tr2 have "i ∈ net_ips s2" by (rule delivered_to_net_ips)
ultimately have "i ∉ dom(netmap s1)" by (auto simp add: net_ips_is_dom_netmap)

from 'wf_net_tree p2' sr2 tr2 have *: "∀j. j ≠ i → netmap s2' j = netmap s2 j"
    and "net_ip_action np (i:deliver(d)) i p2 s2 s2'"

    by (auto dest!: IH2)
from subs subs' tr2 this(2) 'i ∉ dom(netmap s1)'
    have "net_ip_action np (i:deliver(d)) 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 impI)
    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 ∈ net_ips s2' 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 "(SubnetS s t, a, st') ∈ cnet_sos (pnet_sos (trans (pnet np p1)) (trans (pnet np p2)))"
    shows "∃s' t'. st' = SubnetS s' t'"
    using assms by cases (auto dest: partial_net_preserves_subnets)

```

```

lemma complete_net_reachable_is_subnet:

```

```

    assumes "st ∈ reachable (closed (pnet np (p1 || p2))) I"
    shows "∃s t. st = 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 "SubnetS s t ∈ reachable (closed (pnet np (p1 || p2))) I"
    and init: "∧s t. SubnetS s t ∈ init (closed (pnet np (p1 || p2))) ⇒ P s t"
    and step: "∧s t s' t' a. [
        SubnetS s t ∈ reachable (closed (pnet np (p1 || p2))) I;
        P s t; (SubnetS s t, a, SubnetS s' t') ∈ trans (closed (pnet np (p1 || p2))); I a ]
        ⇒ 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 ∈ init (closed (pnet np (p1 || p2)))"
    with init show "P s t" .

```

```

next

```

```

    fix st a s' t'
    assume "st ∈ reachable (closed (pnet np (p1 || p2))) I"
    and tr: "(st, a, SubnetS s' t') ∈ trans (closed (pnet np (p1 || p2)))"
    and "I a"
    and IH: "∧s t. st = SubnetS s t ⇒ P s t"

```

```

from this(1) obtain s t where "st = SubnetS s t"
      and "SubnetS s t ∈ reachable (closed (pnet np (p1 || p2))) I"
  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') ∈ trans (closed (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 ∈ reachable (closed (pnet np n)) TT"
  shows "s ∈ reachable (pnet np n) TT"
  using assms proof (induction rule: reachable.induct)
  fix s s' a
  assume sr: "s ∈ reachable (pnet np n) TT"
    and "(s, a, s') ∈ trans (closed (pnet np n))"
  from this(2) have "(s, a, s') ∈ cnet_sos (trans (pnet np n))" by simp
  thus "s' ∈ reachable (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 ∈ reachable (closed (pnet np ⟨ii; Ri⟩)) TT"
  obtains ξ p q R where "st = 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 ∈ reachable (closed (pnet np (p1 || p2))) TT"
  obtains s t where "st = 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') ∈ trans (closed (pnet np n))"
  shows "∃ a'. (s, a', s') ∈ trans (pnet np n)"
  using assms by (auto elim!: cnet_sos.cases)

lemma reachable_not_in_net_tree_ips [elim]:
  assumes "s ∈ reachable (closed (pnet np n)) TT"
    and "i ∉ net_tree_ips n"
  shows "netmap s i = None"
  using assms proof induction
  fix s
  assume "s ∈ init (closed (pnet np n))"
    and "i ∉ net_tree_ips n"
  thus "netmap s i = None"
  proof (induction n arbitrary: s)
  fix ii R s
  assume "s ∈ init (closed (pnet np ⟨ii; R⟩))"
    and "i ∉ net_tree_ips ⟨ii; R⟩"
  from this(2) have "i ≠ ii" by simp
  moreover from 's ∈ init (closed (pnet np ⟨ii; R⟩))' obtain p where "s = NodeS ii p R"
    by simp (metis pnet.simps(1) pnet_node_init')
  ultimately show "netmap s i = None" by simp
  next
  fix p1 p2 s
  assume IH1: "∧s. s ∈ init (closed (pnet np p1)) ⇒ i ∉ net_tree_ips p1
    ⇒ netmap s i = None"
    and IH2: "∧s. s ∈ init (closed (pnet np p2)) ⇒ i ∉ net_tree_ips p2
    ⇒ netmap s i = None"
    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 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_aadv_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 this(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 ∈ net_tree_ips p1' and 'net_tree_ips p1 = net_ips s1'
  have "i ∈ net_ips s1" by simp
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 (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 *DAWN_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 σ mapping ip addresses to states.

13.1 Open structural operational semantics for sequential process expressions

inductive_set

oseqp_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: " $\sigma' i = \sigma i \Rightarrow$
 $((\sigma, \{l\}broadcast(s_{msg}).p), \quad broadcast (s_{msg} (\sigma i)), \quad (\sigma', p)) \in oseqp_sos$ "

Γi "
ogroupcastT: " $\sigma' i = \sigma i \Rightarrow$
 $((\sigma, \{l\}groupcast(s_{ips}, s_{msg}).p), \quad groupcast (s_{ips} (\sigma i)) (s_{msg} (\sigma i)), (\sigma', p)) \in oseqp_sos$ "

Γi "
ounicastT: " $\sigma' i = \sigma i \Rightarrow$
 $((\sigma, \{l\}unicast(s_{ip}, s_{msg}).p \triangleright q), \quad unicast (s_{ip} (\sigma i)) (s_{msg} (\sigma i)), \quad (\sigma', p)) \in oseqp_sos$ "

Γi "
onotunicastT: " $\sigma' i = \sigma i \Rightarrow$
 $((\sigma, \{l\}unicast(s_{ip}, s_{msg}).p \triangleright q), \quad \neg unicast (s_{ip} (\sigma i)), \quad (\sigma', q)) \in oseqp_sos$ "

Γi "
osendT: " $\sigma' i = \sigma i \Rightarrow$
 $((\sigma, \{l\}send(s_{msg}).p), \quad send (s_{msg} (\sigma i)), \quad (\sigma', p)) \in oseqp_sos$ "

Γi "
odeliverT: " $\sigma' i = \sigma i \Rightarrow$
 $((\sigma, \{l\}deliver(s_{data}).p), \quad deliver (s_{data} (\sigma i)), \quad (\sigma', p)) \in oseqp_sos$ "

Γi "
oreceiveT: " $\sigma' i = u_{msg} msg (\sigma i) \Rightarrow$
 $((\sigma, \{l\}receive(u_{msg}).p), \quad receive msg, \quad (\sigma', p)) \in oseqp_sos$ "

Γi "
oassignT: " $\sigma' i = u (\sigma i) \Rightarrow$
 $((\sigma, \{l\}[\![u]\!] p), \quad \tau, \quad (\sigma', p)) \in oseqp_sos$ "

Γi "
ocallT: " $((\sigma, \Gamma pn), a, (\sigma', p')) \in oseqp_sos \Gamma i \Rightarrow$
 $((\sigma, call(pn)), a, (\sigma', p')) \in oseqp_sos \Gamma i$ "

ochoiceT1: " $((\sigma, p), a, (\sigma', p')) \in oseqp_sos \Gamma i \Rightarrow$
 $((\sigma, p \oplus q), a, (\sigma', p')) \in oseqp_sos \Gamma i$ "

ochoiceT2: " $((\sigma, q), a, (\sigma', q')) \in oseqp_sos \Gamma i \Rightarrow$
 $((\sigma, p \oplus q), a, (\sigma', q')) \in oseqp_sos \Gamma i$ "

| *oguardT*: $"\sigma' i \in g(\sigma i) \implies ((\sigma, \{1\}\langle g \rangle p), \tau, (\sigma', p)) \in \text{oseqp_sos } \Gamma i"$

inductive_cases

oseq_callTE [elim]: $"((\sigma, \text{call}(pn)), a, (\sigma', q)) \in \text{oseqp_sos } \Gamma i"$
and *oseq_choiceTE* [elim]: $"((\sigma, p_1 \oplus p_2), a, (\sigma', q)) \in \text{oseqp_sos } \Gamma i"$

lemma *oseq_broadcastTE* [elim]:

$"[(\sigma, \{1\}\text{broadcast}(s_{msg}). p), a, (\sigma', q)] \in \text{oseqp_sos } \Gamma i;$
 $[[a = \text{broadcast}(s_{msg}(\sigma i)); \sigma' i = \sigma i; q = p] \implies P] \implies P"$
by (*ind_cases* $"((\sigma, \{1\}\text{broadcast}(s_{msg}). p), a, (\sigma', q)) \in \text{oseqp_sos } \Gamma i"$) *simp*

lemma *oseq_groupcastTE* [elim]:

$"[(\sigma, \{1\}\text{groupcast}(s_{ips}, s_{msg}). p), a, (\sigma', q)] \in \text{oseqp_sos } \Gamma i;$
 $[[a = \text{groupcast}(s_{ips}(\sigma i))(s_{msg}(\sigma i)); \sigma' i = \sigma i; q = p] \implies P] \implies P"$
by (*ind_cases* $"((\sigma, \{1\}\text{groupcast}(s_{ips}, s_{msg}). p), a, (\sigma', q)) \in \text{oseqp_sos } \Gamma i"$) *simp*

lemma *oseq_unicastTE* [elim]:

$"[(\sigma, \{1\}\text{unicast}(s_{ip}, s_{msg}). p \triangleright q), a, (\sigma', r)] \in \text{oseqp_sos } \Gamma i;$
 $[[a = \text{unicast}(s_{ip}(\sigma i))(s_{msg}(\sigma i)); \sigma' i = \sigma i; r = p] \implies P;$
 $[[a = \neg\text{unicast}(s_{ip}(\sigma i)); \sigma' i = \sigma i; r = q] \implies P] \implies P"$
by (*ind_cases* $"((\sigma, \{1\}\text{unicast}(s_{ip}, s_{msg}). p \triangleright q), a, (\sigma', r)) \in \text{oseqp_sos } \Gamma i"$) *simp_all*

lemma *oseq_sendTE* [elim]:

$"[(\sigma, \{1\}\text{send}(s_{msg}). p), a, (\sigma', q)] \in \text{oseqp_sos } \Gamma i;$
 $[[a = \text{send}(s_{msg}(\sigma i)); \sigma' i = \sigma i; q = p] \implies P] \implies P"$
by (*ind_cases* $"((\sigma, \{1\}\text{send}(s_{msg}). p), a, (\sigma', q)) \in \text{oseqp_sos } \Gamma i"$) *simp*

lemma *oseq_deliverTE* [elim]:

$"[(\sigma, \{1\}\text{deliver}(s_{data}). p), a, (\sigma', q)] \in \text{oseqp_sos } \Gamma i;$
 $[[a = \text{deliver}(s_{data}(\sigma i)); \sigma' i = \sigma i; q = p] \implies P] \implies P"$
by (*ind_cases* $"((\sigma, \{1\}\text{deliver}(s_{data}). p), a, (\sigma', q)) \in \text{oseqp_sos } \Gamma i"$) *simp*

lemma *oseq_receiveTE* [elim]:

$"[(\sigma, \{1\}\text{receive}(u_{msg}). p), a, (\sigma', q)] \in \text{oseqp_sos } \Gamma i;$
 $\bigwedge \text{msg}. [[a = \text{receive msg}; \sigma' i = u_{msg} \text{ msg }(\sigma i); q = p] \implies P] \implies P"$
by (*ind_cases* $"((\sigma, \{1\}\text{receive}(u_{msg}). p), a, (\sigma', q)) \in \text{oseqp_sos } \Gamma i"$) *simp*

lemma *oseq_assignTE* [elim]:

$"[(\sigma, \{1\}\llbracket u \rrbracket p), a, (\sigma', q)] \in \text{oseqp_sos } \Gamma i; [[a = \tau; \sigma' i = u(\sigma i); q = p] \implies P] \implies P"$
by (*ind_cases* $"((\sigma, \{1\}\llbracket u \rrbracket p), a, (\sigma', q)) \in \text{oseqp_sos } \Gamma i"$) *simp*

lemma *oseq_guardTE* [elim]:

$"[(\sigma, \{1\}\langle g \rangle p), a, (\sigma', q)] \in \text{oseqp_sos } \Gamma i; [[a = \tau; \sigma' i \in g(\sigma i); q = p] \implies P] \implies P"$
by (*ind_cases* $"((\sigma, \{1\}\langle g \rangle p), a, (\sigma', q)) \in \text{oseqp_sos } \Gamma i"$) *simp*

lemmas *oseqpTEs* =

oseq_broadcastTE
oseq_groupcastTE
oseq_unicastTE
oseq_sendTE
oseq_deliverTE
oseq_receiveTE
oseq_assignTE
oseq_callTE
oseq_choiceTE
oseq_guardTE

declare *oseqp_sos.intros* [*intro*]

13.2 Open structural operational semantics for parallel process expressions

inductive_set

oparp_sos :: "ip
 $\implies ((ip \implies 's) \times 's1, 'm \text{ seq_action}) \text{ transition set}$

```

      ⇒ ('s2, 'm seq_action) transition set
      ⇒ ((ip ⇒ 's) × ('s1 × 's2), 'm seq_action) transition set"
for i :: ip
and S :: "((ip ⇒ 's) × 's1, 'm seq_action) transition set"
and T :: "('s2, 'm seq_action) transition set"
where
  oparleft: "[( $((\sigma, s), a, (\sigma', s')) \in S; \bigwedge m. a \neq \text{receive } m$ )] ⇒
    ( $(\sigma, (s, t)), a, (\sigma', (s', t')) \in \text{oparp\_sos } i \ S \ T$ )"
  / oparright: "[( $(t, a, t') \in T; \bigwedge m. a \neq \text{send } m; \sigma' \ i = \sigma \ i$ )] ⇒
    ( $(\sigma, (s, t)), a, (\sigma', (s, t')) \in \text{oparp\_sos } i \ S \ T$ )"
  / oparboth: "[( $((\sigma, s), \text{receive } m, (\sigma', s')) \in S; (t, \text{send } m, t') \in T$ )] ⇒
    ( $(\sigma, (s, t)), \tau, (\sigma', (s', t')) \in \text{oparp\_sos } i \ S \ T$ )"

lemma opar_broadcastTE [elim]:
  "[( $(\sigma, (s, t)), \text{broadcast } m, (\sigma', (s', t')) \in \text{oparp\_sos } i \ S \ T$ ;
    [ $(\sigma, s), \text{broadcast } m, (\sigma', s') \in S; t' = t$ ]] ⇒ P;
    [ $(t, \text{broadcast } m, t') \in T; s' = s; \sigma' \ i = \sigma \ i$ ]] ⇒ P] ⇒ P"
  by (ind_cases " $((\sigma, (s, t)), \text{broadcast } m, (\sigma', (s', t')) \in \text{oparp\_sos } i \ S \ T$ ") simp_all

lemma opar_groupcastTE [elim]:
  "[( $(\sigma, (s, t)), \text{groupcast ips } m, (\sigma', (s', t')) \in \text{oparp\_sos } i \ S \ T$ ;
    [ $(\sigma, s), \text{groupcast ips } m, (\sigma', s') \in S; t' = t$ ]] ⇒ P;
    [ $(t, \text{groupcast ips } m, t') \in T; s' = s; \sigma' \ i = \sigma \ i$ ]] ⇒ P] ⇒ P"
  by (ind_cases " $((\sigma, (s, t)), \text{groupcast ips } m, (\sigma', (s', t')) \in \text{oparp\_sos } i \ S \ T$ ") simp_all

lemma opar_unicastTE [elim]:
  "[( $(\sigma, (s, t)), \text{unicast } i \ m, (\sigma', (s', t')) \in \text{oparp\_sos } i \ S \ T$ ;
    [ $(\sigma, s), \text{unicast } i \ m, (\sigma', s') \in S; t' = t$ ]] ⇒ P;
    [ $(t, \text{unicast } i \ m, t') \in T; s' = s; \sigma' \ i = \sigma \ i$ ]] ⇒ P] ⇒ P"
  by (ind_cases " $((\sigma, (s, t)), \text{unicast } i \ m, (\sigma', (s', t')) \in \text{oparp\_sos } i \ S \ T$ ") simp_all

lemma opar_notunicastTE [elim]:
  "[( $(\sigma, (s, t)), \text{notunicast } i, (\sigma', (s', t')) \in \text{oparp\_sos } i \ S \ T$ ;
    [ $(\sigma, s), \text{notunicast } i, (\sigma', s') \in S; t' = t$ ]] ⇒ P;
    [ $(t, \text{notunicast } i, t') \in T; s' = s; \sigma' \ i = \sigma \ i$ ]] ⇒ P] ⇒ P"
  by (ind_cases " $((\sigma, (s, t)), \text{notunicast } i, (\sigma', (s', t')) \in \text{oparp\_sos } i \ S \ T$ ") simp_all

lemma opar_sendTE [elim]:
  "[( $(\sigma, (s, t)), \text{send } m, (\sigma', (s', t')) \in \text{oparp\_sos } i \ S \ T$ ;
    [ $(\sigma, s), \text{send } m, (\sigma', s') \in S; t' = t$ ]] ⇒ P] ⇒ P"
  by (ind_cases " $((\sigma, (s, t)), \text{send } m, (\sigma', (s', t')) \in \text{oparp\_sos } i \ S \ T$ ") auto

lemma opar_deliverTE [elim]:
  "[( $(\sigma, (s, t)), \text{deliver } d, (\sigma', (s', t')) \in \text{oparp\_sos } i \ S \ T$ ;
    [ $(\sigma, s), \text{deliver } d, (\sigma', s') \in S; t' = t$ ]] ⇒ P;
    [ $(t, \text{deliver } d, t') \in T; s' = s; \sigma' \ i = \sigma \ i$ ]] ⇒ P] ⇒ P"
  by (ind_cases " $((\sigma, (s, t)), \text{deliver } d, (\sigma', (s', t')) \in \text{oparp\_sos } i \ S \ T$ ") simp_all

lemma opar_receiveTE [elim]:
  "[( $(\sigma, (s, t)), \text{receive } m, (\sigma', (s', t')) \in \text{oparp\_sos } i \ S \ T$ ;
    [ $(t, \text{receive } m, t') \in T; s' = s; \sigma' \ i = \sigma \ i$ ]] ⇒ P] ⇒ P"
  by (ind_cases " $((\sigma, (s, t)), \text{receive } m, (\sigma', (s', t')) \in \text{oparp\_sos } i \ S \ T$ ") auto

inductive_cases opar_tauTE: " $((\sigma, (s, t)), \tau, (\sigma', (s', t')) \in \text{oparp\_sos } i \ S \ T$ "

lemmas oparpTEs =
  opar_broadcastTE
  opar_groupcastTE
  opar_unicastTE
  opar_notunicastTE
  opar_sendTE
  opar_deliverTE
  opar_receiveTE

```

```

lemma oparp_sos_cases [elim]:
  assumes "(( $\sigma$ , (s, t)), a, ( $\sigma'$ , (s', t'))))  $\in$  oparp_sos i S T"
  and "[[ (( $\sigma$ , s), a, ( $\sigma'$ , s'))  $\in$  S;  $\bigwedge m$ . a  $\neq$  receive m; t' = t ]  $\implies$  P"
  and "[[ (t, a, t')  $\in$  T;  $\bigwedge m$ . a  $\neq$  send m; s' = s;  $\sigma'$  i =  $\sigma$  i ]  $\implies$  P"
  and " $\bigwedge m$ . [ a =  $\tau$ ; (( $\sigma$ , s), receive m, ( $\sigma'$ , s'))  $\in$  S; (t, send m, t')  $\in$  T ]  $\implies$  P"
  shows "P"
using assms by cases auto

```

```

definition extg :: "('a  $\times$  'b)  $\times$  'c  $\Rightarrow$  'a  $\times$  'b  $\times$  'c"
where "extg  $\equiv$   $\lambda$ (( $\sigma$ , l1), l2). ( $\sigma$ , (l1, l2))"

```

```

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

```

```

lemma extg_range_prod: "extg ' (i1  $\times$  i2) = {( $\sigma$ , (s1, s2)) |  $\sigma$  s1 s2. ( $\sigma$ , s1)  $\in$  i1  $\wedge$  s2  $\in$  i2}"
  unfolding image_def extg_def
  by (rule Collect_cong) (auto split: split_split)

```

```

definition
  opar_comp :: "((ip  $\Rightarrow$  's)  $\times$  's1, 'm seq_action) automaton
   $\Rightarrow$  ip
   $\Rightarrow$  ('s2, 'm seq_action) automaton
   $\Rightarrow$  ((ip  $\Rightarrow$  's)  $\times$  's1  $\times$  's2, 'm seq_action) automaton"
  ("(_ <<_ _)" [102, 0, 103] 102)

```

```

where
  "s <<_i t  $\equiv$  (| init = extg ' (init s  $\times$  init t), trans = oparp_sos i (trans s) (trans t) |)"

```

```

lemma opar_comp_def':
  "s <<_i t = (| init = {( $\sigma$ , (s_l, t_l)) |  $\sigma$  s_l t_l. ( $\sigma$ , s_l)  $\in$  init s  $\wedge$  t_l  $\in$  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  $\times$  init t)"
  unfolding opar_comp_def by simp

```

13.3 Open structural operational semantics for node expressions

inductive_set

```

onode_sos :: "((ip  $\Rightarrow$  's)  $\times$  'l, 'm seq_action) transition set
   $\Rightarrow$  ((ip  $\Rightarrow$  's)  $\times$  'l net_state, 'm node_action) transition set"
for S :: "((ip  $\Rightarrow$  's)  $\times$  'l, 'm seq_action) transition set"

```

where

onode_bcast:

```

"(( $\sigma$ , s), broadcast m, ( $\sigma'$ , s'))  $\in$  S  $\implies$  (( $\sigma$ , NodeS i s R), R:*cast(m), ( $\sigma'$ , NodeS i s' R))  $\in$  onode_sos S"

```

/ onode_gcast:

```

"(( $\sigma$ , s), groupcast D m, ( $\sigma'$ , s'))  $\in$  S  $\implies$  (( $\sigma$ , NodeS i s R), (R  $\cap$  D):*cast(m), ( $\sigma'$ , NodeS i s' R))  $\in$  onode_sos S"

```

/ onode_ucast:

```

"[[ (( $\sigma$ , s), unicast d m, ( $\sigma'$ , s'))  $\in$  S; d  $\in$  R ]  $\implies$  (( $\sigma$ , NodeS i s R), {d}:*cast(m), ( $\sigma'$ , NodeS i s' R))  $\in$  onode_sos S"

```

```

/ onode_notucast: "[[ (( $\sigma$ , s),  $\neg$ unicast d, ( $\sigma'$ , s'))  $\in$  S; d  $\notin$  R;  $\forall j$ . j  $\neq$  i  $\longrightarrow$   $\sigma'$  j =  $\sigma$  j ]  $\implies$  (( $\sigma$ , NodeS i s R),  $\tau$ , ( $\sigma'$ , NodeS i s' R))  $\in$  onode_sos S"

```



```

| onode_deliver: "[ ((σ, s), deliver d, (σ', s')) ∈ S; ∀j. j≠i → σ' j = σ j ]
  ⇒ ((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 ]
  ⇒ ((σ, NodeS i s R), τ, (σ', NodeS i s' R)) ∈ onode_sos S"

| onode_receive:
  "((σ, s), receive m, (σ', s')) ∈ S ⇒ ((σ, NodeS i s R), {i}¬{j}:arrive(m), (σ', NodeS i s' R)) ∈
onode_sos S"

| onode_arrive:
  "σ' i = σ i ⇒ ((σ, NodeS i s R), {j}¬{i}:arrive(m), (σ', NodeS i s R)) ∈ onode_sos S"

| onode_connect1:
  "σ' i = σ i ⇒ ((σ, NodeS i s R), connect(i, i'), (σ', NodeS i s (R ∪ {i'}))) ∈ onode_sos S"

| onode_connect2:
  "σ' i = σ i ⇒ ((σ, NodeS i s R), connect(i', i), (σ', NodeS i s (R ∪ {i'}))) ∈ onode_sos S"

| onode_disconnect1:
  "σ' i = σ i ⇒ ((σ, NodeS i s R), disconnect(i, i'), (σ', NodeS i s (R - {i'}))) ∈ onode_sos S"

| onode_disconnect2:
  "σ' i = σ i ⇒ ((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 ] ⇒ ((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 ] ⇒ ((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¬ni: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 oarrives_or_not:
  assumes "((σ, NodeS i s R), ii¬ni: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
  ⇒ ((ip ⇒ 's) × 'l, 'm seq_action) automaton
  ⇒ ip set
  ⇒ ((ip ⇒ 's) × 'l net_state, 'm node_action) automaton"
  ("⟨_ : _⟩_o" [0, 0, 0] 104)

where
  "⟨i : onp : R_i⟩_o ≡ (| init = {(σ, NodeS i s R_i) | σ s. (σ, s) ∈ init onp},
  trans = onode_sos (trans onp) |)"

```

```

lemma trans_onode_comp:
  "trans ((i : S : R)_o) = onode_sos (trans S)"
  unfolding onode_comp_def by simp

lemma init_onode_comp:
  "init ((i : S : R)_o) = {(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⟩_o) = 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⟩_o) = {(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 (cases s') (auto elim!: onode_sos.cases)
    thus ?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' = (σ', NodeS i ζ' R)"
  using assms proof -
    assume "((σ, NodeS i p R), a, s') ∈ onode_sos S"
    then obtain σ' ζ' R' where "s' = (σ', NodeS i ζ' R)"
      by (cases s') (auto elim!: onode_sos.cases)
    thus ?thesis by simp
  qed

lemma onode_sos_dest_is_net_state'':
  assumes "((σ, NodeS i p R), a, (σ', 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 "s' = 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"
  using assms 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 ζ R where "s = NodeS i ζ R"
  by (metis onode_sos_src_is_net_state)
moreover with assms obtain ζ' R' where "s' = NodeS i ζ' 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 R), a, (σ', NodeS i p' R')) ∈ onode_sos S ⇒
(∧m .    [ a = R:*cast(m);      R' = R; ((σ, p), broadcast m, (σ', p')) ∈ S ] ⇒ P) ⇒
(∧m D.   [ a = (R ∩ D):*cast(m);  R' = R; ((σ, p), groupcast D m, (σ', p')) ∈ S ] ⇒ P) ⇒
(∧d m.   [ a = {d}:*cast(m);     R' = R; ((σ, p), unicast d m, (σ', p')) ∈ S; d ∈ R ] ⇒ P)
⇒
(∧d.     [ a = τ;                R' = R; ((σ, p), ¬unicast d, (σ', p')) ∈ S; d ∉ R ] ⇒ P)
⇒
(∧d.     [ a = i:deliver(d);      R' = R; ((σ, p), deliver d, (σ', p')) ∈ S ] ⇒ P) ⇒
(∧m.     [ a = {i}¬{j}:arrive(m); R' = R; ((σ, p), receive m, (σ', p')) ∈ S ] ⇒ P) ⇒
(
  [ a = τ;                R' = R; ((σ, p), τ, (σ', p')) ∈ S ] ⇒ P) ⇒
(∧m.     [ a = {j}¬{i}:arrive(m); R' = R; p = p'; σ' i = σ i ] ⇒ P) ⇒
(∧i i'.  [ a = connect(i, i');    R' = R ∪ {i'}; p = p'; σ' i = σ i ] ⇒ P) ⇒
(∧i i'.  [ a = connect(i', i);    R' = R ∪ {i'}; p = p'; σ' i = σ i ] ⇒ P) ⇒
(∧i i'.  [ a = disconnect(i, i'); R' = R - {i'}; p = p'; σ' i = σ i ] ⇒ P) ⇒
(∧i i'.  [ a = disconnect(i', i); R' = R - {i'}; p = p'; σ' i = σ i ] ⇒ P) ⇒
(∧i i' i''. [ a = connect(i', i''); R' = R; p = p'; i ≠ i'; i ≠ i''; σ' i = σ i ] ⇒ P) ⇒
(∧i i' i''. [ a = disconnect(i', i''); R' = R; p = p'; i ≠ i'; i ≠ i''; σ' i = σ i ] ⇒ P) ⇒
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

```

opnet_cast1:
" [ ((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 ∪ 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"

```

```

/ opnet_connect:
"[[ ((σ, s), connect(i, i'), (σ', s')) ∈ S; ((σ, t), connect(i, i'), (σ', t')) ∈ T ]
  ⇒ ((σ, SubnetS s t), connect(i, i'), (σ', SubnetS s' t')) ∈ opnet_sos S T"

/ opnet_disconnect:
"[[ ((σ, s), disconnect(i, i'), (σ', s')) ∈ S; ((σ, t), disconnect(i, i'), (σ', t')) ∈ T ]
  ⇒ ((σ, SubnetS s t), disconnect(i, i'), (σ', SubnetS s' t')) ∈ opnet_sos S T"

inductive_cases opartial_castTE [elim]:      "((σ, s), R:*cast(m), (σ', s')) ∈ opnet_sos S T"
and opartial_arriveTE [elim]:              "((σ, s), H¬K:arrive(m), (σ', s')) ∈ opnet_sos S T"
and opartial_deliverTE [elim]:            "((σ, s), i:deliver(d), (σ', s')) ∈ opnet_sos S T"
and opartial_tauTE [elim]:                "((σ, s), τ, (σ', s')) ∈ opnet_sos S T"
and opartial_connectTE [elim]:            "((σ, s), connect(i, i'), (σ', s')) ∈ opnet_sos S T"
and opartial_disconnectTE [elim]:         "((σ, s), disconnect(i, i'), (σ', s')) ∈ opnet_sos S T"
and opartial_newpktTE [elim]:             "((σ, s), i:newpkt(d, di), (σ', s')) ∈ opnet_sos S T"

fun opnet :: "(ip ⇒ (σ ⇒ 's) × 'l, 'm seq_action) automaton)
  ⇒ net_tree ⇒ ((ip ⇒ 's) × 'l net_state, 'm node_action) automaton"

where
  "opnet onp ⟨i; R_i⟩ = ⟨i : onp i : R_i⟩_o"
/ "opnet onp (p_1 || p_2) = (| init = {(σ, SubnetS s_1 s_2) | σ s_1 s_2.
  (σ, s_1) ∈ init (opnet onp p_1)
  ∧ (σ, s_2) ∈ init (opnet onp p_2)
  ∧ net_ips s_1 ∩ net_ips s_2 = {}},
  trans = opnet_sos (trans (opnet onp p_1)) (trans (opnet onp p_2)) |)"

lemma opnet_node_init [elim, simp]:
  assumes "(σ, s) ∈ init (opnet onp ⟨i; R⟩)"
  shows "(σ, s) ∈ {(σ, NodeS i ns R) | σ ns. (σ, ns) ∈ init (onp i)}"
  using assms by (simp add: onode_comp_def)

lemma opnet_node_init' [elim]:
  assumes "(σ, s) ∈ init (opnet onp ⟨i; R⟩)"
  obtains ns where "s = NodeS i ns R"
  and "(σ, ns) ∈ init (onp i)"
  using assms by (auto simp add: onode_comp_def)

lemma opnet_node_trans [elim, simp]:
  assumes "(s, a, s') ∈ trans (opnet onp ⟨i; R⟩)"
  shows "(s, a, s') ∈ onode_sos (trans (onp i))"
  using assms by (simp add: trans_onode_comp)

```

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

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

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

/ ocnet_tau:
"[[ ((σ, s), τ, (σ', s')) ∈ S; ∀j. j ∉ net_ips s → (σ' j = σ j) ]

```

```

    ⇒ ((σ, s), τ, (σ', s')) ∈ ocnet_sos S"

/ ocnet_deliver:
"[[ ((σ, s), i:deliver(d), (σ', s')) ∈ S; ∀j. j ∉ net_ips s → (σ' j = σ j) ]]
⇒ ((σ, s), i:deliver(d), (σ', s')) ∈ ocnet_sos S"

/ ocnet_newpkt:
"[[ ((σ, s), {i}¬K:arrive(newpkt(d, di)), (σ', s')) ∈ S; ∀j. j ∉ net_ips s → (σ' j = σ j) ]]
⇒ ((σ, s), i:newpkt(d, di), (σ', s')) ∈ ocnet_sos S"

inductive_cases oconnect_completeTE: "((σ, s), connect(i, i'), (σ', s')) ∈ ocnet_sos S"
and odisconnect_completeTE: "((σ, s), disconnect(i, i'), (σ', s')) ∈ ocnet_sos S"
and otau_completeTE: "((σ, s), τ, (σ', s')) ∈ ocnet_sos S"
and odeliver_completeTE: "((σ, s), i:deliver(d), (σ', s')) ∈ ocnet_sos S"
and onewpkt_completeTE: "((σ, s), i:newpkt(d, di), (σ', s')) ∈ ocnet_sos S"

lemmas ocompleteTEs = oconnect_completeTE
                    odisconnect_completeTE
                    otau_completeTE
                    odeliver_completeTE
                    onewpkt_completeTE

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

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

lemma ocomplete_no_change [elim]:
assumes "((σ, s), a, (σ', s')) ∈ ocnet_sos T"
and "j ∉ net_ips s"
shows "σ' j = σ j"
using assms by cases simp_all

lemma ocomplete_transE [elim]:
assumes "((σ, ζ), a, (σ', ζ')) ∈ ocnet_sos (trans (opnet onp n))"
obtains a' where "((σ, ζ), a', (σ', ζ')) ∈ trans (opnet onp n)"
using assms by (cases a) (auto elim!: ocompleteTEs [simplified])

abbreviation
  oclosed :: "((ip ⇒ 's) × 'l net_state, ('m::msg) node_action) automaton
            ⇒ ((ip ⇒ 's) × 'l net_state, 'm node_action) automaton"

where
  "oclosed ≡ (λA. A (| trans := ocnet_sos (trans A) |))"

end

```

14 Configure the inv-terms tactic for open sequential processes

```

theory OAWN_SOS_Labels
imports OAWN_SOS Inv_Cterms
begin

```

```

lemma oelimder_guard:
  assumes "p = {l}<fg> qq"
    and "l' ∈ labels Γ q"
    and "((σ, p), a, (σ', q)) ∈ oseqp_sos Γ i"
  obtains p' where "p = {l}<fg> p'"
    and "l' ∈ labels Γ qq"
  using assms by auto

lemma oelimder_assign:
  assumes "p = {l}[[fa]] qq"
    and "l' ∈ labels Γ q"
    and "((σ, p), a, (σ', q)) ∈ oseqp_sos Γ i"
  obtains p' where "p = {l}[[fa]] p'"
    and "l' ∈ labels Γ qq"
  using assms by auto

lemma oelimder_ucast:
  assumes "p = {l}unicast(fip, fmsg).q1 ▷ q2"
    and "l' ∈ labels Γ q"
    and "((σ, p), a, (σ', q)) ∈ oseqp_sos Γ i"
  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 oseqpTEs, auto)

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

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

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

lemma oelimder_deliver:
  assumes "p = {l}deliver(fdata).qq"
    and "l' ∈ labels Γ q"
    and "((σ, p), a, (σ', q)) ∈ oseqp_sos Γ i"
  obtains p' where "p = {l}deliver(fdata).p'"
    and "l' ∈ labels Γ qq"
  using assms by auto

lemma oelimder_receive:
  assumes "p = {l}receive(fmsg).qq"
    and "l' ∈ labels Γ q"
    and "((σ, p), a, (σ', q)) ∈ oseqp_sos Γ i"
  obtains p' where "p = {l}receive(fmsg).p'"
    and "l' ∈ labels Γ qq"

```

```
using assms by auto
```

```
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 `opnet_sos` transitions.

```
lemma opnet_maintains_dom:  
  assumes "(( $\sigma$ , ns), a, ( $\sigma'$ , ns'))  $\in$  trans (opnet np p)"  
  shows "net_ips ns = net_ips ns'"  
  using assms proof (induction p arbitrary:  $\sigma$  ns a  $\sigma'$  ns')  
  fix i R  $\sigma$  ns a  $\sigma'$  ns'  
  assume "(( $\sigma$ , ns), a, ( $\sigma'$ , ns'))  $\in$  trans (opnet np  $\langle$ i; R $\rangle$ )"  
  hence "(( $\sigma$ , ns), a, ( $\sigma'$ , ns'))  $\in$  onode_sos (trans (np i))" ..  
  thus "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 " $\bigwedge$  $\sigma$  ns a  $\sigma'$  ns'. (( $\sigma$ , ns), a, ( $\sigma'$ , ns'))  $\in$  trans (opnet np p1)  $\implies$  net_ips ns = net_ips ns'"  
  and " $\bigwedge$  $\sigma$  ns a  $\sigma'$  ns'. (( $\sigma$ , ns), a, ( $\sigma'$ , ns'))  $\in$  trans (opnet np p2)  $\implies$  net_ips ns = net_ips ns'"  
  and "(( $\sigma$ , ns), a, ( $\sigma'$ , ns'))  $\in$  trans (opnet np (p1  $\parallel$  p2))"  
  thus "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)  $\in$  oreachable (opnet np p) S U)"  
  shows "net_ips ns = net_tree_ips p"  
  using assms proof (induction rule: oreachable_pair_induct)  
  fix  $\sigma$  s  
  assume "(( $\sigma$ , s)  $\in$  init (opnet np p))"  
  thus "net_ips s = net_tree_ips p"  
  proof (induction p arbitrary:  $\sigma$  s)  
  fix p1 p2  $\sigma$  s  
  assume IH1: "(( $\sigma$ , s)  $\in$  init (opnet np p1)  $\implies$  net_ips s = net_tree_ips p1)"  
  and IH2: "(( $\sigma$ , s)  $\in$  init (opnet np p2)  $\implies$  net_ips s = net_tree_ips p2)"  
  and "(( $\sigma$ , s)  $\in$  init (opnet np (p1  $\parallel$  p2)))"  
  thus "net_ips s = net_tree_ips (p1  $\parallel$  p2)"  
  by (clarsimp simp add: net_ips_is_dom_netmap)  
  (metis Un_commute)  
  qed (clarsimp simp add: onode_comps)  
next  
  fix  $\sigma$  s  $\sigma'$  s' a  
  assume "(( $\sigma$ , s)  $\in$  oreachable (opnet np p) S U)"  
  and "net_ips s = net_tree_ips p"
```

```

    and " $((\sigma, s), a, (\sigma', s')) \in \text{trans } (\text{opnet } np \ p)$ "
    and " $S \ \sigma \ \sigma' \ a$ "
  thus "net_ips s' = net_tree_ips p"
  by (simp add: net_ips_is_dom_netmap)
    (metis net_ips_is_dom_netmap opnet_maintains_dom)
qed simp

lemma opnet_net_ips_net_tree_ips_init:
  assumes " $(\sigma, ns) \in \text{init } (\text{opnet } np \ p)$ "
  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 " $((\sigma, \text{SubnetS } s \ t), a, (\sigma', st')) \in \text{opnet\_sos } (\text{trans } (\text{opnet } np \ p1)) \ (\text{trans } (\text{opnet } np \ p2))$ "
  shows " $\exists s' \ t'. st' = \text{SubnetS } s' \ t'$ "
  using assms by cases simp_all

lemma net_par_oreachable_is_subnet:
  assumes " $(\sigma, st) \in \text{oreachable } (\text{opnet } np \ (p1 \parallel p2)) \ S \ U$ "
  shows " $\exists s \ t. st = \text{SubnetS } s \ t$ "
  proof -
    def p  $\equiv$  " $(\sigma, st)$ "
    with assms have "p  $\in$  oreachable (opnet np (p1 || 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
imports AWN OAWN_SOS OInvariants
begin

lemma node_net_state':
  assumes " $s \in \text{oreachable } (\langle i : T : R_i \rangle_o) \ S \ U$ "
  shows " $\exists \sigma \ \zeta \ R. s = (\sigma, \text{NodeS } i \ \zeta \ R)$ "
  using assms proof induction
    fix s
    assume " $s \in \text{init } (\langle i : T : R_i \rangle_o)$ "
    then obtain  $\sigma \ \zeta$  where " $s = (\sigma, \text{NodeS } i \ \zeta \ R_i)$ "
      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 " $U \ (\text{fst } s) \ \sigma'$ "
    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 " $\text{snd } s = \text{NodeS } i \ \zeta \ R$ " 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 " $\exists \sigma \ \zeta \ R. (\sigma', \text{snd } s) = (\sigma, \text{NodeS } i \ \zeta \ R)$ " 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 " $S \ (\text{fst } s) \ (\text{fst } s') \ a$ "
    from ih obtain  $\sigma \ \zeta \ R$  where " $s = (\sigma, \text{NodeS } i \ \zeta \ R)$ " by auto
  end
end

```



```

with tr have " $((\sigma, \text{NodeS } i \zeta R), a, s') \in \text{onode\_sos } (\text{trans } T)$ "
  by (simp add: onode_comps)
then obtain  $\sigma' \zeta' R'$  where " $s' = (\sigma', \text{NodeS } i \zeta' R')$ "
  using onode_sos_dest_is_net_state' by metis
with tr ' $s = (\sigma, \text{NodeS } i \zeta R)$ ' show " $\exists \sigma \zeta R. s' = (\sigma, \text{NodeS } i \zeta R)$ "
  by simp
qed

lemma node_net_state:
  assumes " $(\sigma, s) \in \text{oreachable } (\langle i : T : R_i \rangle_o) S U$ "
  shows " $\exists \zeta R. s = \text{NodeS } i \zeta R$ "
  using assms
  by (metis Pair_inject node_net_state')

lemma node_net_state_trans [elim]:
  assumes sor: " $(\sigma, s) \in \text{oreachable } (\langle i : \zeta_i : R_i \rangle_o) S U$ "
  and str: " $((\sigma, s), a, (\sigma', s')) \in \text{trans } (\langle i : \zeta_i : R_i \rangle_o)$ "
  obtains  $\zeta R \zeta' R'$ 
  where " $s = \text{NodeS } i \zeta R$ "
  and " $s' = \text{NodeS } i \zeta' R'$ "
  proof -
  assume *: " $\bigwedge \zeta R \zeta' R'. s = \text{NodeS } i \zeta R \implies s' = \text{NodeS } i \zeta' R' \implies \text{thesis}$ "
  from sor obtain  $\zeta R$  where " $s = \text{NodeS } i \zeta R$ "
  by (metis node_net_state)
  moreover with str obtain  $\zeta' R'$  where " $s' = \text{NodeS } i \zeta' R'$ "
  by (simp only: onode_comps)
  (metis onode_sos_dest_is_net_state'')
  ultimately show thesis by (rule *)
  qed

lemma nodemap_induct' [consumes, case_names init other local]:
  assumes " $(\sigma, \text{NodeS } ii \zeta R) \in \text{oreachable } (\langle ii : T : R_i \rangle_o) S U$ "
  and init: " $\bigwedge \sigma \zeta. (\sigma, \text{NodeS } ii \zeta R_i) \in \text{init } (\langle ii : T : R_i \rangle_o) \implies P (\sigma, \text{NodeS } ii \zeta R_i)$ "
  and other: " $\bigwedge \sigma \zeta R \sigma' a.
    \llbracket (\sigma, \text{NodeS } ii \zeta R) \in \text{oreachable } (\langle ii : T : R_i \rangle_o) S U;
    U \sigma \sigma'; P (\sigma, \text{NodeS } ii \zeta R) \rrbracket \implies P (\sigma', \text{NodeS } ii \zeta R)$ "
  and local: " $\bigwedge \sigma \zeta R \sigma' \zeta' R' a.
    \llbracket (\sigma, \text{NodeS } ii \zeta R) \in \text{oreachable } (\langle ii : T : R_i \rangle_o) S U;
    ((\sigma, \text{NodeS } ii \zeta R), a, (\sigma', \text{NodeS } ii \zeta' R')) \in \text{trans } (\langle ii : T : R_i \rangle_o);
    S \sigma \sigma' a; P (\sigma, \text{NodeS } ii \zeta R) \rrbracket \implies P (\sigma', \text{NodeS } ii \zeta' R')$ "
  shows " $P (\sigma, \text{NodeS } ii \zeta R)$ "
  using assms(1) proof induction
  fix s
  assume " $s \in \text{init } (\langle ii : T : R_i \rangle_o)$ "
  hence " $s \in \text{oreachable } (\langle ii : T : R_i \rangle_o) S U$ "
  by (rule oreachable_init)
  with ' $s \in \text{init } (\langle ii : T : R_i \rangle_o)$ ' obtain  $\sigma \zeta$  where " $s = (\sigma, \text{NodeS } ii \zeta R_i)$ "
  using node_net_state by (simp add: onode_comps) metis
  with ' $s \in \text{init } (\langle ii : T : R_i \rangle_o)$ ' and init show " $P s$ " by simp
  next
  fix s a  $\sigma'$ 
  assume sr: " $s \in \text{oreachable } (\langle ii : T : R_i \rangle_o) S U$ "
  and " $U (\text{fst } s) \sigma'$ "
  and " $P s$ "
  from sr obtain  $\sigma \zeta R$  where " $s = (\sigma, \text{NodeS } ii \zeta R)$ "
  using node_net_state' by metis
  with sr ' $U (\text{fst } s) \sigma'$ ' ' $P s$ ' show " $P (\sigma', \text{snd } s)$ "
  by simp (metis other)
  next
  fix s a  $s'$ 
  assume sr: " $s \in \text{oreachable } (\langle ii : T : R_i \rangle_o) S U$ "
  and tr: " $(s, a, s') \in \text{trans } (\langle ii : T : R_i \rangle_o)$ "
  and " $S (\text{fst } s) (\text{fst } s') a$ "
  and " $P s$ "

```

from this(1-3) have " $s' \in \text{oreachable } (\langle ii : T : R_i \rangle_o) S U$ "
 by - (erule(2) oreachable_local)
 then obtain $\sigma' \zeta' R'$ where [simp]: " $s' = (\sigma', \text{NodeS } ii \zeta' R')$ "
 using node_net_state' by metis
 from sr and ' $P s'$ ' obtain $\sigma \zeta R$
 where [simp]: " $s = (\sigma, \text{NodeS } ii \zeta R)$ "
 and A1: " $(\sigma, \text{NodeS } ii \zeta R) \in \text{oreachable } (\langle ii : T : R_i \rangle_o) S U$ "
 and A4: " $P (\sigma, \text{NodeS } ii \zeta R)$ "
 using node_net_state' by metis
 with tr and ' $S (\text{fst } s) (\text{fst } s') a'$ '
 have A2: " $((\sigma, \text{NodeS } ii \zeta R), a, (\sigma', \text{NodeS } ii \zeta' R')) \in \text{trans } (\langle ii : T : R_i \rangle_o)$ "
 and A3: " $S \sigma \sigma' a$ " by simp_all
 from A1 A2 A3 A4 have " $P (\sigma', \text{NodeS } ii \zeta' R')$ " by (rule local)
 thus " $P s'$ " by simp
 qed

lemma nodemap_induct [consumes, case_names init step]:

assumes " $(\sigma, \text{NodeS } ii \zeta R) \in \text{oreachable } (\langle ii : T : R_i \rangle_o) S U$ "
 and init: " $\bigwedge \sigma \zeta. (\sigma, \text{NodeS } ii \zeta R_i) \in \text{init } (\langle ii : T : R_i \rangle_o) \implies P \sigma \zeta R_i$ "
 and other: " $\bigwedge \sigma \zeta R \sigma' a.$
 $\llbracket (\sigma, \text{NodeS } ii \zeta R) \in \text{oreachable } (\langle ii : T : R_i \rangle_o) S U;$
 $U \sigma \sigma'; P \sigma \zeta R \rrbracket \implies P \sigma' \zeta R$ "
 and local: " $\bigwedge \sigma \zeta R \sigma' \zeta' R' a.$
 $\llbracket (\sigma, \text{NodeS } ii \zeta R) \in \text{oreachable } (\langle ii : T : R_i \rangle_o) S U;$
 $((\sigma, \text{NodeS } ii \zeta R), a, (\sigma', \text{NodeS } ii \zeta' R')) \in \text{trans } (\langle ii : T : R_i \rangle_o);$
 $S \sigma \sigma' a; P \sigma \zeta R \rrbracket \implies P \sigma' \zeta' R'$ "
 shows " $P \sigma \zeta R$ "

using assms(1) proof (induction " $(\sigma, \text{NodeS } ii \zeta R)$ " arbitrary: $\sigma \zeta R$)

fix $\sigma \zeta R$
 assume a1: " $(\sigma, \text{NodeS } ii \zeta R) \in \text{init } (\langle ii : T : R_i \rangle_o)$ "
 hence " $R = R_i$ " by (simp add: init_onode_comp)
 with a1 have " $(\sigma, \text{NodeS } ii \zeta R_i) \in \text{init } (\langle ii : T : R_i \rangle_o)$ " by simp
 with init and ' $R = R_i$ ' show " $P \sigma \zeta R$ " by simp

next

fix st a $\sigma' \zeta' R'$
 assume " $st \in \text{oreachable } (\langle ii : T : R_i \rangle_o) S U$ "
 and tr: " $(st, a, (\sigma', \text{NodeS } ii \zeta' R')) \in \text{trans } (\langle ii : T : R_i \rangle_o)$ "
 and " $S (\text{fst } st) (\text{fst } (\sigma', \text{NodeS } ii \zeta' R')) a$ "
 and IH: " $\bigwedge \sigma \zeta R. st = (\sigma, \text{NodeS } ii \zeta R) \implies P \sigma \zeta R$ "
 from this(1) obtain $\sigma \zeta R$ where " $st = (\sigma, \text{NodeS } ii \zeta R)$ "
 and " $(\sigma, \text{NodeS } ii \zeta R) \in \text{oreachable } (\langle ii : T : R_i \rangle_o) S U$ "
 by (metis node_net_state')

note this(2)

moreover from tr and ' $st = (\sigma, \text{NodeS } ii \zeta R)$ '
 have " $((\sigma, \text{NodeS } ii \zeta R), a, (\sigma', \text{NodeS } ii \zeta' R')) \in \text{trans } (\langle ii : T : R_i \rangle_o)$ " by simp
 moreover from ' $S (\text{fst } st) (\text{fst } (\sigma', \text{NodeS } ii \zeta' R')) a$ ' and ' $st = (\sigma, \text{NodeS } ii \zeta R)$ '
 have " $S \sigma \sigma' a$ " by simp
 moreover from IH and ' $st = (\sigma, \text{NodeS } ii \zeta R)$ ' have " $P \sigma \zeta R$ ".
 ultimately show " $P \sigma' \zeta' R'$ " by (rule local)

next

fix st $\sigma' \zeta R$
 assume " $st \in \text{oreachable } (\langle ii : T : R_i \rangle_o) S U$ "
 and " $U (\text{fst } st) \sigma'$ "
 and " $\text{snd } st = \text{NodeS } ii \zeta R$ "
 and IH: " $\bigwedge \sigma \zeta R. st = (\sigma, \text{NodeS } ii \zeta R) \implies P \sigma \zeta R$ "
 from this(1,3) obtain σ where " $st = (\sigma, \text{NodeS } ii \zeta R)$ "
 and " $(\sigma, \text{NodeS } ii \zeta R) \in \text{oreachable } (\langle ii : T : R_i \rangle_o) S U$ "
 by (metis surjective_pairing)

note this(2)

moreover from ' $U (\text{fst } st) \sigma'$ ' and ' $st = (\sigma, \text{NodeS } ii \zeta R)$ ' have " $U \sigma \sigma'$ " by simp
 moreover from IH and ' $st = (\sigma, \text{NodeS } ii \zeta R)$ ' have " $P \sigma \zeta R$ ".
 ultimately show " $P \sigma' \zeta R$ " by (rule other)

qed

```

lemma node_addressD [dest, simp]:
  assumes "(σ, NodeS i ζ R) ∈ oreachable (⟨ii : T : R_i⟩_o) S U"
  shows "i = ii"
  using assms by (clarsimp dest!: node_net_state')

lemma node_proc_reachable [dest]:
  assumes "(σ, NodeS i ζ R) ∈ oreachable (⟨ii : T : R_i⟩_o)
    (otherwith S {ii} (oarrivemsg I)) (other U {ii})"
  and sgivesu: "∧ξ ξ'. S ξ ξ' ⇒ U ξ ξ'"
  shows "(σ, ζ) ∈ oreachable T (otherwith S {ii} (orecvmsg I)) (other U {ii})"
proof -
  from assms(1) have "(σ, NodeS ii ζ R) ∈ oreachable (⟨ii : T : R_i⟩_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_i) ∈ init (⟨ii : T : R_i⟩_o)"
  hence "(σ, ζ) ∈ init T" by (auto simp: onode_comps)
  thus "(σ, ζ) ∈ oreachable 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) ∈ oreachable T (otherwith S {ii} (orecvmsg I)) (other U {ii})"
  thus "(σ', ζ) ∈ oreachable 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) ∈ oreachable (⟨ii : T : R_i⟩_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_i⟩_o)"
  and ow: "otherwith S {ii} (oarrivemsg I) σ σ' a"
  and ih: "(σ, ζ) ∈ oreachable 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') ∈ oreachable 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 " $\zeta' = \zeta$ " and " $\sigma' ii = \sigma ii$ " by auto
  from this(2) have "other U {ii}  $\sigma \sigma'$ " by (rule *)
  with ih and ' $\zeta' = \zeta'$ ' show ?thesis by auto
next
  case onode_connect1
  hence " $\zeta' = \zeta$ " and " $\sigma' ii = \sigma ii$ " by auto
  from this(2) have "other U {ii}  $\sigma \sigma'$ " by (rule *)
  with ih and ' $\zeta' = \zeta'$ ' show ?thesis by auto
next
  case onode_connect2
  hence " $\zeta' = \zeta$ " and " $\sigma' ii = \sigma ii$ " by auto
  from this(2) have "other U {ii}  $\sigma \sigma'$ " by (rule *)
  with ih and ' $\zeta' = \zeta'$ ' show ?thesis by auto
next
  case onode_connect_other
  hence " $\zeta' = \zeta$ " and " $\sigma' ii = \sigma ii$ " by auto
  from this(2) have "other U {ii}  $\sigma \sigma'$ " by (rule *)
  with ih and ' $\zeta' = \zeta'$ ' show ?thesis by auto
next
  case onode_disconnect1
  hence " $\zeta' = \zeta$ " and " $\sigma' ii = \sigma ii$ " by auto
  from this(2) have "other U {ii}  $\sigma \sigma'$ " by (rule *)
  with ih and ' $\zeta' = \zeta'$ ' show ?thesis by auto
next
  case onode_disconnect2
  hence " $\zeta' = \zeta$ " and " $\sigma' ii = \sigma ii$ " by auto
  from this(2) have "other U {ii}  $\sigma \sigma'$ " by (rule *)
  with ih and ' $\zeta' = \zeta'$ ' show ?thesis by auto
next
  case onode_disconnect_other
  hence " $\zeta' = \zeta$ " and " $\sigma' ii = \sigma ii$ " by auto
  from this(2) have "other U {ii}  $\sigma \sigma'$ " by (rule *)
  with ih and ' $\zeta' = \zeta'$ ' show ?thesis by auto
qed
qed
qed

```

lemma node_proc_reachable_statelessasm [dest]:

```

  assumes " $(\sigma, \text{NodeS } i \ \zeta \ R) \in \text{oreachable } (\langle ii : T : R_i \rangle_o)$ "
  (otherwith  $(\lambda \_ \_ . \text{True}) \{ii\} (\text{oarrivemsg } I)$ )
  (other  $(\lambda \_ \_ . \text{True}) \{ii\}$ )"
  shows " $(\sigma, \zeta) \in \text{oreachable } T$ "
  (otherwith  $(\lambda \_ \_ . \text{True}) \{ii\} (\text{orecvmsg } I)$ ) (other  $(\lambda \_ \_ . \text{True}) \{ii\}$ )"
  using assms
  by (rule node_proc_reachable) simp_all

```

lemma node_lift:

```

  assumes " $T \models (\text{otherwith } S \{ii\} (\text{orecvmsg } I), \text{other } U \{ii\} \rightarrow) \text{global } P$ "
  and " $\bigwedge \xi \xi' . S \xi \xi' \implies U \xi \xi'$ "
  shows " $\langle ii : T : R_i \rangle_o \models (\text{otherwith } S \{ii\} (\text{oarrivemsg } I), \text{other } U \{ii\} \rightarrow) \text{global } P$ "
  proof (rule oinvariant_oreachableI)
  fix  $\sigma \ \zeta$ 
  assume " $(\sigma, \zeta) \in \text{oreachable } (\langle ii : T : R_i \rangle_o) (\text{otherwith } S \{ii\} (\text{oarrivemsg } I)) (\text{other } U \{ii\})$ "
  moreover then obtain  $i \ s \ R$  where " $\zeta = \text{NodeS } i \ s \ R$ "
  by (metis node_net_state)
  ultimately have " $(\sigma, \text{NodeS } i \ s \ R) \in \text{oreachable } (\langle ii : T : R_i \rangle_o)$ "
  (otherwith  $S \{ii\} (\text{oarrivemsg } I)$ ) (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: " $\bigwedge \sigma \sigma'. \text{ other } U \{i\} \sigma \sigma' \implies Q \sigma \sigma'$ "
 and sgivesu: " $\bigwedge \xi \xi'. S \xi \xi' \implies U \xi \xi'$ "
 shows " $\langle i : T : R_i \rangle_o \models_A \text{ (otherwith } S \{i\} \text{ (oarrivemsg } I), \text{ other } U \{i\} \rightarrow) \text{ globala } (\lambda(\sigma, _, \sigma'). Q \sigma \sigma')$ "
 (is " $_ \models_A (?S, ?U \rightarrow) _$ ")

proof (rule ostep_invariantI, simp)

fix $\sigma s a \sigma' s'$

assume rs: " $(\sigma, s) \in \text{oreachable } (\langle i : T : R_i \rangle_o) ?S ?U$ "
 and tr: " $((\sigma, s), a, (\sigma', s')) \in \text{trans } (\langle i : T : R_i \rangle_o)$ "
 and ow: " $?S \sigma \sigma' a$ "

from ow have *: " $\sigma' i = \sigma i \implies \text{other } U \{i\} \sigma \sigma'$ "

by (clarsimp elim!: otherwithE) (rule otherI, simp_all, metis sgivesu)

from rs tr obtain ζR

where [simp]: " $s = \text{NodeS } i \zeta R$ "
 and " $(\sigma, \text{NodeS } i \zeta R) \in \text{oreachable } (\langle i : T : R_i \rangle_o) ?S ?U$ "
 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 " $((\sigma, \text{NodeS } i \zeta R), a, (\sigma', s')) \in \text{onode_sos } (\text{trans } T)$ "

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 " $\text{otherwith } S \{i\} \text{ (orecvmsg } I) \sigma \sigma' \text{ (broadcast } m)$ "

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 " $\text{otherwith } S \{i\} \text{ (orecvmsg } I) \sigma \sigma' \text{ (groupcast } D m)$ "

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 " $\text{otherwith } S \{i\} \text{ (orecvmsg } I) \sigma \sigma' \text{ (unicast } d m)$ "

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': " $((\sigma, \zeta), \neg \text{unicast } d, (\sigma', \zeta')) \in \text{trans } T$ "

from this(1) and ' $?S \sigma \sigma' a$ ' have " $\text{otherwith } S \{i\} \text{ (orecvmsg } I) \sigma \sigma' \text{ (}\neg \text{unicast } d)$ "

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 " $\text{otherwith } S \{i\} \text{ (orecvmsg } I) \sigma \sigma' \text{ (deliver } d)$ "

by (auto elim!: otherwithE)

with or tr' show ?thesis by (rule ostep_invariantD [OF pinv, simplified])

```

next
  fix ζ'
  assume "a = τ"
  and tr': "((σ, ζ), τ, (σ', ζ')) ∈ trans T"
  from this(1) and 'S σ σ' a' have "otherwith S {i} (orecvmsg I) σ σ' τ"
  by (auto elim!: otherwithE)
  with or tr' show ?thesis by (rule ostep_invariantD [OF pinv, simplified])
next
  fix m ζ'
  assume "a = {i}¬{i}:arrive(m)"
  and tr': "((σ, ζ), receive m, (σ', ζ')) ∈ trans T"
  from this(1) and 'S σ σ' a' have "otherwith S {i} (orecvmsg I) σ σ' (receive m)"
  by (auto elim!: otherwithE)
  with or tr' show ?thesis by (rule ostep_invariantD [OF pinv, simplified])
next
  fix m
  assume "a = {i}¬{i}:arrive(m)"
  and "σ' i = σ i"
  from this(2) have "other U {i} σ σ'" by (rule *)
  thus ?thesis by (rule other)
next
  fix i'
  assume "a = connect(i, i')"
  and "σ' i = σ i"
  from this(2) have "other U {i} σ σ'" by (rule *)
  thus ?thesis by (rule other)
next
  fix i'
  assume "a = connect(i', i)"
  and "σ' i = σ i"
  from this(2) have "other U {i} σ σ'" by (rule *)
  thus ?thesis by (rule other)
next
  fix i' i''
  assume "a = connect(i', i'')"
  and "σ' i = σ i"
  from this(2) have "other U {i} σ σ'" by (rule *)
  thus ?thesis by (rule other)
next
  fix i'
  assume "a = disconnect(i, i')"
  and "σ' i = σ i"
  from this(2) have "other U {i} σ σ'" by (rule *)
  thus ?thesis by (rule other)
next
  fix i'
  assume "a = disconnect(i', i)"
  and "σ' i = σ i"
  from this(2) have "other U {i} σ σ'" by (rule *)
  thus ?thesis by (rule other)
next
  fix i' i''
  assume "a = disconnect(i', i'')"
  and "σ' i = σ i"
  from this(2) have "other U {i} σ σ'" by (rule *)
  thus ?thesis by (rule other)
qed
qed

```

```

lemma node_lift_step_statelessassm [intro]:
  assumes "T ⊨A (λσ _. orecvmsg I σ, other (λ_ _. True) {i} →)
           globala (λ(σ, _, σ'). Q (σ i) (σ' i))"
  and "∧ξ. Q ξ ξ"
  shows "<i : T : R_i>o ⊨A (λσ _. oarrivemsg I σ, other (λ_ _. True) {i} →)"

```

```

                                globala (λ(σ, _, σ'). Q (σ i) (σ' i))"
proof -
  from assms(1)
    have "T ⊢A (otherwith (λ_ _ . True) {i} (orecvmsg I), other (λ_ _ . True) {i} →)
            globala (λ(σ, _, σ'). Q (σ i) (σ' i))"
    by rule auto
  with assms(2) have "<i : T : Ri>o ⊢A (otherwith (λ_ _ . True) {i} (oarrivmsg I),
            other (λ_ _ . True) {i} →)
            globala (λ(σ, _, σ'). Q (σ i) (σ' i))"
    by - (rule node_lift_step, auto)
  thus ?thesis by rule auto
qed

```

lemma node_lift_anycast [intro]:

```

assumes pinv: "T ⊢A (otherwith S {i} (orecvmsg I), other U {i} →)
            globala (λ(σ, a, σ'). anycast (Q σ σ') a)"
  and "∧ξ ξ'. S ξ ξ' ⇒ U ξ ξ'"
shows "<i : T : Ri>o ⊢A (otherwith S {i} (oarrivmsg I), other U {i} →)
            globala (λ(σ, a, σ'). castmsg (Q σ σ') a)"
(is "_ ⊢A (?S, ?U →) _")
proof (rule ostep_invariantI, simp)
  fix σ s a σ' s'
  assume rs: "(σ, s) ∈ oreachable (<i : T : Ri>o) ?S ?U"
  and tr: "((σ, s), a, (σ', s')) ∈ trans (<i : T : Ri>o)"
  and "?S σ σ' a"
  from this(1-2) obtain ζ R
  where [simp]: "s = NodeS i ζ R"
  and "(σ, NodeS i ζ R) ∈ oreachable (<i : T : Ri>o) ?S ?U"
  by (metis node_net_state)
  from this(2) have "(σ, ζ) ∈ oreachable T (otherwith S {i} (orecvmsg I)) ?U"
  by (rule node_proc_reachable [OF _ assms(2)])
  moreover from tr have "((σ, NodeS i ζ R), a, (σ', s')) ∈ onode_sos (trans T)"
  by (simp add: onode_comps)
  ultimately show "castmsg (Q σ σ') a" using ' ?S σ σ' a '
  by - (erule onode_sos.cases, auto elim!: otherwithE dest!: ostep_invariantD [OF pinv])
qed

```

lemma node_lift_anycast_statelessassm [intro]:

```

assumes pinv: "T ⊢A (λσ _ . orecvmsg I σ, other (λ_ _ . True) {i} →)
            globala (λ(σ, a, σ'). anycast (Q σ σ') a)"
shows "<i : T : Ri>o ⊢A (λσ _ . oarrivmsg I σ, other (λ_ _ . True) {i} →)
            globala (λ(σ, a, σ'). castmsg (Q σ σ') a)"
(is "_ ⊢A (?S, _ →) _")
proof -
  from assms(1)
    have "T ⊢A (otherwith (λ_ _ . True) {i} (orecvmsg I), other (λ_ _ . True) {i} →)
            globala (λ(σ, a, σ'). anycast (Q σ σ') a)"
    by rule auto
  hence "<i : T : Ri>o ⊢A (otherwith (λ_ _ . True) {i} (oarrivmsg I), other (λ_ _ . True) {i} →)
            globala (λ(σ, a, σ'). castmsg (Q σ σ') a)"
    by (rule node_lift_anycast) simp_all
  thus ?thesis
    by rule auto
qed

```

lemma node_local_deliver:

```

"<i : ζi : Ri>o ⊢A (S, U →) globala (λ(_, a, _). ∀j. j≠i → (∀d. a ≠ j:deliver(d)))"
proof (rule ostep_invariantI, simp)
  fix σ s a σ' s'
  assume "(σ, s) ∈ oreachable (<i : ζi : Ri>o) S U"
  and "((σ, s), a, (σ', s')) ∈ trans (<i : ζi : Ri>o)"
  and "S σ σ' a"
  moreover from this(1-2) obtain ζ R ζ' R' where "s = NodeS i ζ R" and "s' = NodeS i ζ' R'" ..
  ultimately show "∀j. j≠i → (∀d. a ≠ j:deliver(d))"

```

by (cases a) (auto simp add: onode_comps)
qed

lemma node_tau_deliver_unchanged:

" $\langle i : \zeta_i : R_i \rangle_o \models_A (S, U \rightarrow) \text{globala } (\lambda(\sigma, a, \sigma'). a = \tau \vee (\exists i d. a = i:\text{deliver}(d)) \rightarrow (\forall j. j \neq i \rightarrow \sigma' j = \sigma j))$ "

proof (rule ostep_invariantI, clarsimp simp only: globalasimp snd_conv fst_conv)

fix $\sigma s a \sigma' s' j$

assume " $(\sigma, s) \in \text{oreachable } (\langle i : \zeta_i : R_i \rangle_o) S U$ "

and " $((\sigma, s), a, (\sigma', s')) \in \text{trans } (\langle i : \zeta_i : R_i \rangle_o)$ "

and " $S \sigma \sigma' a$ "

and " $a = \tau \vee (\exists i d. a = i:\text{deliver}(d))$ "

and " $j \neq i$ "

moreover from this(1-2) obtain $\zeta R \zeta' R'$ where " $s = \text{NodeS } i \zeta R$ " and " $s' = \text{NodeS } i \zeta' R'$ " ..

ultimately show " $\sigma' j = \sigma j$ "

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 " $(\sigma, \text{SubnetS } s t) \in \text{oreachable } (\text{opnet onp } (p_1 \parallel p_2)) S U$ "

and init: " $\bigwedge \sigma s t. (\sigma, \text{SubnetS } s t) \in \text{init } (\text{opnet onp } (p_1 \parallel p_2)) \implies P \sigma s t$ "

and other: " $\bigwedge \sigma s t \sigma'. \llbracket (\sigma, \text{SubnetS } s t) \in \text{oreachable } (\text{opnet onp } (p_1 \parallel p_2)) S U; U \sigma \sigma'; P \sigma s t \rrbracket \implies P \sigma' s t$ "

and local: " $\bigwedge \sigma s t \sigma' s' t' a. \llbracket (\sigma, \text{SubnetS } s t) \in \text{oreachable } (\text{opnet onp } (p_1 \parallel p_2)) S U; ((\sigma, \text{SubnetS } s t), a, (\sigma', \text{SubnetS } s' t')) \in \text{trans } (\text{opnet onp } (p_1 \parallel p_2)); S \sigma \sigma' a; P \sigma s t \rrbracket \implies P \sigma' s' t'$ "

shows " $P \sigma s t$ "

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

fix $s t \sigma$

assume " $(\sigma, \text{SubnetS } s t) \in \text{init } (\text{opnet onp } (p_1 \parallel p_2))$ "

with init show " $P \sigma s t$ ".

next

fix $st a s' t' \sigma'$

assume " $st \in \text{oreachable } (\text{opnet onp } (p_1 \parallel p_2)) S U$ "

and tr: " $(st, a, (\sigma', \text{SubnetS } s' t')) \in \text{trans } (\text{opnet onp } (p_1 \parallel p_2))$ "

and " $S (fst st) (fst (\sigma', \text{SubnetS } s' t')) a$ "

and IH: " $\bigwedge s t \sigma. st = (\sigma, \text{SubnetS } s t) \implies P \sigma s t$ "

from this(1) obtain $s t \sigma$ where " $st = (\sigma, \text{SubnetS } s t)$ "

and " $(\sigma, \text{SubnetS } s t) \in \text{oreachable } (\text{opnet onp } (p_1 \parallel p_2)) S U$ "

by (metis net_par_oreachable_is_subnet pair_collapse)

note this(2)

moreover from tr and ' $st = (\sigma, \text{SubnetS } s t)$ '

have " $((\sigma, \text{SubnetS } s t), a, (\sigma', \text{SubnetS } s' t')) \in \text{trans } (\text{opnet onp } (p_1 \parallel p_2))$ " by simp

moreover from ' $S (fst st) (fst (\sigma', \text{SubnetS } s' t')) a$ ' and ' $st = (\sigma, \text{SubnetS } s t)$ '

have " $S \sigma \sigma' a$ " by simp

moreover from IH and ' $st = (\sigma, \text{SubnetS } s t)$ ' have " $P \sigma s t$ ".

ultimately show " $P \sigma' s' t'$ " by (rule local)

next

fix $st \sigma' s t$

assume " $st \in \text{oreachable } (\text{opnet onp } (p_1 \parallel p_2)) S U$ "

and " $U (fst st) \sigma'$ "

and " $\text{snd } st = \text{SubnetS } s t$ "

and IH: " $\bigwedge s t \sigma. st = (\sigma, \text{SubnetS } s t) \implies P \sigma s t$ "

from this(1,3) obtain σ where " $st = (\sigma, \text{SubnetS } s t)$ "

and " $(\sigma, \text{SubnetS } s t) \in \text{oreachable } (\text{opnet onp } (p_1 \parallel p_2)) S U$ "

by (metis pair_collapse)

note this(2)
 moreover from ' $U (fst\ st)\ \sigma'$ ' and ' $st = (\sigma, SubnetS\ s\ t)$ ' have " $U\ \sigma\ \sigma'$ " by simp
 moreover from IH and ' $st = (\sigma, SubnetS\ s\ t)$ ' have " $P\ \sigma\ s\ t$ ".
 ultimately show " $P\ \sigma'\ s\ t$ " by (rule other)
 qed

lemma other_net_tree_ips_par_left:

assumes "other $U (net_tree_ips\ (p_1\ \parallel\ p_2))\ \sigma\ \sigma'$ "
 and " $\bigwedge \xi. U\ \xi\ \xi$ "
 shows "other $U (net_tree_ips\ p_1)\ \sigma\ \sigma'$ "

proof -

from assms(1) obtain ineq: " $\forall i \in net_tree_ips\ (p_1\ \parallel\ p_2). \sigma'\ i = \sigma\ i$ "
 and outU: " $\forall j. j \notin net_tree_ips\ (p_1\ \parallel\ p_2) \longrightarrow U\ (\sigma\ j)\ (\sigma'\ j)$ " ..

show ?thesis

proof (rule otherI)

fix i

assume " $i \in net_tree_ips\ p_1$ "

hence " $i \in net_tree_ips\ (p_1\ \parallel\ p_2)$ " by simp

with ineq show " $\sigma'\ i = \sigma\ i$ " ..

next

fix j

assume " $j \notin net_tree_ips\ p_1$ "

show " $U\ (\sigma\ j)\ (\sigma'\ j)$ "

proof (cases " $j \in net_tree_ips\ p_2$ ")

assume " $j \in net_tree_ips\ p_2$ "

hence " $j \in net_tree_ips\ (p_1\ \parallel\ p_2)$ " by simp

with ineq have " $\sigma'\ j = \sigma\ j$ " ..

thus " $U\ (\sigma\ j)\ (\sigma'\ j)$ "

by simp (rule ' $\bigwedge \xi. U\ \xi\ \xi$ ')

next

assume " $j \notin net_tree_ips\ p_2$ "

with ' $j \notin net_tree_ips\ p_1$ ' have " $j \notin net_tree_ips\ (p_1\ \parallel\ p_2)$ " by simp

with outU show " $U\ (\sigma\ j)\ (\sigma'\ j)$ " by simp

qed

qed

qed

lemma other_net_tree_ips_par_right:

assumes "other $U (net_tree_ips\ (p_1\ \parallel\ p_2))\ \sigma\ \sigma'$ "
 and " $\bigwedge \xi. U\ \xi\ \xi$ "
 shows "other $U (net_tree_ips\ p_2)\ \sigma\ \sigma'$ "

proof -

from assms(1) have "other $U (net_tree_ips\ (p_2\ \parallel\ p_1))\ \sigma\ \sigma'$ "

by (subst net_tree_ips_commute)

thus ?thesis using ' $\bigwedge \xi. U\ \xi\ \xi$ '

by (rule other_net_tree_ips_par_left)

qed

lemma ostep_arrive_invariantD [elim]:

assumes " $p \models_A (\lambda \sigma _ . oarrivemsg\ I\ \sigma, U \rightarrow) P$ "
 and " $(\sigma, s) \in oreachable\ p\ (otherwith\ S\ IPS\ (oarrivemsg\ I))\ U$ "
 and " $((\sigma, s), a, (\sigma', s')) \in trans\ p$ "
 and " $oarrivemsg\ I\ \sigma\ a$ "
 shows " $P\ ((\sigma, s), a, (\sigma', s'))$ "

proof -

from assms(2) have " $(\sigma, s) \in oreachable\ p\ (\lambda \sigma _ a. oarrivemsg\ I\ \sigma\ a)\ U$ "

by (rule oreachable_weakenE) auto

thus " $P\ ((\sigma, s), a, (\sigma', s'))$ "

using assms(3-4) by (rule ostep_invariantD [OF assms(1)])

qed

lemma opnet_sync_action_subnet_oreachable:

assumes " $(\sigma, SubnetS\ s\ t) \in oreachable\ (opnet\ onp\ (p_1\ \parallel\ p_2))$
 ($\lambda \sigma _ . oarrivemsg\ I\ \sigma$) (other $U (net_tree_ips\ (p_1\ \parallel\ p_2))$)"

```

(is "_ ∈ oreachable _ (?S (p1 || p2)) (?U (p1 || p2)))"
and "∧ξ. U ξ ξ"
and act1: "opnet onp p1 ⊨A (λσ _. oarrivemsg I σ, other U (net_tree_ips p1) →)
  globala (λ(σ, a, σ'). castmsg (I σ) a
    ∧ (a = τ ∨ (∃i d. a = i:deliver(d)) →
      ((∀i∈net_tree_ips p1. U (σ i) (σ' i))
        ∧ (∀i. i∉net_tree_ips p1 → σ' i = σ i))))"
and act2: "opnet onp p2 ⊨A (λσ _. oarrivemsg I σ, other U (net_tree_ips p2) →)
  globala (λ(σ, a, σ'). castmsg (I σ) a
    ∧ (a = τ ∨ (∃i d. a = i:deliver(d)) →
      ((∀i∈net_tree_ips p2. U (σ i) (σ' i))
        ∧ (∀i. i∉net_tree_ips p2 → σ' i = σ i))))"
shows "(σ, s) ∈ oreachable (opnet onp p1) (λσ _. oarrivemsg I σ) (other U (net_tree_ips p1))
  ∧ (σ, t) ∈ oreachable (opnet onp p2) (λσ _. oarrivemsg I σ) (other U (net_tree_ips p2))
  ∧ net_tree_ips p1 ∩ net_tree_ips p2 = {}"
using assms(1)
proof (induction rule: oreachable_par_subnet_induct)
  case (init σ s t)
  hence sinit: "(σ, s) ∈ init (opnet onp p1)"
  and tinit: "(σ, t) ∈ init (opnet onp p2)"
  and "net_ips s ∩ net_ips t = {}" by auto
  moreover from sinit have "net_ips s = net_tree_ips p1"
  by (rule opnet_net_ips_net_tree_ips_init)
  moreover from tinit have "net_ips t = net_tree_ips p2"
  by (rule opnet_net_ips_net_tree_ips_init)
  ultimately show ?case by (auto elim: oreachable_init)
next
  case (other σ s t σ')
  hence "other U (net_tree_ips (p1 || p2)) σ σ'"
  and IHs: "(σ, s) ∈ oreachable (opnet onp p1) (?S p1) (?U p1)"
  and IHt: "(σ, t) ∈ oreachable (opnet onp p2) (?S p2) (?U p2)"
  and "net_tree_ips p1 ∩ net_tree_ips p2 = {}" by auto
  have "(σ', s) ∈ oreachable (opnet onp p1) (?S p1) (?U p1)"
  proof -
    from '??U (p1 || p2) σ σ'' and '∧ξ. U ξ ξ' have "??U p1 σ σ'"
    by (rule other_net_tree_ips_par_left)
    with IHs show ?thesis by - (erule(1) oreachable_other')
  qed
  moreover have "(σ', t) ∈ oreachable (opnet onp p2) (?S p2) (?U p2)"
  proof -
    from '??U (p1 || p2) σ σ'' and '∧ξ. U ξ ξ' have "??U p2 σ σ'"
    by (rule other_net_tree_ips_par_right)
    with IHt show ?thesis by - (erule(1) oreachable_other')
  qed
  ultimately show ?case using 'net_tree_ips p1 ∩ net_tree_ips p2 = {}' by simp
next
  case (local σ s t σ' s' t' a)
  hence stor: "(σ, SubnetS s t) ∈ oreachable (opnet onp (p1 || p2)) (?S (p1 || p2)) (?U (p1 || p2))"
  and tr: "((σ, SubnetS s t), a, (σ', SubnetS s' t')) ∈ trans (opnet onp (p1 || p2))"
  and "oarrivemsg I σ a"
  and sor: "(σ, s) ∈ oreachable (opnet onp p1) (?S p1) (?U p1)"
  and tor: "(σ, t) ∈ oreachable (opnet onp p2) (?S p2) (?U p2)"
  and "net_tree_ips p1 ∩ net_tree_ips p2 = {}" by auto
  from tr have "((σ, SubnetS s t), a, (σ', SubnetS s' t'))
    ∈ opnet_sos (trans (opnet onp p1)) (trans (opnet onp p2))" by simp
  hence "(σ', s') ∈ oreachable (opnet onp p1) (?S p1) (?U p1)
    ∧ (σ', t') ∈ oreachable (opnet onp p2) (?S p2) (?U p2)"

```

```

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') ∈ oreachable (opnet onp p₁) (?S p₁) (?U p₁)"
      by - (erule(1) oreachable_local, auto)
  moreover from 'I σ m' tor ttr
    have "(σ', t') ∈ oreachable (opnet onp p₂) (?S p₂) (?U p₂)"
      by - (erule(1) oreachable_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), H¬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') ∈ oreachable (opnet onp p₁) (?S p₁) (?U p₁)"
      by - (erule(1) oreachable_local, auto)
  moreover from 'I σ m' tor ttr
    have "(σ', t') ∈ oreachable (opnet onp p₂) (?S p₂) (?U p₂)"
      by - (erule(1) oreachable_local, auto)
  ultimately show ?thesis ..

next
  fix R m H K
  assume str: "((σ, s), H¬K:arrive(m), (σ', s')) ∈ trans (opnet onp p₁)"
    and ttr: "((σ, t), R:*cast(m), (σ', t')) ∈ trans (opnet onp p₂)"
  from tor ttr have "I σ m"
    by - (drule(1) ostep_invariantD [OF act2], simp_all)
  with sor str
    have "(σ', s') ∈ oreachable (opnet onp p₁) (?S p₁) (?U p₁)"
      by - (erule(1) oreachable_local, auto)
  moreover from 'I σ m' tor ttr
    have "(σ', t') ∈ oreachable (opnet onp p₂) (?S p₂) (?U p₂)"
      by - (erule(1) oreachable_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') ∈ oreachable (opnet onp p₁) (?S p₁) (?U p₁)"
      by - (erule(1) oreachable_local, auto)
  moreover from tor ttr
    have "(σ', t') ∈ oreachable (opnet onp p₂) (?S p₂) (?U p₂)"
      by - (erule(1) oreachable_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') ∈ oreachable (opnet onp p₁) (?S p₁) (?U p₁)"
      by - (erule(1) oreachable_local, auto)
  moreover from tor ttr
    have "(σ', t') ∈ oreachable (opnet onp p₂) (?S p₂) (?U p₂)"
      by - (erule(1) oreachable_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 p₂ → 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 ttr: "((σ, t), 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', s') ∈ oreachable (opnet onp p₁) (?S p₁) (?U p₁)"
  using sor 's' = s' by (clarsimp elim!: oreachable_other')
      (metis otherI '∧ξ. U ξ ξ')+

moreover from tor ttr
  have "(σ', t') ∈ oreachable (opnet onp p₂) (?S p₂) (?U p₂)"
    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 p₂ → 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), 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_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', s') ∈ oreachable (opnet onp p1) (?S p1) (?U p1)"
  using sor 's' = s' by (clarsimp elim!: oreachable_other')
  (metis otherI '∧ξ. U ξ ξ'+)
```

```
moreover from tor ttr
  have "(σ', t') ∈ oreachable (opnet onp p2) (?S p2) (?U p2)"
  by - (erule(1) oreachable_local, auto)
ultimately show ?thesis ..
```

```
qed
with 'net_tree_ips p1 ∩ net_tree_ips p2 = {}' show ?case by simp
qed
```

'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 "(σ, SubnetS s1 s2) ∈ oreachable (opnet onp (p1 || p2)) (λ_ _ . True) (λ_ _ . True)"
shows "(σ, s1) ∈ oreachable (opnet onp p1) (λ_ _ . True) (λ_ _ . True)"
  "(σ, s2) ∈ oreachable (opnet onp p2) (λ_ _ . True) (λ_ _ . True)"
  (is "_ ∈ ?oreachable p2")
```

using assms proof -

```
from assms have "(σ, s1) ∈ ?oreachable p1 ∧ (σ, s2) ∈ ?oreachable p2"
proof (induction rule: oreachable_par_subnet_induct)
```

```
  fix σ s1 s2
  assume "(σ, SubnetS s1 s2) ∈ init (opnet onp (p1 || p2))"
  thus "(σ, s1) ∈ ?oreachable p1 ∧ (σ, s2) ∈ ?oreachable p2"
  by (auto dest: oreachable_init)
```

next

```
case (local σ s1 s2 σ' s1' s2' a)
hence "(σ, SubnetS s1 s2) ∈ ?oreachable (p1 || p2)"
  and sr1: "(σ, s1) ∈ ?oreachable p1"
  and sr2: "(σ, s2) ∈ ?oreachable p2"
  and "((σ, SubnetS s1 s2), a, (σ', SubnetS s1' s2')) ∈ trans (opnet onp (p1 || p2))" by auto
from this(4)
  have "((σ, SubnetS s1 s2), a, (σ', SubnetS s1' s2'))
    ∈ opnet_sos (trans (opnet onp p1)) (trans (opnet onp p2))" by simp
thus "(σ', s1') ∈ ?oreachable p1 ∧ (σ', s2') ∈ ?oreachable p2"
```

proof cases

```
  fix R m H K
  assume "a = R:*cast(m)"
  and tr1: "((σ, s1), R:*cast(m), (σ', s1')) ∈ trans (opnet onp p1)"
  and tr2: "((σ, s2), H¬K:arrive(m), (σ', s2')) ∈ trans (opnet onp p2)"
  from sr1 and tr1 and TrueI have "(σ', s1') ∈ ?oreachable p1"
  by (rule oreachable_local')
  moreover from sr2 and tr2 and TrueI have "(σ', s2') ∈ ?oreachable p2"
  by (rule oreachable_local')
  ultimately show ?thesis ..
```

next

```
  assume "a = τ"
  and "s2' = s2"
  and tr1: "((σ, s1), τ, (σ', s1')) ∈ trans (opnet onp p1)"
  from sr2 and this(2) have "(σ', s2') ∈ ?oreachable p2" by auto
  moreover have "(λ_ _ . True) σ σ'" by (rule TrueI)
  ultimately have "(σ', s2') ∈ ?oreachable p2"
  by (rule oreachable_other')
  moreover from sr1 and tr1 and TrueI have "(σ', s1') ∈ ?oreachable p1"
  by (rule oreachable_local')
qed (insert sr1 sr2, simp_all, (metis (no_types) oreachable_local'
  oreachable_other'+))
```

qed auto

```
thus "(σ, s1) ∈ ?oreachable p1"
  "(σ, s2) ∈ ?oreachable p2" by auto
```

qed

It may also be tempting to try splitting from the assumption $(\sigma, \text{SubnetS } s_1 \ s_2) \in \text{oreachable } (\text{opnet onp } (p_1$

$\parallel p_2$) $(\lambda_ _ _. \text{True}) (\lambda_ _ _. \text{False})$, where the environment step would be trivially true (since the assumption is false), but the lemma cannot be shown when only one side acts, since it must guarantee the assumption for the other side.

lemma lift_opnet_sync_action:

assumes " $\bigwedge \xi. U \xi \xi$ "

and act1: " $\bigwedge i R. \langle i : \text{onp } i : R \rangle_o \models_A (\lambda \sigma _ . \text{oarrivemsg } I \sigma, \text{other } U \{i\} \rightarrow \text{globala } (\lambda(\sigma, a, _). \text{castmsg } (I \sigma) a))$ "

and act2: " $\bigwedge i R. \langle i : \text{onp } i : R \rangle_o \models_A (\lambda \sigma _ . \text{oarrivemsg } I \sigma, \text{other } U \{i\} \rightarrow \text{globala } (\lambda(\sigma, a, \sigma'). (a \neq \tau \wedge (\forall i d. a \neq i:\text{deliver}(d)) \rightarrow S (\sigma i) (\sigma' i))))$ "

and act3: " $\bigwedge i R. \langle i : \text{onp } i : R \rangle_o \models_A (\lambda \sigma _ . \text{oarrivemsg } I \sigma, \text{other } U \{i\} \rightarrow \text{globala } (\lambda(\sigma, a, \sigma'). (a = \tau \vee (\exists d. a = i:\text{deliver}(d)) \rightarrow U (\sigma i) (\sigma' i))))$ "

shows " $\text{opnet onp } p \models_A (\lambda \sigma _ . \text{oarrivemsg } I \sigma, \text{other } U (\text{net_tree_ips } p) \rightarrow \text{globala } (\lambda(\sigma, a, \sigma'). \text{castmsg } (I \sigma) a$

$\wedge (a \neq \tau \wedge (\forall i d. a \neq i:\text{deliver}(d)) \rightarrow (\forall i \in \text{net_tree_ips } p. S (\sigma i) (\sigma' i)))$

$\wedge (a = \tau \vee (\exists i d. a = i:\text{deliver}(d)) \rightarrow ((\forall i \in \text{net_tree_ips } p. U (\sigma i) (\sigma' i))$

$\wedge (\forall i. i \notin \text{net_tree_ips } p \rightarrow \sigma' i = \sigma i))))$ "

(is " $\text{opnet onp } p \models_A (?I, ?U p \rightarrow) ?\text{inv } (\text{net_tree_ips } p)$ ")

proof (induction p)

fix i R

show " $\text{opnet onp } \langle i; R \rangle \models_A (?I, ?U \langle i; R \rangle \rightarrow) ?\text{inv } (\text{net_tree_ips } \langle i; R \rangle)$ "

proof (rule ostep_invariantI, simp only: opnet.simps net_tree_ips.simps)

fix $\sigma s a \sigma' s'$

assume sor: " $(\sigma, s) \in \text{oreachable } (\langle i : \text{onp } i : R \rangle_o) (\lambda \sigma _ . \text{oarrivemsg } I \sigma) (\text{other } U \{i\})$ "

and str: " $((\sigma, s), a, (\sigma', s')) \in \text{trans } (\langle i : \text{onp } i : R \rangle_o)$ "

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 " $a \neq \tau \wedge (\forall i d. a \neq i:\text{deliver}(d)) \rightarrow S (\sigma i) (\sigma' i)$ "

by - (drule(2) ostep_invariantD [OF act2], simp)

moreover have " $a = \tau \vee (\exists i d. a = i:\text{deliver}(d)) \rightarrow U (\sigma i) (\sigma' i)$ "

proof -

from sor str oam have " $a = \tau \vee (\exists d. a = i:\text{deliver}(d)) \rightarrow U (\sigma i) (\sigma' i)$ "

by - (drule(2) ostep_invariantD [OF act3], simp)

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)

ultimately show ?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 " $a = \tau \vee (\exists i d. a = i:\text{deliver}(d)) \rightarrow (\forall j. j \neq i \rightarrow \sigma' j = \sigma j)$ "

by - (drule(2) ostep_invariantD [OF node_tau_deliver_unchanged], simp)

ultimately show " $?\text{inv } \{i\} ((\sigma, s), a, (\sigma', s'))$ " by simp

qed

next

fix $p_1 p_2$

assume inv1: " $\text{opnet onp } p_1 \models_A (?I, ?U p_1 \rightarrow) ?\text{inv } (\text{net_tree_ips } p_1)$ "

and inv2: " $\text{opnet onp } p_2 \models_A (?I, ?U p_2 \rightarrow) ?\text{inv } (\text{net_tree_ips } p_2)$ "

show " $\text{opnet onp } (p_1 \parallel p_2) \models_A (?I, ?U (p_1 \parallel p_2) \rightarrow) ?\text{inv } (\text{net_tree_ips } (p_1 \parallel p_2))$ "

proof (rule ostep_invariantI)

fix $\sigma st a \sigma' st'$

assume " $(\sigma, st) \in \text{oreachable } (\text{opnet onp } (p_1 \parallel p_2)) ?I (?U (p_1 \parallel p_2))$ "

and " $((\sigma, st), a, (\sigma', st')) \in \text{trans } (\text{opnet onp } (p_1 \parallel p_2))$ "

and " $\text{oarrivemsg } I \sigma a$ "

from this(1) obtain s t

where " $st = \text{SubnetS } s t$ "

and *: " $(\sigma, \text{SubnetS } s t) \in \text{oreachable } (\text{opnet onp } (p_1 \parallel p_2)) ?I (?U (p_1 \parallel p_2))$ "

by - (frule net_par_oreachable_is_subnet, metis)

from this(2) and inv1 and inv2

obtain sor: " $(\sigma, s) \in \text{oreachable } (\text{opnet onp } p_1) ?I (?U p_1)$ "

```

    and tor: "(σ, t) ∈ oreachable (opnet onp p2) ?I (?U p2)"
    and "net_tree_ips p1 ∩ net_tree_ips p2 = {}"
    by - (drule opnet_sync_action_subnet_oreachable [OF _ '∧ξ. U ξ ξ'], auto)

from * and '(σ, st), a, (σ', st') ∈ trans (opnet onp (p1 || p2))' and 'st = SubnetS s t'
  obtain s' t' where "st' = SubnetS s' t'"
    and "(σ, SubnetS s t), a, (σ', SubnetS s' t')
      ∈ opnet_sos (trans (opnet onp p1)) (trans (opnet onp p2))"
    by clarsimp (frule opartial_net_preserves_subnets, metis)

from this(2)
  have "castmsg (I σ) a
    ∧ (a ≠ τ ∧ (∀i d. a ≠ i:deliver(d)) → (∀i ∈ net_tree_ips (p1 || p2). S (σ i) (σ' i)))
    ∧ (a = τ ∨ (∃i d. a = i:deliver(d)) → (∀i ∈ net_tree_ips (p1 || p2). U (σ i) (σ' i)))
    ∧ (∀i. i ∉ net_tree_ips (p1 || p2) → σ' i = σ i)"

proof cases
  fix R m H K
  assume "a = R:*cast(m)"
    and str: "(σ, s), R:*cast(m), (σ', s') ∈ trans (opnet onp p1)"
    and ttr: "(σ, t), H¬K:arrive(m), (σ', t') ∈ trans (opnet onp p2)"
  from sor and str have "I σ m ∧ (∀i ∈ net_tree_ips p1. S (σ i) (σ' i))"
    by (auto dest: ostep_invariantD [OF inv1])
  moreover with tor and ttr have "∀i ∈ net_tree_ips p2. S (σ i) (σ' i)"
    by (auto dest: ostep_invariantD [OF inv2])
  ultimately show ?thesis
    using 'a = R:*cast(m)' by auto
next
  fix R m H K
  assume "a = R:*cast(m)"
    and str: "(σ, s), H¬K:arrive(m), (σ', s') ∈ trans (opnet onp p1)"
    and ttr: "(σ, t), R:*cast(m), (σ', t') ∈ trans (opnet onp p2)"
  from tor and ttr have "I σ m ∧ (∀i ∈ net_tree_ips p2. S (σ i) (σ' i))"
    by (auto dest: ostep_invariantD [OF inv2])
  moreover with sor and str have "∀i ∈ net_tree_ips p1. S (σ i) (σ' i)"
    by (auto dest: ostep_invariantD [OF inv1])
  ultimately show ?thesis
    using 'a = R:*cast(m)' by auto
next
  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 and str have "∀i ∈ net_tree_ips p1. S (σ i) (σ' i)"
    by (auto dest: ostep_invariantD [OF inv1])
  moreover from tor and ttr and 'I σ m' have "∀i ∈ net_tree_ips p2. S (σ i) (σ' i)"
    by (auto dest: ostep_invariantD [OF inv2])
  ultimately show ?thesis
    using 'a = (H ∪ H')¬(K ∪ K'):arrive(m)' by auto
next
  fix i d
  assume "a = i:deliver(d)"
    and str: "(σ, s), i:deliver(d), (σ', s') ∈ trans (opnet onp p1)"
  with sor have "(∀i ∈ net_tree_ips p1. U (σ i) (σ' i))
    ∧ (∀i. i ∉ net_tree_ips p1 → σ' i = σ i)"
    by (auto dest!: ostep_invariantD [OF inv1])
  with 'a = i:deliver(d)' and '∧ξ. U ξ ξ' show ?thesis
    by auto
next
  fix i d
  assume "a = i:deliver(d)"
    and ttr: "(σ, t), i:deliver(d), (σ', t') ∈ trans (opnet onp p2)"
  with tor have "(∀i ∈ net_tree_ips p2. U (σ i) (σ' i))
    ∧ (∀i. i ∉ net_tree_ips p2 → σ' i = σ i)"

```

```

    by (auto dest!: ostep_invariantD [OF inv2])
  with 'a = i:deliver(d)' and '∧ξ. U ξ ξ' show ?thesis
  by auto
next
  assume "a = τ"
  and str: "((σ, s), τ, (σ', s')) ∈ trans (opnet onp p₁)"
  with sor have "((∀i∈net_tree_ips p₁. U (σ i) (σ' i))
    ∧ (∀i. i∉net_tree_ips p₁ → σ' i = σ i))"
  by (auto dest!: ostep_invariantD [OF inv1])
  with 'a = τ' and '∧ξ. U ξ ξ' show ?thesis
  by auto
next
  assume "a = τ"
  and ttr: "((σ, t), τ, (σ', t')) ∈ trans (opnet onp p₂)"
  with tor have "((∀i∈net_tree_ips p₂. U (σ i) (σ' i))
    ∧ (∀i. i∉net_tree_ips p₂ → σ' i = σ i))"
  by (auto dest!: ostep_invariantD [OF inv2])
  with 'a = τ' and '∧ξ. U ξ ξ' show ?thesis
  by auto
next
  fix i i'
  assume "a = connect(i, i'"
  and 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)"
  from sor and str have "∀i∈net_tree_ips p₁. S (σ i) (σ' i)"
  by (auto dest: ostep_invariantD [OF inv1])
  moreover from tor and ttr have "∀i∈net_tree_ips p₂. S (σ i) (σ' 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: "((σ, s), disconnect(i, i'), (σ', s')) ∈ trans (opnet onp p₁)"
  and ttr: "((σ, t), disconnect(i, i'), (σ', t')) ∈ trans (opnet onp p₂)"
  from sor and str have "∀i∈net_tree_ips p₁. S (σ i) (σ' i)"
  by (auto dest: ostep_invariantD [OF inv1])
  moreover from tor and ttr have "∀i∈net_tree_ips p₂. S (σ i) (σ' i)"
  by (auto dest: ostep_invariantD [OF inv2])
  ultimately show ?thesis
  using 'a = disconnect(i, i')' by auto
qed
thus "?inv (net_tree_ips (p₁ || p₂)) ((σ, st), a, (σ', st'))" by simp
qed
qed

```

theorem subnet_oreachable:

```

  assumes "(σ, SubnetS s t) ∈ oreachable (opnet onp (p₁ || p₂))
    (otherwith S (net_tree_ips (p₁ || p₂)) (oarrivmsg I))
    (other U (net_tree_ips (p₁ || p₂)))"
  (is "_ ∈ oreachable _ (?S (p₁ || p₂)) (?U (p₁ || p₂))")

```

```

  and "∧ξ. S ξ ξ"
  and "∧ξ. U ξ ξ"

```

```

  and node1: "∧i R. ⟨i : onp i : R⟩ₒ ⊨ₐ (λσ -. oarrivmsg I σ, other U {i} →
    globala (λ(σ, a, _). castmsg (I σ) a))"
  and node2: "∧i R. ⟨i : onp i : R⟩ₒ ⊨ₐ (λσ -. oarrivmsg I σ, other U {i} →
    globala (λ(σ, a, σ'). (a ≠ τ ∧ (∀i d. a ≠ i:deliver(d)) → S (σ i) (σ' i)))"
  and node3: "∧i R. ⟨i : onp i : R⟩ₒ ⊨ₐ (λσ -. oarrivmsg I σ, other U {i} →
    globala (λ(σ, a, σ'). (a = τ ∨ (∃d. a = i:deliver(d)) → U (σ i) (σ' i)))"

```

```

  shows "(σ, s) ∈ oreachable (opnet onp p₁)
    (otherwith S (net_tree_ips p₁) (oarrivmsg I))"

```



```

      (other U (net_tree_ips p1))
    ∧ (σ, t) ∈ oreachable (opnet onp p2)
      (otherwith S (net_tree_ips p2) (oarrivemsg I))
      (other U (net_tree_ips p2))
    ∧ net_tree_ips p1 ∩ net_tree_ips p2 = {}"
using assms(1) proof (induction rule: oreachable_par_subnet_induct)
  case (init σ s t)
  hence sinit: "(σ, s) ∈ init (opnet onp p1)"
  and tinit: "(σ, t) ∈ init (opnet onp p2)"
  and "net_ips s ∩ net_ips t = {}" by auto
  moreover from sinit have "net_ips s = net_tree_ips p1"
  by (rule opnet_net_ips_net_tree_ips_init)
  moreover from tinit have "net_ips t = net_tree_ips p2"
  by (rule opnet_net_ips_net_tree_ips_init)
  ultimately show ?case by (auto elim: oreachable_init)
next
  case (other σ s t σ')
  hence "other U (net_tree_ips (p1 || p2)) σ σ'"
  and IHs: "(σ, s) ∈ oreachable (opnet onp p1) (?S p1) (?U p1)"
  and Iht: "(σ, t) ∈ oreachable (opnet onp p2) (?S p2) (?U p2)"
  and "net_tree_ips p1 ∩ net_tree_ips p2 = {}" by auto

  have "(σ', s) ∈ oreachable (opnet onp p1) (?S p1) (?U p1)"
  proof -
    from ' (?U (p1 || p2)) σ σ'' and '∧ξ. U ξ ξ' have "?U p1 σ σ'"
    by (rule other_net_tree_ips_par_left)
    with IHs show ?thesis by - (erule(1) oreachable_other')
  qed

  moreover have "(σ', t) ∈ oreachable (opnet onp p2) (?S p2) (?U p2)"
  proof -
    from ' (?U (p1 || p2)) σ σ'' and '∧ξ. U ξ ξ' have "?U p2 σ σ'"
    by (rule other_net_tree_ips_par_right)
    with Iht show ?thesis by - (erule(1) oreachable_other')
  qed

  ultimately show ?case using 'net_tree_ips p1 ∩ net_tree_ips p2 = {}' by simp
next
  case (local σ s t σ' s' t' a)
  hence stor: "(σ, SubnetS s t) ∈ oreachable (opnet onp (p1 || p2)) (?S (p1 || p2)) (?U (p1 || p2))"
  and tr: "((σ, SubnetS s t), a, (σ', SubnetS s' t')) ∈ trans (opnet onp (p1 || p2))"
  and "?S (p1 || p2) σ σ' a"
  and sor: "(σ, s) ∈ oreachable (opnet onp p1) (?S p1) (?U p1)"
  and tor: "(σ, t) ∈ oreachable (opnet onp p2) (?S p2) (?U p2)"
  and "net_tree_ips p1 ∩ net_tree_ips p2 = {}" by auto

  have act: "∧p. opnet onp p ⊨A (λσ _. oarrivemsg I σ, other U (net_tree_ips p) →)
    globala (λ(σ, a, σ'). castmsg (I σ) a
      ∧ (a ≠ τ ∧ (∀i d. a ≠ i:deliver(d) →
        (∀i∈net_tree_ips p. S (σ i) (σ' i)))
      ∧ (a = τ ∨ (∃i d. a = i:deliver(d) →
        ((∀i∈net_tree_ips p. U (σ i) (σ' i))
          ∧ (∀i. i∉net_tree_ips p → σ' i = σ i))))))"
  by (rule lift_opnet_sync_action [OF assms(3-6)])

  from ' (?S (p1 || p2) σ σ' a' have "∀j. j ∉ net_tree_ips (p1 || p2) → S (σ j) (σ' j)"
  and "oarrivemsg I σ a"
  by (auto elim!: otherwithE)
  from tr have "((σ, SubnetS s t), a, (σ', SubnetS s' t'))
    ∈ opnet_sos (trans (opnet onp p1)) (trans (opnet onp p2))" by simp
  hence "(σ', s') ∈ oreachable (opnet onp p1) (?S p1) (?U p1)
    ∧ (σ', t') ∈ oreachable (opnet onp p2) (?S p2) (?U p2)"
  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 '∑S (p₁ || p₂) σ σ' a' have "I σ m" by auto

with sor str have "∀i∈net_tree_ips p₁. S (σ i) (σ' i)"
  by - (drule(1) ostep_arrive_invariantD [OF act], simp_all)
moreover from 'I σ m' tor ttr have "∀i∈net_tree_ips p₂. S (σ i) (σ' i)"
  by - (drule(1) ostep_arrive_invariantD [OF act], simp_all)
ultimately have "∀i. S (σ i) (σ' i)"
  using '∑j. j ∉ net_tree_ips (p₁ || p₂) → S (σ j) (σ' j)' by auto

with 'I σ m' sor str
  have "(σ', s') ∈ oreachable (opnet onp p₁) (?S p₁) (?U p₁)"
  by - (erule(1) oreachable_local, auto)
moreover from '∀i. S (σ i) (σ' i)' 'I σ m' tor ttr
  have "(σ', t') ∈ oreachable (opnet onp p₂) (?S p₂) (?U p₂)"
  by - (erule(1) oreachable_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), H¬K:arrive(m), (σ', t')) ∈ trans (opnet onp p₂)"
from sor str have "I σ m"
  by - (drule(1) ostep_arrive_invariantD [OF act], simp_all)
with sor str tor ttr have "∀i. S (σ i) (σ' i)"
  using '∑j. j ∉ net_tree_ips (p₁ || p₂) → S (σ j) (σ' j)'
  by (fastforce dest!: ostep_arrive_invariantD [OF act] ostep_arrive_invariantD [OF act])
with 'I σ m' sor str
  have "(σ', s') ∈ oreachable (opnet onp p₁) (?S p₁) (?U p₁)"
  by - (erule(1) oreachable_local, auto)
moreover from '∀i. S (σ i) (σ' i)' 'I σ m' tor ttr
  have "(σ', t') ∈ oreachable (opnet onp p₂) (?S p₂) (?U p₂)"
  by - (erule(1) oreachable_local, auto)
ultimately show ?thesis ..

next
fix R m H K
assume str: "((σ, s), H¬K:arrive(m), (σ', s')) ∈ trans (opnet onp p₁)"
  and ttr: "((σ, t), R:*cast(m), (σ', t')) ∈ trans (opnet onp p₂)"
from tor ttr have "I σ m"
  by - (drule(1) ostep_arrive_invariantD [OF act], simp_all)
with sor str tor ttr have "∀i. S (σ i) (σ' i)"
  using '∑j. j ∉ net_tree_ips (p₁ || p₂) → S (σ j) (σ' j)'
  by (fastforce dest!: ostep_arrive_invariantD [OF act] ostep_arrive_invariantD [OF act])
with 'I σ m' sor str
  have "(σ', s') ∈ oreachable (opnet onp p₁) (?S p₁) (?U p₁)"
  by - (erule(1) oreachable_local, auto)
moreover from '∀i. S (σ i) (σ' i)' 'I σ m' tor ttr
  have "(σ', t') ∈ oreachable (opnet onp p₂) (?S p₂) (?U p₂)"
  by - (erule(1) oreachable_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 tor have "∀i. S (σ i) (σ' i)"
  using '∑j. j ∉ net_tree_ips (p₁ || p₂) → S (σ j) (σ' j)'
  by (fastforce dest!: ostep_arrive_invariantD [OF act] ostep_arrive_invariantD [OF act])
with sor str
  have "(σ', s') ∈ oreachable (opnet onp p₁) (?S p₁) (?U p₁)"
  by - (erule(1) oreachable_local, auto)
moreover from '∀i. S (σ i) (σ' i)' tor ttr
  have "(σ', t') ∈ oreachable (opnet onp p₂) (?S p₂) (?U p₂)"
  by - (erule(1) oreachable_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 tor have "∀i. S (σ i) (σ' i)"
using '∀j. j ∉ net_tree_ips (p₁ || p₂) → S (σ j) (σ' j)'
by (fastforce dest!: ostep_arrive_invariantD [OF act] ostep_arrive_invariantD [OF act])
with sor str
have "(σ', s') ∈ oreachable (opnet onp p₁) (?S p₁) (?U p₁)"
by - (erule(1) oreachable_local, auto)
moreover from '∀i. S (σ i) (σ' i)' tor ttr
have "(σ', t') ∈ oreachable (opnet onp p₂) (?S p₂) (?U p₂)"
by - (erule(1) oreachable_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_arrive_invariantD [OF act], simp_all)
hence "∀j. j ∉ net_tree_ips p₁ → S (σ j) (σ' j)"
by (auto intro: '∧ξ. S ξ ξ')
with sor 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 sor 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 p₁ → σ' j = σ j'
by (clarsimp elim!: oreachable_other')
(metis otherI '∧ξ. U ξ ξ')+

qed
ultimately show ?thesis ..
next
fix i d
assume "s' = s"
and ttr: "((σ, t), 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_arrive_invariantD [OF act], simp_all)
hence "∀j. j ∉ net_tree_ips p₂ → S (σ j) (σ' j)"
by (auto intro: '∧ξ. 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 '∧ξ. U ξ ξ')+

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: '∧ξ. 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 '∧ξ. U ξ ξ'+)
qed
ultimately show ?thesis by - (rule conjI)
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_arrive_invariantD [OF act], simp_all)
hence "∀j. j ∉ net_tree_ips p₁ → S (σ j) (σ' j)"
  by (auto intro: '∧ξ. S ξ ξ')
with sor 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 sor 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 p₁ → σ' j = σ j'
  by (clarsimp elim!: oreachable_other')
  (metis otherI '∧ξ. U ξ ξ'+)
qed
ultimately show ?thesis ..
qed
with 'net_tree_ips p₁ ∩ net_tree_ips p₂ = {}' show ?case by simp
qed

```

```

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 "∧ii Ri. ⟨ii : onp ii : Ri⟩o
  ⊨ (otherwith S {ii} (oarrivemsg I), other U {ii} →) global (P ii)"

```

```

and "∧ξ. S ξ ξ"
and "∧ξ. U ξ ξ"

```

```

and node1: "∧i R. ⟨i : onp i : R⟩o ⊨A (λσ -. oarrivemsg I σ, other U {i} →)
  globala (λ(σ, a, _). castmsg (I σ) a)"
and node2: "∧i R. ⟨i : onp i : R⟩o ⊨A (λσ -. oarrivemsg I σ, other U {i} →)

```

```

      globala (λ(σ, a, σ'). (a ≠ τ ∧ (∀i d. a ≠ i:deliver(d)) → S (σ i) (σ' i)))"
and node3: "∧i R. ⟨i : onp i : R⟩o ⊨A (λσ -. oarrivemsg I σ, other U {i} →)
      globala (λ(σ, a, σ'). (a = τ ∨ (∃d. a = i:deliver(d)) → U (σ i) (σ' i)))"

shows "opnet onp p ⊨ (otherwith S (net_tree_ips p) (oarrivemsg I),
      other U (net_tree_ips p) →) global (λσ. ∀i∈net_tree_ips p. P i σ)"
  (is "_ ⊨ (?owS p, ?U p →) _")
proof (induction p)
  fix ii Ri
  from assms(1) show "opnet onp ⟨ii; Ri⟩ ⊨ (?owS ⟨ii; Ri⟩, ?U ⟨ii; Ri⟩ →)
      global (λσ. ∀i∈net_tree_ips ⟨ii; Ri⟩. P i σ)" by auto
next
  fix p1 p2
  assume ih1: "opnet onp p1 ⊨ (?owS p1, ?U p1 →) global (λσ. ∀i∈net_tree_ips p1. P i σ)"
  and ih2: "opnet onp p2 ⊨ (?owS p2, ?U p2 →) global (λσ. ∀i∈net_tree_ips p2. P i σ)"
  show "opnet onp (p1 || p2) ⊨ (?owS (p1 || p2), ?U (p1 || p2) →)
      global (λσ. ∀i∈net_tree_ips (p1 || p2). P i σ)"
  unfolding oinvariant_def
  proof
    fix pq
    assume "pq ∈ oreachable (opnet onp (p1 || p2)) (?owS (p1 || p2)) (?U (p1 || p2))"
    moreover then obtain σ s t where "pq = (σ, SubnetS s t)"
      by (metis net_par_oreachable_is_subnet surjective_pairing)
    ultimately have "(σ, SubnetS s t) ∈ oreachable (opnet onp (p1 || p2))
      (?owS (p1 || p2)) (?U (p1 || p2))" by simp
    then obtain sor: "(σ, s) ∈ oreachable (opnet onp p1) (?owS p1) (?U p1)"
      and tor: "(σ, t) ∈ oreachable (opnet onp p2) (?owS p2) (?U p2)"
      by - (drule subnet_oreachable [OF _ _ node1 node2 node3], auto intro: assms(2-3))
    from sor have "∀i∈net_tree_ips p1. P i σ"
      by (auto dest: oinvariantD [OF ih1])
    moreover from tor have "∀i∈net_tree_ips p2. P i σ"
      by (auto dest: oinvariantD [OF ih2])
    ultimately have "∀i∈net_tree_ips (p1 || p2). P i σ" by auto
    with 'pq = (σ, SubnetS s t)' show "global (λσ. ∀i∈net_tree_ips (p1 || p2). P i σ) pq" by simp
  qed
qed
end

```

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 "(σ, ζ) ∈ init (opnet np p)" by simp
thus "(σ, ζ) ∈ oreachable (opnet np p) ?owS U" ..
next
  fix σ ζ σ'
  assume "(σ, ζ) ∈ oreachable (opnet np p) ?owS U"
    and "U σ σ'"
  thus "(σ', ζ) ∈ oreachable (opnet np p) ?owS U"
    by - (rule oreachable_other')
next
  fix σ ζ σ' ζ' a
  assume zor: "(σ, ζ) ∈ oreachable (opnet np p) ?owS U"
    and ztr: "((σ, ζ), a, (σ', ζ')) ∈ trans (oclosed (opnet np p))"
  from this(1) have [simp]: "net_ips ζ = net_tree_ips p"
    by (rule opnet_net_ips_net_tree_ips)
  from ztr have "((σ, ζ), a, (σ', ζ')) ∈ ocnet_sos (trans (opnet np p))" by simp
  thus "(σ', ζ') ∈ oreachable (opnet np p) ?owS U"
  proof cases
    fix i K d di
    assume "a = i:newpkt(d, di)"
      and tr: "((σ, ζ), {i}¬K:arrive(msg_class.newpkt (d, di)), (σ', ζ')) ∈ trans (opnet np p)"
      and "∀j. j ∉ net_ips ζ → σ' j = σ j"
    from this(3) have "∀j. j ∉ net_tree_ips p → σ' j = σ j"
      using 'net_ips ζ = net_tree_ips p' by auto
    hence "otherwith (op=) (net_tree_ips p) inoclosed σ σ' ({i}¬K:arrive(msg_class.newpkt (d, di)))"
      by auto
    with zor tr show ?thesis
      by - (rule oreachable_local')
  next
    assume "a = τ"
      and tr: "((σ, ζ), τ, (σ', ζ')) ∈ trans (opnet np p)"
      and "∀j. j ∉ net_ips ζ → σ' j = σ j"
    from this(3) have "∀j. j ∉ net_tree_ips p → σ' j = σ j"
      using 'net_ips ζ = net_tree_ips p' by auto
    hence "otherwith (op=) (net_tree_ips p) inoclosed σ σ' τ"
      by auto
    with zor tr show ?thesis by - (rule oreachable_local')
  qed (insert 'net_ips ζ = net_tree_ips p',
    auto elim!: oreachable_local' [OF zor])
qed

```

```

lemma oclosed_oreachable_oreachable [elim]:
  assumes "(σ, ζ) ∈ oreachable (oclosed (opnet np p)) (λ_ _ . True) U"
  shows "(σ, ζ) ∈ oreachable (opnet np p) (λ_ _ . True) U"
  using assms by (rule oclosed_oreachable_inclosed [THEN oreachable_weakenE]) simp

```

```

lemma inclosed_closed [intro]:
  assumes cinv: "opnet np p ⊨ (otherwith (op=) (net_tree_ips p) inoclosed, U →) P"
  shows "oclosed (opnet np p) ⊨ (λ_ _ . True, U →) P"
  using assms unfolding oinvariant_def
  by (clarsimp dest!: oclosed_oreachable_inclosed)

```

end

19 Generic invariants on sequential AWN processes

```

theory AWN_Invariants
imports Invariants AWN_SOS AWN_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 Γ , meaning that its transitions, and those of its subterms, are subsumed by those of Γ .

definition

`control_within :: "('s, 'm, 'p, 'l) seqp_env \Rightarrow ('z \times ('s, 'm, 'p, 'l) seqp) set \Rightarrow bool"`

where

`"control_within Γ $\sigma \equiv \forall (\xi, p) \in \sigma. \exists pn. p \in \text{subterms } (\Gamma pn)"$`

lemma control_withinI [intro]:

assumes " $\bigwedge p. p \in \text{Range } \sigma \implies \exists pn. p \in \text{subterms } (\Gamma pn)"$

shows "`control_within Γ σ` "

using `assms unfolding control_within_def` by auto

lemma control_withinD [dest]:

assumes "`control_within Γ σ` "

and " $(\xi, p) \in \sigma$ "

shows " $\exists pn. p \in \text{subterms } (\Gamma pn)"$

using `assms unfolding control_within_def` by blast

lemma control_within_topI [intro]:

assumes " $\bigwedge p. p \in \text{Range } \sigma \implies \exists pn. p = \Gamma pn"$

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 " $\exists pn. p \in \text{subterms } (\Gamma pn)"$

and " $((\xi, p), a, (\xi', p')) \in \text{seqp_sos } \Gamma"$

shows " $\exists pn. p' \in \text{subterms } (\Gamma pn)"$

using `assms`

`proof (induct p)`

fix `p1 p2`

assume `IH1: " $\exists pn. p1 \in \text{subterms } (\Gamma pn) \implies$`

`(($\xi, p1$), a, (ξ', p')) \in seqp_sos $\Gamma \implies$`

`$\exists pn. p' \in \text{subterms } (\Gamma pn)"$`

and `IH2: " $\exists pn. p2 \in \text{subterms } (\Gamma pn) \implies$`

`(($\xi, p2$), a, (ξ', p')) \in seqp_sos $\Gamma \implies$`

`$\exists pn. p' \in \text{subterms } (\Gamma pn)"$`

and " $\exists pn. p1 \oplus p2 \in \text{subterms } (\Gamma pn)"$

and " $((\xi, p1 \oplus p2), a, (\xi', p')) \in \text{seqp_sos } \Gamma"$

from ' $\exists pn. p1 \oplus p2 \in \text{subterms } (\Gamma pn)$ ' obtain `pn`

where " $p1 \in \text{subterms } (\Gamma pn)"$

and " $p2 \in \text{subterms } (\Gamma pn)"$ by auto

from ' $((\xi, p1 \oplus p2), a, (\xi', p')) \in \text{seqp_sos } \Gamma$ '

have " $((\xi, p1), a, (\xi', p')) \in \text{seqp_sos } \Gamma$

$\vee ((\xi, p2), a, (\xi', p')) \in \text{seqp_sos } \Gamma"$ by auto

thus " $\exists pn. p' \in \text{subterms } (\Gamma pn)"$

`proof`

assume " $((\xi, p1), a, (\xi', p')) \in \text{seqp_sos } \Gamma"$

with ' $p1 \in \text{subterms } (\Gamma pn)$ ' show `?thesis` by (auto intro: `IH1`)

next

assume " $((\xi, p2), a, (\xi', p')) \in \text{seqp_sos } \Gamma"$

with ' $p2 \in \text{subterms } (\Gamma pn)$ ' 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 " $(\xi, p) \in \text{reachable A I}$ "

shows " $\exists pn. p \in \text{subterms } (\Gamma pn)"$

using `assms(4)`

`proof (induct rule: reachable_pair_induct)`

fix `ξ p`

assume " $(\xi, p) \in \text{init A}$ "

```

with 'control_within  $\Gamma$  (init A)' show " $\exists pn. p \in \text{subterms } (\Gamma pn)$ " ..
next
fix  $\xi p a \xi' p'$ 
assume " $(\xi, p) \in \text{reachable } A I$ "
and " $\exists pn. p \in \text{subterms } (\Gamma pn)$ "
and " $((\xi, p), a, (\xi', p')) \in \text{trans } A$ "
and " $I a$ "
moreover from this(3) and assms(3) have " $((\xi, p), a, (\xi', p')) \in \text{seqp\_sos } \Gamma$ " by simp
ultimately show " $\exists pn. p' \in \text{subterms } (\Gamma pn)$ "
using 'wellformed  $\Gamma$ '
by (auto elim: seqp_sos_subterms)
qed

```

definition

```

onl :: "('s, 'm, 'p, 'l) seqp_env
       $\Rightarrow$  ('z  $\times$  'l  $\Rightarrow$  bool)
       $\Rightarrow$  'z  $\times$  ('s, 'm, 'p, 'l) seqp
       $\Rightarrow$  bool"

```

where

```

"onl  $\Gamma P \equiv (\lambda(\xi, p). \forall l \in \text{labels } \Gamma p. P (\xi, l))"$ 

```

lemma onlI [intro]:

```

assumes " $\bigwedge l. l \in \text{labels } \Gamma p \implies P (\xi, l)$ "
shows " $\text{onl } \Gamma P (\xi, p)$ "
using assms unfolding onl_def by simp

```

lemmas onlI' [intro] = onlI [simplified atomize_ball]

lemma onlD [dest]:

```

assumes " $\text{onl } \Gamma P (\xi, p)$ "
shows " $\forall l \in \text{labels } \Gamma p. P (\xi, l)$ "
using assms unfolding onl_def by simp

```

lemma onl_invariantI [intro]:

```

assumes init: " $\bigwedge \xi p l. [(\xi, p) \in \text{init } A; l \in \text{labels } \Gamma p] \implies P (\xi, l)$ "
and step: " $\bigwedge \xi p a \xi' p' l'.
  [(\xi, p) \in \text{reachable } A I;
   \forall l \in \text{labels } \Gamma p. P (\xi, l);
   ((\xi, p), a, (\xi', p')) \in \text{trans } A;
   l' \in \text{labels } \Gamma p';
   I a] \implies P (\xi', l')$ "

```

```

shows " $A \models (I \rightarrow) \text{onl } \Gamma P$ "

```

proof (rule invariant_pairI)

```

fix  $\xi p$ 
assume " $(\xi, p) \in \text{init } A$ "
hence " $\forall l \in \text{labels } \Gamma p. P (\xi, l)$ " using init by simp
thus " $\text{onl } \Gamma P (\xi, p)$ " ..

```

next

```

fix  $\xi p a \xi' p'$ 
assume rp: " $(\xi, p) \in \text{reachable } A I$ "
and "onl  $\Gamma P (\xi, p)$ "
and tr: " $((\xi, p), a, (\xi', p')) \in \text{trans } A$ "
and " $I a$ "
from 'onl  $\Gamma P (\xi, p)$ ' have " $\forall l \in \text{labels } \Gamma p. P (\xi, l)$ " ..
with rp tr 'I a' have " $\forall l' \in \text{labels } \Gamma p'. P (\xi', l')$ " by (auto elim: step)
thus " $\text{onl } \Gamma P (\xi', p')$ " ..

```

qed

lemma onl_invariantD [dest]:

```

assumes " $A \models (I \rightarrow) \text{onl } \Gamma P$ "
and " $(\xi, p) \in \text{reachable } A I$ "
and " $l \in \text{labels } \Gamma p$ "
shows " $P (\xi, l)$ "
using assms unfolding onl_def by auto

```



```

lemma onl_invariant_initD [dest]:
  assumes invP: "A  $\models$  (I  $\rightarrow$ ) onl  $\Gamma$  P"
    and init: " $(\xi, p) \in \text{init } A$ "
    and pnl: " $l \in \text{labels } \Gamma p$ "
  shows "P ( $\xi, l$ )"
proof -
  from init have " $(\xi, p) \in \text{reachable } A \ I$ " ..
  with invP show ?thesis using pnl ..
qed

lemma onl_invariant_sterms:
  assumes wf: "wellformed  $\Gamma$ "
    and il: "A  $\models$  (I  $\rightarrow$ ) onl  $\Gamma$  P"
    and rp: " $(\xi, p) \in \text{reachable } A \ I$ "
    and "p'  $\in$  sterms  $\Gamma p$ "
    and "l  $\in$  labels  $\Gamma p$ "
  shows "P ( $\xi, l$ )"
proof -
  from wf 'p'  $\in$  sterms  $\Gamma p$  'l  $\in$  labels  $\Gamma p$ ' have "l  $\in$  labels  $\Gamma p$ "
  by (rule labels_sterms_labels)
  with il rp show "P ( $\xi, l$ )" ..
qed

lemma onl_invariant_sterms_weaken:
  assumes wf: "wellformed  $\Gamma$ "
    and il: "A  $\models$  (I  $\rightarrow$ ) onl  $\Gamma$  P"
    and rp: " $(\xi, p) \in \text{reachable } A \ I$ "
    and "p'  $\in$  sterms  $\Gamma p$ "
    and "l  $\in$  labels  $\Gamma p$ "
    and weaken: " $\bigwedge a. I' a \implies I a$ "
  shows "P ( $\xi, l$ )"
proof -
  from ' $(\xi, p) \in \text{reachable } A \ I$ ' have " $(\xi, p) \in \text{reachable } A \ I$ "
  by (rule reachable_weakenE)
  (erule weaken)
  with assms(1-2) show ?thesis using assms(4-5) by (rule onl_invariant_sterms)
qed

lemma onl_invariant_sterms_TT:
  assumes wf: "wellformed  $\Gamma$ "
    and il: "A  $\models$  onl  $\Gamma$  P"
    and rp: " $(\xi, p) \in \text{reachable } A \ I$ "
    and "p'  $\in$  sterms  $\Gamma p$ "
    and "l  $\in$  labels  $\Gamma p$ "
  shows "P ( $\xi, l$ )"
  using assms by (rule onl_invariant_sterms_weaken) simp

lemma trans_from_sterms:
  assumes " $((\xi, p), a, (\xi', q)) \in \text{seqp\_sos } \Gamma$ "
    and "wellformed  $\Gamma$ "
  shows " $\exists p' \in \text{sterms } \Gamma p. ((\xi, p'), a, (\xi', q)) \in \text{seqp\_sos } \Gamma$ "
  using assms by (induction p rule: sterms_pinduct [OF 'wellformed  $\Gamma$ ']) auto

lemma trans_from_sterms':
  assumes " $((\xi, p'), a, (\xi', q)) \in \text{seqp\_sos } \Gamma$ "
    and "wellformed  $\Gamma$ "
    and "p'  $\in$  sterms  $\Gamma p$ "
  shows " $((\xi, p), a, (\xi', q)) \in \text{seqp\_sos } \Gamma$ "
  using assms by (induction p rule: sterms_pinduct [OF 'wellformed  $\Gamma$ ']) auto

lemma trans_to_dterms:
  assumes " $((\xi, p), a, (\xi', q)) \in \text{seqp\_sos } \Gamma$ "
    and "wellformed  $\Gamma$ "

```

shows " $\forall r \in \text{sterms } \Gamma \ q. \ r \in \text{dterms } \Gamma \ p$ "
 using *assms* by (induction *q*) auto

theorem *cterm_includes_sterms_of_seq_reachable*:

assumes "wellformed Γ "
 and "control_within Γ (init *A*)"
 and "trans *A* = seqp_sos Γ "
 shows " $\bigcup (\text{sterms } \Gamma \ ' \ \text{snd} \ ' \ \text{reachable } A \ I) \subseteq \text{cterm } \Gamma$ "

proof

fix *qs*
 assume " $qs \in \bigcup (\text{sterms } \Gamma \ ' \ \text{snd} \ ' \ \text{reachable } A \ I)$ "
 then obtain ξ and *q* where *: " $(\xi, q) \in \text{reachable } A \ I$ "
 and **: " $qs \in \text{sterms } \Gamma \ q$ " by auto
 from * have " $\bigwedge x. x \in \text{sterms } \Gamma \ q \implies x \in \text{cterm } \Gamma$ "

proof (induction rule: *reachable_pair_induct*)

fix $\xi \ p \ q$
 assume " $(\xi, p) \in \text{init } A$ "
 and " $q \in \text{sterms } \Gamma \ p$ "
 from 'control_within Γ (init *A*)' and ' $(\xi, p) \in \text{init } A$ '
 obtain *pn* where " $p \in \text{subterms } (\Gamma \ pn)$ " by auto
 with 'wellformed Γ ' show " $q \in \text{cterm } \Gamma$ " using ' $q \in \text{sterms } \Gamma \ p$ '
 by (rule *subterms_sterms_in_cterm*)

next

fix *p* $\xi \ a \ \xi' \ q \ x$
 assume " $(\xi, p) \in \text{reachable } A \ I$ "
 and *IH*: " $\bigwedge x. x \in \text{sterms } \Gamma \ p \implies x \in \text{cterm } \Gamma$ "
 and " $((\xi, p), a, (\xi', q)) \in \text{trans } A$ "
 and " $x \in \text{sterms } \Gamma \ q$ "
 from *this*(3) and 'trans *A* = seqp_sos Γ ' have " $((\xi, p), a, (\xi', q)) \in \text{seqp_sos } \Gamma$ " by *simp*
 from *this* and 'wellformed Γ ' obtain *ps*
 where *ps*: " $ps \in \text{sterms } \Gamma \ p$ "
 and *step*: " $((\xi, ps), a, (\xi', q)) \in \text{seqp_sos } \Gamma$ "
 by (rule *trans_from_sterms [THEN bexE]*)
 from *ps* have " $ps \in \text{cterm } \Gamma$ " by (rule *IH*)
 moreover from *step* 'wellformed Γ ' ' $x \in \text{sterms } \Gamma \ q$ ' have " $x \in \text{dterm } \Gamma \ ps$ "
 by (rule *trans_to_dterms [rule_format]*)
 ultimately show " $x \in \text{cterm } \Gamma$ " by (rule *ctermDI*)

qed

thus " $qs \in \text{cterm } \Gamma$ " using ** .

qed

corollary *seq_reachable_in_cterm*:

assumes "wellformed Γ "
 and "control_within Γ (init *A*)"
 and "trans *A* = seqp_sos Γ "
 and " $(\xi, p) \in \text{reachable } A \ I$ "
 and " $p' \in \text{sterms } \Gamma \ p$ "
 shows " $p' \in \text{cterm } \Gamma$ "

using *assms*(1-3)

proof (rule *cterm_includes_sterms_of_seq_reachable [THEN set_mp]*)
 from *assms*(4-5) show " $p' \in \bigcup (\text{sterms } \Gamma \ ' \ \text{snd} \ ' \ \text{reachable } A \ I)$ "
 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*: " $\bigwedge \xi \ p \ l. \ [\ (\xi, p) \in \text{init } A; \ l \in \text{labels } \Gamma \ p \] \implies P \ (\xi, l)$ "
 and *step*: " $\bigwedge p \ l \ \xi \ a \ q \ l' \ \xi' \ pp. \ [\ "$

```

    p∈cterm Γ;
    l∈labels Γ p;
    P (ξ, l);
    ((ξ, p), a, (ξ', q)) ∈ seqp_sos Γ;
    ((ξ, p), a, (ξ', q)) ∈ trans A;
    l'∈labels Γ q;
    (ξ, pp)∈reachable A I;
    p∈sterms Γ pp;
    (ξ', q)∈reachable A I;
    I a
  ] ⇒ P (ξ', l')"
shows "A ⊨ (I →) onl Γ P"
proof
  fix ξ p l
  assume "(ξ, p) ∈ init A"
  and *: "l ∈ labels Γ p"
  with init show "P (ξ, l)" by auto
next
  fix ξ p a ξ' q l'
  assume sr: "(ξ, p) ∈ reachable A I"
  and pl: "∀l∈labels Γ p. P (ξ, l)"
  and tr: "((ξ, p), a, (ξ', q)) ∈ trans A"
  and A6: "l' ∈ labels Γ q"
  and "I a"
  from this(3) and 'trans A = seqp_sos Γ' have tr': "((ξ, p), a, (ξ', q)) ∈ seqp_sos Γ" by simp
  show "P (ξ', l')"
proof -
  from sr and tr and 'I a' have A7: "(ξ', q) ∈ reachable A I" ..
  from tr' obtain p' where "p' ∈ sterms Γ p"
  and "((ξ, p'), a, (ξ', q)) ∈ seqp_sos Γ"
  by (blast dest: trans_from_sterms [OF _ wf])
  from wf cw sp sr this(1) have A1: "p'∈cterm Γ"
  by (rule seq_reachable_in_cterm)
  from labels_not_empty [OF wf] obtain l1 where A2: "l1∈labels Γ p'"
  by blast
  with 'p'∈sterms Γ p' have "l1∈labels Γ p"
  by (rule labels_sterms_labels [OF wf])
  with pl have A3: "P (ξ, l1)" by simp
  from '((ξ, p'), a, (ξ', q)) ∈ seqp_sos Γ' and sp
  have A5: "((ξ, p'), a, (ξ', q)) ∈ trans A" by simp
  with sp have A4: "((ξ, p'), a, (ξ', q)) ∈ seqp_sos Γ" by simp
  from sr 'p'∈sterms Γ p'
  obtain pp where A7: "(ξ, pp)∈reachable A I"
  and A8: "p'∈sterms Γ pp"
  by auto
  from sr tr 'I a' have A9: "(ξ', q) ∈ reachable A I" ..
  from A1 A2 A3 A4 A5 A6 A7 A8 A9 'I a' show ?thesis by (rule step)
qed
qed

```

```

lemma seq_invariant_ctermI:
  assumes wf: "wellformed Γ"
  and "control_within Γ (init A)"
  and "simple_labels Γ"
  and "trans A = seqp_sos Γ"
  and init: "∧ξ p l. [
    (ξ, p) ∈ init A;
    l∈labels Γ p
  ] ⇒ P (ξ, l)"
  and step: "∧p l ξ a q l' ξ' pp pn. [
    wellformed Γ;
    p∈cterm1 (Γ pn);
    not_call p;
    l∈labels Γ p;
  ] ⇒ P (ξ, l)"

```

```

      P (ξ, l);
      ((ξ, p), a, (ξ', q)) ∈ seqp_sos Γ;
      ((ξ, p), a, (ξ', q)) ∈ trans A;
      l' ∈ labels Γ q;
      (ξ, pp) ∈ reachable A I;
      p ∈ sterms Γ pp;
      (ξ', q) ∈ reachable A I;
      I a
    ] ⇒ P (ξ', l')"
  shows "A ⊨ (I →) onl Γ P"
using assms(1-4) proof (rule seq_invariant_ctermI)
  fix ξ p l
  assume "(ξ, p) ∈ init A"
    and "l ∈ labels Γ p"
  thus "P (ξ, l)" by (rule init)
next
  fix p l ξ a q l' ξ' pp
  assume "p ∈ cterms Γ"
    and otherassms: "l ∈ labels Γ p"
    "P (ξ, l)"
    "((ξ, p), a, (ξ', q)) ∈ seqp_sos Γ"
    "((ξ, p), a, (ξ', q)) ∈ trans A"
    "l' ∈ labels Γ q"
    "(ξ, pp) ∈ reachable A I"
    "p ∈ sterms Γ pp"
    "(ξ', q) ∈ reachable A I"
    "I a"
  from this(1) obtain pn where "p ∈ ctermst1(Γ pn)"
    and "not_call p"
  unfolding cterms_def' [OF wf] by auto
  with wf show "P (ξ', l')"
  using otherassms by (rule step)
qed

```

19.2 Step invariants via labelled control terms

definition

```

onll :: "('s, 'm, 'p, 'l) seqp_env
  ⇒ (('z × 'l, 'a) transition ⇒ bool)
  ⇒ ('z × ('s, 'm, 'p, 'l) seqp, 'a) transition ⇒ bool"

```

where

```

"onll Γ P ≡ (λ((ξ, p), a, (ξ', p')). ∀ l ∈ labels Γ p. ∀ l' ∈ labels Γ p'. P ((ξ, l), a, (ξ', l')))"

```

lemma onllI [intro]:

```

assumes "∧ l l'. [ l ∈ labels Γ p; l' ∈ labels Γ p' ] ⇒ P ((ξ, l), a, (ξ', l'))"
  shows "onll Γ P ((ξ, p), a, (ξ', p'))"
using assms unfolding onll_def by simp

```

lemma onllII [intro]:

```

assumes "∀ l ∈ labels Γ p. ∀ l' ∈ labels Γ p'. P ((ξ, l), a, (ξ', l'))"
  shows "onll Γ P ((ξ, p), a, (ξ', p'))"
using assms by auto

```

lemma onllD [dest]:

```

assumes "onll Γ P ((ξ, p), a, (ξ', p'))"
  shows "∀ l ∈ labels Γ p. ∀ l' ∈ labels Γ p'. P ((ξ, l), a, (ξ', l'))"
using assms unfolding onll_def by simp

```

lemma onl_weaken [elim!]: "∧ Γ P Q s. [onl Γ P s; ∧ s. P s ⇒ Q s] ⇒ onl Γ Q s"
 by (clarsimp dest!: onlD intro!: onllI)

lemma onll_weaken [elim!]: "∧ Γ P Q s. [onll Γ P s; ∧ s. P s ⇒ Q s] ⇒ onll Γ Q s"
 by (clarsimp dest!: onllD intro!: onllI)

```

lemma onll_weaken' [elim!]: "∧Γ P Q s. [ onll Γ P ((ξ, p), a, (ξ', p')) ;
    ∧l l'. 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'. [ (ξ, p) ∈ reachable A I ;
    ((ξ, p), a, (ξ', p')) ∈ trans A ;
    I a ;
    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) ∈ reachable A I"
  and "((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 (auto elim!: *)
  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) ∈ reachable A I"
  and "((ξ, p), a, (ξ', p')) ∈ trans A"
  and "I a"
  and lp: "l ∈ labels Γ p"
  and lp': "l' ∈ labels Γ p'"
  shows "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) ∈ reachable A I"
  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)"
  and step: "∧ξ p l ξ' l' a.
    [ (ξ, p) ∈ reachable A I ;
    l ∈ labels Γ p ;
    P (ξ, l) ;
    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) ∈ init A" and "l ∈ labels Γ p"
  thus "P (ξ, l)" by (rule init)
next
  fix ξ p a ξ' p' l'
  assume sr: "(ξ, p) ∈ reachable A I"
  and lp: "∀l ∈ labels Γ p. P (ξ, l)"
  and tr: "((ξ, p), a, (ξ', p')) ∈ trans A"
  and "I a"

```

```

    and lp': "l' ∈ labels Γ p'"
    show "P (ξ', l')"
  proof -
    from lp obtain l where "l ∈ labels Γ p" and "P (ξ, l)"
    using labels_not_empty [OF wf] by auto
    from sinv sr tr 'I a' this(1) lp' have "Q ((ξ, l), a, (ξ', l'))" ..
    with sr 'l ∈ labels Γ p' 'P (ξ, l)' show "P (ξ', l')" using 'I a' by (rule step)
  qed
qed

```

lemma onll_step_invariant_sterms:

```

  assumes wf: "wellformed Γ"
    and si: "A ⊨A (I →) onll Γ P"
    and sr: "(ξ, p) ∈ reachable A I"
    and sos: "((ξ, p), a, (ξ', q)) ∈ trans A"
    and "I a"
    and "l' ∈ labels Γ q"
    and "p' ∈ sterms Γ p"
    and "l ∈ labels Γ p'"
  shows "P ((ξ, l), a, (ξ', l'))"
  proof -
    from wf 'p' ∈ sterms Γ p' 'l ∈ labels Γ p' have "l ∈ labels Γ p"
    by (rule labels_sterms_labels)
    with si sr sos 'I a' show "P ((ξ, l), a, (ξ', l'))" using 'l' ∈ labels Γ q' ..
  qed

```

lemma seq_step_invariant_sterms:

```

  assumes inv: "A ⊨A (I →) onll Γ P"
    and wf: "wellformed Γ"
    and sp: "trans A = seqp_sos Γ"
    and "l' ∈ labels Γ q"
    and sr: "(ξ, p) ∈ reachable A I"
    and tr: "((ξ, p'), a, (ξ', q)) ∈ trans A"
    and "I a"
    and "p' ∈ sterms Γ p"
  shows "∀ l ∈ labels Γ p'. P ((ξ, l), a, (ξ', l'))"
  proof
    from tr and sp have "((ξ, p'), a, (ξ', q)) ∈ seqp_sos Γ" by simp
    hence "((ξ, p), a, (ξ', q)) ∈ seqp_sos Γ"
    using wf 'p' ∈ sterms Γ p' by (rule trans_from_sterms')
    with sp have trp: "((ξ, p), a, (ξ', q)) ∈ trans A" by simp
    fix l assume "l ∈ labels Γ p'"
    with wf inv sr trp 'I a' 'l' ∈ labels Γ q' 'p' ∈ sterms Γ p'
    show "P ((ξ, l), a, (ξ', l'))" by (rule onll_step_invariant_sterms)
  qed

```

lemma seq_step_invariant_sterms_weaken:

```

  assumes "A ⊨A (I →) onll Γ P"
    and "wellformed Γ"
    and "trans A = seqp_sos Γ"
    and "l' ∈ labels Γ q"
    and "(ξ, p) ∈ reachable A I'"
    and "((ξ, p'), a, (ξ', q)) ∈ trans A"
    and "I' a"
    and "p' ∈ sterms Γ p"
    and weaken: "∧ a. I' a ⇒ I a"
  shows "∀ l ∈ labels Γ p'. P ((ξ, l), a, (ξ', l'))"
  proof -
    from 'I' a' have "I a" by (rule weaken)
    from '(ξ, p) ∈ reachable A I'' have Ir: "(ξ, p) ∈ reachable A I"
    by (rule reachable_weakenE) (erule weaken)
    with asms(1-4) show ?thesis
    using '((ξ, p'), a, (ξ', q)) ∈ trans A' 'I a' and 'p' ∈ sterms Γ p'
    by (rule seq_step_invariant_sterms)
  qed

```

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 " $(\xi, p) \in$  reachable A I"
  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  $((\xi, l), a, (\xi', l'))$ "
using assms by (rule seq_step_invariant_sterms_weaken) simp
```

lemma onll_step_invariant_any_sterms:

```
assumes "wellformed  $\Gamma$ "
  and "A  $\models_A$  (I  $\rightarrow$ ) onll  $\Gamma$  P"
  and " $(\xi, p) \in$  reachable A I"
  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  $((\xi, l), a, (\xi', l'))$ "
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: " $\bigwedge p$  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
  ]  $\implies$  P  $((\xi, l), a, (\xi', l'))$ "
shows "A  $\models_A$  (I  $\rightarrow$ ) onll  $\Gamma$  P"
```

proof

```
  fix  $\xi$  p l a  $\xi'$  q l'
  assume sr: " $(\xi, p) \in$  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"
    and A3: " $((\xi, p'), a, (\xi', q)) \in$  seqp_sos  $\Gamma$ "
  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' ∈ sterms Γ p' have "l ∈ labels Γ p'" by simp
with A1 show "P ((ξ, l), a, (ξ', l'))" using A3 A4 A5 A6 A7 A8 'I a'
  by (rule step)
qed

```

```

lemma seq_step_invariant_ctermsI [intro]:
  assumes wf: "wellformed Γ"
    and cw: "control_within Γ (init A)"
    and sl: "simple_labels Γ"
    and sp: "trans A = seqp_sos Γ"
    and step: "∧p l ξ a q l' ξ' pp pn. [|
      wellformed Γ;
      p ∈ ctermsl (Γ pn);
      not_call p;
      l ∈ labels Γ p;
      ((ξ, p), a, (ξ', q)) ∈ seqp_sos Γ;
      ((ξ, p), a, (ξ', q)) ∈ trans A;
      l' ∈ labels Γ q;
      (ξ, pp) ∈ reachable A I;
      p ∈ sterms Γ pp;
      (ξ', q) ∈ reachable A I;
      I a
    |] ⇒ P ((ξ, l), a, (ξ', l'))"
  shows "A ⊨A (I →) onll Γ P"
using assms(1-4) proof (rule seq_step_invariant_ctermI)
  fix p pp l ξ a q l' ξ'
  assume "p ∈ cterms Γ"
    and otherassms: "l ∈ labels Γ p"
      "((ξ, p), a, (ξ', q)) ∈ seqp_sos Γ"
      "((ξ, p), a, (ξ', q)) ∈ trans A"
      "l' ∈ labels Γ q"
      "(ξ, pp) ∈ reachable A I"
      "p ∈ sterms Γ pp"
      "(ξ', q) ∈ reachable A I"
      "I 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 ((ξ, l), a, (ξ', 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 "wellformed Γ"
    and "∃pn. p ∈ subterms (Γ pn)"
    and "((σ, p), a, (σ', p')) ∈ oseqp_sos Γ i"
  shows "∃pn. p' ∈ subterms (Γ pn)"
using assms
proof (induct p)
  fix p1 p2
  assume IH1: "∃pn. p1 ∈ subterms (Γ pn) ⇒
    ((σ, p1), a, (σ', p')) ∈ oseqp_sos Γ i ⇒

```



```

       $\exists pn. p' \in \text{subterms } (\Gamma pn)$ "
and IH2: " $\exists pn. p2 \in \text{subterms } (\Gamma pn) \implies$ 
       $((\sigma, p2), a, (\sigma', p')) \in \text{oseqp\_sos } \Gamma i \implies$ 
       $\exists pn. p' \in \text{subterms } (\Gamma pn)$ "
and " $\exists pn. p1 \oplus p2 \in \text{subterms } (\Gamma pn)$ "
and " $((\sigma, p1 \oplus p2), a, (\sigma', p')) \in \text{oseqp\_sos } \Gamma i$ "
from ' $\exists pn. p1 \oplus p2 \in \text{subterms } (\Gamma pn)$ ' obtain pn
      where " $p1 \in \text{subterms } (\Gamma pn)$ "
      and " $p2 \in \text{subterms } (\Gamma pn)$ " by auto
from ' $((\sigma, p1 \oplus p2), a, (\sigma', p')) \in \text{oseqp\_sos } \Gamma i$ '
  have " $((\sigma, p1), a, (\sigma', p')) \in \text{oseqp\_sos } \Gamma i$ "
     $\vee ((\sigma, p2), a, (\sigma', p')) \in \text{oseqp\_sos } \Gamma i$  by auto
thus " $\exists pn. p' \in \text{subterms } (\Gamma pn)$ "
proof
  assume " $((\sigma, p1), a, (\sigma', p')) \in \text{oseqp\_sos } \Gamma i$ "
  with ' $p1 \in \text{subterms } (\Gamma pn)$ ' show ?thesis by (auto intro: IH1)
next
  assume " $((\sigma, p2), a, (\sigma', p')) \in \text{oseqp\_sos } \Gamma i$ "
  with ' $p2 \in \text{subterms } (\Gamma pn)$ ' show ?thesis by (auto intro: IH2)
qed
qed auto

```

lemma oreachable_subterms:

```

assumes "wellformed  $\Gamma$ "
  and "control_within  $\Gamma$  (init A)"
  and "trans A = oseqp_sos  $\Gamma i$ "
  and " $(\sigma, p) \in \text{oreachable } A S U$ "
shows " $\exists pn. p \in \text{subterms } (\Gamma pn)$ "
using assms(4)
proof (induct rule: oreachable_pair_induct)
  fix  $\sigma p$ 
  assume " $(\sigma, p) \in \text{init } A$ "
  with ' $\text{control\_within } \Gamma$  (init A)' show " $\exists pn. p \in \text{subterms } (\Gamma pn)$ " ..
next
  fix  $\sigma p a \sigma' p'$ 
  assume " $(\sigma, p) \in \text{oreachable } A S U$ "
    and " $\exists pn. p \in \text{subterms } (\Gamma pn)$ "
    and " $((\sigma, p), a, (\sigma', p')) \in \text{trans } A$ "
    and " $S \sigma \sigma' a$ "
  moreover from this(3) and ' $\text{trans } A = \text{oseqp\_sos } \Gamma i$ '
  have " $((\sigma, p), a, (\sigma', p')) \in \text{oseqp\_sos } \Gamma i$ " by simp
  ultimately show " $\exists pn. p' \in \text{subterms } (\Gamma pn)$ "
  using ' $\text{wellformed } \Gamma$ '
  by (auto elim: oseqp_sos_subterms)
qed

```

lemma onl_oinvariantI [intro]:

```

assumes init: " $\bigwedge \sigma p l. [\![ (\sigma, p) \in \text{init } A; l \in \text{labels } \Gamma p ]\!] \implies P (\sigma, l)$ "
  and other: " $\bigwedge \sigma \sigma' p l. [\![ (\sigma, p) \in \text{oreachable } A S U;$ 
       $\forall l \in \text{labels } \Gamma p. P (\sigma, l);$ 
       $U \sigma \sigma' ]\!] \implies \forall l \in \text{labels } \Gamma p. P (\sigma', l)$ "
  and step: " $\bigwedge \sigma p a \sigma' p' l'.$ 
       $[\![ (\sigma, p) \in \text{oreachable } A S U;$ 
       $\forall l \in \text{labels } \Gamma p. P (\sigma, l);$ 
       $((\sigma, p), a, (\sigma', p')) \in \text{trans } A;$ 
       $l' \in \text{labels } \Gamma p';$ 
       $S \sigma \sigma' a ]\!] \implies P (\sigma', l')$ "
shows " $A \models (S, U \rightarrow) \text{onl } \Gamma P$ "
proof
  fix  $\sigma p$ 
  assume " $(\sigma, p) \in \text{init } A$ "
  hence " $\forall l \in \text{labels } \Gamma p. P (\sigma, l)$ " using init by simp
  thus " $\text{onl } \Gamma P (\sigma, p)$ " ..
next

```

```

fix  $\sigma$  p a  $\sigma'$  p'
assume 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 " $S \sigma \sigma' a$ "
from 'onl  $\Gamma P (\sigma, p)$ ' have " $\forall l \in \text{labels } \Gamma p. P (\sigma, l)$ " ..
with rp tr ' $S \sigma \sigma' a$ ' have " $\forall l' \in \text{labels } \Gamma p'. P (\sigma', l')$ " by (auto elim: step)
thus "onl  $\Gamma P (\sigma', p)$ " ..
next
fix  $\sigma \sigma' p$ 
assume " $(\sigma, p) \in \text{oreachable } A S U$ "
  and "onl  $\Gamma P (\sigma, p)$ "
  and " $U \sigma \sigma'$ "
from 'onl  $\Gamma P (\sigma, p)$ ' have " $\forall l \in \text{labels } \Gamma p. P (\sigma, l)$ " by auto
with ' $(\sigma, p) \in \text{oreachable } A S U$ ' have " $\forall l \in \text{labels } \Gamma p. P (\sigma', l)$ "
  using ' $U \sigma \sigma'$ ' by (rule other)
thus "onl  $\Gamma P (\sigma', p)$ " by auto
qed

```

lemma global_oinvariantI [intro]:

```

assumes init: " $\bigwedge \sigma p. (\sigma, p) \in \text{init } A \implies P \sigma$ "
  and other: " $\bigwedge \sigma \sigma' p l. \llbracket (\sigma, p) \in \text{oreachable } A S U; P \sigma; U \sigma \sigma' \rrbracket \implies P \sigma'$ "
  and step: " $\bigwedge \sigma p a \sigma' p'. \llbracket (\sigma, p) \in \text{oreachable } A S U; P \sigma; ((\sigma, p), a, (\sigma', p')) \in \text{trans } A; S \sigma \sigma' a \rrbracket \implies P \sigma'$ "
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  $\sigma'$  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 " $S \sigma \sigma' a$ "
from ' $(\lambda(\sigma, \_). P \sigma) (\sigma, p)$ ' have " $P \sigma$ " by simp
with rp have " $P \sigma'$ "
  using tr ' $S \sigma \sigma' a$ ' by (rule step)
thus " $(\lambda(\sigma, \_). P \sigma) (\sigma', p)$ " by simp
next
fix  $\sigma \sigma' p$ 
assume " $(\sigma, p) \in \text{oreachable } A S U$ "
  and " $(\lambda(\sigma, \_). P \sigma) (\sigma, p)$ "
  and " $U \sigma \sigma'$ "
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: " $\bigwedge s s' a. S s s' a \implies S' s s' a$ "

```

```

    and weakenU: " $\bigwedge s s'. U s s' \implies U' s s'$ "
  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: " $(\sigma, p) \in \text{init } A$ "
    and pnl: " $l \in \text{labels } \Gamma p$ "
  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'  $\in$  sterms  $\Gamma p$ "
    and "l  $\in$  labels  $\Gamma p'$ "
  shows "P ( $\sigma$ , l)"
proof -
  from wf 'p'  $\in$  sterms  $\Gamma p'$  'l  $\in$  labels  $\Gamma p'$ ' have "l  $\in$  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'  $\in$  sterms  $\Gamma p$ "
    and "l  $\in$  labels  $\Gamma p'$ "
    and weakenS: " $\bigwedge \sigma \sigma' a. S \sigma \sigma' a \implies S' \sigma \sigma' a$ "
    and weakenU: " $\bigwedge \sigma \sigma'. U \sigma \sigma' \implies U' \sigma \sigma'$ "
  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'  $\in$  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_dsterms:

```

```

assumes " $((\sigma, p), a, (\sigma', q)) \in \text{oseqp\_sos } \Gamma \ i$ "
  and "wellformed  $\Gamma$ "
shows " $\forall r \in \text{sterms } \Gamma \ q. r \in \text{dterms } \Gamma \ p$ "
using assms by (induction q) auto

```

theorem cterms_includes_sterms_of_oseq_reachable:

```

assumes "wellformed  $\Gamma$ "
  and "control_within  $\Gamma \ (\text{init } A)$ "
  and "trans  $A = \text{oseqp\_sos } \Gamma \ i$ "
shows " $\bigcup (\text{sterms } \Gamma \ ' \ \text{snd } ' \ \text{oreachable } A \ S \ U) \subseteq \text{cterms } \Gamma$ "
proof
  fix qs
  assume "qs  $\in \bigcup (\text{sterms } \Gamma \ ' \ \text{snd } ' \ \text{oreachable } A \ S \ U)$ "
  then obtain  $\xi$  and q where *: " $(\xi, q) \in \text{oreachable } A \ S \ U$ "
    and **: "qs  $\in \text{sterms } \Gamma \ q$ " by auto
  from * have " $\bigwedge x. x \in \text{sterms } \Gamma \ q \implies x \in \text{cterms } \Gamma$ "
  proof (induction rule: oreachable_pair_induct)
    fix  $\sigma \ p \ q$ 
    assume " $(\sigma, p) \in \text{init } A$ "
      and "q  $\in \text{sterms } \Gamma \ p$ "
    from 'control_within  $\Gamma \ (\text{init } A)$ ' and ' $(\sigma, p) \in \text{init } A$ '
      obtain pn where "p  $\in \text{subterms } (\Gamma \ pn)$ " by auto
    with 'wellformed  $\Gamma$ ' show "q  $\in \text{cterms } \Gamma$ " using 'q  $\in \text{sterms } \Gamma \ p$ '
      by (rule subterms_sterms_in_cterms)
  next
    fix p  $\sigma \ a \ \sigma' \ q \ x$ 
    assume " $(\sigma, p) \in \text{oreachable } A \ S \ U$ "
      and IH: " $\bigwedge x. x \in \text{sterms } \Gamma \ p \implies x \in \text{cterms } \Gamma$ "
      and " $((\sigma, p), a, (\sigma', q)) \in \text{trans } A$ "
      and "x  $\in \text{sterms } \Gamma \ q$ "
    from this(3) and 'trans  $A = \text{oseqp\_sos } \Gamma \ i$ '
      have step: " $((\sigma, p), a, (\sigma', q)) \in \text{oseqp\_sos } \Gamma \ i$ " by simp
    from step 'wellformed  $\Gamma$ ' obtain ps
      where ps: "ps  $\in \text{sterms } \Gamma \ p$ "
      and step': " $((\sigma, ps), a, (\sigma', q)) \in \text{oseqp\_sos } \Gamma \ i$ "
      by (rule otrans_from_sterms [THEN bexE])
    from ps have "ps  $\in \text{cterms } \Gamma$ " by (rule IH)
    moreover from step' 'wellformed  $\Gamma$ ' 'x  $\in \text{sterms } \Gamma \ q$ ' have "x  $\in \text{dterms } \Gamma \ ps$ "
      by (rule otrans_to_dterms [rule_format])
    ultimately show "x  $\in \text{cterms } \Gamma$ " by (rule ctermsDI)
  qed
  thus "qs  $\in \text{cterms } \Gamma$ " using ** .
qed

```

corollary oseq_reachable_in_cterms:

```

assumes "wellformed  $\Gamma$ "
  and "control_within  $\Gamma \ (\text{init } A)$ "
  and "trans  $A = \text{oseqp\_sos } \Gamma \ i$ "
  and " $(\sigma, p) \in \text{oreachable } A \ S \ U$ "
  and "p'  $\in \text{sterms } \Gamma \ p$ "
shows "p'  $\in \text{cterms } \Gamma$ "
using assms(1-3)
proof (rule cterms_includes_sterms_of_oseq_reachable [THEN set_mp])
  from assms(4-5) show "p'  $\in \bigcup (\text{sterms } \Gamma \ ' \ \text{snd } ' \ \text{oreachable } A \ S \ U)$ "
    by (auto elim!: rev_bexI)
qed

```

lemma oseq_invariant_ctermI:

```

assumes wf: "wellformed  $\Gamma$ "
  and cw: "control_within  $\Gamma \ (\text{init } A)$ "
  and sl: "simple_labels  $\Gamma$ "
  and sp: "trans  $A = \text{oseqp\_sos } \Gamma \ i$ "
  and init: " $\bigwedge \sigma \ p \ l. \llbracket$ 
      ( $\sigma, p$ )  $\in \text{init } A$ ;"

```

```

      l ∈ labels Γ p
    ] ⇒ 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' ] ⇒ P (σ', l)"
and local: "∧p l σ a q l' σ' pp. [
  p ∈ cterms Γ;
  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) ∈ oreachable A S U;
  p ∈ sterms Γ pp;
  (σ', q) ∈ oreachable A S U;
  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"
  with init show "P (σ, l)" by auto
next
  fix σ p a σ' q l'
  assume sr: "(σ, p) ∈ oreachable A S U"
  and pl: "∀l ∈ labels Γ p. P (σ, l)"
  and tr: "((σ, p), a, (σ', q)) ∈ trans A"
  and A6: "l' ∈ labels Γ q"
  and "S σ σ' a"
  thus "P (σ', l'"
proof -
  from sr and tr and 'S σ σ' a' have A7: "(σ', q) ∈ oreachable A S U"
  by - (rule oreachable_local')
  from tr and sp have tr': "((σ, p), a, (σ', q)) ∈ oseqp_sos Γ i" by simp
  then obtain p' where "p' ∈ sterms Γ p"
  and A4: "((σ, p'), a, (σ', q)) ∈ oseqp_sos Γ i"
  by (blast dest: otrans_from_sterms [OF wf])
  from wf cw sp sr this(1) have A1: "p' ∈ cterms Γ"
  by (rule oseq_reachable_in_cterms)
  from labels_not_empty [OF wf] obtain l1 where A2: "l1 ∈ labels Γ p'"
  by blast
  with 'p' ∈ sterms Γ p' have "l1 ∈ labels Γ p"
  by (rule labels_sterms_labels [OF wf])
  with pl have A3: "P (σ, l1)" by simp
  from sr 'p' ∈ sterms Γ p'
  obtain pp where A7: "(σ, pp) ∈ oreachable A S U"
  and A8: "p' ∈ sterms Γ pp"
  by auto
  from sr tr 'S σ σ' a' have A9: "(σ', q) ∈ oreachable A S U"
  by - (rule oreachable_local')
  from sp and '((σ, p'), a, (σ', q)) ∈ oseqp_sos Γ i'
  have A5: "((σ, p'), a, (σ', q)) ∈ trans A" by simp
  from A1 A2 A3 A4 A5 A6 A7 A8 A9 'S σ σ' a' show ?thesis by (rule local)
qed
next
  fix σ σ' p l
  assume sr: "(σ, p) ∈ oreachable A S U"
  and "∀l ∈ labels Γ p. P (σ, l)"
  and "U σ σ'"
  show "∀l ∈ labels Γ p. P (σ', l)"
proof

```

```

fix l
assume "l ∈ labels Γ p"
with '∀ l ∈ labels Γ p. P (σ, l)' have "P (σ, l)" ..
with sr and 'l ∈ labels Γ p'
  show "P (σ', l)" using 'U σ σ'' by (rule other)
qed
qed

```

```

lemma oseq_invariant_ctermsI:
  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) ∈ oreachable 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) ∈ oreachable A S U;
    p ∈ sterms Γ pp;
    (σ', q) ∈ oreachable 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) ∈ oreachable 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) ∈ oreachable A S U"
  "p ∈ sterms Γ pp"
  "(σ', q) ∈ oreachable 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 ( $\sigma'$ ,  $l'$ )"
    using otherassms by (rule local)
qed

```

20.2 Open step invariants via labelled control terms

lemma onll_ostep_invariantI [intro]:

```

assumes *: " $\bigwedge \sigma p l a \sigma' p' l'. \llbracket (\sigma, p) \in \text{oreachable } A S U;$ 
           $((\sigma, p), a, (\sigma', p')) \in \text{trans } A;$ 
           $S \sigma \sigma' a;$ 
           $l \in \text{labels } \Gamma p;$ 
           $l' \in \text{labels } \Gamma p' \rrbracket$ 
           $\implies P ((\sigma, l), a, (\sigma', l'))"$ 

```

shows " $A \models_A (S, U \rightarrow) \text{onll } \Gamma P$ "

proof

fix $\sigma p \sigma' p' a$

assume " $(\sigma, p) \in \text{oreachable } A S U$ "

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

and " $S \sigma \sigma' a$ "

hence " $\forall l \in \text{labels } \Gamma p. \forall l' \in \text{labels } \Gamma p'. P ((\sigma, l), a, (\sigma', l'))$ " by (auto elim!: *)

thus " $\text{onll } \Gamma P ((\sigma, p), a, (\sigma', p'))$ " ..

qed

lemma onll_ostep_invariantE [elim]:

```

assumes "A  $\models_A (S, U \rightarrow) \text{onll } \Gamma P$ "
  and " $(\sigma, p) \in \text{oreachable } A S U$ "
  and " $((\sigma, p), a, (\sigma', p')) \in \text{trans } A$ "
  and " $S \sigma \sigma' a$ "
  and lp: " $l \in \text{labels } \Gamma p$ "
  and lp': " $l' \in \text{labels } \Gamma p'$ "
shows "P (( $\sigma, l$ ), a, ( $\sigma', l'$ ))"

```

proof -

from assms(1-4) have " $\text{onll } \Gamma P ((\sigma, p), a, (\sigma', p'))$ " ..

with lp lp' show "P ((σ, l), a, (σ', l'))" by auto

qed

lemma onll_ostep_invariantD [dest]:

```

assumes "A  $\models_A (S, U \rightarrow) \text{onll } \Gamma P$ "
  and " $(\sigma, p) \in \text{oreachable } A S U$ "
  and " $((\sigma, p), a, (\sigma', p')) \in \text{trans } A$ "
  and " $S \sigma \sigma' a$ "
shows " $\forall l \in \text{labels } \Gamma p. \forall l' \in \text{labels } \Gamma p'. P ((\sigma, l), a, (\sigma', l'))"$ 
using assms by auto

```

lemma onll_ostep_invariant_weakenD [dest]:

```

assumes "A  $\models_A (S', U' \rightarrow) \text{onll } \Gamma P$ "
  and " $(\sigma, p) \in \text{oreachable } A S U$ "
  and " $((\sigma, p), a, (\sigma', p')) \in \text{trans } A$ "
  and " $S' \sigma \sigma' a$ "
  and weakenS: " $\bigwedge s s' a. S s s' a \implies S' s s' a$ "
  and weakenU: " $\bigwedge s s'. U s s' \implies U' s s'$ "
shows " $\forall l \in \text{labels } \Gamma p. \forall l' \in \text{labels } \Gamma p'. P ((\sigma, l), a, (\sigma', l'))"$ 

```

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 ' $A \models_A (S', U' \rightarrow) \text{onll } \Gamma P$ ' show ?thesis

using ' $((\sigma, p), a, (\sigma', p')) \in \text{trans } A$ ' and ' $S' \sigma \sigma' a$ ' ..

qed

lemma onll_ostep_to_invariantI [intro]:

```

assumes sinv: "A  $\models_A (S, U \rightarrow) \text{onll } \Gamma Q$ "
  and wf: "wellformed  $\Gamma$ "

```

```

and init: " $\bigwedge \sigma \ l \ p. \llbracket (\sigma, p) \in \text{init } A; \ l \in \text{labels } \Gamma \ p \rrbracket \implies P(\sigma, l)$ "
and other: " $\bigwedge \sigma \ \sigma' \ p \ l. \llbracket (\sigma, p) \in \text{oreachable } A \ S \ U; \ l \in \text{labels } \Gamma \ p; \ P(\sigma, l); \ U \ \sigma \ \sigma' \rrbracket \implies P(\sigma', l)$ "
and local: " $\bigwedge \sigma \ p \ l \ \sigma' \ l' \ a. \llbracket (\sigma, p) \in \text{oreachable } A \ S \ U; \ l \in \text{labels } \Gamma \ p; \ P(\sigma, l); \ Q((\sigma, l), a, (\sigma', l')); \ S \ \sigma \ \sigma' \ a \rrbracket \implies P(\sigma', l')$ "
shows "A  $\models (S, U \rightarrow) \text{ onl } \Gamma \ P$ "
proof
  fix  $\sigma \ p \ l$ 
  assume " $(\sigma, p) \in \text{init } A$ " and " $l \in \text{labels } \Gamma \ p$ "
  thus " $P(\sigma, l)$ " by (rule init)
next
  fix  $\sigma \ p \ a \ \sigma' \ p' \ l'$ 
  assume sr: " $(\sigma, p) \in \text{oreachable } A \ S \ U$ "
  and lp: " $\forall l \in \text{labels } \Gamma \ p. P(\sigma, l)$ "
  and tr: " $((\sigma, p), a, (\sigma', p')) \in \text{trans } A$ "
  and "S  $\sigma \ \sigma' \ a$ "
  and lp': " $l' \in \text{labels } \Gamma \ p'$ "
  show " $P(\sigma', l')$ "
  proof -
    from lp obtain l where " $l \in \text{labels } \Gamma \ p$ " and " $P(\sigma, l)$ "
    using labels_not_empty [OF wf] by auto
    from sr tr 'S  $\sigma \ \sigma' \ a$ ' this(1) lp' have " $Q((\sigma, l), a, (\sigma', l'))$ " ..
    with sr ' $l \in \text{labels } \Gamma \ p$ ' ' $P(\sigma, l)$ ' show " $P(\sigma', l')$ " using 'S  $\sigma \ \sigma' \ a$ ' by (rule local)
  qed
next
  fix  $\sigma \ \sigma' \ p \ l$ 
  assume " $(\sigma, p) \in \text{oreachable } A \ S \ U$ "
  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
    fix l
    assume " $l \in \text{labels } \Gamma \ p$ "
    with ' $\forall l \in \text{labels } \Gamma \ p. P(\sigma, l)$ ' have " $P(\sigma, l)$ " ..
    with ' $(\sigma, p) \in \text{oreachable } A \ S \ U$ ' and ' $l \in \text{labels } \Gamma \ p$ '
    show " $P(\sigma', l)$ " using ' $U \ \sigma \ \sigma'$ ' by (rule other)
  qed
qed
qed

lemma onll_ostep_invariant_sterms:
  assumes wf: "wellformed  $\Gamma$ "
  and si: "A  $\models_A (S, U \rightarrow) \text{ onll } \Gamma \ P$ "
  and sr: " $(\sigma, p) \in \text{oreachable } A \ S \ U$ "
  and sos: " $((\sigma, p), a, (\sigma', q)) \in \text{trans } A$ "
  and "S  $\sigma \ \sigma' \ a$ "
  and " $l' \in \text{labels } \Gamma \ q$ "
  and " $p' \in \text{sterms } \Gamma \ p$ "
  and " $l \in \text{labels } \Gamma \ p'$ "
  shows " $P((\sigma, l), a, (\sigma', l'))$ "
  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 si sr sos 'S  $\sigma \ \sigma' \ a$ ' show " $P((\sigma, l), a, (\sigma', l'))$ " using ' $l' \in \text{labels } \Gamma \ q$ ' ..
  qed

lemma oseq_step_invariant_sterms:
  assumes inv: "A  $\models_A (S, U \rightarrow) \text{ onll } \Gamma \ P$ "

```



```

and wf: "wellformed  $\Gamma$ "
and sp: "trans A = oseqp_sos  $\Gamma$  i"
and "l'  $\in$  labels  $\Gamma$  q"
and sr: " $(\sigma, p) \in$  oreachable A S U"
and tr: " $((\sigma, p'), a, (\sigma', q)) \in$  trans A"
and "S  $\sigma \sigma' a$ "
and "p'  $\in$  sterms  $\Gamma$  p"
shows " $\forall l \in$  labels  $\Gamma$  p'. P  $((\sigma, l), a, (\sigma', l'))$ "
proof
from assms(3, 6) have " $((\sigma, p'), a, (\sigma', q)) \in$  oseqp_sos  $\Gamma$  i" by simp
hence " $((\sigma, p), a, (\sigma', q)) \in$  oseqp_sos  $\Gamma$  i"
  using wf 'p'  $\in$  sterms  $\Gamma$  p' by (rule otrans_from_sterms')
with assms(3) have trp: " $((\sigma, p), a, (\sigma', q)) \in$  trans A" by simp
fix l assume "l  $\in$  labels  $\Gamma$  p'"
with wf inv sr trp 'S  $\sigma \sigma' a$ ' 'l'  $\in$  labels  $\Gamma$  q' 'p'  $\in$  sterms  $\Gamma$  p'
  show "P  $((\sigma, l), a, (\sigma', l'))$ "
  by - (erule(7) onll_ostep_invariant_sterms)
qed

```

lemma oseq_step_invariant_sterms_weaken:

```

assumes inv: "A  $\models_A$  (S, U  $\rightarrow$ ) onll  $\Gamma$  P"
and wf: "wellformed  $\Gamma$ "
and sp: "trans A = oseqp_sos  $\Gamma$  i"
and "l'  $\in$  labels  $\Gamma$  q"
and sr: " $(\sigma, p) \in$  oreachable A S' U'"
and tr: " $((\sigma, p'), a, (\sigma', q)) \in$  trans A"
and "S'  $\sigma \sigma' a$ "
and "p'  $\in$  sterms  $\Gamma$  p"
and weakenS: " $\bigwedge \sigma \sigma' a. S' \sigma \sigma' a \implies S \sigma \sigma' a$ "
and weakenU: " $\bigwedge \sigma \sigma'. U' \sigma \sigma' \implies U \sigma \sigma'$ "
shows " $\forall l \in$  labels  $\Gamma$  p'. P  $((\sigma, l), a, (\sigma', l'))$ "

```

proof -

```

from 'S'  $\sigma \sigma' a$ ' have "S  $\sigma \sigma' a$ " by (rule weakenS)
from ' $(\sigma, p) \in$  oreachable A S' U''
  have Ir: " $(\sigma, p) \in$  oreachable A S U"
  by (rule oreachable_weakenE)
  (erule weakenS, erule weakenU)
with assms(1-4) show ?thesis
  using tr 'S  $\sigma \sigma' a$ ' 'p'  $\in$  sterms  $\Gamma$  p'
  by (rule oseq_step_invariant_sterms)
qed

```

lemma onll_ostep_invariant_any_sterms:

```

assumes wf: "wellformed  $\Gamma$ "
and si: "A  $\models_A$  (S, U  $\rightarrow$ ) onll  $\Gamma$  P"
and sr: " $(\sigma, p) \in$  oreachable A S U"
and sos: " $((\sigma, p), a, (\sigma', q)) \in$  trans A"
and "S  $\sigma \sigma' a$ "
and "l'  $\in$  labels  $\Gamma$  q"
shows " $\forall p' \in$  sterms  $\Gamma$  p.  $\forall l \in$  labels  $\Gamma$  p'. P  $((\sigma, l), a, (\sigma', l'))$ "
by (intro ballI) (rule onll_ostep_invariant_sterms [OF assms])

```

lemma oseq_step_invariant_ctermI [intro]:

```

assumes wf: "wellformed  $\Gamma$ "
and cw: "control_within  $\Gamma$  (init A)"
and sl: "simple_labels  $\Gamma$ "
and sp: "trans A = oseqp_sos  $\Gamma$  i"
and local: " $\bigwedge p l \sigma a q l' \sigma' pp. [$ 
  p  $\in$  cterms  $\Gamma$ ;
  l  $\in$  labels  $\Gamma$  p;
   $((\sigma, p), a, (\sigma', q)) \in$  oseqp_sos  $\Gamma$  i;
   $((\sigma, p), a, (\sigma', q)) \in$  trans A;
  l'  $\in$  labels  $\Gamma$  q;
   $(\sigma, pp) \in$  oreachable A S U;

```

```

      p∈sterms Γ pp;
      (σ', q) ∈ oreachable A S U;
      S σ σ' a
    ] ⇒ P ((σ, l), a, (σ', l'))"
shows "A ⊨A (S, U →) onll Γ P"
proof
  fix σ p l a σ' q l'
  assume sr: "(σ, p) ∈ oreachable A S U"
  and tr: "((σ, p), a, (σ', q)) ∈ trans A"
  and "S σ σ' a"
  and pl: "l ∈ labels Γ p"
  and A5: "l' ∈ labels Γ q"
  from this(2) and sp have "((σ, p), a, (σ', q)) ∈ oseqp_sos Γ i" by simp
  then obtain p' where "p' ∈ sterms Γ p"
    and A3: "((σ, p'), a, (σ', q)) ∈ oseqp_sos Γ i"
  by (blast dest: otrans_from_sterms [OF _ wf])
  from this(2) and sp have A4: "((σ, p'), a, (σ', q)) ∈ trans A" by simp
  from wf cw sp sr 'p'∈sterms Γ p' have A1: "p'∈cterms Γ"
  by (rule oseq_reachable_in_cterms)
  from sr 'p'∈sterms Γ p'
  obtain pp where A6: "(σ, pp)∈oreachable A S U"
    and A7: "p'∈sterms Γ pp"
  by auto
  from sr tr 'S σ σ' a' have A8: "(σ', q)∈oreachable A S U"
  by - (erule(2) oreachable_local')
  from wf cw sp sr have "∃pn. p ∈ subterms (Γ pn)"
  by (rule oreachable_subterms)
  with sl wf have "∀p'∈sterms Γ p. l ∈ labels Γ p'"
  using pl by (rule simple_labels_in_sterms)
  with 'p' ∈ sterms Γ p' have "l ∈ labels Γ p'" by simp
  with A1 show "P ((σ, l), a, (σ', l'))" using A3 A4 A5 A6 A7 A8 'S σ σ' a'
  by (rule local)
qed

```

```

lemma oseq_step_invariant_ctermsI [intro]:
  assumes wf: "wellformed Γ"
  and "control_within Γ (init A)"
  and "simple_labels Γ"
  and "trans A = oseqp_sos Γ i"
  and local: "∧p l σ a q l' σ' pp pn. [
    wellformed Γ;
    p∈ctermsl (Γ pn);
    not_call p;
    l∈labels Γ p;
    ((σ, p), a, (σ', q)) ∈ oseqp_sos Γ i;
    ((σ, p), a, (σ', q)) ∈ trans A;
    l'∈labels Γ q;
    (σ, pp) ∈ oreachable A S U;
    p∈sterms Γ pp;
    (σ', q) ∈ oreachable A S U;
    S σ σ' a
  ] ⇒ P ((σ, l), a, (σ', l'))"
shows "A ⊨A (S, U →) onll Γ P"
using assms(1-4) proof (rule oseq_step_invariant_ctermsI)
  fix p l σ a q l' σ' pp
  assume "p ∈ cterms Γ"
  and otherassms: "l ∈ labels Γ p"
  "((σ, p), a, (σ', q)) ∈ oseqp_sos Γ i"
  "((σ, p), a, (σ', q)) ∈ trans A"
  "l' ∈ labels Γ q"
  "(σ, pp) ∈ oreachable A S U"
  "p ∈ sterms Γ pp"
  "(σ', q) ∈ oreachable A S U"
  "S σ σ' a"

```

```

from this(1) obtain pn where "p ∈ cterms1(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 "[ ps ∈ sterms Γ p1; ((σ i, ps), a, σ' i, p') ∈ seqp_sos Γ ]"
    ⇒ "((σ, p1), a, (σ', p')) ∈ oseqp_sos Γ i"
  and "[ 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

```

21 Transfer standard invariants into open invariants

```

theory OAWN_Convert
imports AWN_SOS_Labels AWN_Invariants
        OAWN_SOS OAWN_Invariants
begin

definition initiali :: "'i ⇒ (('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, 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

lemma closed_from_initialiD [dest]:
  assumes "initiali i OI CI"
    and "(ξ, p) ∈ CI"
  shows "∃σ. σ i = ξ ∧ (σ, p) ∈ OI"
  using assms unfolding initiali_def by auto

definition
  seq1 :: "'i ⇒ (('s × 'l) ⇒ bool) ⇒ (('i ⇒ 's) × 'l) ⇒ bool"
where
  "seq1 i P ≡ (λ(σ, p). P (σ i, p))"

lemma seq1I [intro]:
  "P (fst s i, snd s) ⇒ seq1 i P s"
  by (clarsimp simp: seq1_def)

lemma same_seq1 [elim]:
  assumes "∀j∈{i}. σ' j = σ j"
    and "seq1 i P (σ', s)"
  shows "seq1 i P (σ, s)"
  using assms unfolding seq1_def by (clarsimp)

lemma seq1simp:
  "seq1 i P (σ, p) = P (σ i, p)"
  unfolding seq1_def by simp

lemma other_steps_resp_local [intro!, simp]: "other_steps (other A I) I"
  by (clarsimp elim!: otherE)

lemma seq1_onl_swap:
  "seq1 i (onl Γ P) = onl Γ (seq1 i P)"
  unfolding seq1_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

```

```

fix  $\sigma \sigma' l ms p$ 
assume " $\sigma' i = \sigma i$ "
  and " $\forall j \in \{i\}. \zeta j = \sigma j$ "
hence " $((\zeta, \{l\}broadcast(ms).p), broadcast (ms (\sigma i)), (\sigma', p)) \in oseqp\_sos \Gamma i$ "
  by (metis obroadcastT singleton_iff)
with " $\forall j \in \{i\}. \zeta j = \sigma j'$ " show " $\exists \zeta'. (\forall j \in \{i\}. \zeta' j = \sigma' j) \wedge$ 
  ( $(\zeta, \{l\}broadcast(ms).p), broadcast (ms (\sigma i)), (\zeta', p)) \in oseqp\_sos \Gamma i$ "
  by blast
next
fix  $\sigma \sigma' :: "nat \Rightarrow 's"$  and  $fmsg :: "'m \Rightarrow 's \Rightarrow 's"$  and  $msg l p$ 
assume *: " $\sigma' i = fmsg msg (\sigma i)$ "
  and **: " $\forall j \in \{i\}. \zeta j = \sigma j$ "
hence " $\forall j \in \{i\}. (\zeta(i := fmsg msg (\zeta i))) j = \sigma' j$ " by clarsimp
moreover from * **
  have " $((\zeta, \{l\}receive(fmsg).p), receive msg, (\zeta(i := fmsg msg (\zeta i)), p)) \in oseqp\_sos \Gamma i$ "
  by (metis fun_upd_same oreceiveT)
ultimately show " $\exists \zeta'. (\forall j \in \{i\}. \zeta' j = \sigma' j) \wedge$ 
  ( $(\zeta, \{l\}receive(fmsg).p), receive msg, (\zeta', p)) \in oseqp\_sos \Gamma i$ "
  by blast
next
fix  $\sigma' \sigma l p$  and  $fas :: "'s \Rightarrow 's"$ 
assume *: " $\sigma' i = fas (\sigma i)$ "
  and **: " $\forall j \in \{i\}. \zeta j = \sigma j$ "
hence " $\forall j \in \{i\}. (\zeta(i := fas (\zeta i))) j = \sigma' j$ " by clarsimp
moreover from * ** have " $((\zeta, \{l\}[fas] p), \tau, (\zeta(i := fas (\zeta i)), p)) \in oseqp\_sos \Gamma i$ "
  by (metis fun_upd_same oassignT)
ultimately show " $\exists \zeta'. (\forall j \in \{i\}. \zeta' j = \sigma' j) \wedge ((\zeta, \{l\}[fas] p), \tau, (\zeta', p)) \in oseqp\_sos \Gamma i$ "
  by blast
next
fix  $g :: "'s \Rightarrow 's set"$  and  $\sigma \sigma' l p$ 
assume *: " $\sigma' i \in g (\sigma i)$ "
  and **: " $\forall j \in \{i\}. \zeta j = \sigma j$ "
hence " $\forall j \in \{i\}. (SOME \zeta'. \zeta' i = \sigma' i) j = \sigma' j$ " by simp (metis (lifting, full_types) some_eq_ex)
moreover with * ** have " $((\zeta, \{l\}(g) p), \tau, (SOME \zeta'. \zeta' i = \sigma' i, p)) \in oseqp\_sos \Gamma i$ "
  by simp (metis oguardT step_seq_tau)
ultimately show " $\exists \zeta'. (\forall j \in \{i\}. \zeta' j = \sigma' j) \wedge ((\zeta, \{l\}(g) p), \tau, (\zeta', p)) \in oseqp\_sos \Gamma i$ "
  by blast
next
fix  $\sigma pn a \sigma' p'$ 
assume " $((\sigma, \Gamma pn), a, (\sigma', p')) \in oseqp\_sos \Gamma i$ "
  and IH: " $\forall j \in \{i\}. \zeta j = \sigma j \implies \exists \zeta'. (\forall j \in \{i\}. \zeta' j = \sigma' j) \wedge ((\zeta, \Gamma pn), a, (\zeta', p')) \in oseqp\_sos \Gamma i$ "
  and " $\forall j \in \{i\}. \zeta j = \sigma j$ "
then obtain  $\zeta'$  where " $\forall j \in \{i\}. \zeta' j = \sigma' j$ "
  and " $((\zeta, \Gamma pn), a, (\zeta', p')) \in oseqp\_sos \Gamma i$ "
  by blast
thus " $\exists \zeta'. (\forall j \in \{i\}. \zeta' j = \sigma' j) \wedge ((\zeta, call(pn)), a, (\zeta', p')) \in oseqp\_sos \Gamma i$ "
  by blast
next
fix  $\sigma p a \sigma' p' q$ 
assume " $((\sigma, p), a, (\sigma', p')) \in oseqp\_sos \Gamma i$ "
  and " $\forall j \in \{i\}. \zeta j = \sigma j \implies \exists \zeta'. (\forall j \in \{i\}. \zeta' j = \sigma' j) \wedge ((\zeta, p), a, (\zeta', p')) \in oseqp\_sos$ 
 $\Gamma i$ "
  and " $\forall j \in \{i\}. \zeta j = \sigma j$ "
then obtain  $\zeta'$  where " $\forall j \in \{i\}. \zeta' j = \sigma' j$ "
  and " $((\zeta, p), a, (\zeta', p')) \in oseqp\_sos \Gamma i$ "
  by blast
thus " $\exists \zeta'. (\forall j \in \{i\}. \zeta' j = \sigma' j) \wedge ((\zeta, p \oplus q), a, (\zeta', p')) \in oseqp\_sos \Gamma i$ "
  by blast
next
fix  $\sigma p a \sigma' q q'$ 
assume " $((\sigma, q), a, (\sigma', q')) \in oseqp\_sos \Gamma i$ "
  and " $\forall j \in \{i\}. \zeta j = \sigma j \implies \exists \zeta'. (\forall j \in \{i\}. \zeta' j = \sigma' j) \wedge ((\zeta, q), a, (\zeta', q')) \in oseqp\_sos$ 
 $\Gamma i$ "

```

```

    and "∀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)+)
qed

```

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, p), a, (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, {l}broadcast(ms).p), broadcast (ms (σ i)), (ξ', p)) ∈ seqp_sos Γ"
  by clarsimp

```

next

```

  fix fmsg :: "'m ⇒ 's ⇒ 's" and msg :: 'm and σ' σ l p
  assume "σ' i = fmsg msg (σ i)"
    and "σ i = ξ"
  have "((ξ, {l}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'
  show "∃ξ'. σ' i = ξ' ∧ ((ξ, {l}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) ∈ reachable OA 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) ∈ reachable A I"
  using assms(1) proof (induction rule: reachable_pair_induct)
  fix σ p
  assume "(σ, p) ∈ init OA"
  with 'initiali i (init OA) (init A)' obtain ξ where "σ i = ξ"
    and "(ξ, p) ∈ init A"
  by auto
  from '(ξ, p) ∈ init A' have "(ξ, p) ∈ reachable A I" ..
  with 'σ i = ξ' show "∃ξ. σ i = ξ ∧ (ξ, p) ∈ reachable A I"
  by auto

```

next

```

  fix σ p σ' p' a
  assume "(σ, p) ∈ reachable OA I"
    and IH: "∃ξ. σ i = ξ ∧ (ξ, p) ∈ reachable A I"
    and otr: "((σ, p), a, (σ', p')) ∈ trans OA"
    and "I a"
  from IH obtain ξ where "σ i = ξ"
    and cr: "(ξ, p) ∈ reachable A I"

```

```

  by clarsimp
  from otr and spo have " $((\sigma, p), a, (\sigma', p')) \in \text{oseqp\_sos } \Gamma \ i$ " by simp
  with ' $\sigma \ i = \xi$ ' obtain  $\xi'$  where " $\sigma' \ i = \xi'$ "
    and " $((\xi, p), a, (\xi', p')) \in \text{seqp\_sos } \Gamma$ "
  by (auto dest!: oseq_step_is_seq_step)
  from this(2) and sp have ctr: " $((\xi, p), a, (\xi', p')) \in \text{trans } A$ " by simp
  from ' $(\xi, p) \in \text{reachable } A \ I$ ' and ctr and ' $I \ a$ '
  have " $(\xi', p') \in \text{reachable } A \ I$ " ..
  with ' $\sigma' \ i = \xi'$ ' show " $\exists \xi. \sigma' \ i = \xi \wedge (\xi, p') \in \text{reachable } A \ I$ "
  by blast
qed

```

```

lemma reachable_oseq_seqp_sos':
  assumes "s ∈ reachable OA I"
    and "initiali 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 - (cases s, auto dest: reachable_oseq_seqp_sos)

```

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 "initiali 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} →) (seq1 i P)"
proof -
  have "OA ⊨ (I →) (seq1 i P)"
  proof (rule invariant_arbitraryI)
    fix s
    assume "s ∈ reachable OA I"
    with 'initiali 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 "seq1 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 "seq1 i P (σ', s)"
  thus "seq1 i P (σ, s)" ..
qed
qed

```

definition

```

seqll :: "'i ⇒ (((s × 'l) × 'a × (s × 'l)) ⇒ bool)
        ⇒ (((i ⇒ 's) × 'l) × 'a × ((i ⇒ 's) × 'l)) ⇒ bool"

```

where

```

"seqll i P ≡ (λ((σ, 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 "seqll i P ((σ1', s), a, (σ2', s'))"
  shows "seqll 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  $\models_A$  (I  $\rightarrow$ ) 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  $\models_A$  (act I, other ANY {i}  $\rightarrow$ ) (seqll i P)"
proof -
  have "OA  $\models_A$  (I  $\rightarrow$ ) (seqll i P)"
proof (rule step_invariant_arbitraryI)
  fix σ p a σ' p'
  assume or: "(σ, p) ∈ reachable OA I"
  and otr: "((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 otr 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 ctr: "((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  $\models_A$  (I  $\rightarrow$ ) 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 ΓQMSG :: "((m::msg) list, 'm, unit, unit label) seqp_env"
where

```



```

" $\Gamma_{QMSG} () = \text{labelled } () \text{ (receive } (\lambda \text{msg msgs. msgs @ [msg]). \text{call}())$ 
 $\oplus \langle \text{msgs. msgs} \neq [] \rangle$ 
  ( $\text{send } (\lambda \text{msgs. hd msgs). \llbracket \text{msgs. tl msgs} \rrbracket \text{call}()$ )
 $\oplus \text{receive } (\lambda \text{msg msgs. msgs @ [msg]). \text{call}())$ )"

```

```

definition  $\sigma_{QMSG} :: "(('m::msg) \text{list} \times ('m \text{list}, 'm, \text{unit}, \text{unit label}) \text{seqp}) \text{set}"$ 
where " $\sigma_{QMSG} \equiv \{([], \Gamma_{QMSG} ())\}$ "

```

```

abbreviation qmsg
  :: "(('m::msg) \text{list} \times ('m \text{list}, 'm, \text{unit}, \text{unit label}) \text{seqp}, 'm \text{seq\_action}) \text{automaton}"
where
  "qmsg  $\equiv (\text{init} = \sigma_{QMSG}, \text{trans} = \text{seqp\_sos } \Gamma_{QMSG})$ "

```

```

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

```

```

lemma  $\sigma_{QMSG}.\text{not\_empty}$  [simp, intro]: " $\sigma_{QMSG} \neq \{\}$ "
  unfolding  $\sigma_{QMSG}.\text{def}$  by simp

```

```

lemma  $\sigma_{QMSG}.\text{exists}$  [simp]: " $\exists \text{qmsg } q. (qmsg, q) \in \sigma_{QMSG}$ "
  unfolding  $\sigma_{QMSG}.\text{def}$  by simp

```

```

lemma qmsg_wf [simp]: "wellformed  $\Gamma_{QMSG}$ "
  by (rule wf\_no\_direct\_calls) auto

```

```

lemmas qmsg_labels_not_empty [simp] = labels_not_empty [OF qmsg_wf]

```

```

lemma qmsg_control_within [simp]: "control\_within  $\Gamma_{QMSG} (\text{init } \text{qmsg})$ "
  unfolding  $\sigma_{QMSG}.\text{def}$  by (rule control\_withinI) (auto simp del: \Gamma_{QMSG}.\text{simps})

```

```

lemma qmsg_simple_labels [simp]: "simple_labels  $\Gamma_{QMSG}$ "
  unfolding simple_labels_def by auto

```

```

lemma qmsg_trans: "trans qmsg = seqp\_sos  $\Gamma_{QMSG}$ "
  by simp

```

```

lemma  $\sigma_{QMSG}.\text{labels}$  [simp]: " $(\xi, q) \in \sigma_{QMSG} \implies \text{labels } \Gamma_{QMSG} q = \{() \}$ "
  unfolding  $\sigma_{QMSG}.\text{def}$  by simp

```

```

lemma qmsg_proc_cases [dest]:
  fixes p pn
  shows " $p \in \text{cterm} \text{sl } (\Gamma_{QMSG} \text{pn}) \implies p \in \text{cterm} \text{sl } (\Gamma_{QMSG} ())$ "
  using assms by simp

```

```

declare
   $\Gamma_{QMSG}.\text{simps}$  [cterm} \text{sl}_\text{env}]
  qmsg\_proc\_cases [cterm} \text{sl}_\text{cases}]
  seq\_invariant\_cterm} \text{sl}_\text{I} [OF qmsg\_wf qmsg\_control\_within qmsg\_simple\_labels qmsg\_trans, cterm} \text{sl}_\text{intros}]
  seq\_step\_invariant\_cterm} \text{sl}_\text{I} [OF qmsg\_wf qmsg\_control\_within qmsg\_simple\_labels qmsg\_trans, cterm} \text{sl}_\text{intros}]

```

end

23 Lifting rules for parallel compositions with QMSG

```

theory Qmsg_Lifting
imports Qmsg OAWN_SOS Inv_Cterms OAWN_Invariants
begin

```

```

lemma oseq_no_change_on_send:
  fixes  $\sigma s a \sigma' s'$ 
  assumes " $((\sigma, s), a, (\sigma', s')) \in \text{oseq\_sos } \Gamma i$ "
  shows "case a of
    broadcast m  $\implies \sigma' i = \sigma i$ 
  | groupcast ips m  $\implies \sigma' i = \sigma i$ "

```

```

| unicast ips m    ⇒ σ' i = σ i
| ¬unicast ips    ⇒ σ' i = σ i
| send m          ⇒ σ' i = σ i
| deliver m       ⇒ σ' i = σ i
| _ ⇒ True"
using assms by induction simp_all

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 Γ_QMSG)"
  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 Γ_QMSG)"
  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_QMSG"
    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 Γ_QMSG (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 ⊧_A onll Γ_QMSG (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 ⊧_A onll Γ_QMSG (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 ⊧_A onll Γ_QMSG (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_oreachable:
  assumes "( $\sigma$ ,  $\zeta$ )  $\in$  oreachable (A  $\langle\langle$   $i$  qmsg) (otherwith S {i} (orecvmsg R)) (other U {i})"
    (is "_  $\in$  oreachable _ ?owS _")
  and pinv: " $A \models_A$  (otherwith S {i} (orecvmsg R), other U {i}  $\rightarrow$ )
    globala ( $\lambda(\sigma, \_, \sigma'). U (\sigma i) (\sigma' i)$ )"
  and ustutter: " $\bigwedge \xi. U \xi \xi$ "
  and sgivesu: " $\bigwedge \xi \xi'. S \xi \xi' \implies U \xi \xi'$ "
  and upreservesq: " $\bigwedge \sigma \sigma' m. \llbracket \forall j. U (\sigma j) (\sigma' j); R \sigma m \rrbracket \implies R \sigma' m$ "
shows "( $\sigma$ , fst  $\zeta$ )  $\in$  oreachable A ?owS (other U {i})
   $\wedge$  snd  $\zeta \in$  reachable qmsg (recvmsg (R  $\sigma$ ))
   $\wedge$  ( $\forall m \in \text{set} (\text{fst} (\text{snd} \zeta)). R \sigma m$ )"
using assms(1) proof (induction rule: oreachable_pair_induct)
  fix  $\sigma$  pq
  assume "( $\sigma$ , pq)  $\in$  init (A  $\langle\langle$   $i$  qmsg)"
  then obtain p ms q where "pq = (p, (ms, q))"
    and "( $\sigma$ , p)  $\in$  init A"
    and "(ms, q)  $\in$  init qmsg"
  by (clarsimp simp del:  $\Gamma_{QMSG\_simps}$ )
  from this(2) have "( $\sigma$ , p)  $\in$  oreachable A ?owS (other U {i})" ..
  moreover from '(ms, q)  $\in$  init qmsg' have "(ms, q)  $\in$  reachable qmsg (recvmsg (R  $\sigma$ ))" ..
  moreover from '(ms, q)  $\in$  init qmsg' have "ms = []"
  unfolding  $\sigma_{QMSG\_def}$  by simp
  ultimately show "( $\sigma$ , fst pq)  $\in$  oreachable A ?owS (other U {i})
     $\wedge$  snd pq  $\in$  reachable qmsg (recvmsg (R  $\sigma$ ))
     $\wedge$  ( $\forall m \in \text{set} (\text{fst} (\text{snd} pq)). R \sigma m$ )"
  using 'pq = (p, (ms, q))' by simp
next
  note  $\Gamma_{QMSG\_simps}$  [simp del]
  case (other  $\sigma$  pq  $\sigma'$ )
  hence "( $\sigma$ , fst pq)  $\in$  oreachable A ?owS (other U {i})"
    and "other U {i}  $\sigma \sigma'$ "
    and qr: "snd pq  $\in$  reachable qmsg (recvmsg (R  $\sigma$ ))"
    and " $\forall m \in \text{set} (\text{fst} (\text{snd} pq)). R \sigma m$ "
  by simp_all
  from 'other U {i}  $\sigma \sigma'$ ' and ustutter have " $\forall j. U (\sigma j) (\sigma' j)$ "
  by (clarsimp elim!: otherE) metis
  from 'other U {i}  $\sigma \sigma'$ '
  and '( $\sigma$ , fst pq)  $\in$  oreachable A ?owS (other U {i})'
  have "( $\sigma'$ , fst pq)  $\in$  oreachable A ?owS (other U {i})"
  by - (rule oreachable_other')
  moreover have " $\forall m \in \text{set} (\text{fst} (\text{snd} pq)). R \sigma' m$ "
proof
  fix m assume "m  $\in$  set (fst (snd pq))"
  with ' $\forall m \in \text{set} (\text{fst} (\text{snd} pq)). R \sigma m$ ' have "R  $\sigma m$ " ..
  with ' $\forall j. U (\sigma j) (\sigma' j)$ ' show "R  $\sigma' m$ " by (rule upreservesq)
qed
moreover from qr have "snd pq  $\in$  reachable qmsg (recvmsg (R  $\sigma'$ ))"
proof
  fix a
  assume "recvmsg (R  $\sigma$ ) a"
  thus "recvmsg (R  $\sigma'$ ) a"
  proof (rule recvmsgE [where R=R])
    fix m assume "R  $\sigma m$ "
    with ' $\forall j. U (\sigma j) (\sigma' j)$ ' show "R  $\sigma' m$ " by (rule upreservesq)
  qed
qed
ultimately show ?case using qr by simp
next
  case (local  $\sigma$  pq  $\sigma'$  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: "(( $\sigma$ , (p, (ms, q))), a, ( $\sigma'$ , (p', (ms', q'))))"

```

```

      ∈ oparp_sos i (trans A) (seqp_sos  $\Gamma_{QMSG}$ )"
    and por: "( $\sigma$ , p) ∈ oreachable A ?owS (other U {i})"
    and qr: "(ms, q) ∈ reachable qmsg (recvmsg (R  $\sigma$ ))"
    and "∀m∈set ms. R  $\sigma$  m"
    and "?owS  $\sigma$   $\sigma'$  a"
  by (simp_all del:  $\Gamma_{QMSG}$ _simps)

from '?owS  $\sigma$   $\sigma'$  a' have "∀j. j≠i → S ( $\sigma$  j) ( $\sigma'$  j)"
  by (clarsimp dest!: otherwith_syncD)
with sgivesu have "∀j. j≠i → U ( $\sigma$  j) ( $\sigma'$  j)" by simp

from '?owS  $\sigma$   $\sigma'$  a' have "orecvmsg R  $\sigma$  a" by (rule otherwithE)
hence "recvmsg (R  $\sigma$ ) a" ..

from pqtr have "( $\sigma'$ , p') ∈ oreachable A ?owS (other U {i})
  ∧ (ms', q') ∈ reachable qmsg (recvmsg (R  $\sigma'$ ))
  ∧ (∀m∈set ms'. R  $\sigma'$  m)"

proof
  assume "(( $\sigma$ , p), a, ( $\sigma'$ , p')) ∈ trans A"
  and "∧m. a ≠ receive m"
  and "(ms', q') = (ms, q)"
  from this(1) have ptr: "(( $\sigma$ , p), a, ( $\sigma'$ , p')) ∈ trans A" by simp
  with pinv por and '?owS  $\sigma$   $\sigma'$  a' have "U ( $\sigma$  i) ( $\sigma'$  i)"
  by (auto dest!: ostep_invariantD)
  with '∀j. j≠i → U ( $\sigma$  j) ( $\sigma'$  j)' have "∀j. U ( $\sigma$  j) ( $\sigma'$  j)" by auto

  hence recvmsg': "∧a. recvmsg (R  $\sigma$ ) a ⇒ recvmsg (R  $\sigma'$ ) a"
  by (auto elim!: recvmsgE [where R=R] upreservesq)

  from por ptr '?owS  $\sigma$   $\sigma'$  a' have "( $\sigma'$ , p') ∈ oreachable A ?owS (other U {i})"
  by - (rule oreachable_local')

  moreover have "(ms', q') ∈ reachable qmsg (recvmsg (R  $\sigma'$ ))"
  proof -
    from qr and '(ms', q') = (ms, q)'
    have "(ms', q') ∈ reachable qmsg (recvmsg (R  $\sigma$ ))" by simp
    thus ?thesis by (rule reachable_weakenE) (erule recvmsg')
  qed

  moreover have "∀m∈set ms'. R  $\sigma'$  m"
  proof
    fix m
    assume "m∈set ms'"
    with '(ms', q') = (ms, q)' have "m∈set ms" by simp
    with '∀m∈set ms. R  $\sigma$  m' have "R  $\sigma$  m" ..
    with '∀j. U ( $\sigma$  j) ( $\sigma'$  j)' show "R  $\sigma'$  m"
    by (rule upreservesq)
  qed

  ultimately show
    "( $\sigma'$ , p') ∈ oreachable A ?owS (other U {i})
    ∧ (ms', q') ∈ reachable qmsg (recvmsg (R  $\sigma'$ ))
    ∧ (∀m∈set ms'. R  $\sigma'$  m)" by simp_all

next
  assume qtr: "((ms, q), a, (ms', q')) ∈ seqp_sos  $\Gamma_{QMSG}$ "
  and "∧m. a ≠ send m"
  and "p' = p"
  and " $\sigma'$  i =  $\sigma$  i"

  from this(4) and '∧ξ. U ξ ξ' have "U ( $\sigma$  i) ( $\sigma'$  i)" by simp
  with '∀j. j≠i → U ( $\sigma$  j) ( $\sigma'$  j)' have "∀j. U ( $\sigma$  j) ( $\sigma'$  j)" by auto

  hence recvmsg': "∧a. recvmsg (R  $\sigma$ ) a ⇒ recvmsg (R  $\sigma'$ ) a"
  by (auto elim!: recvmsgE [where R=R] upreservesq)

```

```

from qtr have tqtr: " $((ms, q), a, (ms', q')) \in \text{trans } qmsg$ " by simp

from ' $\forall j. U (\sigma j) (\sigma' j)$ ' and ' $\sigma' i = \sigma i$ ' have "other  $U \{i\} \sigma \sigma'$ " by auto
with por and ' $p' = p$ '
  have " $(\sigma', p') \in \text{oreachable } A \text{ ?owS } (\text{other } U \{i\})$ "
  by (auto dest: oreachable_other)

moreover have " $(ms', q') \in \text{reachable } qmsg (\text{recvmsg } (R \sigma'))$ "
proof (rule reachable_weakenE [where P="recvmsg (R  $\sigma$ )"])
  from qr tqtr ' $\text{recvmsg } (R \sigma) a$ ' show " $(ms', q') \in \text{reachable } qmsg (\text{recvmsg } (R \sigma))$ " ..
qed (rule recvmsg')

moreover have " $\forall m \in \text{set } ms'. R \sigma' m$ "
proof
  fix m
  assume "m  $\in$  set ms'"
  moreover have "case a of receive m  $\Rightarrow$   $ms' = ms @ [m] \mid \_ \Rightarrow \text{set } ms' \subseteq \text{set } ms$ "
  proof -
    from qr have " $(ms, q) \in \text{reachable } qmsg TT$ " ..
    thus ?thesis using tqtr
      by (auto dest!: step_invariantD [OF qmsg_queue_contents])
  qed
  ultimately have " $R \sigma' m$ " using ' $\forall m \in \text{set } ms. R \sigma m$ ' and ' $\text{orecvmsg } R \sigma a$ '
  by (cases a) auto
  with ' $\forall j. U (\sigma j) (\sigma' j)$ ' show " $R \sigma' m$ "
  by (rule upreservesq)
qed

ultimately show " $(\sigma', p') \in \text{oreachable } A \text{ ?owS } (\text{other } U \{i\})$ 
 $\wedge (ms', q') \in \text{reachable } qmsg (\text{recvmsg } (R \sigma'))$ 
 $\wedge (\forall m \in \text{set } ms'. R \sigma' m)$ " by simp

next
fix m
assume "a =  $\tau$ "
  and " $((\sigma, p), \text{receive } m, (\sigma', p')) \in \text{trans } A$ "
  and " $((ms, q), \text{send } m, (ms', q')) \in \text{seqp\_sos } \Gamma_{QMSG}$ "
from this(2-3)
  have ptr: " $((\sigma, p), \text{receive } m, (\sigma', p')) \in \text{trans } A$ "
  and qtr: " $((ms, q), \text{send } m, (ms', q')) \in \text{trans } qmsg$ " by simp_all

from qr have " $(ms, q) \in \text{reachable } qmsg TT$ " ..
with qtr have "m  $\in$  set ms"
  by (auto dest!: step_invariantD [OF qmsg_send_from_queue])
with ' $\forall m \in \text{set } ms. R \sigma m$ ' have " $R \sigma m$ " ..
hence " $\text{orecvmsg } R \sigma (\text{receive } m)$ " by simp

with ' $\forall j. j \neq i \longrightarrow S (\sigma j) (\sigma' j)$ ' have "?owS  $\sigma \sigma' (\text{receive } m)$ "
  by (auto intro!: otherwithI)
with pinv por ptr have " $U (\sigma i) (\sigma' i)$ "
  by (auto dest!: ostep_invariantD)
with ' $\forall j. j \neq i \longrightarrow U (\sigma j) (\sigma' j)$ ' have " $\forall j. U (\sigma j) (\sigma' j)$ " by auto
hence recvmsg': " $\bigwedge a. \text{recvmsg } (R \sigma) a \Longrightarrow \text{recvmsg } (R \sigma') a$ "
  by (auto elim!: recvmsgE [where R=R] upreservesq)

from por ptr have " $(\sigma', p') \in \text{oreachable } A \text{ ?owS } (\text{other } U \{i\})$ "
  using ' $\text{?owS } \sigma \sigma' (\text{receive } m)$ ' by - (erule(1) oreachable_local, simp)

moreover have " $(ms', q') \in \text{reachable } qmsg (\text{recvmsg } (R \sigma'))$ "
proof (rule reachable_weakenE [where P="recvmsg (R  $\sigma$ )"])
  have " $\text{recvmsg } (R \sigma) (\text{send } m)$ " by simp
  with qr qtr show " $(ms', q') \in \text{reachable } qmsg (\text{recvmsg } (R \sigma))$ " ..
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 upreservesq)
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: Γ_QMSG_simps)
qed

lemma par_qmsg_oreachable_statelessasm:
  assumes "(σ, ζ) ∈ oreachable A (⟨⟨i qmsg)
    (λσ _. orecvmsg (λ_. R) σ) (other (λ_ _. True) {i})"
  and ustutter: "∧ξ. U ξ ξ"
  shows "(σ, fst ζ) ∈ oreachable A (λσ _. orecvmsg (λ_. R) σ) (other (λ_ _. True) {i})
    ∧ snd ζ ∈ reachable qmsg (recvmsg R)
    ∧ (∀m∈set (fst (snd ζ)). R m)"
  proof -
    from assms(1)
      have "(σ, ζ) ∈ oreachable A (⟨⟨i qmsg)
        (otherwith (λ_ _. True) {i} (orecvmsg (λ_. R)))
        (other (λ_ _. True) {i})" by auto
  moreover
    have "A ⊨A (otherwith (λ_ _. True) {i} (orecvmsg (λ_. R)),
      other (λ_ _. True) {i} →) globala (λ(σ, _, σ'). True)"
      by auto
  ultimately
    obtain "(σ, fst ζ) ∈ oreachable A
      (otherwith (λ_ _. True) {i} (orecvmsg (λ_. R))) (other (λ_ _. True) {i})"
      and *: "snd ζ ∈ reachable qmsg (recvmsg R)"
      and **: "(∀m∈set (fst (snd ζ)). R m)"
      by (auto dest!: par_qmsg_oreachable)
  from this(1)
    have "(σ, fst ζ) ∈ oreachable A (λσ _. orecvmsg (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} (orecvmsg R), other U {i} →) global P"
  and "∧ξ. U ξ ξ"
  and "∧ξ ξ'. S ξ ξ' ⇒ U ξ ξ'"
  and "∧σ σ' m. [ [∀j. U (σ j) (σ' j); R σ m ] ⇒ R σ' m"
  and "A ⊨A (otherwith S {i} (orecvmsg R), other U {i} →)
    globala (λ(σ, _, σ'). U (σ i) (σ' i))"
  shows "A (⟨⟨i qmsg ⊨ (otherwith S {i} (orecvmsg R), other U {i} →) global P"
  proof (rule oinvariant_oreachableI)
    fix σ ζ
    assume "(σ, ζ) ∈ oreachable A (⟨⟨i qmsg) (otherwith S {i} (orecvmsg R)) (other U {i})"
    then obtain s where "(σ, s) ∈ oreachable A (otherwith S {i} (orecvmsg R)) (other U {i})"
      by (auto dest!: par_qmsg_oreachable [OF _ assms(5,2-4)])
  
```

```

with assms(1) show "global P ( $\sigma$ ,  $\zeta$ )"
  by (auto dest: oinvariant_weakenD [OF assms(1)])
qed

```

lemma lift_step_into_qmsg:

```

assumes inv: "A  $\vdash_A$  (otherwith S {i} (orecvmsg R), other U {i}  $\rightarrow$ ) globala P"
  and ustutter: " $\bigwedge \xi. U \xi \xi$ "
  and sgivesu: " $\bigwedge \xi \xi'. S \xi \xi' \implies U \xi \xi'$ "
  and upreservesq: " $\bigwedge \sigma \sigma' m. \llbracket \forall j. U (\sigma j) (\sigma' j); R \sigma m \rrbracket \implies R \sigma' m$ "
  and self_sync: "A  $\vdash_A$  (otherwith S {i} (orecvmsg R), other U {i}  $\rightarrow$ )
    globala ( $\lambda(\sigma, \_, \sigma'). U (\sigma i) (\sigma' i)$ )"

  and recv_stutter: " $\bigwedge \sigma \sigma' m. \llbracket \forall j. U (\sigma j) (\sigma' j); \sigma' i = \sigma i \rrbracket \implies P (\sigma, \text{receive } m, \sigma')$ "
  and receive_right: " $\bigwedge \sigma \sigma' m. P (\sigma, \text{receive } m, \sigma') \implies P (\sigma, \tau, \sigma')$ "
shows "A  $\llbracket_i$  qmsg  $\vdash_A$  (otherwith S {i} (orecvmsg R), other U {i}  $\rightarrow$ ) globala P"
  (is " $\_ \vdash_A$  (?owS, ?U  $\rightarrow$ )  $\_$ ")

```

proof (rule ostep_invariantI)

```

fix  $\sigma \zeta a \sigma' \zeta'$ 
assume or: " $(\sigma, \zeta) \in \text{oreachable } (A \llbracket_i$  qmsg) ?owS ?U"
  and otr: " $((\sigma, \zeta), a, (\sigma', \zeta')) \in \text{trans } (A \llbracket_i$  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 msgs q s' msgs' q'
  where " $\zeta = (s, (\text{msgs}, q))$ " " $\zeta' = (s', (\text{msgs}', q'))$ "
  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"
  " $(\text{msgs}, q) \in \text{reachable qmsg } (\text{recvmsg } (R \sigma))$ "
  " $(\forall m \in \text{set } \text{msgs}. R \sigma m)$ "
  by (auto dest: par_qmsg_oreachable [OF _ self_sync ustutter sgivesu]
    elim!: upreservesq)
from 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 " $\bigwedge m. a \neq \text{send } m$ "
  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 " $(\exists m. a = \text{receive } m) \vee (a = \tau)$ "

proof -

```

from ' $(\text{msgs}, q) \in \text{reachable qmsg } (\text{recvmsg } (R \sigma))$ '
  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 ' $\bigwedge m. a \neq \text{send } m$ '
  by (auto dest!: step_invariantD [OF 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)

```

```

next
  fix m
  assume "a = τ"
  and "((σ, s), receive m, (σ', s')) ∈ trans A"
  and "((msgs, q), send m, (msgs', q')) ∈ seqp_sos ΓQMSG"

  from '(msgs, q) ∈ reachable qmsg (recvmsg (R σ))'
  have "(msgs, q) ∈ reachable qmsg TT" ..
  moreover from '((msgs, q), send m, (msgs', q')) ∈ seqp_sos ΓQMSG'
  have "((msgs, q), send m, (msgs', q')) ∈ trans qmsg" by simp
  ultimately have "m ∈ set msgs"
  by (auto dest!: step_invariantD [OF qmsg_send_from_queue])

  with '∀m ∈ set msgs. R σ m' have "R σ m" ..
  with '?owS σ σ' a' have "?owS σ σ' (receive m)"
  by (auto dest!: otherwith_syncD)

  with '((σ, s), receive m, (σ', s')) ∈ trans A'
  have "globala P ((σ, s), receive m, (σ', s'))"
  using '(σ, s) ∈ oreachable A ?owS ?U'
  by - (rule ostep_invariantD [OF inv])
  hence "P (σ, receive m, σ')" by simp
  hence "P (σ, τ, σ')" by (rule receive_right)
  with 'a = τ' show "globala P ((σ, s), 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_statelessasm:

```

assumes "A ⊨A (λσ _. orecvmsg (λ_. R) σ, other (λ_ _. True) {i} →) globala P"
  and "∧σ σ' m. σ' i = σ i ⇒ P (σ, receive m, σ')"
  and "∧σ σ' m. 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 j) ≠ {}"
  and trans: "∧s a s' σ σ'. [ σ i = fst (sr s);
    σ' i = fst (sr s');
    (s, a, s') ∈ trans (np i) ]"

```


$\implies ((\sigma, \text{snd } (sr\ s)), a, (\sigma', \text{snd } (sr\ s'))) \in \text{trans } (\text{onp } i)$ "

begin

lemma *init_pnet_p_NodeS*:

assumes "*NodeS i s R* \in *init (pnet np p)*"
 shows "*p* = $\langle i; R \rangle$ "
 using *assms* by (*cases p*) (auto *simp* add: *node_comps*)

lemma *init_pnet_p_SubnetS*:

assumes "*SubnetS s1 s2* \in *init (pnet np p)*"
 obtains *p1 p2* where "*p* = (*p1* \parallel *p2*)"
 and "*s1* \in *init (pnet np p1)*"
 and "*s2* \in *init (pnet np p2)*"
 using *assms* by (*cases p*) (auto *simp* add: *node_comps*)

lemma *init_pnet_fst_sr_netgmap*:

assumes "*s* \in *init (pnet np p)*"
 and "*i* \in *net_ips s*"
 and "*wf_net_tree p*"
 shows "*the (fst (netgmap sr s) i)* \in (*fst* \circ *sr*) ' *init (np i)*"

using *assms* *proof* (*induction s* *arbitrary: p*)

fix *ii s Ri p*

assume "*NodeS ii s Ri* \in *init (pnet np p)*"
 and "*i* \in *net_ips (NodeS ii s Ri)*"
 and "*wf_net_tree p*"

note *this(1)*

moreover then have "*p* = $\langle ii; Ri \rangle$ "

by (*rule init_pnet_p_NodeS*)

ultimately have "*s* \in *init (np ii)*"

by (*clarsimp simp: node_comps*)

with '*i* \in *net_ips (NodeS ii s Ri)*'

show "*the (fst (netgmap sr (NodeS ii s Ri)) i)* \in (*fst* \circ *sr*) ' *init (np i)*"
 by *clarsimp*

next

fix *s1 s2 p*

assume *IH1*: " $\bigwedge p. s1 \in \text{init } (\text{pnet } np\ p)$ "

$\implies i \in \text{net_ips } s1$

$\implies \text{wf_net_tree } p$

$\implies \text{the } (\text{fst } (\text{netgmap } sr\ s1)\ i) \in (\text{fst } \circ\ sr) ' \text{init } (np\ i)$ "

and *IH2*: " $\bigwedge p. s2 \in \text{init } (\text{pnet } np\ p)$ "

$\implies i \in \text{net_ips } s2$

$\implies \text{wf_net_tree } p$

$\implies \text{the } (\text{fst } (\text{netgmap } sr\ s2)\ i) \in (\text{fst } \circ\ sr) ' \text{init } (np\ i)$ "

and "*SubnetS s1 s2* \in *init (pnet np p)*"

and "*i* \in *net_ips (SubnetS s1 s2)*"

and "*wf_net_tree p*"

from *this(3)* obtain *p1 p2* where "*p* = (*p1* \parallel *p2*)"

and "*s1* \in *init (pnet np p1)*"

and "*s2* \in *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* \cap *net_tree_ips p2* = $\{\}$ "

by *auto*

from '*i* \in *net_ips (SubnetS s1 s2)*' have "*i* \in *net_ips s1* \vee *i* \in *net_ips s2*"

by *simp*

thus "*the (fst (netgmap sr (SubnetS s1 s2)) i)* \in (*fst* \circ *sr*) ' *init (np i)*"

proof

assume "*i* \in *net_ips s1*"

hence "*i* \notin *net_ips s2*"

proof -

from '*s1* \in *init (pnet np p1)*' and '*i* \in *net_ips s1*' have "*i* \in *net_tree_ips p1*" ..

with '*net_tree_ips p1* \cap *net_tree_ips p2* = $\{\}$ ' have "*i* \notin *net_tree_ips p2*" by *auto*

with '*s2* \in *init (pnet np p2)*' show ?*thesis* ..

```

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 ∘ sr) ' init (np i)"
    by (rule IH1)
ultimately show ?thesis by simp
next
assume "i ∈ net_ips s2"
moreover with 's2 ∈ init (pnet np p2)' have "the (fst (netgmap sr s2) i) ∈ (fst ∘ 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 ∘ sr) ' init (np i)) } ⊆ init (opnet onp p)"

using assms proof (induction p)
  fix i R
  assume "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 ∘ sr) ' init (np j))} ⊆ init (opnet onp ⟨i; R⟩)"
    by (clarsimp simp add: node_comps onode_comps)
    (rule set_mp [OF init], auto)
next
  fix p1 p2
  assume IH1: "wf_net_tree p1
    ⇒ {(σ, snd (netgmap sr s)) | σ s. s ∈ init (pnet np p1)
      ∧ (∀ i. if i ∈ net_tree_ips p1 then σ i = the (fst (netgmap sr s) i)
              else σ i ∈ (fst ∘ sr) ' init (np i))} ⊆ init (opnet onp p1)"
    (is "_ ⇒ ?S1 ⊆ _")
  and 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 ∘ sr) ' init (np i))} ⊆ init (opnet onp p2)"
    (is "_ ⇒ ?S2 ⊆ _")
  and "wf_net_tree (p1 || 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 ∘ 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 ∘ 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
    }
  }

```

```

}
moreover
{ fix i
  assume "i ∉ net_tree_ips p1"
  have "σ i ∈ (fst ∘ sr) 'init (np i)"
  proof (cases "i ∈ net_tree_ips p2")
    assume "i ∉ net_tree_ips p2"
    with 'i ∉ net_tree_ips p1' and σ_desc show ?thesis
      by simp
  next
    assume "i ∈ net_tree_ips p2"
    with 's2 ∈ init (pnet np p2)' have "i ∈ net_ips s2" ..
    with 's2 ∈ init (pnet np p2)' have "the (fst (netgmap sr s2) i) ∈ (fst ∘ sr) 'init (np i)"
      using 'wf_net_tree p2' by (rule init_pnet_fst_sr_netgmap)
    with 'i ∈ net_tree_ips p2' and 'i ∈ net_ips s2' show ?thesis
      using σ_desc by simp
  qed
}
ultimately show ?thesis
  using 's1 ∈ init (pnet np p1)' and σ_desc by auto
qed
hence "(σ, snd (netgmap sr s1)) ∈ init (opnet onp p1)"
  by (rule set_mp [OF IH1 [OF 'wf_net_tree p1']])

have "(σ, snd (netgmap sr s2)) ∈ ?S2"
proof -
{ fix i
  assume "i ∈ net_tree_ips p2"
  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 s2) i)"
    by simp
}
moreover
{ fix i
  assume "i ∉ net_tree_ips p2"
  have "σ i ∈ (fst ∘ sr) 'init (np i)"
  proof (cases "i ∈ net_tree_ips p1")
    assume "i ∉ net_tree_ips p1"
    with 'i ∉ net_tree_ips p2' and σ_desc show ?thesis
      by simp
  next
    assume "i ∈ net_tree_ips p1"
    with 's1 ∈ init (pnet np p1)' have "i ∈ net_ips s1" ..
    with 's1 ∈ init (pnet np p1)' have "the (fst (netgmap sr s1) i) ∈ (fst ∘ sr) 'init (np i)"
      using 'wf_net_tree p1' by (rule init_pnet_fst_sr_netgmap)
    moreover from 's2 ∈ init (pnet np p2)' and 'i ∉ net_tree_ips p2' have "i ∉ net_ips s2" ..
    ultimately show ?thesis
      using 'i ∈ net_tree_ips p1' 'i ∈ net_ips s1' and 'i ∉ net_tree_ips p2' σ_desc by simp
  qed
}
ultimately show ?thesis
  using 's2 ∈ init (pnet np p2)' and σ_desc by auto
qed
hence "(σ, snd (netgmap sr s2)) ∈ init (opnet onp p2)"
  by (rule set_mp [OF IH2 [OF 'wf_net_tree p2']])

with '(σ, snd (netgmap sr s1)) ∈ init (opnet onp p1)'
show "(σ, snd (netgmap sr (SubnetS s1 s2))) ∈ init (opnet onp (p1 || p2))"
  using 'net_tree_ips p1 ∩ net_tree_ips p2 = {}'
    'net_ips s1 = net_tree_ips p1'
    'net_ips s2 = 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 "(λ i. 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
ultimately show ?thesis
  by (rule set_rev_mp [rotated])
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 → σ' 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 Ri ns ns' ζ
    assume "(ns, connect(i, i'), ns') ∈ trans (pnet np ⟨ii; Ri⟩)"

```

```

    and "netgmap sr ns = netmask (net_tree_ips ⟨ii; Ri⟩) (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' R') ∈ node_sos (trans (np ii))"
    by simp
  moreover then have "s' = s" by auto
  ultimately have "((σ, NodeS ni (snd (sr s)) R), connect(i, i'), (σ, NodeS ni (snd (sr s)) R'))
    ∈ onode_sos (trans (onp ii))"

  by - (rule node_connectTE', 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 ⟨ii; Ri⟩) (σ, ζ)'
  show "((σ, snd (netgmap sr ns)), connect(i, i'), (σ, snd (netgmap sr ns')))) ∈ trans (opnet onp
⟨ii; Ri⟩)
    ∧ netgmap sr ns' = netmask (net_tree_ips ⟨ii; Ri⟩) (σ, snd (netgmap sr ns'))"
  by (simp add: onode_comps)
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 (opne
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 (opne
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 pnet_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 pnet_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_split1)
with '(s1, connect(i, i'), s1') ∈ trans (pnet np n1)'
  and 's1 ∈ reachable (pnet np n1) TT'

```

```

n1)"
  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, 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, connect(i, i'), s2') ∈ trans (pnet np n2)'
  and 's2 ∈ reachable (pnet np n2) TT'
n2)"
  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'))"
  using 'wf_net_tree n2' unfolding atomize_conj by (rule IH2)

have "((σ, snd (netgmap sr s)), connect(i, i'), (σ, snd (netgmap sr s')))"
  ∈ trans (opnet onp (n1 || n2))"

proof -
n1)'
  from '((σ, snd (netgmap sr s1)), connect(i, i'), (σ, snd (netgmap sr s1'))) ∈ trans (opnet onp
n2)'
  and '((σ, snd (netgmap sr s2)), connect(i, i'), (σ, snd (netgmap sr s2'))) ∈ trans (opnet onp
n2)'
  have "((σ, SubnetS (snd (netgmap sr s1)) (snd (netgmap sr s2))), connect(i, i'),
  (σ, SubnetS (snd (netgmap sr s1')) (snd (netgmap sr s2'))))"
  ∈ opnet_sos (trans (opnet onp n1)) (trans (opnet onp n2))"
  by (rule opnet_connect)
with 's = SubnetS s1 s2' 's' = SubnetS s1' s2'' show ?thesis by simp
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, ζ)' have "ζ = snd (netgmap sr s)" by simp
ultimately show " ∃σ' ζ'. ((σ, ζ), connect(i, i'), (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

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, ζ)"
  and "wf_net_tree n"
  obtains σ' ζ' where "((σ, ζ), disconnect(i, i'), (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
  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'))"

proof (induction n arbitrary: s s' ζ)
  fix ii Ri ns ns' ζ
  assume "(ns, disconnect(i, i'), ns') ∈ trans (pnet np ⟨ii; Ri⟩)"
  and "netgmap sr ns = netmask (net_tree_ips ⟨ii; Ri⟩) (σ, ζ)"
  from this(1) have "(ns, disconnect(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, disconnect(i, i'), NodeS ni s' R') ∈ node_sos (trans (np ii))"
  by simp
moreover then have "s' = s" by auto
ultimately have "((σ, NodeS ni (snd (sr s)) R), disconnect(i, i'), (σ, NodeS ni (snd (sr s)) R'))
  ∈ onode_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 ⟨ii; R_i⟩) (σ, ζ)'
  show "((σ, snd (netgmap sr ns)), disconnect(i, i'), (σ, snd (netgmap sr ns')))) ∈ trans (opnet
onp ⟨ii; R_i⟩)
  ∧ netgmap sr ns' = netmask (net_tree_ips ⟨ii; R_i⟩) (σ, snd (netgmap sr ns'))"
  by (simp add: onode_comps)
next
fix n1 n2 s s' ζ
assume IH1: "∧s s' ζ. (s, disconnect(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)), disconnect(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, disconnect(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)), disconnect(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, disconnect(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, disconnect(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, disconnect(i, i'), s1') ∈ trans (pnet np n1)"
  and "(s2, disconnect(i, i'), s2') ∈ trans (pnet np n2)"
  by (rule partial_disconnectTE) 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 pnet_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 pnet_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_split1)
with '(s1, disconnect(i, i'), s1') ∈ trans (pnet np n1)'
  and 's1 ∈ reachable (pnet np n1) TT'
  have "((σ, snd (netgmap sr s1)), disconnect(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, 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'
  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)

have "((σ, snd (netgmap sr s)), disconnect(i, i'), (σ, snd (netgmap sr s'))))
  ∈ trans (opnet onp (n1 || n2))"

proof -
  from '((σ, snd (netgmap sr s1)), disconnect(i, i'), (σ, snd (netgmap sr s1')))) ∈ trans (opnet
onp n1)'
  and '((σ, snd (netgmap sr s2)), disconnect(i, i'), (σ, snd (netgmap sr s2')))) ∈ trans (opnet
onp n2)'
  have "((σ, SubnetS (snd (netgmap sr s1)) (snd (netgmap sr s2))), disconnect(i, i'),
  (σ, SubnetS (snd (netgmap sr s1')) (snd (netgmap sr s2'))))
  ∈ opnet_sos (trans (opnet onp n1)) (trans (opnet onp n2))"
    by (rule opnet_disconnect)
  with 's = SubnetS s1 s2' 's' = SubnetS s1' s2'' show ?thesis by simp
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)), disconnect(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, ζ)' have "ζ = snd (netgmap sr s)" by simp
ultimately show "∃σ' ζ'. ((σ, ζ), disconnect(i, i'), (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

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 ∉ net_ips ζ → σ' j = σ 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' ≡ "λj. 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 Ri ns ns' ζ
  assume "(ns, τ, ns') ∈ trans (pnet np ⟨ii; Ri⟩)"
    and nsr: "ns ∈ reachable (pnet np ⟨ii; Ri⟩) TT"
    and "netgmap sr ns = netmask (net_tree_ips ⟨ii; Ri⟩) (σ, ζ)"
    and "netgmap sr ns' = netmask (net_tree_ips ⟨ii; Ri⟩) (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 ii))"
    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; Ri⟩) (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; Ri⟩) (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)' '∀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; Ri⟩)"
      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 ∈ reachable (pnet np n1) TT
    ⇒ wf_net_tree n1
    ⇒ i ∈ net_ips s
    ⇒ net_ip_action np τ i n1 s s'
    ⇒ ((σ, snd (netgmap sr s)), τ, (σ', snd (netgmap sr s'))) ∈ trans (opnet onp n1)"
and IH2: "∧ s s' ζ. (s, τ, s') ∈ trans (pnet np n2)
    ⇒ netgmap sr s = netmask (net_tree_ips n2) (σ, ζ)
    ⇒ netgmap sr s' = netmask (net_tree_ips n2) (σ', snd (netgmap sr s'))
    ⇒ s ∈ reachable (pnet np n2) TT
    ⇒ wf_net_tree n2
    ⇒ i ∈ net_ips s
    ⇒ net_ip_action np τ i n2 s s'
    ⇒ ((σ, snd (netgmap sr s)), τ, (σ', snd (netgmap sr s'))) ∈ trans (opnet onp n2)"
and tr: "(s, τ, 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 τ i (n1 || n2) s s'"
from tr have "(s, τ, 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_tauTE) 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)), τ, (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 τ i (n1 || n2) s s''
        have "(s1, τ, s1') ∈ trans (pnet np n1)"
            and "net_ip_action np τ i n1 s1 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))'
  '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 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'' 'net_ip_action np τ i (n1 || n2) s s''
  have "(s2, τ, s2') ∈ trans (pnet np n2)"
    and "net_ip_action np τ i n2 s2 s2'"
    and "s1' = s1" by simp_all

from 'net_ips s2 = net_tree_ips n2' and '(s2, τ, 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, τ, s2') ∈ trans (pnet np n2)'
  'netgmap sr s2 = netmask (net_tree_ips n2) (σ, snd (netgmap sr s2))'
  'netgmap sr s2' = netmask (net_tree_ips n2) (σ', snd (netgmap sr s2'))'
  's2 ∈ reachable (pnet np n2) TT'
  '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'' 's1' = s1' show ?thesis
  by (simp del: step_node_tau) (erule opnet_tau2)
qed
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 ζ → σ' j = σ j" by (metis net_ips_netgmap)
ultimately have "((σ, ζ), τ, (σ', snd (netgmap sr s')))) ∈ trans (opnet onp n)
  ∧ (∀j. j ∉ net_ips 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 "∃σ' ζ'. ((σ, ζ), τ, (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
qed

```

lemma transfer_deliver:

```

assumes "(s, i:deliver(d), 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), i:deliver(d), (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

```

from assms(4,2,1) obtain "i ∈ net_ips s"
  and "∀j. j ≠ i → netmap s' j = netmap s j"
  and "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 "∀j. j ≠ i → fst (netgmap sr s') j = 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 σ' ≡ "λj. 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)), i:deliver(d), (σ', snd (netgmap sr s')) ∈ trans (opnet onp n)"
    using assms(2,4) 'i ∈ net_ips s' and 'net_ip_action np (i:deliver(d)) i n s s'"

```

proof (induction n arbitrary: s s' ζ)

```

  fix ii Ri ns ns' ζ
  assume "(ns, i:deliver(d), ns') ∈ trans (pnet np ⟨ii; Ri⟩)"
    and nsr: "ns ∈ reachable (pnet np ⟨ii; Ri⟩) TT"
    and "netgmap sr ns = netmask (net_tree_ips ⟨ii; Ri⟩) (σ, ζ)"
    and "netgmap sr ns' = netmask (net_tree_ips ⟨ii; Ri⟩) (σ', snd (netgmap sr ns'))"
    and "i ∈ net_ips ns"

```

```

  from this(1) have "(ns, i:deliver(d), 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_sos_dest)

```

```

  moreover from 'i ∈ net_ips ns' and 'ns = NodeS ii s R' have "ii = i" by simp
  ultimately have ntr: "(NodeS i s R, i:deliver(d), NodeS i s' R') ∈ node_sos (trans (np ii))"
    by simp

```

```

  hence "R' = R" by (metis net_state.inject(1) node_deliverTE')

```

```

  from ntr have "(s, deliver d, s') ∈ trans (np i)"
    by (rule node_deliverTE') simp

```

```

  from 'netgmap sr ns = netmask (net_tree_ips ⟨ii; Ri⟩) (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; Ri)) (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)

with 'ns = NodeS ii s R' 'ns' = NodeS ii s' R' 'ii = i'
  show "((σ, snd (netgmap sr ns)), i:deliver(d), (σ', snd (netgmap sr ns')))) ∈ trans (opnet onp (ii;
Ri))"
    by (simp add: onode_comps)
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 n1) (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 n2) (σ, ζ)
  ⇒ netgmap sr s' = netmask (net_tree_ips 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 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 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)
  'netgmap sr s2 = netmask (net_tree_ips n2) (σ, snd (netgmap sr s2))
  'netgmap sr s2' = netmask (net_tree_ips n2) (σ', snd (netgmap sr s2'))
  's2 ∈ reachable (pnet np n2) TT
  'wf_net_tree n2
  'i ∈ net_ips s2
  'net_ip_action np (i:deliver(d)) i n2 s2 s2'
have "((σ, snd (netgmap sr s2)), i:deliver(d), (σ', snd (netgmap 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
qed
with 'ζ = snd (netgmap sr s)' have "((σ, ζ), i:deliver(d), (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 ζ → σ' j = σ j" by (metis net_ips_netgmap)
ultimately have "((σ, ζ), i:deliver(d), (σ', snd (netgmap sr s')))) ∈ trans (opnet onp n)
  ∧ (∀j. j∉net_ips 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 "∃σ' ζ'. ((σ, ζ), i:deliver(d), (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
qed

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 "netgmap 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 'netgmap sr s = netmask (net_tree_ips n) (σ, ζ)'
      have "ζ = snd (netgmap sr s)" by simp
    from 'netgmap sr s' = netmask (net_tree_ips n) (σ', ζ)''
      have "ζ' = snd (netgmap sr s')"
        and "netgmap sr s' = netmask (net_tree_ips n) (σ', snd (netgmap sr s'))"
        by simp_all
    from assms(1-3) 'netgmap sr s' = netmask (net_tree_ips n) (σ', snd (netgmap sr s'))' assms(5)
      have "((σ, snd (netgmap sr s)), H¬K:arrive(m), (σ', snd (netgmap sr s')))) ∈ trans (opnet onp n)"
    proof (induction n arbitrary: s s' ζ H K)
      fix ii Ri ns ns' ζ H K
      assume "(ns, H¬K:arrive(m), ns') ∈ trans (pnet np ⟨ii; Ri⟩)"
        and nsr: "ns ∈ reachable (pnet np ⟨ii; Ri⟩) TT"
        and "netgmap sr ns = netmask (net_tree_ips ⟨ii; Ri⟩) (σ, ζ)"
        and "netgmap sr ns' = netmask (net_tree_ips ⟨ii; Ri⟩) (σ', snd (netgmap 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 = {}"
  from 'netgmap sr ns = netmask (net_tree_ips ⟨ii; R_i⟩) (σ, ζ)' and '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_i⟩) (σ', snd (netgmap sr ns'))'
    and 'ns' = NodeS ii s' R'
    have "σ' ii = fst (sr s'" by simp (metis map_upd_Some_unfold)
  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), {ii}¬{ }:arrive(m), (σ', NodeS ii (snd (sr s')) R))
    ∈ onode_sos (trans (onp ii))"

    by (rule onode_receive)
  with 'H={ii}' and 'K={}'
    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
  assume "H = {}"
    and "s = s'"
    and "K = {ii}"
  from 's = s'' 'netgmap sr ns' = netmask (net_tree_ips ⟨ii; R_i⟩) (σ', snd (netgmap sr ns'))'
    'netgmap sr ns = netmask (net_tree_ips ⟨ii; R_i⟩) (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={}' '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
qed
with 'ns = NodeS ii s R' 'ns' = NodeS ii s' R'
  show "((σ, snd (netgmap sr ns)), H-K:arrive(m), (σ', snd (netgmap sr ns'))))
    ∈ trans (opnet onp ⟨ii; R_i⟩)"

  by (simp add: onode_comps)
next
fix n1 n2 s s' ζ H K
assume IH1: "∧s s' ζ H K. (s, H-K:arrive(m), s') ∈ trans (pnet np n1)
  ⇒ s ∈ reachable (pnet np n1) TT
  ⇒ netgmap sr s = netmask (net_tree_ips n1) (σ, ζ)
  ⇒ netgmap sr s' = netmask (net_tree_ips n1) (σ', snd (netgmap sr s'))
  ⇒ wf_net_tree n1
  ⇒ ((σ, snd (netgmap sr s)), H-K:arrive(m), σ', snd (netgmap sr s'))
    ∈ trans (opnet onp n1)"
and IH2: "∧s s' ζ H K. (s, H-K:arrive(m), s') ∈ trans (pnet np n2)
  ⇒ s ∈ reachable (pnet np n2) TT
  ⇒ netgmap sr s = netmask (net_tree_ips n2) (σ, ζ)
  ⇒ netgmap sr s' = netmask (net_tree_ips n2) (σ', snd (netgmap sr s'))
  ⇒ wf_net_tree n2
  ⇒ ((σ, snd (netgmap sr s)), H-K:arrive(m), σ', snd (netgmap 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: "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)"

```



```

from this(3) have "(s, H¬K:arrive(m), s') ∈ pnet_sos (trans (pnet np n1))
                                                    (trans (pnet np n2))"

  by simp
thus "((σ, snd (netgmap sr s)), H¬K:arrive(m), (σ', snd (netgmap sr s'))))
      ∈ 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¬K1:arrive(m), s1') ∈ trans (pnet np n1)"
    and tr2: "(s2, H2¬K2:arrive(m), s2') ∈ trans (pnet np n2)"
    and "H = H1 ∪ H2"
    and "K = K1 ∪ K2"

  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)
  with tr1 have "net_ips s1' = net_tree_ips n1" by (metis pnet_maintains_dom)

  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)
  with tr2 have "net_ips s2' = net_tree_ips n2" by (metis pnet_maintains_dom)

  from '(s1, H1¬K1:arrive(m), s1') ∈ trans (pnet np n1)'
    's1 ∈ reachable (pnet np n1) TT'
  have "((σ, snd (netgmap sr s1)), H1¬K1:arrive(m), (σ', snd (netgmap sr s1'))))
      ∈ trans (opnet onp n1)"

  proof (rule IH1 [OF _ _ _ _ 'wf_net_tree n1'])
    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'
    show "netgmap sr s1 = netmask (net_tree_ips n1) (σ, snd (netgmap sr s1))"
      by (rule netgmap_subnet_split1)
  next
    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'
    show "netgmap sr s1' = netmask (net_tree_ips n1) (σ', snd (netgmap sr s1'))"
      by (rule netgmap_subnet_split1)
  qed

  moreover from '(s2, H2¬K2:arrive(m), s2') ∈ trans (pnet np n2)'
    's2 ∈ reachable (pnet np n2) TT'
  have "((σ, snd (netgmap sr s2)), H2¬K2:arrive(m), (σ', snd (netgmap sr s2'))))
      ∈ 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 "netgmap sr s2 = netmask (net_tree_ips n2) (σ, snd (netgmap sr s2))"
      by (rule netgmap_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 "netgmap sr s2' = netmask (net_tree_ips n2) (σ', snd (netgmap sr s2'))"
      by (rule netgmap_subnet_split2)

```

```

      qed
      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)
    qed
  qed
  with 'ζ = snd (netgmap sr s)' and 'ζ' = snd (netgmap sr s)
  show "((σ, ζ), H¬K:arrive(m), (σ', ζ')) ∈ trans (opnet onp n)"
    by simp
  qed

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
  qed

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_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 "fst (netgmap sr s') = fst (netmask (net_tree_ips n) (σ', snd (netgmap sr s')))"
  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"
  unfolding σ'_def by simp

from 'netgmap sr s = netmask (net_tree_ips n) (σ, ζ)'
  have "ζ = snd (netgmap sr s)" by simp

from assms(1-3) 'netgmap sr s' = netmask (net_tree_ips n) (σ', snd (netgmap sr s'))' assms(4)
  have "((σ, snd (netgmap sr s)), mR:*cast(m), (σ', snd (netgmap sr s'))) ∈ trans (opnet onp n)"
proof (induction n arbitrary: s s' ζ mR)
  fix ii Ri ns ns' ζ mR
  assume "(ns, mR:*cast(m), ns') ∈ trans (pnet np ⟨ii; Ri⟩)"
    and nsr: "ns ∈ reachable (pnet np ⟨ii; Ri⟩) TT"
    and "netgmap sr ns = netmask (net_tree_ips ⟨ii; Ri⟩) (σ, ζ)"
    and "netgmap sr ns' = netmask (net_tree_ips ⟨ii; Ri⟩) (σ', snd (netgmap sr ns'))"
  from this(1) have "(ns, mR:*cast(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_castTE')
  ultimately have "(NodeS ii s R, mR:*cast(m), NodeS ii s' R) ∈ node_sos (trans (np ii))"
    by simp

from 'netgmap sr ns = netmask (net_tree_ips ⟨ii; Ri⟩) (σ, ζ)' and 'ns = NodeS ii s R'
  have "σ ii = fst (sr s)"
  by simp (metis map_upd_Some_unfold)
from 'netgmap sr ns' = netmask (net_tree_ips ⟨ii; Ri⟩) (σ', snd (netgmap sr ns'))'
  and 'ns' = NodeS ii s' R'
  have "σ' ii = fst (sr s')" by simp (metis map_upd_Some_unfold)

from '(NodeS ii s R, mR:*cast(m), NodeS ii s' R) ∈ node_sos (trans (np ii))'
  have "((σ, NodeS ii (snd (sr s)) R), mR:*cast(m), (σ', NodeS ii (snd (sr s')) R))
    ∈ onode_sos (trans (onp ii))"
proof (rule node_castTE)
  assume "(s, broadcast m, s') ∈ trans (np ii)"
    and "R = mR"
  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 (onp ii)"
  by (rule trans)
  hence "((σ, NodeS ii (snd (sr s)) R), R:*cast(m), (σ', NodeS ii (snd (sr s')) R))
    ∈ onode_sos (trans (onp ii))"
  by (rule onode_bcast)
  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 (rule onode_gcast)

```

```

R))
with 'mR = R ∩ D' 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, unicast d m, s') ∈ trans (np ii)"
    and "d ∈ R"
    and "mR = {d}"
from 'σ ii = fst (sr s)' 'σ' ii = fst (sr s')' and this(1)
    have "((σ, snd (sr s)), unicast d m, (σ', snd (sr s')))) ∈ trans (onp ii)"
    by (rule trans)
hence "((σ, NodeS ii (snd (sr s)) R), {d}:*cast(m), (σ', NodeS ii (snd (sr s')) R))
                                     ∈ onode_sos (trans (onp ii)))"
    using 'd∈R' by (rule onode_ucast)
with 'mR={d}' 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
qed
with 'ns = NodeS ii s R' 'ns' = NodeS ii s' R'
    show "((σ, snd (netgmap sr ns)), mR:*cast(m), (σ', snd (netgmap sr ns'))))
                                     ∈ trans (opnet onp ⟨ii; R_i⟩)"
    by (simp add: onode_comps)
next
fix n1 n2 s s' ζ mR
assume IH1: "∧s s' ζ mR. (s, mR:*cast(m), s') ∈ trans (pnet np n1)
    ⇒ s ∈ reachable (pnet np n1) TT
    ⇒ netgmap sr s = netmask (net_tree_ips n1) (σ, ζ)
    ⇒ netgmap sr s' = netmask (net_tree_ips n1) (σ', snd (netgmap sr s'))
    ⇒ wf_net_tree n1
    ⇒ ((σ, snd (netgmap sr s)), mR:*cast(m), σ', snd (netgmap sr s'))
       ∈ trans (opnet onp n1)"
and IH2: "∧s s' ζ mR. (s, mR:*cast(m), s') ∈ trans (pnet np n2)
    ⇒ s ∈ reachable (pnet np n2) TT
    ⇒ netgmap sr s = netmask (net_tree_ips n2) (σ, ζ)
    ⇒ netgmap sr s' = netmask (net_tree_ips n2) (σ', snd (netgmap sr s'))
    ⇒ wf_net_tree n2
    ⇒ ((σ, snd (netgmap sr s)), mR:*cast(m), σ', snd (netgmap sr s'))
       ∈ trans (opnet onp n2)"
and "(s, mR:*cast(m), 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)"
from this(3) have "(s, mR:*cast(m), s') ∈ pnet_sos (trans (pnet np n1)) (trans (pnet np n2))"
    by simp
then obtain s1 s1' s2 s2' H K
    where "s = SubnetS s1 s2"
    and "s' = SubnetS s1' s2'"
    and "H ⊆ mR"
    and "K ∩ mR = {}"
    and trtr: "((s1, mR:*cast(m), s1') ∈ trans (pnet np n1)
    ∧ (s2, H¬K:arrive(m), s2') ∈ trans (pnet np n2))
    ∨ ((s1, H¬K:arrive(m), s1') ∈ trans (pnet np n1)
    ∧ (s2, mR:*cast(m), s2') ∈ trans (pnet np n2))"
    by (rule partial_castTE)metis+
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)

```

```

with trtr have "net_ips s1' = net_tree_ips n1" by (metis pnet_maintains_dom)

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)
with trtr have "net_ips s2' = net_tree_ips n2" by (metis pnet_maintains_dom)

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_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 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 trtr show "((σ, snd (netgmap sr s)), mR:*cast(m), (σ', snd (netgmap sr s'))))
  ∈ trans (opnet onp (n1 || n2))"

proof (elim disjE conjE)
  assume "(s1, mR:*cast(m), s1') ∈ trans (pnet np n1)"
  and "(s2, H-K:arrive(m), s2') ∈ trans (pnet np n2)"
  from '(s1, mR:*cast(m), s1') ∈ trans (pnet np n1)'
    's1 ∈ reachable (pnet np n1) TT'
    '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'))'
    'wf_net_tree n1'
  have "((σ, snd (netgmap sr s1)), mR:*cast(m), (σ', snd (netgmap sr s1')))) ∈ trans (opnet onp
n1)"
    by (rule IH1)

  moreover from '(s2, H-K:arrive(m), s2') ∈ trans (pnet np n2)'
    's2 ∈ reachable (pnet np n2) TT'
    'netgmap sr s2 = netmask (net_tree_ips n2) (σ, snd (netgmap sr s2))'
    'netgmap sr s2' = netmask (net_tree_ips n2) (σ', snd (netgmap sr s2'))'
    'wf_net_tree n2'
  have "((σ, snd (netgmap sr s2)), H-K:arrive(m), (σ', snd (netgmap sr s2')))) ∈ trans (opnet onp
n2)"
    by (rule transfer_arrive')

  ultimately have "((σ, SubnetS (snd (netgmap sr s1)) (snd (netgmap sr s2))), mR:*cast(m),
  (σ', SubnetS (snd (netgmap sr s1')) (snd (netgmap sr s2'))))
  ∈ opnet_sos (trans (opnet onp n1)) (trans (opnet onp n2))"
    using 'H ⊆ mR' and 'K ∩ mR = {}' by (rule opnet_sos.intros(1))
  with 's = SubnetS s1 s2' 's' = SubnetS s1' s2'' show ?thesis by simp
next
  assume "(s1, H-K:arrive(m), s1') ∈ trans (pnet np n1)"
  and "(s2, mR:*cast(m), s2') ∈ trans (pnet np n2)"

```

```

from '(s1, H¬K:arrive(m), s1') ∈ trans (pnet np n1)
  's1 ∈ reachable (pnet np n1) TT'
  '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'))'
  'wf_net_tree n1'
have "((σ, snd (netgmap sr s1)), H¬K:arrive(m), (σ', snd (netgmap sr s1')))) ∈ trans (opnet onp
n1)"
  by (rule transfer_arrive')

moreover from '(s2, mR:*cast(m), s2') ∈ trans (pnet np n2)
  's2 ∈ reachable (pnet np n2) TT'
  'netgmap sr s2 = netmask (net_tree_ips n2) (σ, snd (netgmap sr s2))'
  'netgmap sr s2' = netmask (net_tree_ips n2) (σ', snd (netgmap sr s2'))'
  'wf_net_tree n2'
have "((σ, snd (netgmap sr s2)), mR:*cast(m), (σ', snd (netgmap sr s2')))) ∈ trans (opnet onp
n2)"
  by (rule IH2)

ultimately have "((σ, SubnetS (snd (netgmap sr s1)) (snd (netgmap sr s2))), mR:*cast(m),
  (σ', SubnetS (snd (netgmap sr s1')) (snd (netgmap sr s2'))))
  ∈ opnet_sos (trans (opnet onp n1)) (trans (opnet onp n2))"
  using 'H ⊆ mR' and 'K ∩ mR = {}' by (rule opnet_sos.intros(2))
with 's = SubnetS s1 s2' 's' = SubnetS s1' s2'' show ?thesis by simp
qed
qed
with 'ζ = snd (netgmap sr s)' have "((σ, ζ), mR:*cast(m), (σ', snd (netgmap sr s')))) ∈ trans (opnet
onp n)"
  by simp
moreover from '∀j. j∉net_ips (snd (netgmap sr s)) → σ' j = σ j' 'ζ = snd (netgmap sr s)'
  have "∀j. j∉net_ips ζ → σ' j = σ j" by simp
moreover note 'netgmap sr s' = netmask (net_tree_ips n) (σ', snd (netgmap sr s'))'
ultimately show "∃σ' ζ'. ((σ, ζ), mR:*cast(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
qed

lemma transfer_pnet_action:
  assumes "s ∈ reachable (pnet np n) TT"
    and "netgmap sr s = netmask (net_tree_ips n) (σ, ζ)"
    and "wf_net_tree n"
    and "(s, a, s') ∈ trans (pnet np n)"
  obtains σ' ζ' where "((σ, ζ), a, (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
  show "∃σ' ζ'. ((σ, ζ), a, (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)'"

proof (cases a)
  case node_cast
  with assms(4) show ?thesis
    by (auto elim!: transfer_cast [OF _ assms(1-3)])
next
  case node_deliver
  with assms(4) show ?thesis
    by (auto elim!: transfer_deliver [OF _ assms(1-3)])
next
  case node_arrive
  with assms(4) show ?thesis
    by (auto elim!: transfer_arrive [OF _ assms(1-3)])
next
  case node_connect
  with assms(4) show ?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
qed

```

lemma transfer_action_pnet_closed:

```

assumes "(s, a, s') ∈ trans (closed (pnet np n))"
obtains a' where "(s, a', s') ∈ trans (pnet np n)"
  and "∧σ ζ σ' ζ'. [ ((σ, ζ), a', (σ', ζ')) ∈ trans (opnet onp n);
    (∀j. j∉net_ips ζ → σ' j = σ j) ]
    ⇒ ((σ, ζ), a, (σ', ζ')) ∈ trans (oclosed (opnet onp n))"

```

proof (atomize_elim)

```

from assms have "(s, a, s') ∈ cnet_sos (trans (pnet np n))" by simp

```

```

thus "∃a'. (s, a', s') ∈ trans (pnet np n)"

```

```

  ∧ (∀σ ζ σ' ζ'. ((σ, ζ), a', (σ', ζ')) ∈ trans (opnet onp n)
    → (∀j. j ∉ net_ips ζ → σ' j = σ j)
    → ((σ, ζ), a, (σ', ζ')) ∈ trans (oclosed (opnet 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 ∈ reachable (closed (pnet np n)) TT"
  and "netgmap sr s = netmask (net_tree_ips n) (σ, ζ)"
  and "wf_net_tree n"
  and "(s, a, s') ∈ trans (closed (pnet np n))"

```

```

obtains σ' ζ' where "((σ, ζ), a, (σ', ζ')) ∈ trans (oclosed (opnet onp n))"
  and "netgmap sr s' = netmask (net_tree_ips n) (σ', ζ')"
```

proof atomize_elim

```

from assms(1) have "s ∈ reachable (pnet np n) TT" ..

```

```

from assms(4)

```

```

  show "∃σ' ζ'. ((σ, ζ), a, (σ', ζ')) ∈ trans (oclosed (opnet onp n))
    ∧ netgmap sr s' = netmask (net_tree_ips n) (σ', ζ')"
```

```

  by (cases a)

```

```

    ((elim transfer_action_pnet_closed
      transfer_pnet_action [OF 's ∈ reachable (pnet np n) TT' assms(2-3)]?)?,
      (auto intro!: exI)[1])+

```

qed

lemma pnet_reachable_transfer':

```

assumes "wf_net_tree n"

```

```

  and "s ∈ reachable (closed (pnet np n)) TT"

```

```

  shows "netgmap sr s ∈ netmask (net_tree_ips n) 'oreachable (oclosed (opnet onp n)) (λ_ _ . True)
  U"
```

```

    (is " _ ∈ ?f ' ?oreachable n")

```

using assms(2) proof induction

```

  fix s

```

```

  assume "s ∈ init (closed (pnet np n))"

```

```

  hence "s ∈ init (pnet np n)" by simp

```

```

  with 'wf_net_tree n' have "netgmap sr s ∈ netmask (net_tree_ips n) 'init (opnet onp n)"

```

```

    by (rule init_pnet_opnet)

```

```

hence "netgmap sr s ∈ netmask (net_tree_ips n) ‘ init (oclosed (opnet onp n))"
  by simp
moreover have "netmask (net_tree_ips n) ‘ init (oclosed (opnet onp n))
  ⊆ netmask (net_tree_ips n) ‘ ?oreachable n"
  by (intro image_mono subsetI) (rule oreachable_init)
ultimately show "netgmap sr s ∈ netmask (net_tree_ips n) ‘ ?oreachable n"
  by (rule set_rev_mp)
next
fix s a s'
assume "s ∈ reachable (closed (pnet np n)) TT"
  and "netgmap sr s ∈ netmask (net_tree_ips n) ‘ ?oreachable n"
  and "(s, a, s') ∈ trans (closed (pnet np n))"
from this(2) obtain σ ζ where "netgmap sr s = netmask (net_tree_ips n) (σ, ζ)"
  and "(σ, ζ) ∈ ?oreachable n"
  by clarsimp
from 's ∈ reachable (closed (pnet np n)) TT' this(1) 'wf_net_tree n'
  and '(s, a, s') ∈ trans (closed (pnet np n))'
  obtain σ' ζ' where "((σ, ζ), a, (σ', ζ')) ∈ trans (oclosed (opnet onp n))"
  and "netgmap sr s' = netmask (net_tree_ips n) (σ', ζ')"
  by (rule transfer_action)
from '(σ, ζ) ∈ ?oreachable n' and this(1) have "(σ', ζ') ∈ ?oreachable n"
  by (rule oreachable_local) simp
with 'netgmap sr s' = netmask (net_tree_ips n) (σ', ζ')'
  show "netgmap sr s' ∈ netmask (net_tree_ips n) ‘ ?oreachable n" by (rule image_eqI)
qed

```

definition

```
someinit :: "nat ⇒ 'g"
```

where

```
"someinit i ≡ SOME x. x ∈ (fst o sr) ‘ init (np i)"
```

definition

```
initmissing :: "(nat ⇒ 'g option) × 'a ⇒ (nat ⇒ 'g) × 'a"
```

where

```
"initmissing σ = (λi. case (fst σ) i of None ⇒ someinit i | Some s ⇒ s, snd σ)"
```

lemma initmissing_def':

```
"initmissing = apfst (default someinit)"
```

```
by (auto simp add: initmissing_def default_def)
```

lemma netmask_initmissing_netgmap:

```
"netmask (net_ips s) (initmissing (netgmap sr s)) = netgmap sr s"
```

```
proof (intro prod_eqI ext)
```

```
fix i
```

```
show "fst (netmask (net_ips s) (initmissing (netgmap sr s))) i = fst (netgmap sr s) i"
```

```
unfolding initmissing_def by (clarsimp split: option.split)
```

```
qed (simp add: initmissing_def)
```

lemma snd_initmissing [simp]:

```
"snd (initmissing x) = snd x"
```

```
using assms unfolding initmissing_def by simp
```

lemma initmissing_snd_netgmap [simp]:

```
assumes "initmissing (netgmap sr s) = (σ, ζ)"
```

```
shows "snd (netgmap sr s) = ζ"
```

```
using assms unfolding initmissing_def by simp
```

lemma in_net_ips_fst_init_missing [simp]:

```
assumes "i ∈ net_ips s"
```

```
shows "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_oreachable_netmask [elim]:

```

assumes "initmissing (netgmap sr s) ∈ oreachable (oclosed (opnet onp n)) (λ_ _ . True) U"
      (is "_ ∈ ?oreachable n")

```

```

and "net_ips s = net_tree_ips n"

```

```

shows "netgmap sr s ∈ netmask (net_tree_ips n) ' ?oreachable n"

```

proof -

```

obtain σ ζ where "initmissing (netgmap sr s) = (σ, ζ)" by (metis surj_pair)

```

```

with assms(1) have "(σ, ζ) ∈ ?oreachable 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 ?thesis

```

```

      using 'initmissing (netgmap sr s) = (σ, ζ)' by simp

```

```

  qed simp

```

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) ∈ ?oreachable n'

```

```

  have "netmask (net_ips s) (σ, ζ) ∈ netmask (net_ips s) ' ?oreachable n"

```

```

    by (rule imageI)

```

```

ultimately show ?thesis

```

```

  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) ∈ oreachable (oclosed (opnet onp n)) (λ_ _ . True) U"

```

```

      (is "_ ∈ ?oreachable 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 ∘ 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

```

```

next
  assume "i ∉ net_tree_ips n"
  with 's ∈ init (pnet np n)' have "i ∉ net_ips s" ..
  hence "fst (initmissing (netgmap sr s)) i = someinit i" by simp
  moreover have "someinit i ∈ (fst ∘ sr) 'init (np i)'"
  unfolding someinit_def proof (rule someI_ex)
    from init_notempty show "∃x. x ∈ (fst ∘ sr) 'init (np i)'" by auto
  qed
  ultimately show "fst (initmissing (netgmap sr s)) i ∈ (fst ∘ sr) 'init (np i)'"
    by simp
qed
qed
qed
hence "initmissing (netgmap sr s) ∈ init (oclosed (opnet onp n))" by simp
thus "initmissing (netgmap sr s) ∈ ?oreachable n" ..
next
  fix s a s'
  assume "s ∈ reachable (closed (pnet np n)) TT"
    and "(s, a, s') ∈ trans (closed (pnet np n))"
    and "initmissing (netgmap sr s) ∈ ?oreachable n"
  from this(1) have "s ∈ 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) ∈ ?oreachable n'
    have "netgmap sr s ∈ netmask (net_tree_ips n) ' ?oreachable n"
      by (rule initmissing_oreachable_netmask)

  obtain σ ζ where "(σ, ζ) = initmissing (netgmap sr s)" by (metis surj_pair)
  with 'initmissing (netgmap sr s) ∈ ?oreachable n'
    have "(σ, ζ) ∈ ?oreachable n" by simp
  from '(σ, ζ) = initmissing (netgmap sr s)' and 'net_ips s = net_tree_ips n' [symmetric]
    have "netgmap sr s = netmask (net_tree_ips n) (σ, ζ)"
      by (clarsimp simp add: netmask_initmissing_netgmap)

  with 's ∈ reachable (closed (pnet np n)) TT'
    obtain σ' ζ' where "((σ, ζ), a, (σ', ζ')) ∈ trans (oclosed (opnet onp n))"
      and "netgmap sr s' = netmask (net_tree_ips n) (σ', ζ)'"
    using 'wf_net_tree n' and '(s, a, s') ∈ trans (closed (pnet np n))'
      by (rule transfer_action)

  from '(σ, ζ) ∈ ?oreachable n' have "net_ips ζ = net_tree_ips n"
    by (rule opnet_net_ips_net_tree_ips [OF oclosed_oreachable_oreachable])
  with '((σ, ζ), a, (σ', ζ')) ∈ trans (oclosed (opnet onp n))'
    have "∀j. j ∉ net_tree_ips n → σ' j = σ j"
      by (clarsimp elim!: ocomplete_no_change)
  have "initmissing (netgmap sr s') = (σ', ζ)'"
  proof (intro prod_eqI ext)
    fix i
    from 'netgmap sr s' = netmask (net_tree_ips n) (σ', ζ)''
      '∀j. j ∉ net_tree_ips n → σ' j = σ j'
      '(σ, ζ) = initmissing (netgmap sr s)''
      'net_ips s = net_tree_ips n'
    show "fst (initmissing (netgmap sr s')) i = fst (σ', ζ') i"
      unfolding initmissing_def by simp
  next
    from 'netgmap sr s' = netmask (net_tree_ips n) (σ', ζ)''
      show "snd (initmissing (netgmap sr s')) = snd (σ', ζ)'" by simp
  qed
  moreover from '(σ, ζ) ∈ ?oreachable n' '((σ, ζ), a, (σ', ζ')) ∈ trans (oclosed (opnet onp n))'
    have "(σ', ζ') ∈ ?oreachable n"
      by (rule oreachable_local) (rule TrueI)

  ultimately show "initmissing (netgmap sr s') ∈ ?oreachable n"
    by simp
qed

```

definition

```
netglobal :: "(nat ⇒ 'g) ⇒ bool) ⇒ 's net_state ⇒ bool"
```

where

```
"netglobal P ≡ (λs. P (fst (initmissing (netgmap sr s))))"
```

lemma netglobalsimp [simp]:

```
"netglobal P s = P (fst (initmissing (netgmap sr s)))"
unfolding netglobal_def by simp
```

lemma netglobale [elim]:

```
assumes "netglobal P s"
  and "∧σ. [ P σ; fst (initmissing (netgmap sr s)) = σ ] ⇒ Q σ"
shows "netglobal Q s"
using assms by simp
```

lemma netglobal_weakenE [elim]:

```
assumes "p ⊨ netglobal P"
  and "∧σ. P σ ⇒ Q σ"
shows "p ⊨ 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:

```
assumes "wf_net_tree n"
  and "oclosed (opnet onp n) ⊨ (λ_ _ . True, U →) global P"
shows "closed (pnet np n) ⊨ netglobal P"
unfolding invariant_def proof
  fix s
  assume "s ∈ reachable (closed (pnet np n)) TT"
  with assms(1)
  have "initmissing (netgmap sr s) ∈ oreachable (oclosed (opnet onp n)) (λ_ _ . True) U"
  by (rule pnet_reachable_transfer)
  with assms(2) have "global P (initmissing (netgmap sr s))" ..
  thus "netglobal P s" by simp
qed
```

end**locale openproc_parq =**

```
op: openproc np onp sr for np :: "ip ⇒ ('s, ('m::msg) seq_action) automaton" and onp sr
+ fixes qp :: "('t, 'm seq_action) automaton"
  assumes init_qp_notempty: "init qp ≠ {}"
```

```
sublocale openproc_parq ⊆ openproc "λi. np i ⟨⟨ qp"
  "λi. onp i ⟨⟨i qp"
  "λ(p, q). (fst (sr p), (snd (sr p), q))"
```

proof unfold_locales

```
fix i
show "{ (σ, ζ) | σ ζ s. s ∈ init (np i ⟨⟨ qp)
  ∧ (σ i, ζ) = ((λ(p, q). (fst (sr p), (snd (sr p), q))) s)
  ∧ (∀j. j ≠ i → σ j ∈ (fst o (λ(p, q). (fst (sr p), (snd (sr p), q))))
  ' init (np j ⟨⟨ qp)) } ⊆ init (onp i ⟨⟨i qp)"
(is "?S ⊆ _")
```

proof

```
fix s
assume "s ∈ ?S"
then obtain σ p lq
  where "s = (σ, (snd (sr p), lq))"
  and "lq ∈ init qp"
  and "p ∈ init (np i)"
```

```

    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) ⟨⟨ 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) ⟨⟨ 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!: oparright)
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) ⟨⟨ qp ⟩⟩ ≠ {}"
  by (clarsimp simp add: init_notempty init_qp_notempty)
qed

```

end

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  $\equiv$   $\lambda(d,dip). \text{Newpkt } d \text{ dip}"$ "
definition eq_newpkt_def: "eq_newpkt  $m \equiv$  case  $m$  of  $\text{Newpkt } d \text{ dip} \Rightarrow \text{True} \mid \_ \Rightarrow \text{False}"$ "

instance by intro_classes (simp add: eq_newpkt_def)
end

definition pkt :: "nat  $\times$  nat  $\Rightarrow$  msg"
where "pkt  $\equiv$   $\lambda(no, sip). \text{Pkt } no \text{ sip}"$ "

lemma pkt_simp [simp]:
  "pkt( $no, sip$ ) = Pkt  $no \text{ sip}"$ "
  unfolding pkt_def by simp

lemma not_eq_newpkt_pkt [simp]: " $\neg \text{eq\_newpkt } (\text{Pkt } no \text{ sip})"$ "
  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  $\Rightarrow$  state"
where "toy_init  $i \equiv$  (
  ip =  $i$ ,
  no = 0,
  nhip =  $i$ ,

  msg = (SOME  $x. \text{True}$ ),
  num = (SOME  $x. \text{True}$ ),
  sip = (SOME  $x. \text{True}$ )
)"
```

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

```
definition clear_locals :: "state ⇒ state"
where "clear_locals ξ = ξ (|
  msg      := (SOME x. True),
  num      := (SOME x. True),
  sip      := (SOME x. True)
|)"
```

```
lemma clear_locals_but_not_globals [simp]:
  "ip (clear_locals ξ) = ip ξ"
  "no (clear_locals ξ) = no ξ"
  "nhip (clear_locals ξ) = nhip ξ"
  unfolding clear_locals_def by auto
```

```
definition is_newpkt
where "is_newpkt ξ ≡ case msg ξ of
      Newpkt data dip ⇒ { ξ(|num := data|) }
    | _ ⇒ {}"
```

```
definition is_pkt
where "is_pkt ξ ≡ case msg ξ of
      Pkt num' sip' ⇒ { ξ(| num := num', sip := sip' |) }
    | _ ⇒ {}"
```

```
lemmas is_msg_defs =
  is_pkt_def is_newpkt_def
```

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

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

```
lemma is_msg_inv_no [simp]:
  "ξ' ∈ is_pkt ξ    ⇒ no ξ' = no ξ"
  "ξ' ∈ is_newpkt ξ ⇒ no ξ' = no ξ"
  unfolding is_msg_defs
  by (cases "msg ξ", clarsimp+)
```

```
lemma is_msg_inv_nhip [simp]:
  "ξ' ∈ is_pkt ξ    ⇒ nhip ξ' = nhip ξ"
  "ξ' ∈ is_newpkt ξ ⇒ nhip ξ' = nhip ξ"
  unfolding is_msg_defs
  by (cases "msg ξ", clarsimp+)
```

```
lemma is_msg_inv_msg [simp]:
  "ξ' ∈ is_pkt ξ    ⇒ msg ξ' = msg ξ"
  "ξ' ∈ is_newpkt ξ ⇒ msg ξ' = msg ξ"
  unfolding is_msg_defs
  by (cases "msg ξ", clarsimp+)
```

```
datatype pseqp =
  PToy
```

```
fun nat_of_seqp :: "pseqp ⇒ nat"
where
```

```

"nat_of_seqp PToy = 1"

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

abbreviation Toy
where
  "Toy ≡ λ_. [[clear_locals]] call(PToy)"

fun ΓTOY :: "(state, msg, pseqp, pseqp label) seqp_env"
where
  "ΓTOY 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>
      Toy()
    )
  ))"

declare ΓTOY.simps [simp del, code del]
lemmas ΓTOY.simps [simp, code] = ΓTOY.simps [simplified]

fun ΓTOY_skeleton
where "ΓTOY_skeleton PToy = seqp_skeleton (ΓTOY PToy)"

lemma ΓTOY_skeleton_wf [simp]:
  "wellformed ΓTOY_skeleton"
proof (rule, intro allI)
  fix pn pn'
  show "call(pn') ∉ stermsl (ΓTOY_skeleton pn)"
  by (cases pn) simp_all
qed

declare ΓTOY_skeleton.simps [simp del, code del]
lemmas ΓTOY_skeleton.simps [simp, code] = ΓTOY_skeleton.simps [simplified ΓTOY.simps seqp_skeleton.simps]

lemma toy_proc_cases [dest]:
  fixes p pn
  assumes "p ∈ ctermsl (ΓTOY pn)"
  shows "p ∈ ctermsl (ΓTOY PToy)"
using assms
by (cases pn) simp_all

definition σTOY :: "ip ⇒ (state × (state, msg, pseqp, pseqp label) seqp) set"
where "σTOY i ≡ {(toy_init i, ΓTOY PToy)}"

abbreviation ptoy
  :: "ip ⇒ (state × (state, msg, pseqp, pseqp label) seqp, msg seq_action) automaton"
where

```

```

"ptoy i  $\equiv$  ( $\mid$  init =  $\sigma_{TOY}$  i, trans = seqp_sos  $\Gamma_{TOY}$   $\mid$ )"

lemma toy_trans: "trans (ptoy i) = seqp_sos  $\Gamma_{TOY}$ "
  by simp

lemma toy_control_within [simp]: "control_within  $\Gamma_{TOY}$  (init (ptoy i))"
  unfolding  $\sigma_{TOY\_def}$  by (rule control_withinI) (auto simp del:  $\Gamma_{TOY\_simps}$ )

lemma toy_wf [simp]:
  "wellformed  $\Gamma_{TOY}$ "
  proof (rule, intro allI)
    fix pn pn'
    show "call(pn')  $\notin$  sterms1 ( $\Gamma_{TOY}$  pn)"
      by (cases pn) simp_all
  qed

lemmas toy_labels_not_empty [simp] = labels_not_empty [OF toy_wf]

lemma toy_ex_label [intro]: " $\exists$  l. l  $\in$  labels  $\Gamma_{TOY}$  p"
  by (metis toy_labels_not_empty all_not_in_conv)

lemma toy_ex_labelE [elim]:
  assumes " $\forall$  l  $\in$  labels  $\Gamma_{TOY}$  p. P l p"
    and " $\exists$  p l. P l p  $\implies$  Q"
  shows "Q"
  using assms by (metis toy_ex_label)

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

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

By default, we no longer let the simplifier descend into process terms.

declare seqp_congs [cong]

declare
   $\Gamma_{TOY\_simps}$  [cterms_env]
  toy_proc_cases [cterms1_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  $\sigma_{OTOY} :: ((ip \implies state) \times ((state, msg, pseqp, pseqp label) seqp)) set$ 
  where " $\sigma_{OTOY} \equiv \{(toy\_init, \Gamma_{TOY} PToy)\}$ "

abbreviation optoy
  :: " $ip \implies ((ip \implies state) \times (state, msg, pseqp, pseqp label) seqp, msg seq\_action)$  automaton"
  where
    "optoy i  $\equiv$  ( $\mid$  init =  $\sigma_{OTOY}$ , trans = oseqp_sos  $\Gamma_{TOY}$  i  $\mid$ )"

lemma initiali_toy [intro!, simp]: "initiali i (init (optoy i)) (init (ptoy i))"
  unfolding  $\sigma_{TOY\_def}$   $\sigma_{OTOY\_def}$  by rule simp_all

lemma oaadv_control_within [simp]: "control_within  $\Gamma_{TOY}$  (init (optoy i))"
  unfolding  $\sigma_{OTOY\_def}$  by (rule control_withinI) (auto simp del:  $\Gamma_{TOY\_simps}$ )

```


lemma σ_{OTOY_labels} [simp]: " $(\sigma, p) \in \sigma_{OTOY} \implies labels \Gamma_{TOY} p = \{PToy-:0\}$ "
 unfolding σ_{OTOY_def} by simp

lemma $otoy_trans$: " $trans (optoy i) = oseqp_sos \Gamma_{TOY} i$ "
 by simp

declare

$oseq_invariant_ctermI$ [OF toy_wf $oaadv_control_within$ toy_simple_labels $otoy_trans$, $ctermI_intros$]
 $oseq_step_invariant_ctermI$ [OF toy_wf $oaadv_control_within$ toy_simple_labels $otoy_trans$, $ctermI_intros$]

26.4 Predicates

definition msg_sender :: " $msg \Rightarrow ip$ "
 where " $msg_sender m \equiv case m of Pkt _ ipc \Rightarrow ipc$ "

lemma msg_sender_sims [simp]:
 " $\bigwedge d sip. msg_sender (Pkt d sip) = sip$ "
 unfolding msg_sender_def by simp_all

abbreviation not_Pkt :: " $msg \Rightarrow bool$ "
 where " $not_Pkt m \equiv case m of Pkt _ _ \Rightarrow False \mid _ \Rightarrow True$ "

definition $nos_increase$:: " $state \Rightarrow state \Rightarrow bool$ "
 where " $nos_increase \xi \xi' \equiv (no \xi \leq no \xi')$ "

definition msg_num_ok :: " $(ip \Rightarrow state) \Rightarrow msg \Rightarrow bool$ "
 where " $msg_num_ok \sigma m \equiv case m of Pkt num' sip' \Rightarrow num' \leq no (\sigma sip') \mid _ \Rightarrow True$ "

lemma msg_num_okI [intro]:
 assumes " $\bigwedge num' sip'. m = Pkt num' sip' \implies num' \leq no (\sigma sip')$ "
 shows " $msg_num_ok \sigma m$ "
 using $assms$ unfolding $msg_num_ok_def$
 by (auto split: $msg.split$)

lemma $msg_num_ok_Pkt$ [simp]:
 " $msg_num_ok \sigma (Pkt data src) = (data \leq no (\sigma src))$ "
 unfolding $msg_num_ok_def$ by simp

lemma $msg_num_ok_pkt$ [simp]:
 " $msg_num_ok \sigma (pkt(data, src)) = (data \leq no (\sigma src))$ "
 unfolding $msg_num_ok_def$ by simp

lemma $msg_num_ok_Newpkt$ [simp]:
 " $msg_num_ok \sigma (Newpkt data dst)$ "
 unfolding $msg_num_ok_def$ by simp

lemma $msg_num_ok_newpkt$ [simp]:
 " $msg_num_ok \sigma (newpkt(data, dst))$ "
 unfolding $msg_num_ok_def$ by simp

26.5 Sequential Invariants

lemma $seq_no_leq_num$:
 " $ptoy i \models onl \Gamma_{TOY} (\lambda(\xi, l). l \in \{PToy-:7..PToy-:9\} \longrightarrow no \xi \leq num \xi)$ "
 by inv_ctermI

lemma $seq_nos_increases$:
 " $ptoy i \models_A onll \Gamma_{TOY} (\lambda((\xi, _), _, (\xi', _)). nos_increase \xi \xi')$ "
 unfolding $nos_increase_def$
 proof -
 show " $ptoy i \models_A onll \Gamma_{TOY} (\lambda((\xi, _), _, (\xi', _)). no \xi \leq no \xi')$ "
 by (inv_ctermI 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 onll ΓTOY (λ((ξ, _), a, _). anycast (λm. msg_sender m = ip ξ) a)"  
by inv_cterms
```

lemma ip_constant:

```
"ptoy i ⊨ onl ΓTOY (λ(ξ, _). ip ξ = i)"  
by inv_cterms (simp add: σTOY-def)
```

lemma nhip_eq_ip:

```
"ptoy i ⊨ onl ΓTOY (λ(ξ, l). l ∈ {PToy-:2..PToy-:8} → nhip ξ = ip ξ)"  
by inv_cterms
```

lemma seq_msg_num_ok:

```
"ptoy i ⊨A onll ΓTOY (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 ⊨ onl ΓTOY (λ(ξ, l). 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 "l ∈ labels ΓTOY 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 ΓTOY 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 seqll_onll_swap]
```

lemmas oseq_no_leq_num =

```
  open_seq_invariant [OF seq_no_leq_num initiali_toy otoy_trans toy_trans,  
    simplified seq1_onl_swap]
```

lemma all_nos_increase:

```
  shows "optoy i ⊨A (otherwith nos_increase {i} S,  
    other nos_increase {i} →  
    onll ΓTOY (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: " $\bigwedge \sigma \sigma' m. \llbracket P \sigma m; \text{other } Q \{i\} \sigma \sigma' \rrbracket \implies P \sigma' m$ "
    and local: " $\bigwedge \sigma m. P \sigma m \implies P (\sigma(i := \sigma i(\text{msg} := m))) m$ "
  shows "optoy  $i \models (\text{otherwith } Q \{i\} (\text{orecvmsg } P), \text{other } Q \{i\} \rightarrow)$ 
        onl  $\Gamma_{\text{TOY}} (\lambda(\sigma, l). l \in \{\text{PToy-:1}\} \rightarrow P \sigma (\text{msg} (\sigma i)))$ "
proof (inv_cterms, intro impI)
  fix  $\sigma \sigma' l$ 
  assume "l = PToy-:1  $\rightarrow P \sigma (\text{msg} (\sigma i))$ "
    and "l = PToy-:1"
    and "other  $Q \{i\} \sigma \sigma'$ "
  from this(1-2) have "P  $\sigma (\text{msg} (\sigma i))$ " ..
  hence "P  $\sigma' (\text{msg} (\sigma i))$ " using 'other  $Q \{i\} \sigma \sigma'$ '
    by (rule other)
  moreover from 'other  $Q \{i\} \sigma \sigma'$ ' have " $\sigma' i = \sigma i$ " ..
  ultimately show "P  $\sigma' (\text{msg} (\sigma' i))$ " by simp
next
  fix  $\sigma \sigma' \text{msg}$ 
  assume "otherwith  $Q \{i\} (\text{orecvmsg } P) \sigma \sigma' (\text{receive msg})$ "
    and " $\sigma' i = \sigma i(\text{msg} := \text{msg})$ "
  from this(1) have "P  $\sigma \text{msg}$ "
    and " $\forall j. j \neq i \rightarrow Q (\sigma j) (\sigma' j)$ " by auto
  from this(1) have "P ( $\sigma(i := \sigma i(\text{msg} := \text{msg}))$ )  $\text{msg}$ " by (rule local)
  thus "P  $\sigma' \text{msg}$ "
proof (rule other)
  from ' $\sigma' i = \sigma i(\text{msg} := \text{msg})$ ' and ' $\forall j. j \neq i \rightarrow Q (\sigma j) (\sigma' j)$ '
  show "other  $Q \{i\} (\sigma(i := \sigma i(\text{msg} := \text{msg}))$ )  $\sigma'$ "
    by - (rule otherI, auto)
qed
qed

lemma msg_num_ok_other_nos_increase [elim]:
  assumes "msg_num_ok  $\sigma m$ "
    and "other nos_increase  $\{i\} \sigma \sigma'$ "
  shows "msg_num_ok  $\sigma' m$ "
proof (cases m)
  fix num sip
  assume "m = Pkt num sip"
  with 'msg_num_ok  $\sigma m$ ' have "num  $\leq \text{no} (\sigma \text{sip})$ " by simp
  also from 'other nos_increase  $\{i\} \sigma \sigma'$ ' have "no ( $\sigma \text{sip}$ )  $\leq \text{no} (\sigma' \text{sip})$ "
    by (rule otherE) (metis eq_iff nos_increased)
  finally have "num  $\leq \text{no} (\sigma' \text{sip})$ " .
  with 'm = Pkt num sip' show ?thesis
    by simp
qed simp

lemma msg_num_ok_no_leq_no [simp, elim]:
  assumes "msg_num_ok  $\sigma m$ "
    and " $\forall j. \text{no} (\sigma j) \leq \text{no} (\sigma' j)$ "
  shows "msg_num_ok  $\sigma' m$ "
using assms(1) proof (cases m)
  fix num sip
  assume "m = Pkt num sip"
  with 'msg_num_ok  $\sigma m$ ' have "num  $\leq \text{no} (\sigma \text{sip})$ " by simp
  also from ' $\forall j. \text{no} (\sigma j) \leq \text{no} (\sigma' j)$ ' have "no ( $\sigma \text{sip}$ )  $\leq \text{no} (\sigma' \text{sip})$ "
    by simp
  finally have "num  $\leq \text{no} (\sigma' \text{sip})$ " .
  with 'm = Pkt num sip' show ?thesis

```

```

    by simp
qed (simp add: assms(1))

```

lemma oreceived_msg_num_ok:

```

"optoy i  $\models$  (otherwith nos_increase {i} (orecvmsg msg_num_ok),
  other nos_increase {i}  $\rightarrow$ )
  onl  $\Gamma_{TOY}$  ( $\lambda(\sigma, l). l \in \{PToy-:1..\}$   $\rightarrow$  msg_num_ok  $\sigma$  (msg ( $\sigma$  i)))"
(is "_  $\models$  (?S, ?U  $\rightarrow$ ) _")
proof (inv_cterms inv add: oseq_step_invariant_sterms [OF all_nos_increase toy_wf otoy_trans],
  intro impI, elim impE)
  fix  $\sigma$   $\sigma'$ 
  assume "msg_num_ok  $\sigma$  (msg ( $\sigma$  i))"
  and "other nos_increase {i}  $\sigma$   $\sigma'$ "
  moreover from this(2) have "msg ( $\sigma'$  i) = msg ( $\sigma$  i)"
  by (clarsimp elim!: otherE)
  ultimately show "msg_num_ok  $\sigma'$  (msg ( $\sigma'$  i))"
  by (auto)
next
  fix p l  $\sigma$  a q l'  $\sigma'$  pp p' m
  assume a1: "( $\sigma', p') \in$  oreachable (optoy i) ?S ?U"
  and a2: "PToy-:1  $\in$  labels  $\Gamma_{TOY}$  p'"
  and a3: " $\sigma' i = \sigma i$  (msg := m)"
  have inv: "optoy i  $\models$  (?S, ?U  $\rightarrow$ ) onl  $\Gamma_{TOY}$  ( $\lambda(\sigma, l). l \in \{PToy-:1\}$   $\rightarrow$  msg_num_ok  $\sigma$  (msg ( $\sigma$  i)))"
  proof (rule oreceived_msg_inv)
    fix  $\sigma$   $\sigma'$  m
    assume "msg_num_ok  $\sigma$  m"
    and "other nos_increase {i}  $\sigma$   $\sigma'$ "
    thus "msg_num_ok  $\sigma'$  m" ..
  next
    fix  $\sigma$  m
    assume "msg_num_ok  $\sigma$  m"
    thus "msg_num_ok ( $\sigma(i := \sigma i$  (msg := m))) m"
    by (cases m) auto
  qed
  from a1 a2 a3 show "msg_num_ok  $\sigma'$  m"
  by (clarsimp dest!: oinvariantD [OF inv] onlD)
qed simp

```

lemma is_pkt_handler_num_leq_no:

```

shows "optoy i  $\models$  (otherwith nos_increase {i} (orecvmsg msg_num_ok),
  other nos_increase {i}  $\rightarrow$ )
  onl  $\Gamma_{TOY}$  ( $\lambda(\sigma, l). l \in \{PToy-:6..PToy-:10\}$   $\rightarrow$  num ( $\sigma$  i)  $\leq$  no ( $\sigma$  (sip ( $\sigma$  i))))"
proof -
  { fix  $\sigma$   $\sigma'$ 
    assume " $\forall j. no$  ( $\sigma$  j)  $\leq$  no ( $\sigma'$  j)"
    and "num ( $\sigma$  i)  $\leq$  no ( $\sigma$  (sip ( $\sigma$  i)))"
    have "num ( $\sigma$  i)  $\leq$  no ( $\sigma'$  (sip ( $\sigma$  i)))"
    proof -
      note ' $num$  ( $\sigma$  i)  $\leq$  no ( $\sigma$  (sip ( $\sigma$  i)))'
      also from ' $\forall j. no$  ( $\sigma$  j)  $\leq$  no ( $\sigma'$  j)' have "no ( $\sigma$  (sip ( $\sigma$  i)))  $\leq$  no ( $\sigma'$  (sip ( $\sigma$  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  $\sigma$   $\sigma'$ 
    assume *: "num ( $\sigma$  i)  $\leq$  no ( $\sigma$  (sip ( $\sigma$  i)))"
    and "other nos_increase {i}  $\sigma$   $\sigma'$ "
    from this(2) obtain " $\forall i \in \{i\}. \sigma' i = \sigma i$ "
    and " $\forall j. j \notin \{i\} \rightarrow nos\_increase$  ( $\sigma$  j) ( $\sigma'$  j)" ..
    show "num ( $\sigma'$  i)  $\leq$  no ( $\sigma'$  (sip ( $\sigma'$  i)))"

```

```

proof (cases "sip (σ i) = i")
  assume "sip (σ i) = i"
  with * '∀i∈{i}. σ' i = σ i'
  show ?thesis by simp
next
  assume "sip (σ i) ≠ i"
  with '∀j. j ∉ {i} → nos_increase (σ j) (σ' j)'
  have "no (σ (sip (σ i))) ≤ no (σ' (sip (σ i)))" by simp
  with * '∀i∈{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) ≤ no (σ' j)"
  and "σ' i ∈ is_pkt (σ i)"
show "num (σ' i) ≤ 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) ≤ no (σ (sip (σ' i)))"
    by simp
  also from '∀j. no (σ j) ≤ no (σ' j)' have "no (σ (sip (σ' i))) ≤ 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
qed

```

```

lemmas oseq_ip_constant =
  open_seq_invariant [OF ip_constant initiali_toy otoy_trans toy_trans,
    simplified seq1_onl_swap]

```

```

lemmas oseq_nhip_eq_i =
  open_seq_invariant [OF nhip_eq_i initiali_toy otoy_trans toy_trans,
    simplified seq1_onl_swap]

```

```

lemmas oseq_nhip_eq_ip =
  open_seq_invariant [OF nhip_eq_ip initiali_toy otoy_trans toy_trans,
    simplified seq1_onl_swap]

```

```

lemma oseq_bigger_than_next:
  shows "optoy i ⊨ (otherwith nos_increase {i} (orecvmsg msg_num_ok),
    other nos_increase {i} →) global (λσ. no (σ i) ≤ no (σ (nhip (σ i))))"
  (is "_ ⊨ (?S, ?U →) ?P")
proof -
  have nhipinv: "optoy i ⊨ (?S, ?U →)
    onl ΓTOY (λ(σ, l). l∈{PToy-:2..PToy-:8}
      → nhip (σ i) = ip (σ i))"
  by (rule oinvariant_weakenE [OF oseq_nhip_eq_ip]) (auto simp: seqlsimp)
  have ipinv: "optoy i ⊨ (?S, ?U →) onl ΓTOY (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) ≤ 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)))" ..
} note * = this
have "optoy i ⊨ (otherwith nos_increase {i} (orecvmsg 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_oinvariant_sterms [OF toy_wf oseq_no_leq_num [THEN oinvariant_anyact]]
  oseq_step_invariant_sterms [OF all_nos_increase toy_wf otoy_trans]
  onl_oinvariant_sterms [OF toy_wf is_pkt_handler_num_leq_no]
  onl_oinvariant_sterms [OF toy_wf nhipinv]
  onl_oinvariant_sterms [OF toy_wf ipinv]
  simp add: seqlsimp seqllsimp
  simp del: nos_increase_simp
  solve: *)
fix σ p l
assume "(σ, p) ∈ σOTOY"
thus "no (σ i) ≤ no (σ (nhip (σ i)))"
  by (simp add: σOTOY_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)))"
  and "other nos_increase {i} σ σ'"
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 σ 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 oinvariant_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  $\models_A$  (act TT, other U {i}  $\rightarrow$ ) globala ( $\lambda(\sigma, a, \_)$ . anycast (msg_num_ok  $\sigma$ ) a)"
  by (rule ostep_invariant_weakenE [OF open_seq_step_invariant
    [OF seq_msg_num_ok initiali_toy otoy_trans toy_trans, simplified seq_l_onl_swap]])
    (auto simp: seqllsimp dest!: onl1D elim!: anycast_weakenE intro!: msg_num_okI)

```

26.7 Lifting

```

lemma opar_bigger_than_next:
  shows "optoy i  $\langle\langle_i$  qmsg  $\models$  (otherwith nos_increase {i} (orecvmsg msg_num_ok),
    other nos_increase {i}  $\rightarrow$ ) global ( $\lambda\sigma$ . no ( $\sigma$  i)  $\leq$  no ( $\sigma$  (nhip ( $\sigma$  i)))))"
  proof (rule lift_into_qmsg [OF oseq_bigger_than_next])
    fix  $\sigma$   $\sigma'$  m
    assume " $\forall j$ . nos_increase ( $\sigma$  j) ( $\sigma'$  j)"
      and "msg_num_ok  $\sigma$  m"
    from this(2) show "msg_num_ok  $\sigma'$  m"
    proof (cases m, simp only: msg_num_ok_Pkt)
      fix num' sip'
      assume "num'  $\leq$  no ( $\sigma$  sip)"
      also from ' $\forall j$ . nos_increase ( $\sigma$  j) ( $\sigma'$  j)' have "no ( $\sigma$  sip)  $\leq$  no ( $\sigma'$  sip)"
        by simp
      finally show "num'  $\leq$  no ( $\sigma'$  sip)" .
    qed simp
  qed simp
next
  show "optoy i  $\models_A$  (otherwith nos_increase {i} (orecvmsg msg_num_ok), other nos_increase {i}  $\rightarrow$ )
    globala ( $\lambda(\sigma, \_, \sigma')$ . nos_increase ( $\sigma$  i) ( $\sigma'$  i))"
  by (rule ostep_invariant_weakenE [OF open_seq_step_invariant
    [OF seq_nos_increases initiali_toy otoy_trans toy_trans]])
    (auto simp: seqllsimp dest!: onl1D)
  qed simp

```

```

lemma onode_bigger_than_next:
  " $\langle_i$  : optoy i  $\langle\langle_i$  qmsg :  $R_i$  $\rangle_o$ 
   $\models$  (otherwith nos_increase {i} (oarrivemsg msg_num_ok), other nos_increase {i}  $\rightarrow$ )
    global ( $\lambda\sigma$ . no ( $\sigma$  i)  $\leq$  no ( $\sigma$  (nhip ( $\sigma$  i))))"
  by (rule node_lift [OF opar_bigger_than_next])

```

```

lemma node_local_nos_increase:
  " $\langle_i$  : optoy i  $\langle\langle_i$  qmsg :  $R_i$  $\rangle_o$   $\models_A$  ( $\lambda\sigma$   $\_$ . oarrivemsg ( $\lambda\sigma$   $\_$ . True)  $\sigma$ , other ( $\lambda\sigma$   $\_$ . True) {i}  $\rightarrow$ )
    globala ( $\lambda(\sigma, \_, \sigma')$ . nos_increase ( $\sigma$  i) ( $\sigma'$  i))"
  proof (rule node_lift_step_statelessassm)
    have "optoy i  $\models_A$  ( $\lambda\sigma$   $\_$ . orecvmsg ( $\lambda\sigma$   $\_$ . True)  $\sigma$ , other ( $\lambda\sigma$   $\_$ . True) {i}  $\rightarrow$ )
      globala ( $\lambda(\sigma, \_, \sigma')$ . nos_increase ( $\sigma$  i) ( $\sigma'$  i))"
    by (rule ostep_invariant_weakenE [OF oseq_nos_increases])
      (auto simp: seqllsimp dest!: onl1D)
    thus "optoy i  $\langle\langle_i$  qmsg  $\models_A$  ( $\lambda\sigma$   $\_$ . orecvmsg ( $\lambda\sigma$   $\_$ . True)  $\sigma$ , other ( $\lambda\sigma$   $\_$ . True) {i}  $\rightarrow$ )
      globala ( $\lambda(\sigma, \_, \sigma')$ . nos_increase ( $\sigma$  i) ( $\sigma'$  i))"
    by (rule lift_step_into_qmsg_statelessassm) auto
  qed simp

```

```

lemma opnet_bigger_than_next:
  "opnet ( $\lambda i$ . optoy i  $\langle\langle_i$  qmsg) n
   $\models$  (otherwith nos_increase (net_tree_ips n) (oarrivemsg msg_num_ok),
    other nos_increase (net_tree_ips n)  $\rightarrow$ )
    global ( $\lambda\sigma$ .  $\forall i \in$  net_tree_ips n. no ( $\sigma$  i)  $\leq$  no ( $\sigma$  (nhip ( $\sigma$  i))))"
  proof (rule pnet_lift [OF onode_bigger_than_next])
    fix i  $R_i$ 
    have " $\langle_i$  : optoy i  $\langle\langle_i$  qmsg :  $R_i$  $\rangle_o$   $\models_A$  ( $\lambda\sigma$   $\_$ . oarrivemsg msg_num_ok  $\sigma$ , other ( $\lambda\sigma$   $\_$ . True) {i}  $\rightarrow$ )
      globala ( $\lambda(\sigma, a, \_)$ . castmsg (msg_num_ok  $\sigma$ ) a)"
    proof (rule node_lift_anycast_statelessassm)
      have "optoy i  $\models_A$  ( $\lambda\sigma$   $\_$ . orecvmsg ( $\lambda\sigma$   $\_$ . True)  $\sigma$ , other ( $\lambda\sigma$   $\_$ . True) {i}  $\rightarrow$ )
        globala ( $\lambda(\sigma, a, \_)$ . anycast (msg_num_ok  $\sigma$ ) a)"
      by (rule ostep_invariant_weakenE [OF oseq_msg_num_ok]) auto
    qed
  qed

```

```

hence "optoy i <<i qmsg ⊢A (λσ -. orecvmsg (λ_ -. True) σ, other (λ_ -. True) {i} →)
      globala (λ(σ, a, _). anycast (msg_num_ok σ) a)"
  by (rule lift_step_into_qmsg_statelessassm) auto
thus "optoy i <<i qmsg ⊢A (λσ -. orecvmsg msg_num_ok σ, other (λ_ -. True) {i} →)
      globala (λ(σ, a, _). anycast (msg_num_ok σ) a)"
  by (rule ostep_invariant_weakenE) auto
qed
thus "<i : optoy i <<i qmsg : Ri>o ⊢A (λσ -. oarrivemsg msg_num_ok σ, other nos_increase {i} →)
      globala (λ(σ, a, _). castmsg (msg_num_ok σ) a)"
  by (rule ostep_invariant_weakenE) auto
next
fix i Ri
show "<i : optoy i <<i qmsg : Ri>o ⊢A (λσ -. oarrivemsg msg_num_ok σ,
      other nos_increase {i} →)
      globala (λ(σ, a, σ'). a ≠ τ ∧ (∀i d. a ≠ i:deliver(d)) → nos_increase (σ i) (σ' i))"
  by (rule ostep_invariant_weakenE [OF node_local_nos_increase]) auto
next
fix i R
show "<i : optoy i <<i qmsg : R>o ⊢A (λσ -. oarrivemsg msg_num_ok σ,
      other nos_increase {i} →)
      globala (λ(σ, a, σ'). a = τ ∨ (∃d. a = i:deliver(d)) → nos_increase (σ i) (σ' i))"
  by (rule ostep_invariant_weakenE [OF node_local_nos_increase]) auto
qed simp_all

lemma ocnet_bigger_than_next:
"oclosed (opnet (λi. optoy i <<i qmsg) n)
 ⊢ (λ_ _ . True, other nos_increase (net_tree_ips n) →)
  global (λσ. ∀i∈net_tree_ips n. no (σ i) ≤ no (σ (nhip (σ i))))"
proof (rule inclosed_closed)
show "opnet (λi. optoy i <<i qmsg) n
 ⊢ (otherwith op = (net_tree_ips n) inoclosed, other nos_increase (net_tree_ips n) →)
  global (λσ. ∀i∈net_tree_ips n. no (σ i) ≤ no (σ (nhip (σ i))))"
proof (rule oinvariant_weakenE [OF opnet_bigger_than_next])
fix s s':: "nat ⇒ state" and a :: "msg node_action"
assume "otherwith op = (net_tree_ips n) inoclosed s s' a"
thus "otherwith nos_increase (net_tree_ips n) (oarrivemsg msg_num_ok) s s' a"
proof (rule otherwithE, intro otherwithI)
assume "inoclosed s a"
and "∀j. j ∉ net_tree_ips n → s j = s' j"
and "otherwith (op=) (net_tree_ips n) inoclosed s s' a"
thus "oarrivemsg msg_num_ok s a"
  by (cases a) auto
qed auto
qed simp
qed

```

26.8 Transfer

definition

```
initmissing :: "(nat ⇒ state option) × 'a ⇒ (nat ⇒ state) × 'a"
```

where

```
"initmissing σ = (λi. case (fst σ) i of None ⇒ toy_init i | Some s ⇒ s, snd σ)"
```

lemma not_in_net_ips_fst_init_missing [simp]:

```
assumes "i ∉ net_ips σ"
```

```
shows "fst (initmissing (netgmap fst σ)) i = toy_init i"
```

```
using assms unfolding initmissing_def by simp
```

lemma fst_initmissing_netgmap_pair_fst [simp]:

```
"fst (initmissing (netgmap (λ(p, q). (fst (id p), snd (id p), q)) s))"
```

```
= fst (initmissing (netgmap fst s))"
```

```
unfolding initmissing_def by auto
```

interpretation toy_openproc: openproc ptoy 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)} ⊆ σOTOY"
      unfolding σTOY_def σOTOY_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 opq.initmissing_def opq.someinit_def initmissing_def
unfolding  $\sigma_{TOY\_def}$   $\sigma_{QMSG\_def}$  by (clarsimp cong: option.case_cong)
thus "openproc.initmissing ( $\lambda i.$  ptoy i  $\langle\langle$  qmsg) ( $\lambda(p, q).$  (fst (id p), snd (id p), q)) = initmissing"
  by (rule ext)

have " $\bigwedge P \sigma.$  openproc.netglobal ( $\lambda i.$  ptoy i  $\langle\langle$  qmsg) ( $\lambda(p, q).$  (fst (id p), snd (id p), q)) P  $\sigma$ 
  = netglobal P  $\sigma$ "

unfolding opq.netglobal_def netglobal_def opq.initmissing_def initmissing_def opq.someinit_def
unfolding  $\sigma_{TOY\_def}$   $\sigma_{QMSG\_def}$ 
by (clarsimp cong: option.case_cong
    simp del: One_nat_def
    simp add: fst_initmissing_netgmap_default_toy_init_netlift
    [symmetric, unfolded initmissing_def])
thus "openproc.netglobal ( $\lambda i.$  ptoy i  $\langle\langle$  qmsg) ( $\lambda(p, q).$  (fst (id p), snd (id p), q)) = netglobal"
  by auto
qed

```

26.9 Final result

```

lemma bigger_than_next:
  assumes "wf_net_tree n"
  shows "closed (pnet ( $\lambda i.$  ptoy i  $\langle\langle$  qmsg) n)  $\models$  netglobal ( $\lambda\sigma.$   $\forall i.$  no ( $\sigma$  i)  $\leq$  no ( $\sigma$  (nhip ( $\sigma$  i))))"
    (is "_  $\models$  netglobal ( $\lambda\sigma.$   $\forall i.$  ?inv  $\sigma$  i)")
  proof -
    from 'wf_net_tree n'
      have proto: "closed (pnet ( $\lambda i.$  ptoy i  $\langle\langle$  qmsg) n)
         $\models$  netglobal ( $\lambda\sigma.$   $\forall i \in \text{net\_tree\_ips } n.$  no ( $\sigma$  i)  $\leq$  no ( $\sigma$  (nhip ( $\sigma$  i))))"
        by (rule toy_openproc_par_qmsg.close_opnet [OF _ ocnet_bigger_than_next])
    show ?thesis
  unfolding invariant_def opnet_sos.opnet_tau1
  proof (rule, simp only: toy_openproc_par_qmsg.netglobalsimp
    fst_initmissing_netgmap_pair_fst, rule allI)
    fix  $\sigma$  i
    assume sr: " $\sigma \in \text{reachable (closed (pnet ( $\lambda i.$  ptoy i  $\langle\langle$  qmsg) n)) TT}$ "
    hence " $\forall i \in \text{net\_tree\_ips } n.$  ?inv (fst (initmissing (netgmap fst  $\sigma$ ))) i"
      by - (drule invariantD [OF proto],
        simp only: toy_openproc_par_qmsg.netglobalsimp
        fst_initmissing_netgmap_pair_fst)
    thus "?inv (fst (initmissing (netgmap fst  $\sigma$ ))) i"
  proof (cases "i  $\in$  net_tree_ips n")
    assume "i  $\notin$  net_tree_ips n"
    from sr have " $\sigma \in \text{reachable (pnet ( $\lambda i.$  ptoy i  $\langle\langle$  qmsg) n) TT}$ " ..
    hence "net_ips  $\sigma$  = net_tree_ips n" ..
    with 'i  $\notin$  net_tree_ips n' have "i  $\notin$  net_ips  $\sigma$ " by simp
    hence "(fst (initmissing (netgmap fst  $\sigma$ ))) i = 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

- [1] T. Bourke, R. J. van Glabbeek, and P. Höfner. Mechanizing node and network invariants based on process algebra, 2014. Submitted for publication.

- [2] A. Fehnker, R. J. van Glabbeek, P. Höfner, A. McIver, M. Portmann, and W. L. Tan. A process algebra for wireless mesh networks used for modelling, verifying and analysing AODV. Technical Report 5513, NICTA, 2013.