

The pi-calculus

Jesper Bengtson

February 6, 2026

Abstract

We formalise the pi-calculus using the nominal datatype package, based on ideas from the nominal logic by Pitts et al., and demonstrate an implementation in Isabelle/HOL. The purpose is to derive powerful induction rules for the semantics in order to conduct machine checkable proofs, closely following the intuitive arguments found in manual proofs. In this way we have covered many of the standard theorems of bisimulation equivalence and congruence, both late and early, and both strong and weak in a uniform manner. We thus provide one of the most extensive formalisations of a the pi-calculus ever done inside a theorem prover.

A significant gain in our formulation is that agents are identified up to alpha-equivalence, thereby greatly reducing the arguments about bound names. This is a normal strategy for manual proofs about the pi-calculus, but that kind of hand waving has previously been difficult to incorporate smoothly in an interactive theorem prover. We show how the nominal logic formalism and its support in Isabelle accomplishes this and thus significantly reduces the tedium of conducting completely formal proofs. This improves on previous work using weak higher order abstract syntax since we do not need extra assumptions to filter out exotic terms and can keep all arguments within a familiar first-order logic.

Contents

1	Overview	1
2	Formalisation	2

1 Overview

The following results of the pi-calculus meta-theory are formalised, where the notation (e) means that the results cover the early operational semantics and (l) the late one.

- strong bisimilarity is preserved by all operators except the input-prefix (e/l)
- strong equivalence is a congruence (e/l)
- weak bisimilarity is preserved by all operators except the input-prefix and sum (e/l)
- weak congruence is a congruence (e/l)
- strong equivalence respect the laws of structural congruence (l)
- all strongly equivalent agents are also weakly congruent which in turn are weakly bisimilar. Moreover, strongly equivalent agents are also strongly bisimilar (e/l)
- all late equivalences are included in their early counterparts.
- as a corollary of the last three points, all mentioned equivalences respect the laws of structural congruence
- the axiomatisation of the finite fragment of strong late bisimilarity is sound and complete
- The Hennessy lemma (l)

The file naming convention is hopefully self-explanatory, where the prefixes *Strong* and *Weak* denote that the file covers theories required to formalise properties of strong and weak bisimilarity respectively; if the file name contains *Early* or *Late* the theories work with the early or the late operational semantics of the pi-calculus respectively; if the file name contains *Sim* the theories cover simulation, file names containing *Bisim* cover bisimulation, and file names containing *Cong* cover weak congruence; files with the suffix *Pres* deal with theories that reason about preservation properties of operators such as a certain simulation or bisimulation being preserved by a certain operator; files with the suffix *SC* reason about structural congruence.

For a complete exposition of all of theories, please consult Bengtson's Ph. D. thesis [1]. A shorter presentation can be found in our LMCS article 'Formalising the pi-calculus using nominal logic' from 2009 [3]. A recollection of the axiomatisation results can be found in the SOS article 'A completeness proof for bisimulation in the pi-calculus using Isabelle' from 2007 [2].

2 Formalisation

```
theory Agent
  imports HOL-Nominal.Nominal
```

begin

lemma *pt-id*:

fixes $x :: 'a$
and $a :: 'x$

assumes $pt: pt \text{ TYPE}('a) \text{ TYPE}('x)$
and $at: at \text{ TYPE}('x)$
shows $[(a, a)] \cdot x = x$

proof –

have $x = ([::'x \text{ prm}]) \cdot x$
by (*simp add: pt1* [*OF pt*])
also have $[(a, a)] \cdot x = ([::'x \text{ prm}]) \cdot x$
by (*simp add: pt3* [*OF pt*] *at-ds1* [*OF at*])
finally show *?thesis* **by** *simp*

qed

lemma *pt-swap*:

fixes $x :: 'a$
and $a :: 'x$
and $b :: 'x$

assumes $pt: pt \text{ TYPE}('a) \text{ TYPE}('x)$
and $at: at \text{ TYPE}('x)$

shows $[(a, b)] \cdot x = [(b, a)] \cdot x$

proof –

show *?thesis* **by** (*simp add: pt3* [*OF pt*] *at-ds5* [*OF at*])

qed

atom-decl *name*

lemmas *name-fresh-abs* = *fresh-abs-fun-iff* [*OF pt-name-inst*, *OF at-name-inst*,
OF fs-name1]

lemmas *name-bij* = *at-bij* [*OF at-name-inst*]

lemmas *name-supp-abs* = *abs-fun-supp* [*OF pt-name-inst*, *OF at-name-inst*, *OF*
fs-name1]

lemmas *name-abs-eq* = *abs-fun-eq* [*OF pt-name-inst*, *OF at-name-inst*]

lemmas *name-supp* = *at-supp* [*OF at-name-inst*]

lemmas *name-calc* = *at-calc* [*OF at-name-inst*]

lemmas *name-fresh-fresh* = *pt-fresh-fresh* [*OF pt-name-inst*, *OF at-name-inst*]

lemmas *name-fresh-left* = *pt-fresh-left* [*OF pt-name-inst*, *OF at-name-inst*]

lemmas *name-fresh-right* = *pt-fresh-right* [*OF pt-name-inst*, *OF at-name-inst*]

lemmas *name-id* [*simp*] = *pt-id* [*OF pt-name-inst*, *OF at-name-inst*]

lemmas *name-swap-bij* [*simp*] = *pt-swap-bij* [*OF pt-name-inst*, *OF at-name-inst*]

lemmas *name-swap* = *pt-swap* [*OF pt-name-inst*, *OF at-name-inst*]

lemmas *name-rev-per* = *pt-rev-pi* [*OF pt-name-inst*, *OF at-name-inst*]

lemmas *name-per-rev* = *pt-pi-rev* [*OF pt-name-inst*, *OF at-name-inst*]

lemmas *name-exists-fresh* = *at-exists-fresh* [*OF at-name-inst*, *OF fs-name1*]

lemmas *name-perm-compose* = *pt-perm-compose*[*OF pt-name-inst, OF at-name-inst*]

nominal-datatype *pi* = *PiNil* ($\langle \mathbf{0} \rangle$)
 | *Output name name pi* ($\langle \{-\} \cdot \rightarrow$ [120, 120, 110] 110)
 | *Tau pi* ($\langle \tau \cdot \rightarrow$ [120] 110)
 | *Input name «name» pi* ($\langle \{-\} \cdot \rightarrow$ [120, 120, 110] 110)
 | *Match name name pi* ($\langle \{-\} \cdot \rightarrow$ [120, 120, 110] 110)
 | *Mismatch name name pi* ($\langle \{-\} \cdot \rightarrow$ [120, 120, 110] 110)
 | *Sum pi pi* (**infixr** $\langle \oplus \rangle$ 90)
 | *Par pi pi* (**infixr** $\langle \parallel \rangle$ 85)
 | *Res «name» pi* ($\langle \nu \cdot \rightarrow$ [100, 100] 100)
 | *Bang pi* ($\langle ! \cdot \rightarrow$ [110] 110)

lemmas *name-fresh*[*simp*] = *at-fresh*[*OF at-name-inst*]

lemma *alphaInput*:

fixes *a* :: *name*
and *x* :: *name*
and *P* :: *pi*
and *c* :: *name*

assumes *A1*: *c* $\#$ *P*

shows $a \langle x \rangle \cdot P = a \langle c \rangle \cdot ([x, c] \cdot P)$

proof(*cases x = c*)

assume *x=c*

thus *?thesis* **by** (*simp*)

next

assume $x \neq c$

with *A1* **show** *?thesis*

by(*simp add: pi.inject alpha name-fresh-left name-calc*)

qed

lemma *alphaRes*:

fixes *a* :: *name*
and *P* :: *pi*
and *c* :: *name*

assumes *A1*: *c* $\#$ *P*

shows $\langle \nu a \rangle P = \langle \nu c \rangle ([a, c] \cdot P)$

proof(*cases a=c*)

assume *a=c*

thus *?thesis* **by** *simp*

next

assume $a \neq c$

with *A1* **show** *?thesis*

by(*simp add: pi.inject alpha fresh-left name-calc*)

qed

definition $\text{subst-name} :: \text{name} \Rightarrow \text{name} \Rightarrow \text{name} \Rightarrow \text{name} \quad (\langle \cdot \text{[-::=]} \rangle [110, 110, 110] 110)$

where

$a[b::=c] \equiv \text{if } (a = b) \text{ then } c \text{ else } a$

declare $\text{subst-name-def}[\text{simp}]$

lemma $\text{subst-name-eqvt}[\text{eqvt}]$:

fixes $p :: \text{name prm}$

and $a :: \text{name}$

and $b :: \text{name}$

and $c :: \text{name}$

shows $p \cdot (a[b::=c]) = (p \cdot a)[(p \cdot b)::=(p \cdot c)]$

by($\text{auto simp add: at-bij[OF at-name-inst]}$)

nominal-primrec ($\text{freshness-context: } (c::\text{name}, d::\text{name})$)

$\text{subs} :: \text{pi} \Rightarrow \text{name} \Rightarrow \text{name} \Rightarrow \text{pi} \quad (\langle \cdot \text{[-::=]} \rangle [100, 100, 100] 100)$

where

$\mathbf{0}[c::=d] = \mathbf{0}$

$| \tau.(P)[c::=d] = \tau.(P[c::=d])$

$| a\{b\}.P[c::=d] = (a[c::=d])\{(b[c::=d])\}.(P[c::=d])$

$| \llbracket x \neq a; x \neq c; x \neq d \rrbracket \Longrightarrow (a\langle x \rangle.P)[c::=d] = (a[c::=d])\langle x \rangle.(P[c::=d])$

$| [a \frown b]P[c::=d] = [(a[c::=d]) \frown (b[c::=d])](P[c::=d])$

$| [a \neq b]P[c::=d] = [(a[c::=d]) \neq (b[c::=d])](P[c::=d])$

$| (P \oplus Q)[c::=d] = (P[c::=d]) \oplus (Q[c::=d])$

$| (P \parallel Q)[c::=d] = (P[c::=d]) \parallel (Q[c::=d])$

$| \llbracket x \neq c; x \neq d \rrbracket \Longrightarrow (\langle \nu x \rangle P)[c::=d] = \langle \nu x \rangle (P[c::=d])$

$| !P[c::=d] = !(P[c::=d])$

apply($\text{simp-all add: abs-fresh}$)

apply(finite-guess) $+$

by(fresh-guess) $+$

lemma forget :

fixes $a :: \text{name}$

and $P :: \text{pi}$

and $b :: \text{name}$

assumes $a \nmid P$

shows $P[a::=b] = P$

using assms

by($\text{nominal-induct } P \text{ avoiding: } a \text{ b rule: } \text{pi.strong-induct}$)

($\text{auto simp add: name-fresh-abs}$)

```

lemma fresh-fact2[rule-format]:
  fixes  $P :: pi$ 
  and  $a :: name$ 
  and  $b :: name$ 

  assumes  $a \neq b$ 

  shows  $a \# P[a::=b]$ 
using assms
by(nominal-induct P avoiding: a b rule: pi.strong-induct)
  (auto simp add: name-fresh-abs)

lemma subst-identity[simp]:
  fixes  $P :: pi$ 
  and  $a :: name$ 

  shows  $P[a::=a] = P$ 
by(nominal-induct P avoiding: a rule: pi.strong-induct) auto

lemma renaming:
  fixes  $P :: pi$ 
  and  $a :: name$ 
  and  $b :: name$ 
  and  $c :: name$ 

  assumes  $c \# P$ 

  shows  $P[a::=b] = ([c, a] \cdot P)[c::=b]$ 
using assms
by(nominal-induct P avoiding: a b c rule: pi.strong-induct)
  (auto simp add: name-calc name-fresh-abs)

lemma fresh-fact1:
  fixes  $P :: pi$ 
  and  $a :: name$ 
  and  $b :: name$ 
  and  $c :: name$ 

  assumes  $a \# P$ 
  and  $a \neq c$ 

  shows  $a \# P[b::=c]$ 
using assms
by(nominal-induct P avoiding: a b c rule: pi.strong-induct)
  (auto simp add: name-fresh-abs)

lemma eqvt-subs[eqvt]:

```

```

fixes  $p :: \text{name } prm$ 
and  $P :: \text{pi}$ 
and  $a :: \text{name}$ 
and  $b :: \text{name}$ 

shows  $p \cdot (P[a::=b]) = (p \cdot P)[(p \cdot a)::=(p \cdot b)]$ 
by(nominal-induct  $P$  avoiding: a b rule: pi.strong-induct)
  (auto simp add: name-bij)

```

```

lemma substInput[simp]:

```

```

fixes  $x :: \text{name}$ 
and  $b :: \text{name}$ 
and  $c :: \text{name}$ 
and  $a :: \text{name}$ 
and  $P :: \text{pi}$ 

```

```

assumes  $x \neq b$ 
and  $x \neq c$ 

```

```

shows  $(a \langle x \rangle . P)[b::=c] = (a[b::=c] \langle x \rangle . (P[b::=c]))$ 

```

```

proof –

```

```

obtain  $y :: \text{name}$  where  $y \neq a$  and  $y \# P$  and  $y \neq b$  and  $y \neq c$ 

```

```

  by(generate-fresh name) (auto simp add: fresh-prod)

```

```

from  $\langle y \# P \rangle$  have  $a \langle x \rangle . P = a \langle y \rangle . ([x, y] \cdot P)$  by(simp add: alphaInput)

```

```

moreover have  $(a[b::=c] \langle x \rangle . (P[b::=c])) = (a[b::=c] \langle y \rangle . ([x, y] \cdot P)[b::=c])$ 

```

```

(is  $?LHS = ?RHS$ )

```

```

proof –

```

```

from  $\langle y \# P \rangle$   $\langle y \neq c \rangle$  have  $y \# P[b::=c]$  by(rule fresh-fact1)

```

```

hence  $?LHS = (a[b::=c] \langle y \rangle . ([x, y] \cdot (P[b::=c])))$  by(simp add: alphaInput)

```

```

moreover with  $\langle x \neq b \rangle$   $\langle x \neq c \rangle$   $\langle y \neq b \rangle$   $\langle y \neq c \rangle$  have  $\dots = ?RHS$ 

```

```

  by(auto simp add: eqvt-subs name-calc)

```

```

ultimately show  $?thesis$  by simp

```

```

qed

```

```

ultimately show  $?thesis$  using  $\langle y \neq a \rangle$   $\langle y \neq b \rangle$   $\langle y \neq c \rangle$  by simp

```

```

qed

```

```

lemma injPermSubst:

```

```

fixes  $P :: \text{pi}$ 
and  $a :: \text{name}$ 
and  $b :: \text{name}$ 

```

```

assumes  $b \# P$ 

```

```

shows  $[(a, b)] \cdot P = P[a::=b]$ 

```

```

using assms

```

```

by(nominal-induct  $P$  avoiding: a b rule: pi.strong-induct)

```

```

  (auto simp add: name-calc name-fresh-abs)

```

```

lemma substRes2:
  fixes  $P :: pi$ 
  and  $a :: name$ 
  and  $b :: name$ 

  assumes  $b \# P$ 

  shows  $\langle \nu a \rangle P = \langle \nu b \rangle (P[a::=b])$ 
proof(case-tac a = b)
  assume  $a = b$ 
  thus ?thesis by auto
next
  assume  $a \neq b$ 
  moreover with  $\langle b \# P \rangle$  show ?thesis
    apply(simp add: pi.inject abs-fun-eq[OF pt-name-inst, OF at-name-inst])
    apply auto
    apply(simp add: renaming)
    apply(simp add: pt-swap[OF pt-name-inst, OF at-name-inst])
    apply(simp add: renaming)
    apply(simp add: pt-fresh-left[OF pt-name-inst, OF at-name-inst])
    by(force simp add: at-calc[OF at-name-inst])
qed

```

```

lemma freshRes:
  fixes  $P :: pi$ 
  and  $a :: name$ 

  shows  $a \# \langle \nu a \rangle P$ 
by(simp add: name-fresh-abs)

```

```

lemma substRes3:
  fixes  $P :: pi$ 
  and  $a :: name$ 
  and  $b :: name$ 

  assumes  $b \# P$ 

  shows  $(\langle \nu a \rangle P)[a::=b] = \langle \nu b \rangle (P[a::=b])$ 
proof –
  have  $(\langle \nu a \rangle P)[a::=b] = \langle \nu a \rangle P$ 
    using freshRes by(simp add: forget)
  thus ?thesis using  $\langle b \# P \rangle$  by(simp add: substRes2)
qed

```

```

lemma suppSubst:
  fixes  $P :: pi$ 
  and  $a :: name$ 
  and  $b :: name$ 

```

shows $\text{supp}(P[a::=b]) \subseteq \text{insert } b ((\text{supp } P) - \{a\})$
apply(*nominal-induct* P *avoiding: a b rule: pi.strong-induct,*
simp-all add: pi.supp name-supp-abs name-supp supp-prod)
by(*blast*)+

primrec $\text{seqSubs} :: \text{pi} \Rightarrow (\text{name} \times \text{name}) \text{list} \Rightarrow \text{pi} (\lambda \cdot [\langle - \rangle] \rangle [100,100] 100)$
where
 $P[\langle [] \rangle] = P$
 $| P[\langle (x\#\sigma) \rangle] = (P[(\text{fst } x)::=(\text{snd } x))][\langle \sigma \rangle]$

primrec $\text{seq-subst-name} :: \text{name} \Rightarrow (\text{name} \times \text{name}) \text{list} \Rightarrow \text{name}$ **where**
 $\text{seq-subst-name } a [] = a$
 $| \text{seq-subst-name } a (x\#\sigma) = \text{seq-subst-name } (a[(\text{fst } x)::=(\text{snd } x)]) \sigma$

lemma *freshSeqSubstName:*
fixes $x :: \text{name}$
and $a :: \text{name}$
and $s :: (\text{name} \times \text{name}) \text{list}$

assumes $x \neq a$
and $x \# s$

shows $x \neq \text{seq-subst-name } a s$
using *assms*
apply(*induct s arbitrary: a*)
apply *simp*
apply(*case-tac aa = fst(a)*)
by (*force simp add: fresh-list-cons fresh-prod*)+

lemma *seqSubstZero[simp]:*
fixes $\sigma :: (\text{name} \times \text{name}) \text{list}$

shows $\mathbf{0}[\langle \sigma \rangle] = \mathbf{0}$
by(*induct* σ , *auto*)

lemma *seqSubstTau[simp]:*
fixes $P :: \text{pi}$
and $\sigma :: (\text{name} \times \text{name}) \text{list}$

shows $(\tau.(P))[\langle \sigma \rangle] = \tau.(P[\langle \sigma \rangle])$
by(*induct* σ *arbitrary: P, auto*)

lemma *seqSubstOutput[simp]:*
fixes $a :: \text{name}$
and $b :: \text{name}$
and $P :: \text{pi}$

and $\sigma :: (\text{name} \times \text{name}) \text{ list}$

shows $(a\{b\}.P)[\langle\sigma\rangle] = (\text{seq-subst-name } a \ \sigma)\{(\text{seq-subst-name } b \ \sigma)\}.(P[\langle\sigma\rangle])$
by(*induct* σ *arbitrary: a b P, auto*)

lemma *seqSubstInput[simp]*:
fixes $a :: \text{name}$
and $x :: \text{name}$
and $P :: \text{pi}$
and $\sigma :: (\text{name} \times \text{name}) \text{ list}$

assumes $x \# \sigma$

shows $(a\langle x \rangle.P)[\langle\sigma\rangle] = (\text{seq-subst-name } a \ \sigma)\langle x \rangle.(P[\langle\sigma\rangle])$
using *assms*
by(*induct* σ *arbitrary: a x P*) (*auto simp add: fresh-list-cons fresh-prod*)

lemma *seqSubstMatch[simp]*:
fixes $a :: \text{name}$
and $b :: \text{name}$
and $P :: \text{pi}$
and $\sigma :: (\text{name} \times \text{name}) \text{ list}$

shows $([a\frown b]P)[\langle\sigma\rangle] = [(\text{seq-subst-name } a \ \sigma)\frown(\text{seq-subst-name } b \ \sigma)](P[\langle\sigma\rangle])$
by(*induct* σ *arbitrary: a b P, auto*)

lemma *seqSubstMismatch[simp]*:
fixes $a :: \text{name}$
and $b :: \text{name}$
and $P :: \text{pi}$
and $\sigma :: (\text{name} \times \text{name}) \text{ list}$

shows $([a\neq b]P)[\langle\sigma\rangle] = [(\text{seq-subst-name } a \ \sigma)\neq(\text{seq-subst-name } b \ \sigma)](P[\langle\sigma\rangle])$
by(*induct* σ *arbitrary: a b P, auto*)

lemma *seqSubstSum[simp]*:
fixes $P :: \text{pi}$
and $Q :: \text{pi}$
and $\sigma :: (\text{name} \times \text{name}) \text{ list}$

shows $(P \oplus Q)[\langle\sigma\rangle] = (P[\langle\sigma\rangle]) \oplus (Q[\langle\sigma\rangle])$
by(*induct* σ *arbitrary: P Q, auto*)

lemma *seqSubstPar[simp]*:
fixes $P :: \text{pi}$
and $Q :: \text{pi}$
and $\sigma :: (\text{name} \times \text{name}) \text{ list}$

shows $(P \parallel Q)[\langle\sigma\rangle] = (P[\langle\sigma\rangle]) \parallel (Q[\langle\sigma\rangle])$

```

by(induct  $\sigma$  arbitrary:  $P Q$ , auto)

lemma seqSubstRes[simp]:
  fixes  $x :: name$ 
  and  $P :: pi$ 
  and  $\sigma :: (name \times name) list$ 

  assumes  $x \# \sigma$ 

  shows  $\langle \nu x \rangle P[\langle \sigma \rangle] = \langle \nu x \rangle (P[\langle \sigma \rangle])$ 
using assms
by(induct  $\sigma$  arbitrary:  $x P$ ) (auto simp add: fresh-list-cons fresh-prod)

lemma seqSubstBang[simp]:
  fixes  $P :: pi$ 
  and  $s :: (name \times name) list$ 

  shows  $!(P)[\langle \sigma \rangle] = !(P[\langle \sigma \rangle])$ 
by(induct  $\sigma$  arbitrary:  $P$ , auto)

lemma seqSubstEqvt[eqvt, simp]:
  fixes  $P :: pi$ 
  and  $\sigma :: (name \times name) list$ 
  and  $p :: name prm$ 

  shows  $p \cdot (P[\langle \sigma \rangle]) = (p \cdot P)[\langle (p \cdot \sigma) \rangle]$ 
by(induct  $\sigma$  arbitrary:  $P$ , auto simp add: eqvt-subs)

lemma seqSubstAppend[simp]:
  fixes  $P :: pi$ 
  and  $\sigma :: (name \times name) list$ 
  and  $\sigma' :: (name \times name) list$ 

  shows  $P[\langle (\sigma @ \sigma') \rangle] = (P[\langle \sigma \rangle])[\langle \sigma' \rangle]$ 
by(induct  $\sigma$  arbitrary:  $P$ , auto)

lemma freshSubstChain[intro]:
  fixes  $P :: pi$ 
  and  $\sigma :: (name \times name) list$ 
  and  $a :: name$ 

  assumes  $a \# P$ 
  and  $a \# \sigma$ 

  shows  $a \# P[\langle \sigma \rangle]$ 
using assms
by(induct  $\sigma$  arbitrary:  $a P$ , auto simp add: fresh-list-cons fresh-prod fresh-fact1)

end

```

theory *Late-Semantics*

imports *Agent*

begin

nominal-datatype *subject* = *InputS name*
| *BoundOutputS name*

nominal-datatype *freeRes* = *OutputR name name* ($\langle \cdot \rangle$ [130, 130]
110)
| *TauR* ($\langle \tau \rangle$ 130)

nominal-datatype *residual* = *BoundR subject «name» pi* ($\langle \cdot \rangle$ [80, 80,
80] 80)
| *FreeR freeRes pi* ($\langle \cdot \rangle$ [80, 80] 80)

lemmas *residualInject* = *residual.inject freeRes.inject subject.inject*

abbreviation *Transitions-Inputjudge* :: *name* \Rightarrow *name* \Rightarrow *pi* \Rightarrow *residual* ($\langle \cdot \rangle$
 $\langle \cdot \rangle$ [80, 80, 80] 80)
where $a \langle x \rangle \prec P' \equiv ((\text{InputS } a) \langle x \rangle \prec P')$

abbreviation *Transitions-BoundOutputjudge* :: *name* \Rightarrow *name* \Rightarrow *pi* \Rightarrow *residual*
($\langle \cdot \rangle$ [80, 80, 80] 80)
where $a \langle \nu x \rangle \prec P' \equiv (\text{BoundR } (\text{BoundOutputS } a) x P')$

inductive *transitions* :: *pi* \Rightarrow *residual* \Rightarrow *bool* ($\langle \cdot \rangle$ [80, 80] 80)

where

Tau: $\tau.(P) \mapsto \tau \prec P$
| *Input*: $x \neq a \implies a \langle x \rangle . P \mapsto a \langle x \rangle \prec P$
| *Output*: $a \{ b \} . P \mapsto a [b] \prec P$

| *Match*: $\llbracket P \mapsto R s \rrbracket \implies [b \frown b] P \mapsto R s$
| *Mismatch*: $\llbracket P \mapsto R s ; a \neq b \rrbracket \implies [a \neq b] P \mapsto R s$

| *Open*: $\llbracket P \mapsto a [b] \prec P' ; a \neq b \rrbracket \implies \langle \nu b \rangle P \mapsto a \langle \nu b \rangle \prec P'$
| *Sum1*: $\llbracket P \mapsto R s \rrbracket \implies (P \oplus Q) \mapsto R s$
| *Sum2*: $\llbracket Q \mapsto R s \rrbracket \implies (P \oplus Q) \mapsto R s$

| *Par1B*: $\llbracket P \mapsto a \langle x \rangle \prec P' ; x \# P ; x \# Q ; x \# a \rrbracket \implies P \parallel Q \mapsto a \langle x \rangle \prec (P' \parallel Q)$
| *Par1F*: $\llbracket P \mapsto \alpha \prec P' \rrbracket \implies P \parallel Q \mapsto \alpha \prec (P' \parallel Q)$
| *Par2B*: $\llbracket Q \mapsto a \langle x \rangle \prec Q' ; x \# P ; x \# Q ; x \# a \rrbracket \implies P \parallel Q \mapsto a \langle x \rangle \prec (P \parallel Q')$
| *Par2F*: $\llbracket Q \mapsto \alpha \prec Q' \rrbracket \implies P \parallel Q \mapsto \alpha \prec (P \parallel Q')$

| *Comm1*: $\llbracket P \mapsto a \langle x \rangle \prec P' ; Q \mapsto a [b] \prec Q' ; x \# P ; x \# Q ; x \neq a ; x \neq b ; x \# Q' \rrbracket \implies P \parallel Q \mapsto \tau \prec P' [x ::= b] \parallel Q'$
| *Comm2*: $\llbracket P \mapsto a [b] \prec P' ; Q \mapsto a \langle x \rangle \prec Q' ; x \# P ; x \# Q ; x \neq a ; x$

$\neq b; x \# P'] \implies P \parallel Q \mapsto \tau \prec P' \parallel Q'[x::=b]$
| *Close1*: $\llbracket P \mapsto a\langle x \rangle \prec P'; Q \mapsto a\langle \nu y \rangle \prec Q'; x \# P; x \# Q; y \# P;$
 $y \# Q; x \neq a; x \# Q'; y \neq a; y \# P'; x \neq y \rrbracket \implies P \parallel Q \mapsto \tau \prec$
 $\langle \nu y \rangle (P'[x::=y] \parallel Q')$
| *Close2*: $\llbracket P \mapsto a\langle \nu y \rangle \prec P'; Q \mapsto a\langle x \rangle \prec Q'; x \# P; x \# Q; y \# P;$
 $y \# Q; x \neq a; x \# P'; y \neq a; y \# Q'; x \neq y \rrbracket \implies P \parallel Q \mapsto \tau \prec$
 $\langle \nu y \rangle (P' \parallel Q'[x::=y])$

| *ResB*: $\llbracket P \mapsto a\langle x \rangle \prec P'; y \# a; y \neq x; x \# P; x \# a \rrbracket \implies \langle \nu y \rangle P \mapsto$
 $a\langle x \rangle \prec \langle \nu y \rangle P'$
| *ResF*: $\llbracket P \mapsto \alpha \prec P'; y \# \alpha \rrbracket \implies \langle \nu y \rangle P \mapsto \alpha \prec \langle \nu y \rangle P'$

| *Bang*: $\llbracket P \parallel !P \mapsto Rs \rrbracket \implies !P \mapsto Rs$

equivariance transitions

nominal-inductive transitions

by(*auto simp add: abs-fresh fresh-fact2*)

lemma *alphaBoundResidual*:

fixes $a :: \text{subject}$

and $x :: \text{name}$

and $P :: \text{pi}$

and $x' :: \text{name}$

assumes $A1: x' \# P$

shows $a\langle x \rangle \prec P = a\langle x' \rangle \prec ([(x, x')] \cdot P)$

proof(*cases x=x'*)

assume $x=x'$

thus *?thesis by simp*

next

assume $x \neq x'$

with $A1$ **show** *?thesis*

by(*simp add: residualInject alpha name-fresh-left name-calc*)

qed

lemma *freshResidual*:

fixes $P :: \text{pi}$

and $Rs :: \text{residual}$

and $x :: \text{name}$

assumes $P \mapsto Rs$

and $x \# P$

shows $x \# Rs$

using *assms*

by(*nominal-induct rule: transitions.strong-induct*)

(*auto simp add: abs-fresh fresh-fact2 fresh-fact1*)

```

lemma freshBoundDerivative:
  assumes  $P \mapsto a\langle x \rangle \prec P'$ 
  and  $y \# P$ 

  shows  $y \# a$ 
  and  $y \neq x \implies y \# P'$ 
apply –
using assms
by(fastforce dest: freshResidual simp add: abs-fresh)

lemma freshFreeDerivative:
  fixes  $P :: pi$ 
  and  $\alpha :: freeRes$ 
  and  $P' :: pi$ 
  and  $y :: name$ 

  assumes  $P \mapsto \alpha \prec P'$ 
  and  $y \# P$ 

  shows  $y \# \alpha$ 
  and  $y \# P'$ 
apply –
using assms
by(fastforce dest: freshResidual)

lemma substTrans[simp]:
  fixes  $b :: name$ 
  and  $P :: pi$ 
  and  $a :: name$ 
  and  $c :: name$ 

  assumes  $b \# P$ 

  shows  $(P[a ::= b])[b ::= c] = P[a ::= c]$ 
using assms
apply(simp add: injPermSubst[THEN sym])
apply(simp add: renaming)
by(simp add: pt-swap[OF pt-name-inst, OF at-name-inst])

lemma Input:
  fixes  $a :: name$ 
  and  $x :: name$ 
  and  $P :: pi$ 

  shows  $a\langle x \rangle.P \mapsto a\langle x \rangle \prec P$ 
proof –
  obtain  $y :: name$  where  $y \neq a$  and  $y \# P$ 
  by(generate-fresh name, auto simp add: fresh-prod)
  from  $\langle y \# P \rangle$  have  $a\langle x \rangle.P = a\langle y \rangle.([(x, y)] \cdot P)$  and  $a\langle x \rangle \prec P = a\langle y \rangle \prec$ 

```

```

(( $(x, y)$ ) ·  $P$ )
  by(auto simp add: alphaBoundResidual alphaInput)
  with  $\langle y \neq a \rangle$  show ?thesis by(force intro: Input)
qed

declare perm-fresh-fresh[simp] name-swap[simp] fresh-prod[simp]

```

lemma *Par1B*:

```

fixes  $P :: pi$ 
and  $a :: subject$ 
and  $x :: name$ 
and  $P' :: pi$ 
and  $Q :: pi$ 

```

```

assumes  $P \mapsto_a \langle x \rangle \prec P'$ 
and  $x \# Q$ 

```

```

shows  $P \parallel Q \mapsto_a \langle x \rangle \prec P' \parallel Q$ 

```

proof –

```

obtain  $y :: name$  where  $y \# P$  and  $y \# P'$  and  $y \# Q$  and  $y \# a$ 
  by(generate-fresh name, auto)
from  $\langle P \mapsto_a \langle x \rangle \prec P' \rangle \langle y \# P' \rangle$  have  $P \mapsto_a \langle y \rangle \prec ((x, y) \cdot P')$ 
  by(simp add: alphaBoundResidual)
hence  $P \parallel Q \mapsto_a \langle y \rangle \prec ((x, y) \cdot P') \parallel Q$  using  $\langle y \# P \rangle \langle y \# Q \rangle \langle y \# a \rangle$ 
  by(rule Par1B)
with  $\langle x \# Q \rangle \langle y \# P' \rangle \langle y \# Q \rangle$  show ?thesis
  by(subst alphaBoundResidual[where  $x'=y$ ]) auto

```

qed

lemma *Par2B*:

```

fixes  $Q :: pi$ 
and  $a :: subject$ 
and  $x :: name$ 
and  $Q' :: pi$ 
and  $P :: pi$ 

```

```

assumes  $QTrans: Q \mapsto_a \langle x \rangle \prec Q'$ 
and  $x \# P$ 

```

```

shows  $P \parallel Q \mapsto_a \langle x \rangle \prec P \parallel Q'$ 

```

proof –

```

obtain  $y :: name$  where  $y \# Q$  and  $y \# Q'$  and  $y \# P$  and  $y \# a$ 
  by(generate-fresh name, auto simp add: fresh-prod)
from  $QTrans \langle y \# Q' \rangle$  have  $Q \mapsto_a \langle y \rangle \prec ((x, y) \cdot Q')$ 
  by(simp add: alphaBoundResidual)
hence  $P \parallel Q \mapsto_a \langle y \rangle \prec P \parallel ((x, y) \cdot Q')$  using  $\langle y \# P \rangle \langle y \# Q \rangle \langle y \# a \rangle$ 
  by(rule Par2B)
moreover have  $a \langle y \rangle \prec P \parallel ((x, y) \cdot Q') = a \langle x \rangle \prec P \parallel Q'$ 

```

proof –

from $\langle y \# Q' \rangle \langle x \# P \rangle$ **have** $x \# P \parallel ((x, y) \cdot Q')$ **by**(*auto simp add: calc-atm fresh-left*)
with $\langle x \# P \rangle \langle y \# P \rangle$ **show** *?thesis* **by**(*simp only: alphaBoundResidual, auto simp add: name-swap name-fresh-fresh*)
qed
ultimately show *?thesis* **by** *simp*
qed

lemma *Comm1*:

fixes $P :: pi$
and $a :: name$
and $x :: name$
and $P' :: pi$
and $Q :: pi$
and $b :: name$
and $Q' :: pi$

assumes $PTrans: P \mapsto a \langle x \rangle \prec P'$
and $QTrans: Q \mapsto a[b] \prec Q'$

shows $P \parallel Q \mapsto \tau \prec P'[x::=b] \parallel Q'$

proof –

obtain $y :: name$ **where** $y \# P$ **and** $y \# P'$ **and** $y \# Q$ **and** $y \neq a$ **and** $y \neq b$ **and** $y \# Q'$

by(*generate-fresh name, auto simp add: fresh-prod*)

from $PTrans \langle y \# P' \rangle$ **have** $P \mapsto a \langle y \rangle \prec ((x, y) \cdot P')$

by(*simp add: alphaBoundResidual*)

hence $P \parallel Q \mapsto \tau \prec ((x, y) \cdot P')[y::=b] \parallel Q'$

using $QTrans \langle y \# P \rangle \langle y \# Q \rangle \langle y \neq a \rangle \langle y \neq b \rangle \langle y \# Q' \rangle$

by(*rule Comm1*)

with $\langle y \# P' \rangle$ **show** *?thesis* **by**(*simp add: renaming name-swap*)

qed

lemma *Comm2*:

fixes $P :: pi$
and $a :: name$
and $b :: name$
and $P' :: pi$
and $Q :: pi$
and $x :: name$
and $Q' :: pi$

assumes $PTrans: P \mapsto a[b] \prec P'$

and $QTrans: Q \mapsto a \langle x \rangle \prec Q'$

shows $P \parallel Q \mapsto \tau \prec P' \parallel (Q'[x::=b])$

proof –

obtain $y :: name$ **where** $y \# P$ **and** $y \# P'$ **and** $y \# Q$ **and** $y \neq a$ **and** $y \neq b$ **and** $y \# Q'$

```

  by(generate-fresh name, auto simp add: fresh-prod)
from QTrans ⟨y # Q'⟩ have Q ⟶a⟨y⟩ < ((x, y)] · Q')
  by(simp add: alphaBoundResidual)
with PTrans have P || Q ⟶τ < P' || (((x, y)] · Q')[y::=b])
using ⟨y # P⟩ ⟨y # Q⟩ ⟨y ≠ a⟩ ⟨y ≠ b⟩ ⟨y # P'⟩
  by(rule Comm2)
with ⟨y # Q'⟩ show ?thesis by(simp add: renaming name-swap)
qed

```

lemma *Close1*:

```

fixes P :: pi
and a :: name
and x :: name
and P' :: pi
and Q :: pi
and y :: name
and Q' :: pi

```

```

assumes PTrans: P ⟶a⟨x⟩ < P'
and QTrans: Q ⟶a⟨νy⟩ < Q'
and y # P

```

shows $P \parallel Q \xrightarrow{\tau} \langle \nu y \rangle (P'[x::=y] \parallel Q')$

proof –

```

obtain x'::name where x' # P and x' # P' and x' # Q and x' # Q' and x' ≠ a

```

```

  by(generate-fresh name, auto simp add: fresh-prod)

```

```

obtain y'::name where y' # P and y' # Q' and y' # Q

```

```

  and y' # P' and y' ≠ x' and y' ≠ y and y' ≠ a

```

```

  by(generate-fresh name, auto simp add: fresh-prod)

```

```

from PTrans ⟨x' # P'⟩ have P ⟶a⟨x'⟩ < ((x, x')] · P')

```

```

  by(simp add: alphaBoundResidual)

```

```

moreover from QTrans ⟨y' # Q'⟩ have Q ⟶a⟨νy'⟩ < ((y, y')] · Q')

```

```

  by(simp add: alphaBoundResidual)

```

```

ultimately have P || Q ⟶τ < νy' > (((x, x')] · P')[x'::=y'] || (((y, y')] · Q'))

```

```

  using ⟨y' # P⟩ ⟨y' # Q⟩ ⟨x' # P⟩ ⟨x' # Q⟩ ⟨y' ≠ x'⟩ ⟨y' ≠ a⟩ ⟨x' ≠ a⟩

```

```

  ⟨y' # P'⟩ ⟨y' # Q'⟩ ⟨x' # P'⟩ ⟨x' # Q'⟩

```

```

  apply(rule-tac Close1)

```

```

  by assumption (auto simp add: fresh-left calc-atm)

```

```

moreover have < νy' > (((x, x')] · P')[x'::=y'] || (((y, y')] · Q')) = < νy > (P'[x::=y]
|| Q')

```

proof –

```

  from ⟨x' # P'⟩ have (((x, x')] · P')[x'::=y'] = P'[x::=y'] by(simp add: renaming
name-swap)

```

```

  moreover have y # (P'[x::=y'] || (((y, y')] · Q'))

```

```

  proof(case-tac y = x)

```

```

    assume y = x

```

```

    with ⟨y' # Q'⟩ ⟨y' ≠ y⟩ show ?thesis by(auto simp add: fresh-fact2 fresh-left
calc-atm)

```

```

  next

```

```

    assume  $y \neq x$ 
    with  $\langle y \# P \rangle$   $PTrans$  have  $y \# P'$  by(force dest: freshBoundDerivative)
    with  $\langle y' \# Q' \rangle$   $\langle y' \neq y \rangle$  show ?thesis by(auto simp add: fresh-left calc-atm
fresh-fact1)
  qed
  ultimately show ?thesis using  $\langle y' \# P' \rangle$  apply(simp only: alphaRes)
  by(auto simp add: name-swap eqvt-subs calc-atm renaming)
  qed
  ultimately show ?thesis by simp
  qed

```

lemma *Close2*:

```

fixes  $P :: pi$ 
and  $a :: name$ 
and  $y :: name$ 
and  $P' :: pi$ 
and  $Q :: pi$ 
and  $x :: name$ 
and  $Q' :: pi$ 

```

```

assumes  $PTrans: P \mapsto a \langle \nu y \rangle \prec P'$ 
and  $QTrans: Q \mapsto a \langle x \rangle \prec Q'$ 
and  $y \# Q$ 

```

shows $P \parallel Q \mapsto \tau \prec \langle \nu y \rangle (P' \parallel (Q'[x::=y]))$

proof –

```

obtain  $x'::name$  where  $x' \# P$  and  $x' \# Q'$  and  $x' \# Q$  and  $x' \# P'$  and  $x' \neq a$ 
  by(generate-fresh name, auto simp add: fresh-prod)
obtain  $y'::name$  where  $y' \# P$  and  $y' \# P'$  and  $y' \# Q$ 
  and  $y' \# Q'$  and  $y' \neq x'$  and  $y' \neq y$  and  $y' \neq a$ 
  by(generate-fresh name, auto simp add: fresh-prod)
from  $PTrans \langle y' \# P' \rangle$  have  $P \mapsto a \langle \nu y' \rangle \prec ((y, y') \cdot P')$ 
  by(simp add: alphaBoundResidual)
moreover from  $QTrans \langle x' \# Q' \rangle$  have  $Q \mapsto a \langle x' \rangle \prec ((x, x') \cdot Q')$ 
  by(simp add: alphaBoundResidual)
ultimately have  $P \parallel Q \mapsto \tau \prec \langle \nu y' \rangle (((y, y') \cdot P') \parallel (((x, x') \cdot Q')[x'::=y']))$ 
  using  $\langle y' \# P \rangle \langle y' \# Q \rangle \langle x' \# P \rangle \langle x' \# Q \rangle \langle y' \neq x' \rangle \langle x' \neq a \rangle \langle y' \neq a \rangle$ 
   $\langle x' \# P' \rangle \langle x' \# Q' \rangle \langle y' \# P' \rangle \langle y' \# Q' \rangle$ 
  by(rule-tac Close2) (assumption | auto simp add: fresh-left calc-atm) +
  moreover have  $\langle \nu y' \rangle (((y, y') \cdot P') \parallel (((x, x') \cdot Q')[x'::=y'])) = \langle \nu y \rangle (P' \parallel (Q'[x::=y]))$ 
  proof –
    from  $\langle x' \# Q' \rangle$  have  $((x, x') \cdot Q')[x'::=y'] = Q'[x::=y']$  by(simp add: renaming name-swap)
    moreover have  $y \# (((y, y') \cdot P') \parallel (Q'[x::=y']))$ 
    proof(case-tac y = x)
      assume  $y = x$ 
      with  $\langle y' \# P' \rangle \langle y' \neq y \rangle$  show ?thesis by(auto simp add: fresh-fact2 fresh-left calc-atm)
    qed
  qed

```

```

next
  assume  $y \neq x$ 
  with  $\langle y \# Q \rangle$   $QTrans$  have  $y \# Q'$  by(force dest: freshBoundDerivative)
  with  $\langle y' \# P' \rangle$   $\langle y' \neq y \rangle$  show ?thesis by(auto simp add: fresh-left calc-atm
fresh-fact1)
qed
ultimately show ?thesis using  $\langle y' \# Q' \rangle$  apply(simp only: alphaRes)
  by(auto simp add: name-swap eqvt-subs calc-atm renaming)
qed
ultimately show ?thesis by simp
qed

```

lemma *ResB*:

```

fixes  $P :: pi$ 
and  $a :: subject$ 
and  $x :: name$ 
and  $P' :: pi$ 
and  $y :: name$ 

assumes  $PTrans: P \mapsto a \langle x \rangle \prec P'$ 
and  $y \# a$ 
and  $y \neq x$ 

shows  $\langle \nu y \rangle P \mapsto a \langle x \rangle \prec \langle \nu y \rangle P'$ 
proof -
  obtain  $z$  where  $z \# P$  and  $z \# a$  and  $z \neq y$  and  $z \# P'$ 
  by(generate-fresh name, auto simp add: fresh-prod)
  from  $PTrans \langle z \# P' \rangle$  have  $P \mapsto a \langle z \rangle \prec ((x, z) \cdot P')$  by(simp add: al-
phaBoundResidual)
  with  $\langle z \# P \rangle \langle z \# a \rangle \langle z \neq y \rangle \langle y \# a \rangle$  have  $\langle \nu y \rangle P \mapsto a \langle z \rangle \prec \langle \nu y \rangle ((x, z) \cdot P')$ 
  by(rule-tac ResB) auto
  moreover have  $a \langle z \rangle \prec \langle \nu y \rangle ((x, z) \cdot P') = a \langle x \rangle \prec \langle \nu y \rangle P'$ 
  proof -
    from  $\langle z \# P' \rangle \langle y \neq x \rangle$  have  $x \# \langle \nu y \rangle ((x, z) \cdot P')$  by(auto simp add: abs-fresh
fresh-left calc-atm)
    with  $\langle y \neq x \rangle \langle z \neq y \rangle$  show ?thesis by(simp add: alphaBoundResidual name-swap
calc-atm)
  qed
  ultimately show ?thesis by simp
qed

```

lemma *outputInduct*[consumes 1, case-names *Output Match Mismatch Sum1 Sum2 Par1 Par2 Res Bang*]:

```

fixes  $P :: pi$ 
and  $a :: name$ 
and  $b :: name$ 
and  $P' :: pi$ 
and  $F :: 'a::fs-name \Rightarrow pi \Rightarrow name \Rightarrow name \Rightarrow pi \Rightarrow bool$ 
and  $C :: 'a::fs-name$ 

```

assumes *Trans*: $P \mapsto a[b] \prec P'$
and $\bigwedge a b P C. F C (a\{b\}.P) a b P$
and $\bigwedge P a b P' c C. \llbracket P \mapsto \text{OutputR } a b \prec P'; \bigwedge C. F C P a b P' \rrbracket \implies F C$
 $([c \frown c]P) a b P'$
and $\bigwedge P a b P' c d C. \llbracket P \mapsto \text{OutputR } a b \prec P'; \bigwedge C. F C P a b P'; c \neq d \rrbracket$
 $\implies F C ([c \neq d]P) a b P'$
and $\bigwedge P a b P' Q C. \llbracket P \mapsto \text{OutputR } a b \prec P'; \bigwedge C. F C P a b P' \rrbracket \implies F C$
 $(P \oplus Q) a b P'$
and $\bigwedge Q a b Q' P C. \llbracket Q \mapsto \text{OutputR } a b \prec Q'; \bigwedge C. F C Q a b Q' \rrbracket \implies F$
 $C (P \oplus Q) a b Q'$
and $\bigwedge P a b P' Q C. \llbracket P \mapsto \text{OutputR } a b \prec P'; \bigwedge C. F C P a b P' \rrbracket \implies F C$
 $(P \parallel Q) a b (P' \parallel Q)$
and $\bigwedge Q a b Q' P C. \llbracket Q \mapsto \text{OutputR } a b \prec Q'; \bigwedge C. F C Q a b Q' \rrbracket \implies F$
 $C (P \parallel Q) a b (P \parallel Q')$
and $\bigwedge P a b P' x C. \llbracket P \mapsto \text{OutputR } a b \prec P'; x \neq a; x \neq b; x \# C; \bigwedge C. F$
 $C P a b P' \rrbracket \implies$
 $F C (\langle \nu x \rangle P) a b (\langle \nu x \rangle P')$
and $\bigwedge P a b P' C. \llbracket P \parallel !P \mapsto \text{OutputR } a b \prec P'; \bigwedge C. F C (P \parallel !P) a b P' \rrbracket$
 $\implies F C (!P) a b P'$

shows $F C P a b P'$

proof –

from *Trans show ?thesis*

by(*nominal-induct x2 == OutputR a b < P' avoiding: C arbitrary: P' rule:*
transitions.strong-induct,

auto simp add: residualInject freeRes.inject intro: assms)

qed

lemma *inputInduct[consumes 2, case-names Input Match Mismatch Sum1 Sum2*
Par1 Par2 Res Bang]:

fixes $P :: pi$

and $a :: name$

and $x :: name$

and $P' :: pi$

and $F :: ('a::fs-name) \Rightarrow pi \Rightarrow name \Rightarrow name \Rightarrow pi \Rightarrow bool$

and $C :: 'a::fs-name$

assumes $a: P \mapsto a\langle x \rangle \prec P'$
and $x \# P$
and *cInput*: $\bigwedge a x P C. F C (a\langle x \rangle.P) a x P$
and *cMatch*: $\bigwedge P a x P' b C. \llbracket P \mapsto a\langle x \rangle \prec P'; \bigwedge C. F C P a x P' \rrbracket \implies$
 $F C ([b \frown b]P) a x P'$
and *cMismatch*: $\bigwedge P a x P' b c C. \llbracket P \mapsto a\langle x \rangle \prec P'; \bigwedge C. F C P a x P'; b$
 $\neq c \rrbracket \implies F C ([b \neq c]P) a x P'$
and *cSum1*: $\bigwedge P Q a x P' C. \llbracket P \mapsto a\langle x \rangle \prec P'; \bigwedge C. F C P a x P' \rrbracket \implies$
 $F C (P \oplus Q) a x P'$
and *cSum2*: $\bigwedge P Q a x Q' C. \llbracket Q \mapsto a\langle x \rangle \prec Q'; \bigwedge C. F C Q a x Q' \rrbracket \implies$
 $F C (P \oplus Q) a x Q'$

and $cPar1B$: $\bigwedge P P' Q a x C. \llbracket P \mapsto a \langle x \rangle \prec P'; x \# P; x \# Q; x \neq a; \bigwedge C. F C P a x P' \rrbracket \implies$
 $F C (P \parallel Q) a x (P' \parallel Q)$

and $cPar2B$: $\bigwedge P Q Q' a x C. \llbracket Q \mapsto a \langle x \rangle \prec Q'; x \# P; x \# Q; x \neq a; \bigwedge C. F C Q a x Q' \rrbracket \implies$
 $F C (P \parallel Q) a x (P \parallel Q')$

and $cResB$: $\bigwedge P P' a x y C. \llbracket P \mapsto a \langle x \rangle \prec P'; y \neq a; y \neq x; y \# C; \bigwedge C. F C P a x P' \rrbracket \implies F C (\nu y \rangle P) a x (\nu y \rangle P')$

and $cBang$: $\bigwedge P a x P' C. \llbracket P \parallel !P \mapsto a \langle x \rangle \prec P'; \bigwedge C. F C (P \parallel !P) a x P' \rrbracket \implies$
 $F C (!P) a x P'$

shows $F C P a x P'$

proof –

from $a \langle x \# P \rangle$ **show** $?thesis$

proof(*nominal-induct* $x2 == a \langle x \rangle \prec P'$ *avoiding: C a x P' rule: transitions.strong-induct*)

case($Tau P$)

thus $?case$ **by**(*simp add: residualInject*)

next

case($Input x a P C a' x' P'$)

have $x \# x'$ **by fact** **hence** $x \neq x'$ **by** *simp*

moreover **have** $a \langle x \rangle \prec P = a' \langle x' \rangle \prec P'$ **by fact**

ultimately **have** $aeqa'$: $a = a'$ **and** $PeqP'$: $P = [(x, x')] \cdot P'$

by(*simp add: residualInject freeRes.inject subject.inject name-abs-eq*)+

have $F C (a \langle x' \rangle.([(x, x')] \cdot P)) a x' ([(x, x')] \cdot P)$ **by**(*rule cInput*)

moreover **have** $x \# P'$ **by fact**

ultimately **show** $?case$ **using** $PeqP' aeqa'$ **by**(*simp add: alphaInput name-swap*)

next

case($Output P a b$)

thus $?case$ **by**(*simp add: residualInject*)

next

case($Match P b Rs a x$)

thus $?case$

by(*force intro: cMatch simp add: residualInject*)

next

case($Mismatch P Rs a b C a x$)

thus $?case$

by(*force intro: cMismatch simp add: residualInject*)

next

case($Open P P' a b C a' x P'$)

thus $?case$ **by**(*simp add: residualInject*)

next

case($Sum1 P Q Rs C$)

thus $?case$ **by**(*force intro: cSum1*)

next

case($Sum2 P Q Rs C$)

thus $?case$ **by**(*force intro: cSum2*)

next

```

case(Par1B P a x P' Q C a' x' P'')
have x # x' by fact hence xineqx': x ≠ x' by simp
moreover have Eq: a«x» < (P' || Q) = a'<x'> < P'' by fact
hence aeqa': a = InputS a' by(simp add: residualInject)
have x' # P || Q by fact
hence x' # P and x' # Q by simp+
have P''eq: P'' = ((x, x') · P') || Q
proof -
  from Eq xineqx' have (P' || Q) = ((x, x') · P')
  by(simp add: residualInject name-abs-eq)
  hence (((x, x') · (P' || Q)) = P'' by simp
  with ⟨x' # Q⟩⟨x # Q⟩ show ?thesis by(simp add: name-fresh-fresh)
qed

have x # P'' by fact
with P''eq ⟨x ≠ x'⟩ have x' # P' by(simp add: name-fresh-left name-calc)

have PTrans: P ⟶ a«x» < P' by fact
with ⟨x' # P'⟩ aeqa' have P ⟶ a'<x'> < (((x, x') · P')
  by(simp add: alphaBoundResidual)
moreover have ∧C. F C P a' x' (((x, x') · P')
proof -
  fix C
  have ∧C a' x' P''. [[a«x» < P' = a'<x'> < P'']; x' # P] ⟹ F C P a' x' P''
by fact
  moreover with aeqa' xineqx' ⟨x' # P'⟩ have a«x» < P' = a'<x'> < (((x,
x') · P')
  by(simp add: residualInject name-abs-eq name-fresh-left name-calc)
  ultimately show F C P a' x' (((x, x') · P') using ⟨x' # P'⟩ by blast
qed
moreover from PTrans ⟨x' # P'⟩ have x' # a by(auto dest: freshBoundDeriva-
tive)
ultimately have F C (P || Q) a' x' (((x, x') · P') || Q) using ⟨x' # Q⟩ aeqa'
⟨x' # P'⟩
  by(rule-tac cPar1B) auto
with P''eq show ?case by simp
next
case(Par1F P P' Q α)
thus ?case by(simp add: residualInject)
next
case(Par2B Q a x Q' P C a' x' Q'')
have x # x' by fact hence xineqx': x ≠ x' by simp
moreover have Eq: a«x» < (P || Q') = a'<x'> < Q'' by fact
hence aeqa': a = InputS a' by(simp add: residualInject)
have x # P by fact
have x' # P || Q by fact
hence x' # P and x' # Q by simp+
have Q''eq: Q'' = P || (((x, x') · Q')
proof -

```

```

from Eq xineq  $x'$  have  $(P \parallel Q') = [(x, x')] \cdot Q''$ 
  by (simp add: residualInject name-abs-eq)
hence  $([(x, x')] \cdot (P \parallel Q')) = Q''$  by simp
with  $\langle x' \# P \rangle \langle x \# P \rangle$  show ?thesis by (simp add: name-fresh-fresh)
qed

have  $x \# Q''$  by fact
with  $Q''eq \langle x \neq x' \rangle$  have  $x' \# Q'$  by (simp add: name-fresh-left name-calc)

have QTrans:  $Q \mapsto a\langle x \rangle \prec Q'$  by fact
with  $\langle x' \# Q' \rangle aeqa'$  have  $Q \mapsto a'\langle x' \rangle \prec ([x, x'] \cdot Q')$ 
  by (simp add: alphaBoundResidual)
moreover have  $\bigwedge C. F C Q a' x' ([x, x'] \cdot Q')$ 
proof –
  fix  $C$ 
  have  $\bigwedge C a' x' Q''. [a\langle x \rangle \prec Q' = a'\langle x' \rangle \prec Q''; x' \# Q] \implies F C Q a' x'$ 
   $Q''$  by fact
  moreover with  $aeqa' xineq x' \langle x' \# Q' \rangle$  have  $a\langle x \rangle \prec Q' = a'\langle x' \rangle \prec ([x, x'] \cdot Q')$ 
  by (simp add: residualInject name-abs-eq name-fresh-left name-calc)
  ultimately show  $F C Q a' x' ([x, x'] \cdot Q')$  using  $\langle x' \# Q \rangle aeqa'$  by blast
qed
moreover from QTrans  $\langle x' \# Q \rangle$  have  $x' \# a$  by (force dest: freshBoundDerivative)
ultimately have  $F C (P \parallel Q) a' x' (P \parallel ([x, x'] \cdot Q'))$  using  $\langle x' \# P \rangle aeqa'$ 
 $\langle x' \# Q \rangle$ 
  by (rule-tac cPar2B) auto
with  $Q''eq$  show ?case by simp
next
  case (Par2F  $P P' Q \alpha$ )
  thus ?case by (simp add: residualInject)
next
  case (Comm1  $P P' Q Q' a b x$ )
  thus ?case by (simp add: residualInject)
next
  case (Comm2  $P P' Q Q' a b x$ )
  thus ?case by (simp add: residualInject)
next
  case (Close1  $P P' Q Q' a x y$ )
  thus ?case by (simp add: residualInject)
next
  case (Close2  $P P' Q Q' a x y$ )
  thus ?case by (simp add: residualInject)
next
  case (ResB  $P a x P' y C a' x' P''$ )
  have  $x \# x'$  by fact hence xineq  $x' : x \neq x'$  by simp
  moreover have Eq:  $a\langle x \rangle \prec (\nu y \langle P' \rangle) = a'\langle x' \rangle \prec P''$  by fact
  hence aeqa':  $a = InputS a'$  by (simp add: residualInject)
  have  $y \# x'$  by fact hence yineq  $x' : y \neq x'$  by simp

```

```

moreover have  $x' \# \langle \nu y \rangle P$  by fact
ultimately have  $x' \# P$  by(simp add: name-fresh-abs)
have  $y \neq x$  and  $y \text{ineq} a: y \# a$  and  $y \text{Fresh} C: y \# C$  by fact+

have  $P'' \text{eq}: P'' = \langle \nu y \rangle ([ (x, x') ] \cdot P')$ 
proof –
  from Eq  $x \text{ineq} x'$  have  $\langle \nu y \rangle P' = [(x, x')] \cdot P''$ 
  by(simp add: residualInject name-abs-eq)
  hence  $([(x, x')] \cdot \langle \nu y \rangle P')$   $= P''$  by simp
  with  $y \text{ineq} x' \langle y \neq x \rangle$  show ?thesis by(simp add: name-fresh-fresh)
qed

have  $x \# P''$  by fact
with  $P'' \text{eq} \langle y \neq x \rangle \langle x \neq x' \rangle$  have  $x' \# P'$  by(simp add: name-fresh-left name-calc
name-fresh-abs)

have  $P \mapsto a \langle x \rangle \prec P'$  by fact
with  $\langle x' \# P' \rangle \text{aeqa}'$  have  $P \mapsto a' \langle x' \rangle \prec ([ (x, x') ] \cdot P')$ 
  by(simp add: alphaBoundResidual)
moreover have  $\bigwedge C. F C P a' x' ([ (x, x') ] \cdot P')$ 
proof –
  fix C
  have  $\bigwedge C a' x' P''. [a \langle x \rangle \prec P' = a' \langle x' \rangle \prec P''; x' \# P] \implies F C P a' x' P''$ 
by fact
  moreover with  $\text{aeqa}' x \text{ineq} x' \langle x' \# P' \rangle$  have  $a \langle x \rangle \prec P' = a' \langle x' \rangle \prec ([ (x, x') ] \cdot P')$ 
  by(simp add: residualInject name-abs-eq name-fresh-left name-calc)
  ultimately show  $F C P a' x' ([ (x, x') ] \cdot P')$  using  $\langle x' \# P' \rangle \text{aeqa}'$  by blast
qed
  ultimately have  $F C (\langle \nu y \rangle P) a' x' (\langle \nu y \rangle ([ (x, x') ] \cdot P'))$  using  $y \text{ineq} x'$ 
 $y \text{ineq} a y \text{Fresh} C \text{aeqa}'$ 
  by(force intro: cResB)
  with  $P'' \text{eq}$  show ?case by simp
next
  case(ResF P P'  $\alpha$  y)
  thus ?case by(simp add: residualInject)
next
  case(Bang P Rs)
  thus ?case by(force intro: cBang)
qed
qed

lemma boundOutputInduct[consumes 2, case-names Match Mismatch Open Sum1
Sum2 Par1 Par2 Res Bang]:
  fixes P :: pi
  and a :: name
  and x :: name
  and P' :: pi
  and F :: ('a::fs-name)  $\Rightarrow$  pi  $\Rightarrow$  name  $\Rightarrow$  name  $\Rightarrow$  pi  $\Rightarrow$  bool

```

and $C :: 'a::fs\text{-name}$

assumes $a: P \mapsto a \langle \nu x \rangle \prec P'$

and $x \# P$

and $cMatch: \bigwedge P a x P' b C. \llbracket P \mapsto a \langle \nu x \rangle \prec P'; \bigwedge C. F C P a x P' \rrbracket \implies F C ([b \frown b]P) a x P'$

and $cMismatch: \bigwedge P a x P' b c C. \llbracket P \mapsto a \langle \nu x \rangle \prec P'; \bigwedge C. F C P a x P'; b \neq c \rrbracket \implies F C ([b \neq c]P) a x P'$

and $cOpen: \bigwedge P a x P' C. \llbracket P \mapsto (\text{OutputR } a x) \prec P'; a \neq x \rrbracket \implies F C (\langle \nu x \rangle P) a x P'$

and $cSum1: \bigwedge P Q a x P' C. \llbracket P \mapsto a \langle \nu x \rangle \prec P'; \bigwedge C. F C P a x P' \rrbracket \implies F C (P \oplus Q) a x P'$

and $cSum2: \bigwedge P Q a x Q' C. \llbracket Q \mapsto a \langle \nu x \rangle \prec Q'; \bigwedge C. F C Q a x Q' \rrbracket \implies F C (P \oplus Q) a x Q'$

and $cPar1B: \bigwedge P P' Q a x C. \llbracket P \mapsto a \langle \nu x \rangle \prec P'; x \# Q; \bigwedge C. F C P a x P' \rrbracket \implies$

$F C (P \parallel Q) a x (P' \parallel Q)$

and $cPar2B: \bigwedge P Q Q' a x C. \llbracket Q \mapsto a \langle \nu x \rangle \prec Q'; x \# P; \bigwedge C. F C Q a x Q' \rrbracket \implies$

$F C (P \parallel Q) a x (P \parallel Q')$

and $cResB: \bigwedge P P' a x y C. \llbracket P \mapsto a \langle \nu x \rangle \prec P'; y \neq a; y \neq x; y \# C; \bigwedge C. F C P a x P' \rrbracket \implies F C (\langle \nu y \rangle P) a x (\langle \nu y \rangle P')$

and $cBang: \bigwedge P a x P' C. \llbracket P \parallel !P \mapsto a \langle \nu x \rangle \prec P'; \bigwedge C. F C (P \parallel !P) a x P' \rrbracket \implies$

$F C (!P) a x P'$

shows $F C P a x P'$

proof –

from $a \langle x \# P \rangle$ **show** $?thesis$

proof(*nominal-induct* $x2 == a \langle \nu x \rangle \prec P'$ *avoiding: C a x P' rule: transitions.strong-induct*)

case(*Tau P*)

thus $?case$ **by**(*simp add: residualInject*)

next

case(*Input P a x*)

thus $?case$ **by**(*simp add: residualInject*)

next

case(*Output P a b*)

thus $?case$ **by**(*simp add: residualInject*)

next

case(*Match P Rs b C a x*)

thus $?case$

by(*force intro: cMatch simp add: residualInject*)

next

case(*Mismatch P Rs a b C c x*)

thus $?case$

by(*force intro: cMismatch simp add: residualInject*)

next

case(*Sum1 P Q Rs C*)

thus $?case$ **by**(*force intro: cSum1*)

```

next
  case(Sum2 P Q Rs C)
  thus ?case by(force intro: cSum2)
next
  case(Open P a b P' C a' x P'')
  have b # x by fact hence bineqx: b ≠ x by simp
  moreover have a<νb> < P' = a'<νx> < P'' by fact
  ultimately have aeqa': a=a' and P'eqP'': P'' = [(b, x)] · P'
    by(simp add: residualInject name-abs-eq)+
  have x # <νb>P by fact
  with bineqx have x # P by(simp add: name-fresh-abs)
  have aineqb: a ≠ b by fact

  have PTrans: P ↦ a[b] < P' by fact
  with <x # P> have xineqa: x ≠ a by(force dest: freshFreeDerivative)
  from PTrans have ([(b, x)] · P) ↦ [(b, x)] · (a[b] < P') by(rule transi-
tions.eqt)
  with P'eqP'' xineqa aineqb have Trans: ([(b, x)] · P) ↦ a[x] < P''
    by(auto simp add: name-calc)
  hence F C (<νx>([(b, x)] · P)) a x P'' using xineqa by(blast intro: cOpen)
  with <x # P> aeqa' show ?case by(simp add: alphaRes)
next
  case(Par1B P a x P' Q C a' x' P'')
  have x # x' by fact hence xineqx': x ≠ x' by simp
  moreover have Eq: a«x» < (P' || Q) = a'<νx'> < P'' by fact
  hence aeqa': a = BoundOutputS a' by(simp add: residualInject)
  have x # Q by fact
  have x' # P || Q by fact
  hence x' # P and x' # Q by simp+
  have P''eq: P'' = [(x, x')] · P' || Q
  proof -
    from Eq xineqx' have (P' || Q) = [(x, x')] · P''
      by(simp add: residualInject name-abs-eq)
    hence ([(x, x')] · (P' || Q)) = P'' by simp
    with <x' # Q><x # Q> show ?thesis by(simp add: name-fresh-fresh)
  qed

  have x # P'' by fact
  with P''eq <x ≠ x'> have x' # P' by(simp add: name-fresh-left name-calc)

  have P ↦ a«x» < P' by fact
  with <x' # P'> aeqa' have P ↦ a'<νx'> < ([(x, x')] · P')
    by(simp add: alphaBoundResidual)
  moreover have ∧C. F C P a' x' ([(x, x')] · P')
  proof -
    fix C
    have ∧C a' x' P''. [a«x» < P' = a'<νx'> < P''; x' # P] ⇒ F C P a' x'
P'' by fact
    moreover with aeqa' xineqx' <x' # P'> have a«x» < P' = a'<νx'> < ([(x,

```

$x'] \cdot P')$
 by(*simp add: residualInject name-abs-eq name-fresh-left name-calc*)
 ultimately show $F C P a' x' (((x, x') \cdot P')$ using $\langle x' \# P \rangle aeqa'$ by *blast*
 qed
 ultimately have $F C (P \parallel Q) a' x' (((x, x') \cdot P') \parallel Q)$ using $\langle x' \# Q \rangle aeqa'$
 by(*blast intro: cPar1B*)
 with $P''eq$ show ?*case* by *simp*
 next
 case(*Par1F P P' Q α*)
 thus ?*case* by(*simp add: residualInject*)
 next
 case(*Par2B Q a x Q' P C a' x' Q''*)
 have $x \# x'$ by *fact* hence $xineqx'$: $x \neq x'$ by *simp*
 moreover have $Eq: a\langle x \rangle \prec (P \parallel Q') = a'\langle \nu x' \rangle \prec Q''$ by *fact*
 hence $aeqa'$: $a = BoundOutputS a'$ by(*simp add: residualInject*)
 have $x \# P$ by *fact*
 have $x' \# P \parallel Q$ by *fact*
 hence $x' \# P$ and $x' \# Q$ by *simp+*
 have $Q''eq$: $Q'' = P \parallel ((x, x') \cdot Q')$
 proof –
 from Eq $xineqx'$ have $(P \parallel Q') = [(x, x') \cdot Q''$
 by(*simp add: residualInject name-abs-eq*)
 hence $([(x, x') \cdot (P \parallel Q') = Q''$ by *simp*
 with $\langle x' \# P \rangle \langle x \# P \rangle$ show ?*thesis* by(*simp add: name-fresh-fresh*)
 qed

 have $x \# Q''$ by *fact*
 with $Q''eq \langle x \neq x' \rangle$ have $x' \# Q'$ by(*simp add: name-fresh-left name-calc*)

 have $Q \mapsto a\langle x \rangle \prec Q'$ by *fact*
 with $\langle x' \# Q' \rangle aeqa'$ have $Q \mapsto a'\langle \nu x' \rangle \prec ((x, x') \cdot Q')$
 by(*simp add: alphaBoundResidual*)
 moreover have $\bigwedge C. F C Q a' x' ((x, x') \cdot Q')$
 proof –
 fix C
 have $\bigwedge C a' x' Q''. \llbracket a\langle x \rangle \prec Q' = a'\langle \nu x' \rangle \prec Q''; x' \# Q \rrbracket \implies F C Q a' x'$
 Q'' by *fact*
 moreover with $aeqa' xineqx' \langle x' \# Q' \rangle$ have $a\langle x \rangle \prec Q' = a'\langle \nu x' \rangle \prec ((x,$
 $x') \cdot Q')$
 by(*simp add: residualInject name-abs-eq name-fresh-left name-calc*)
 ultimately show $F C Q a' x' ((x, x') \cdot Q')$ using $\langle x' \# Q \rangle aeqa'$ by *blast*
 qed
 ultimately have $F C (P \parallel Q) a' x' (P \parallel ((x, x') \cdot Q'))$ using $\langle x' \# P \rangle$
 by(*blast intro: cPar2B*)
 with $Q''eq$ show ?*case* by *simp*
 next
 case(*Par2F P P' Q α*)
 thus ?*case* by(*simp add: residualInject*)
 next

```

    case(Comm1 P P' Q Q' a b x)
    thus ?case by(simp add: residualInject)
next
    case(Comm2 P P' Q Q' a b x)
    thus ?case by(simp add: residualInject)
next
    case(Close1 P P' Q Q' a x y)
    thus ?case by(simp add: residualInject)
next
    case(Close2 P P' Q Q' a x y)
    thus ?case by(simp add: residualInject)
next
    case(ResB P a x P' y C a' x' P'')
    have x # x' by fact hence xineqx': x ≠ x' by simp
    moreover have Eq: a«x» < (<νy>P') = a'<νx'> < P'' by fact
    hence aeqa': a = BoundOutputS a' by(simp add: residualInject)
    have y # x' by fact hence yineqx': y ≠ x' by simp
    moreover have x' # <νy>P by fact
    ultimately have x' # P by(simp add: name-fresh-abs)
    have y ≠ x and y # a and yFreshC: y # C by fact+

    have P''eq: P'' = <νy>([(x, x')] · P')
    proof -
      from Eq yineqx' have <νy>P' = [(x, x')] · P''
      by(simp add: residualInject name-abs-eq)
      hence ([(x, x')] · (<νy>P')) = P'' by simp
      with yineqx' ⟨y ≠ x⟩ show ?thesis by(simp add: name-fresh-fresh)
    qed

    have x # P'' by fact
    with P''eq ⟨y ≠ x⟩ ⟨x ≠ x'⟩ have x' # P' by(simp add: name-fresh-left name-calc
name-fresh-abs)

    have P ⟶a«x» < P' by fact
    with ⟨x' # P'⟩ aeqa' have P ⟶a'<νx'> < ([(x, x')] · P')
    by(simp add: alphaBoundResidual)
    moreover have ∧C. F C P a' x' ([(x, x')] · P')
    proof -
      fix C
      have ∧C a' x' P''. [[a«x» < P' = a'<νx'> < P''; x' # P]] ⟹ F C P a' x'
P'' by fact
      moreover with aeqa' yineqx' ⟨x' # P'⟩ have a«x» < P' = a'<νx'> < ([(x,
x')] · P')
      by(simp add: residualInject name-abs-eq name-fresh-left name-calc)
      ultimately show F C P a' x' ([(x, x')] · P') using ⟨x' # P'⟩ aeqa' by blast
    qed
    ultimately have F C (<νy>P) a' x' (<νy>([(x, x')] · P')) using yineqx' ⟨y
# a⟩ yFreshC aeqa'
    by(force intro: cResB)

```

```

with P''eq show ?case by simp
next
case(ResF P P' α y)
thus ?case by(simp add: residualInject)
next
case(Bang P Rs)
thus ?case by(force intro: cBang)
qed
qed

```

lemma tauInduct[consumes 1, case-names Tau Match Mismatch Sum1 Sum2 Par1 Par2 Comm1 Comm2 Close1 Close2 Res Bang]:

```

fixes P :: pi
and P' :: pi
and F :: 'a::fs-name ⇒ pi ⇒ pi ⇒ bool
and C :: 'a::fs-name

assumes Trans: P ⟶τ < P'
and A1: ⋀P C. F C (τ.(P)) P
and A2: ⋀P P' c C. [P ⟶τ < P'; ⋀C. F C P P'] ⇒ F C ([c↔c]P) P'
and A3: ⋀P P' c d C. [P ⟶τ < P'; ⋀C. F C P P'; c ≠ d] ⇒ F C ([c≠d]P) P'

and A4: ⋀P P' Q C. [P ⟶τ < P'; ⋀C. F C P P'] ⇒ F C (P ⊕ Q) P'
and A5: ⋀Q Q' P C. [Q ⟶τ < Q'; ⋀C. F C Q Q'] ⇒ F C (P ⊕ Q) Q'
and A6: ⋀P P' Q C. [P ⟶τ < P'; ⋀C. F C P P'] ⇒ F C (P ∥ Q) (P' ∥ Q)
and A7: ⋀Q Q' P C. [Q ⟶τ < Q'; ⋀C. F C Q Q'] ⇒ F C (P ∥ Q) (P ∥ Q')
and A8: ⋀P a x P' Q b Q' C. [P ⟶(BoundR (InputS a) x P'); Q ⟶OutputR a b < Q'; x # P; x # Q; x # C] ⇒ F C (P ∥ Q) (P'[x::=b] ∥ Q')
and A9: ⋀P a b P' Q x Q' C. [P ⟶OutputR a b < P'; Q ⟶(BoundR (InputS a) x Q'); x # P; x # Q; x # C] ⇒ F C (P ∥ Q) (P' ∥ Q'[x::=b])
and A10: ⋀P a x P' Q y Q' C. [P ⟶(BoundR (InputS a) x P'); Q ⟶a<νy> < Q'; x # P; x # Q; x # C; y # P; y # Q; y # C; x ≠ y] ⇒ F C (P ∥ Q) (<νy>(P'[x::=y] ∥ Q'))
and A11: ⋀P a y P' Q x Q' C. [P ⟶a<νy> < P'; Q ⟶(BoundR (InputS a) x Q'); x # P; x # Q; x # C; y # P; y # Q; y # C; x ≠ y] ⇒ F C (P ∥ Q) (<νy>(P' ∥ Q'[x::=y]))
and A12: ⋀P P' x C. [P ⟶τ < P'; x # C; ⋀C. F C P P'] ⇒ F C (<νx>P) (<νx>P')
and A13: ⋀P P' C. [P ∥ !P ⟶τ < P'; ⋀C. F C (P ∥ !P) P'] ⇒ F C (!P) P'

```

shows F C P P'

proof –

from Trans show ?thesis

by(nominal-induct x2==τ < P' avoiding: C arbitrary: P' rule: transitions.strong-induct, auto simp add: residualInject intro: assms)

qed

inductive bangPred :: pi ⇒ pi ⇒ bool

where

$aux1: \text{bangPred } P (!P)$
 $| aux2: \text{bangPred } P (P \parallel !P)$

inductive-cases $nilCases'$ [*simplified pi.distinct residual.distinct*]: $\mathbf{0} \mapsto Rs$
inductive-cases $tauCases'$ [*simplified pi.distinct residual.distinct*]: $\tau.(P) \mapsto Rs$
inductive-cases $inputCases'$ [*simplified pi.inject residualInject*]: $a \langle b \rangle . P \mapsto Rs$
inductive-cases $outputCases'$ [*simplified pi.inject residualInject*]: $a \{ b \} . P \mapsto Rs$
inductive-cases $matchCases'$ [*simplified pi.inject residualInject*]: $[a \frown b] P \mapsto Rs$
inductive-cases $mismatchCases'$ [*simplified pi.inject residualInject*]: $[a \neq b] P \mapsto Rs$
inductive-cases $sumCases'$ [*simplified pi.inject residualInject*]: $P \oplus Q \mapsto Rs$
inductive-cases $parCasesB'$ [*simplified pi.distinct residual.distinct*]: $P \parallel Q \mapsto b \ll y \gg \prec P'$
inductive-cases $parCasesF'$ [*simplified pi.distinct residual.distinct*]: $P \parallel Q \mapsto \alpha \prec P'$
inductive-cases $resCases'$ [*simplified pi.distinct residual.distinct*]: $\langle \nu x \rangle P \mapsto Rs$
inductive-cases $resCasesB'$ [*simplified pi.distinct residual.distinct*]: $\langle \nu x \rangle P \mapsto a \ll y' \gg \prec P'$
inductive-cases $resCasesF'$ [*simplified pi.distinct residual.distinct*]: $\langle \nu x \rangle P \mapsto \alpha \prec P'$
inductive-cases $bangCases$ [*simplified pi.distinct residual.distinct*]: $!P \mapsto Rs$

lemma $tauCases$ [*consumes 1, case-names cTau*]:

fixes $P :: pi$
and $\alpha :: freeRes$
and $P' :: pi$

assumes $\tau.(P) \mapsto \alpha \prec P'$
and $[\alpha = \tau; P = P'] \implies Prop (\tau) P$

shows $Prop \alpha P'$

using *assms*

by(*erule-tac tauCases', auto simp add: pi.inject residualInject*)

lemma $outputCases$ [*consumes 1, case-names cOutput*]:

fixes $a :: name$
and $b :: name$
and $P :: pi$
and $\alpha :: freeRes$
and $P' :: pi$

assumes $a \{ b \} . P \mapsto \alpha \prec P'$
and $[\alpha = a[b]; P = P'] \implies Prop (a[b]) P$

shows $Prop \alpha P'$

using *assms*

by(*erule-tac outputCases', auto simp add: residualInject*)

lemma $zeroTrans$ [*dest*]:

```

fixes  $R_s :: residual$ 

assumes  $0 \mapsto R_s$ 

shows  $False$ 
using  $assms$ 
by( $induct\ rule: nilCases', auto$ )

lemma  $resZeroTrans[dest]$ :
  fixes  $x :: name$ 
  and  $R_s :: residual$ 

  assumes  $\langle \nu x \rangle 0 \mapsto R_s$ 

  shows  $False$ 
using  $assms$ 
by( $induct\ rule: resCases', auto\ simp\ add: pi.inject\ alpha'$ )

lemma  $matchTrans[dest]$ :
  fixes  $a :: name$ 
  and  $b :: name$ 
  and  $P :: pi$ 
  and  $R_s :: residual$ 

  assumes  $[a \curvearrowright b]P \mapsto R_s$ 
  and  $a \neq b$ 

  shows  $False$ 
using  $assms$ 
by( $induct\ rule: matchCases', auto$ )

lemma  $mismatchTrans[dest]$ :
  fixes  $a :: name$ 
  and  $P :: pi$ 
  and  $R_s :: residual$ 

  assumes  $[a \neq a]P \mapsto R_s$ 

  shows  $False$ 
using  $assms$ 
by( $induct\ rule: mismatchCases', auto$ )

lemma  $inputCases[consumes\ 4, case-names\ cInput]$ :
  fixes  $a :: name$ 
  and  $x :: name$ 
  and  $P :: pi$ 
  and  $P' :: pi$ 

  assumes  $Input: a \langle x \rangle . P \mapsto b \langle y \rangle \prec y P'$ 

```

and $y \neq a$
and $y \neq x$
and $y \# P$
and $A: \llbracket b = \text{InputS } a; yP' = ((x, y)) \cdot P \rrbracket \implies \text{Prop } (\text{InputS } a) y ((x, y)) \cdot P$

shows $\text{Prop } b y yP'$

proof –

note assms

moreover from $\text{Input } \langle y \neq a \rangle \langle y \neq x \rangle \langle y \# P \rangle$ **have** $y \# b$

by($\text{force dest: freshBoundDerivative simp add: abs-fresh}$)

moreover obtain $z::\text{name}$ **where** $z \neq y$ **and** $z \neq x$ **and** $z \# P$ **and** $z \neq a$ **and** $z \# b$ **and** $z \# yP'$

by($\text{generate-fresh name, auto simp add: fresh-prod}$)

moreover obtain $z'::\text{name}$ **where** $z' \neq y$ **and** $z' \neq x$ **and** $z' \neq z$ **and** $z' \# P$ **and** $z' \neq a$ **and** $z' \# b$ **and** $z' \# yP'$

by($\text{generate-fresh name, auto simp add: fresh-prod}$)

ultimately show $?thesis$

by($\text{cases rule: transitions.strong-cases}$ [**where** $x=y$ **and** $b=z$ **and** $xa=z$ **and** $xb=z$ **and** $xc=z$ **and** $xd=z$ **and** $xe=z$

and $xf=z$ **and** $xg=z$ **and** $y=z'$ **and** $ya=z'$

and $yb=y$ **and** $yc=z'$])

($\text{auto simp add: pi.inject residualInject alpha abs-fresh fresh-prod fresh-left calc-atm}$)+

qed

lemma $\text{tauBoundTrans}[dest]:$

fixes $P :: pi$

and $a :: \text{subject}$

and $x :: \text{name}$

and $P' :: pi$

assumes $\tau.(P) \mapsto_a \langle x \rangle \prec P'$

shows False

using assms

by – ($\text{ind-cases } \tau.(P) \mapsto_a \langle x \rangle \prec P'$)

lemma $\text{tauOutputTrans}[dest]:$

fixes $P :: pi$

and $a :: \text{name}$

and $b :: \text{name}$

and $P' :: pi$

assumes $\tau.(P) \mapsto_a [b] \prec P'$

shows False

using assms

by – ($\text{ind-cases } \tau.(P) \mapsto_a [b] \prec P'$, $\text{auto simp add: residualInject}$)

lemma *inputFreeTrans*[*dest*]:

fixes *a* :: *name*
and *x* :: *name*
and *P* :: *pi*
and α :: *freeRes*
and *P'* :: *pi*

assumes $a\langle x \rangle.P \mapsto \alpha \prec P'$

shows *False*

using *assms*

by - (*ind-cases* $a\langle x \rangle.P \mapsto \alpha \prec P'$)

lemma *inputBoundOutputTrans*[*dest*]:

fixes *a* :: *name*
and *x* :: *name*
and *P* :: *pi*
and *b* :: *name*
and *y* :: *name*
and *P'* :: *pi*

assumes $a\langle x \rangle.P \mapsto b\langle \nu y \rangle \prec P'$

shows *False*

using *assms*

by - (*ind-cases* $a\langle x \rangle.P \mapsto b\langle \nu y \rangle \prec P'$, *auto simp add: residualInject*)

lemma *outputTauTrans*[*dest*]:

fixes *a* :: *name*
and *b* :: *name*
and *P* :: *pi*
and *P'* :: *pi*

assumes $a\{b\}.P \mapsto \tau \prec P'$

shows *False*

using *assms*

by - (*ind-cases* $a\{b\}.P \mapsto \tau \prec P'$, *auto simp add: residualInject*)

lemma *outputBoundTrans*[*dest*]:

fixes *a* :: *name*
and *b* :: *name*
and *P* :: *pi*
and *c* :: *subject*
and *x* :: *name*
and *P'* :: *pi*

assumes $a\{b\}.P \mapsto c\langle x \rangle \prec P'$

shows *False*
using *assms*
by – (*ind-cases* $a\{b\}.P \mapsto c\langle x \rangle \prec P'$)

lemma *outputIneqTrans[dest]*:

fixes $a :: \text{name}$
and $b :: \text{name}$
and $P :: \text{pi}$
and $c :: \text{name}$
and $d :: \text{name}$
and $P' :: \text{pi}$

assumes $a\{b\}.P \mapsto c[d] \prec P'$
and $a \neq c \vee b \neq d$

shows *False*
using *assms*
by – (*ind-cases* $a\{b\}.P \mapsto c[d] \prec P'$, *auto simp add: residualInject pi.inject alpha'*)

lemma *outputFreshTrans[dest]*:

fixes $a :: \text{name}$
and $b :: \text{name}$
and $P :: \text{pi}$
and $\alpha :: \text{freeRes}$
and $P' :: \text{pi}$

assumes $a\{b\}.P \mapsto \alpha \prec P'$
and $a \# \alpha \vee b \# \alpha$

shows *False*
using *assms*
by – (*ind-cases* $a\{b\}.P \mapsto \alpha \prec P'$, *auto simp add: residualInject pi.inject alpha'*)

lemma *inputIneqTrans[dest]*:

fixes $a :: \text{name}$
and $x :: \text{name}$
and $P :: \text{pi}$
and $b :: \text{subject}$
and $y :: \text{name}$
and $P' :: \text{pi}$

assumes $a\langle x \rangle.P \mapsto b\langle y \rangle \prec P'$
and $a \# b$

shows *False*
using *assms*
by – (*ind-cases* $a\langle x \rangle.P \mapsto b\langle y \rangle \prec P'$, *auto simp add: residualInject pi.inject*)

lemma *resTauBoundTrans*[*dest*]:

fixes *x* :: *name*
and *P* :: *pi*
and *a* :: *subject*
and *y* :: *name*
and *P'* :: *pi*

assumes $\langle \nu x \rangle \tau.(P) \mapsto a \langle y \rangle \prec P'$

shows *False*

using *assms*

by $-$ (*ind-cases* $\langle \nu x \rangle \tau.(P) \mapsto a \langle y \rangle \prec P'$, *auto simp add: residualInject pi.inject alpha'*)

lemma *resTauOutputTrans*[*dest*]:

fixes *x* :: *name*
and *P* :: *pi*
and *a* :: *name*
and *b* :: *name*
and *P'* :: *pi*

assumes $\langle \nu x \rangle \tau.(P) \mapsto a[b] \prec P'$

shows *False*

using *assms*

by $-$ (*ind-cases* $\langle \nu x \rangle \tau.(P) \mapsto a[b] \prec P'$, *auto simp add: residualInject pi.inject alpha'*)

lemma *resInputFreeTrans*[*dest*]:

fixes *x* :: *name*
fixes *a* :: *name*
and *y* :: *name*
and *P* :: *pi*
and α :: *freeRes*
and *P'* :: *pi*

assumes $\langle \nu x \rangle a \langle y \rangle . P \mapsto \alpha \prec P'$

shows *False*

using *assms*

by $-$ (*ind-cases* $\langle \nu x \rangle a \langle y \rangle . P \mapsto \alpha \prec P'$, *auto simp add: pi.inject residualInject alpha'*)

lemma *resInputBoundOutputTrans*[*dest*]:

fixes *x* :: *name*
and *a* :: *name*
and *y* :: *name*
and *P* :: *pi*

```

and  $b :: name$ 
and  $z :: name$ 
and  $P' :: pi$ 

assumes  $\langle \nu x \rangle a \langle y \rangle . P \mapsto b \langle \nu z \rangle \prec P'$ 

shows False
using assms
by - (ind-cases  $\langle \nu x \rangle a \langle y \rangle . P \mapsto b \langle \nu z \rangle \prec P'$ , auto simp add: pi.inject residualInject alpha')

lemma resOutputTauTrans[dest]:
  fixes  $x :: name$ 
  and  $a :: name$ 
  and  $b :: name$ 
  and  $P :: pi$ 
  and  $P' :: pi$ 

  assumes  $\langle \nu x \rangle a \{b\} . P \mapsto \tau \prec P'$ 

  shows False
using assms
by - (ind-cases  $\langle \nu x \rangle a \{b\} . P \mapsto \tau \prec P'$ , auto simp add: residualInject pi.inject alpha')

lemma resOutputInputTrans[dest]:
  fixes  $x :: name$ 
  and  $a :: name$ 
  and  $b :: name$ 
  and  $P :: pi$ 
  and  $c :: name$ 
  and  $y :: name$ 
  and  $P' :: pi$ 

  assumes  $\langle \nu x \rangle a \{b\} . P \mapsto c \langle y \rangle \prec P'$ 

  shows False
using assms
by - (ind-cases  $\langle \nu x \rangle a \{b\} . P \mapsto c \langle y \rangle \prec P'$ , auto simp add: pi.inject residualInject alpha')

lemma resOutputOutputTrans[dest]:
  fixes  $x :: name$ 
  and  $a :: name$ 
  and  $P :: pi$ 
  and  $b :: name$ 
  and  $y :: name$ 
  and  $P' :: pi$ 

```

```

assumes  $\langle \nu x \rangle a\{x\}.P \mapsto b[y] \prec P'$ 

shows False
using assms
by - (ind-cases  $\langle \nu x \rangle a\{x\}.P \mapsto b[y] \prec P'$ , auto simp add: pi.inject residualInject
alpha' calc-atm)

lemma resTrans[dest]:
  fixes  $x :: \text{name}$ 
  and  $b :: \text{name}$ 
  and  $Rs :: \text{residual}$ 
  and  $y :: \text{name}$ 

  shows  $\langle \nu x \rangle x\{b\}.P \mapsto Rs \implies \text{False}$ 
  and  $\langle \nu x \rangle x\langle y \rangle.P \mapsto Rs \implies \text{False}$ 
apply(ind-cases  $\langle \nu x \rangle x\{b\}.P \mapsto Rs$ , auto simp add: pi.inject alpha' calc-atm)
by(ind-cases  $\langle \nu x \rangle x\langle y \rangle.P \mapsto Rs$ , auto simp add: pi.inject alpha' calc-atm abs-fresh
fresh-left)

lemma matchCases[consumes 1, case-names cMatch]:
  fixes  $a :: \text{name}$ 
  and  $b :: \text{name}$ 
  and  $P :: \text{pi}$ 
  and  $Rs :: \text{residual}$ 
  and  $F :: \text{name} \Rightarrow \text{name} \Rightarrow \text{bool}$ 

  assumes  $[a \frown b]P \mapsto Rs$ 
  and  $\llbracket P \mapsto Rs; a = b \rrbracket \implies F a a$ 

  shows  $F a b$ 
using assms
by(induct rule: matchCases', auto)

lemma mismatchCases[consumes 1, case-names cMismatch]:
  fixes  $a :: \text{name}$ 
  and  $b :: \text{name}$ 
  and  $P :: \text{pi}$ 
  and  $Rs :: \text{residual}$ 
  and  $F :: \text{name} \Rightarrow \text{name} \Rightarrow \text{bool}$ 

  assumes Trans:  $[a \neq b]P \mapsto Rs$ 
  and  $cMatch$ :  $\llbracket P \mapsto Rs; a \neq b \rrbracket \implies F a b$ 

  shows  $F a b$ 
using assms
by(induct rule: mismatchCases', auto)

lemma sumCases[consumes 1, case-names cSum1 cSum2]:
  fixes  $P :: \text{pi}$ 

```

```

and  $Q :: pi$ 
and  $Rs :: residual$ 

assumes  $Trans: P \oplus Q \mapsto Rs$ 
and  $cSum1: P \mapsto Rs \implies Prop$ 
and  $cSum2: Q \mapsto Rs \implies Prop$ 

shows  $Prop$ 
using  $assms$ 
by( $induct$  rule:  $sumCases'$ ,  $auto$ )

lemma  $name-abs-alpha$ :
  fixes  $a :: name$ 
  and  $b :: name$ 
  and  $P :: pi$ 

  assumes  $b \# P$ 

  shows  $[a].P = [b].([a, b] \cdot P)$ 
proof( $cases$   $a=b$ ,  $auto$ )
  assume  $a \neq b$ 
  with  $assms$  show  $?thesis$ 
  by( $force$   $intro$ :  $abs-fun-eq3$ [ $OF$   $pt-name-inst$ ,  $OF$   $at-name-inst$ ]
     $simp$   $add$ :  $name-swap$   $name-calc$   $name-fresh-left$ )
qed

lemma  $parCasesB$ [ $consumes$  3,  $case-names$   $cPar1$   $cPar2$ ]:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $a :: subject$ 
  and  $x :: name$ 
  and  $PQ' :: pi$ 
  and  $C :: 'a::fs-name$ 

  assumes  $P \parallel Q \mapsto a\langle x \rangle \prec PQ'$ 
  and  $x \# P$ 
  and  $x \# Q$ 
  and  $\bigwedge P'. P \mapsto a\langle x \rangle \prec P' \implies Prop (P' \parallel Q)$ 
  and  $\bigwedge Q'. Q \mapsto a\langle x \rangle \prec Q' \implies Prop (P \parallel Q')$ 

  shows  $Prop PQ'$ 
proof –
  note  $assms$ 
  moreover from  $\langle P \parallel Q \mapsto a\langle x \rangle \prec PQ' \rangle \langle x \# P \rangle \langle x \# Q \rangle$  have  $x \# a$ 
  by( $force$   $dest$ :  $freshBoundDerivative$ )
  moreover obtain  $y::name$  where  $y \neq x$  and  $y \# P$  and  $y \# Q$  and  $y \# a$  and
 $y \# PQ'$ 
  by( $generate-fresh$   $name$ ,  $auto$   $simp$   $add$ :  $fresh-prod$ )
  moreover obtain  $z::name$  where  $z \neq y$  and  $z \neq x$  and  $z \# P$  and  $z \# Q$  and

```

$z \# a$ and $z \# PQ'$
 by(*generate-fresh name, auto simp add: fresh-prod*)
 ultimately show ?thesis
 by(*cases rule: transitions.strong-cases[where $x=y$ and $b=y$ and $xa=x$ and $xb=x$ and $xc=y$ and $xd=y$ and $xe=y$ and $xf=y$ and $xg=y$ and $y=z$ and $ya = z$ and $yb=z$ and $yc=z$]*)
 (*auto simp add: pi.inject residualInject alpha abs-fresh fresh-prod*)+
 qed

lemma *parCasesF*[*consumes 1, case-names cPar1 cPar2 cComm1 cComm2 cClose1 cClose2*]:

fixes $P :: pi$
 and $Q :: pi$
 and $\alpha :: freeRes$
 and $P' :: pi$
 and $C :: 'a::fs-name$
 and $F :: freeRes \Rightarrow pi \Rightarrow bool$

assumes *Trans*: $P \parallel Q \mapsto \alpha \prec PQ'$
 and *icPar1F*: $\bigwedge P'. \llbracket P \mapsto \alpha \prec P' \rrbracket \Longrightarrow F \alpha (P' \parallel Q)$
 and *icPar2F*: $\bigwedge Q'. \llbracket Q \mapsto \alpha \prec Q' \rrbracket \Longrightarrow F \alpha (P \parallel Q')$
 and *icComm1*: $\bigwedge P' Q' a b x. \llbracket P \mapsto a \langle x \rangle \prec P'; Q \mapsto a[b] \prec Q'; x \# P; x \# Q; x \neq a; x \neq b; x \# Q'; x \# C; \alpha = \tau \rrbracket \Longrightarrow F (\tau) (P'[x::=b] \parallel Q')$
 and *icComm2*: $\bigwedge P' Q' a b x. \llbracket P \mapsto a[b] \prec P'; Q \mapsto a \langle x \rangle \prec Q'; x \# P; x \# Q; x \neq a; x \neq b; x \# P'; x \# C; \alpha = \tau \rrbracket \Longrightarrow F (\tau) (P' \parallel Q'[x::=b])$
 and *icClose1*: $\bigwedge P' Q' a x y. \llbracket P \mapsto a \langle x \rangle \prec P'; Q \mapsto a \langle \nu y \rangle \prec Q'; x \# P; x \# Q; x \neq a; x \neq y; x \# Q'; y \# P; y \# Q; y \neq a; y \# P'; x \# C; y \# C; \alpha = \tau \rrbracket \Longrightarrow$

$$F (\tau) (\langle \nu y \rangle (P'[x::=y] \parallel Q'))$$
 and *icClose2*: $\bigwedge P' Q' a x y. \llbracket P \mapsto a \langle \nu y \rangle \prec P'; Q \mapsto a \langle x \rangle \prec Q'; x \# P; x \# Q; x \neq a; x \neq y; x \# P'; y \# P; y \# Q; y \neq a; y \# Q'; x \# C; y \# C; \alpha = \tau \rrbracket \Longrightarrow$

$$F (\tau) (\langle \nu y \rangle (P' \parallel Q'[x::=y]))$$

shows $F \alpha PQ'$
 proof –
 note *assms*
 moreover obtain $x::name$ where $x \# P$ and $x \# Q$ and $x \# \alpha$ and $x \# PQ'$ and $x \# C$
 by(*generate-fresh name, auto simp add: fresh-prod*)
 moreover obtain $y::name$ where $y \# P$ and $y \# Q$ and $y \# \alpha$ and $y \# PQ'$ and $y \# C$ and $x \neq y$
 by(*generate-fresh name, auto simp add: fresh-prod*)
 ultimately show ?thesis
 by(*cases rule: transitions.strong-cases[where $x=x$ and $b=x$ and $xa=x$ and $xb=x$ and $xc=x$ and $xd=x$ and $xe=x$ and $xf=x$ and $xg=x$ and $y=y$ and $ya=y$]*)

and $yb=y$ **and** $yc=y$)
 (auto simp add: pi.inject residualInject alpha abs-fresh fresh-prod)+
qed

lemma *resCasesF*[consumes 1, case-names *cRes*]:

fixes $x :: name$
and $P :: pi$
and $\alpha :: freeRes$
and $P' :: pi$
and $C :: 'a::fs-name$

assumes $\langle \nu x \rangle P \mapsto \alpha \prec xP'$
and $\bigwedge P'. \llbracket P \mapsto \alpha \prec P'; x \# \alpha \rrbracket \implies F (\langle \nu x \rangle P')$

shows $F xP'$

proof –

note *assms*

moreover from $\langle \langle \nu x \rangle P \mapsto \alpha \prec xP' \rangle$ **have** $x \# \alpha$ **and** $x \# xP'$

by(force dest: freshFreeDerivative simp add: abs-fresh)+

moreover obtain $y::name$ **where** $y \neq x$ **and** $y \# P$ **and** $y \# \alpha$ **and** $y \# xP'$

by(generate-fresh name, auto simp add: fresh-prod)

moreover obtain $z::name$ **where** $z \neq y$ **and** $z \neq x$ **and** $z \# P$ **and** $z \# \alpha$ **and**
 $z \# xP'$

by(generate-fresh name, auto simp add: fresh-prod)

ultimately show *?thesis*

by(cases rule: transitions.strong-cases[**where** $x=y$ **and** $b=y$ **and** $xa=y$ **and**
 $xb=y$ **and** $xc=y$ **and** $xd=y$ **and** $xe=y$

and $xf=y$ **and** $xg=y$ **and** $y=z$ **and** $ya=z$

and $yb=z$ **and** $yc=x$)

(auto simp add: pi.inject residualInject alpha abs-fresh fresh-prod)+

qed

lemma *resCasesB*[consumes 3, case-names *cOpen cRes*]:

fixes $x :: name$
and $P :: pi$
and $a :: subject$
and $y :: name$
and $yP' :: pi$
and $C :: 'a::fs-name$

assumes *Trans*: $\langle \nu y \rangle P \mapsto a \langle x \rangle \prec yP'$

and *xineqy*: $x \neq y$

and *xineqy*: $x \# P$

and *rcOpen*: $\bigwedge b P'. \llbracket P \mapsto b[y] \prec P'; b \neq y; a = BoundOutputS b \rrbracket \implies F$
 (*BoundOutputS* b) ($[(x, y)] \cdot P'$)

and *rcResB*: $\bigwedge P'. \llbracket P \mapsto a \langle x \rangle \prec P'; y \# a \rrbracket \implies F a (\langle \nu y \rangle P')$

shows $F a yP'$

proof –

note *assms*
moreover from $\langle \langle \nu y \rangle P \mapsto a \langle x \rangle \prec y P' \rangle \langle x \neq y \rangle$ **have** $y \# a$ **and** $y \# y P'$
by(*force dest: freshBoundDerivative simp add: abs-fresh*)
moreover from $\langle \langle \nu y \rangle P \mapsto a \langle x \rangle \prec y P' \rangle \langle x \# P \rangle$ **have** $x \# a$
by(*force dest: freshBoundDerivative simp add: abs-fresh*)
moreover obtain $z :: \text{name}$ **where** $z \neq y$ **and** $z \neq x$ **and** $z \# P$ **and** $z \# a$ **and**
 $z \# y P'$
by(*generate-fresh name, auto simp add: fresh-prod*)
moreover obtain $z' :: \text{name}$ **where** $z' \neq y$ **and** $z' \neq x$ **and** $z' \neq z$ **and** $z' \# P$
and $z' \# a$ **and** $z' \# y P'$
by(*generate-fresh name, auto simp add: fresh-prod*)
ultimately show *?thesis*
by(*cases rule: transitions.strong-cases*[**where** $x=z$ **and** $b=y$ **and** $xa=z$ **and**
 $xb=z$ **and** $xc=z$ **and** $xd=z$ **and** $xe=z$ **and** $xf=z$ **and** $xg=x$ **and** $y=z'$ **and** $ya=z'$
and $yb=y$ **and** $yc=z'$])
(auto simp add: pi.inject residualInject alpha abs-fresh fresh-prod fresh-left calc-atm)
qed

lemma *bangInduct*[*consumes 1, case-names cPar1B cPar1F cPar2B cPar2F cComm1*
cComm2 cClose1 cClose2 cBang]:

fixes $F :: 'a :: \text{fs-name} \Rightarrow \text{pi} \Rightarrow \text{residual} \Rightarrow \text{bool}$
and $P :: \text{pi}$
and $Rs :: \text{residual}$
and $C :: 'a :: \text{fs-name}$

assumes *Trans*: $!P \mapsto Rs$
and *cPar1B*: $\bigwedge a x P' C. \llbracket P \mapsto a \langle x \rangle \prec P'; x \# P; x \# C \rrbracket \Longrightarrow F C (P \parallel !P) (a \langle x \rangle \prec P' \parallel !P)$
and *cPar1F*: $\bigwedge \alpha P' C. \llbracket P \mapsto \alpha \prec P' \rrbracket \Longrightarrow F C (P \parallel !P) (\alpha \prec P' \parallel !P)$
and *cPar2B*: $\bigwedge a x P' C. \llbracket !P \mapsto a \langle x \rangle \prec P'; x \# P; x \# C; \bigwedge C. F C (!P) (a \langle x \rangle \prec P') \rrbracket \Longrightarrow$
 $F C (P \parallel !P) (a \langle x \rangle \prec P \parallel P')$
and *cPar2F*: $\bigwedge \alpha P' C. \llbracket !P \mapsto \alpha \prec P'; \bigwedge C. F C (!P) (\alpha \prec P') \rrbracket \Longrightarrow F C (P \parallel !P) (\alpha \prec P \parallel P')$
and *cComm1*: $\bigwedge a x P' b P'' C. \llbracket P \mapsto a \langle x \rangle \prec P'; !P \mapsto (\text{OutputR } a \ b) \prec P''; x \# C; \bigwedge C. F C (!P) ((\text{OutputR } a \ b) \prec P'') \rrbracket \Longrightarrow$
 $F C (P \parallel !P) (\tau \prec (P'[x::=b]) \parallel P'')$
and *cComm2*: $\bigwedge a b P' x P'' C. \llbracket P \mapsto (\text{OutputR } a \ b) \prec P'; !P \mapsto a \langle x \rangle \prec P''; x \# C; \bigwedge C. F C (!P) (a \langle x \rangle \prec P'') \rrbracket \Longrightarrow$
 $F C (P \parallel !P) (\tau \prec P' \parallel (P''[x::=b]))$
and *cClose1*: $\bigwedge a x P' y P'' C. \llbracket P \mapsto a \langle x \rangle \prec P'; !P \mapsto a \langle \nu y \rangle \prec P''; y \# P; x \# C; y \# C; \bigwedge C. F C (!P) (a \langle \nu y \rangle \prec P'') \rrbracket \Longrightarrow$
 $F C (P \parallel !P) (\tau \prec \langle \nu y \rangle ((P'[x::=y]) \parallel P''))$
and *cClose2*: $\bigwedge a y P' x P'' C. \llbracket P \mapsto a \langle \nu y \rangle \prec P'; !P \mapsto a \langle x \rangle \prec P''; y$

```

# P; x # C; y # C;
                                 $\bigwedge C. F C (!P) (a \langle x \rangle \prec P'') \implies$ 
                                 $F C (P \parallel !P) (\tau \prec \langle \nu y \rangle (P' \parallel (P''[x::=y])))$ 
and    $cBang: \bigwedge Rs C. \llbracket P \parallel !P \mapsto Rs; \bigwedge C. F C (P \parallel !P) Rs \rrbracket \implies F C (!P)$ 
Rs

shows  $F C (!P) Rs$ 
proof –
have  $\bigwedge X Y C. \llbracket X \mapsto Y; bangPred P X \rrbracket \implies F C X Y$ 
proof –
  fix  $X Y C$ 
  assume  $X \mapsto Y$  and  $bangPred P X$ 
  thus  $F C X Y$ 
  proof(nominal-induct avoiding: C rule: transitions.strong-induct)
    case(Tau Pa)
    thus ?case
    apply –
    by(ind-cases bangPred P ( $\tau.(Pa)$ ))
  next
    case(Input x a Pa)
    thus ?case
    apply –
    by(ind-cases bangPred P ( $a \langle x \rangle . Pa$ ))
  next
    case(Output a b Pa)
    thus ?case
    apply –
    by(ind-cases bangPred P ( $a \{ b \} . Pa$ ))
  next
    case(Match Pa Rs b)
    thus ?case
    apply –
    by(ind-cases bangPred P ( $[b \frown b] Pa$ ))
  next
    case(Mismatch Pa Rs a b)
    thus ?case
    apply –
    by(ind-cases bangPred P ( $[a \neq b] Pa$ ))
  next
    case(Open Pa a b Pa')
    thus ?case
    apply –
    by(ind-cases bangPred P ( $\langle \nu b \rangle Pa$ ))
  next
    case(Sum1 Pa Rs Q)
    thus ?case
    apply –
    by(ind-cases bangPred P ( $Pa \oplus Q$ ))
  next

```

```

case(Sum2 Q Rs Pa)
thus ?case
  apply -
  by(ind-cases bangPred P (Pa ⊕ Q))
next
case(Par1B Pa a x Pa' Q)
thus ?case
  apply -
  by(ind-cases bangPred P (Pa || Q), auto intro: cPar1B simp add: pi.inject)
next
case(Par1F Pa α Pa' Q)
thus ?case
  apply -
  by(ind-cases bangPred P (Pa || Q), auto intro: cPar1F simp add: pi.inject)
next
case(Par2B Qa a x Qa' Pa)
thus ?case
  apply -
  by(ind-cases bangPred P (Pa || Qa), auto intro: cPar2B aux1 simp add:
pi.inject)
next
case(Par2F Qa α Qa' Pa)
thus ?case
  apply -
  by(ind-cases bangPred P (Pa || Qa), auto intro: cPar2F aux1 simp add:
pi.inject)
next
case(Comm1 Pa a x Pa' Q b Q')
thus ?case
  apply -
  by(ind-cases bangPred P (Pa || Q), auto intro: cComm1 aux1 simp add:
pi.inject)
next
case(Comm2 Pa a b Pa' Q x Q')
thus ?case
  apply -
  by(ind-cases bangPred P (Pa || Q), auto intro: cComm2 aux1 simp add:
pi.inject)
next
case(Close1 Pa a x Pa' Q y Q')
thus ?case
  apply -
  by(ind-cases bangPred P (Pa || Q), auto intro: cClose1 aux1 simp add:
pi.inject)
next
case(Close2 Pa a y Pa' Q x Q')
thus ?case
  apply -
  by(ind-cases bangPred P (Pa || Q), auto intro: cClose2 aux1 simp add:

```

```

pi.inject)
  next
    case(ResB Pa a x P' y)
    thus ?case
      apply -
      by(ind-cases bangPred P (<νy>Pa))
  next
    case(ResF Pa α P' y)
    thus ?case
      apply -
      by(ind-cases bangPred P (<νy>Pa))
  next
    case(Bang Pa Rs)
    thus ?case
      apply -
      by(ind-cases bangPred P (!Pa), auto intro: cBang aux2 simp add: pi.inject)
  qed
qed

  with Trans show ?thesis by(force intro: bangPred.aux1)
qed

end

theory Late-Semantics1
  imports Late-Semantics
begin

free-constructors case-subject for
  InputS
| BoundOutputS
by(auto simp add: subject.inject)
  (metis Rep-subject-inverse subject.constr-rep(1,2) subject-Rep.exhaust)

free-constructors case-freeRes for
  OutputR
| TauR
by(auto simp add: freeRes.inject)
  (metis Abs-freeRes-cases Abs-freeRes-inverse freeRes.constr-rep(1,2) freeRes-Rep.exhaust)

end

theory Rel
  imports Agent
begin

definition eqvt :: (('a::pt-name) × ('a::pt-name)) set ⇒ bool
  where eqvt Rel ≡ (∀ x (perm::name prm). x ∈ Rel ⟶ perm · x ∈ Rel)

```

```

lemma eqvtRelI:
  fixes Rel :: ('a::pt-name × 'a) set
  and P :: 'a
  and Q :: 'a
  and perm :: name prm

  assumes eqvt Rel
  and (P, Q) ∈ Rel

  shows (perm · P, perm · Q) ∈ Rel
using assms
by(auto simp add: eqvt-def)

lemma eqvtRelE:
  fixes Rel :: ('a::pt-name × 'a) set
  and P :: 'a
  and Q :: 'a
  and perm :: name prm

  assumes eqvt Rel
  and (perm · P, perm · Q) ∈ Rel

  shows (P, Q) ∈ Rel
proof -
  have rev perm · (perm · P) = P and rev perm · (perm · Q) = Q
    by(simp add: pt-rev-pi[OF pt-name-inst, OF at-name-inst])+
  with assms show ?thesis
    by(force dest: eqvtRelI[of - - - rev perm])
qed

lemma eqvtTrans[intro]:
  fixes Rel :: ('a::pt-name × 'a) set
  and Rel' :: ('a × 'a) set

  assumes EqvtRel: eqvt Rel
  and EqvtRel': eqvt Rel'

  shows eqvt (Rel O Rel')
using assms
by(force simp add: eqvt-def)

lemma eqvtUnion[intro]:
  fixes Rel :: ('a::pt-name × 'a) set
  and Rel' :: ('a × 'a) set

  assumes EqvtRel: eqvt Rel
  and EqvtRel': eqvt Rel'

  shows eqvt (Rel ∪ Rel')

```

```

using assms
by(force simp add: eqvt-def)

definition substClosed :: (pi × pi) set ⇒ (pi × pi) set where
  substClosed Rel ≡ {(P, Q) | P Q. ∀σ. (P[<σ>], Q[<σ>]) ∈ Rel}

lemma eqvtSubstClosed:
  fixes Rel :: (pi × pi) set

  assumes eqvtRel: eqvt Rel

  shows eqvt (substClosed Rel)
proof(simp add: eqvt-def substClosed-def, auto)
  fix P Q perm s

  assume ∀s. (P[<s>], Q[<s>]) ∈ Rel
  hence (P[<(rev (perm::name prm) · s)>], Q[<(rev perm · s)>]) ∈ Rel by simp
  with eqvtRel have (perm · (P[<(rev perm · s)>]), perm · (Q[<(rev perm · s)>]))
  ∈ Rel
  by(rule eqvtRelI)
  thus ((perm · P)[<s>], (perm · Q)[<s>]) ∈ Rel
  by(simp add: name-per-rev)
qed

lemma substClosedSubset:
  fixes Rel :: (pi × pi) set

  shows substClosed Rel ⊆ Rel
proof(auto simp add: substClosed-def)
  fix P Q
  assume ∀s. (P[<s>], Q[<s>]) ∈ Rel
  hence (P[<[]>], Q[<[]>]) ∈ Rel by blast
  thus (P, Q) ∈ Rel by simp
qed

lemma partUnfold:
  fixes P :: pi
  and Q :: pi
  and σ :: (name × name) list
  and Rel :: (pi × pi) set

  assumes (P, Q) ∈ substClosed Rel

  shows (P[<σ>], Q[<σ>]) ∈ substClosed Rel
using assms
proof(auto simp add: substClosed-def)
  fix σ'
  assume ∀σ. (P[<σ>], Q[<σ>]) ∈ Rel
  hence (P[<(σ@σ')>], Q[<(σ@σ')>]) ∈ Rel by blast

```

```

thus (( $P[\langle\sigma\rangle]$ )[ $\langle\sigma'\rangle$ ], ( $Q[\langle\sigma\rangle]$ )[ $\langle\sigma'\rangle$ ])  $\in$  Rel
  by simp
qed

inductive-set bangRel :: ( $pi \times pi$ ) set  $\Rightarrow$  ( $pi \times pi$ ) set
for Rel :: ( $pi \times pi$ ) set
where
  BRBang: ( $P, Q \in Rel \Rightarrow (!P, !Q) \in bangRel Rel$ )
| BRPar: ( $R, T \in Rel \Rightarrow (P, Q) \in (bangRel Rel) \Rightarrow (R \parallel P, T \parallel Q) \in (bangRel Rel)$ )
| BRRes: ( $P, Q \in bangRel Rel \Rightarrow (\langle\nu a\rangle P, \langle\nu a\rangle Q) \in bangRel Rel$ )

inductive-cases BRBangCases': ( $P, !Q \in bangRel Rel$ )
inductive-cases BRParCases': ( $P, Q \parallel !Q \in bangRel Rel$ )
inductive-cases BRResCases': ( $P, \langle\nu x\rangle Q \in bangRel Rel$ )

lemma eqvtBangRel:
fixes Rel :: ( $pi \times pi$ ) set

  assumes eqvtRel: eqvt Rel

  shows eqvt(bangRel Rel)
proof(simp add: eqvt-def, auto)
fix  $P Q$  perm
assume ( $P, Q \in bangRel Rel$ )
thus ((perm::name prm)  $\cdot P$ , perm  $\cdot Q$ )  $\in$  bangRel Rel
proof(induct)
  fix  $P Q$ 
  assume ( $P, Q \in Rel$ )
  with eqvtRel have (perm  $\cdot P$ , perm  $\cdot Q$ )  $\in$  Rel
  by(rule eqvtRelI)
  thus (perm  $\cdot !P$ , perm  $\cdot !Q$ )  $\in$  bangRel Rel
  by(force intro: BRBang)
next
fix  $P Q R T$ 
assume  $R: (R, T) \in Rel$ 
assume  $BR: (perm \cdot P, perm \cdot Q) \in bangRel Rel$ 

  from eqvtRel R have (perm  $\cdot R$ , perm  $\cdot T$ )  $\in$  Rel
  by(rule eqvtRelI)

  with BR show (perm  $\cdot (R \parallel P)$ , perm  $\cdot (T \parallel Q)$ )  $\in$  bangRel Rel
  by(force intro: BRPar)
next
fix  $P Q a$ 
assume (perm  $\cdot P$ , perm  $\cdot Q$ )  $\in$  bangRel Rel
thus (perm  $\cdot \langle\nu a\rangle P$ , perm  $\cdot \langle\nu a\rangle Q$ )  $\in$  bangRel Rel
  by(force intro: BRRes)
qed

```

qed

lemma *BRBangCases*[consumes 1, case-names *BRBang*]:

fixes $P :: pi$
and $Q :: pi$
and $Rel :: (pi \times pi) \text{ set}$
and $F :: pi \Rightarrow bool$

assumes $(P, !Q) \in \text{bangRel } Rel$
and $\bigwedge P. (P, Q) \in Rel \Longrightarrow F (!P)$

shows $F P$
using *assms*
by(*induct rule: BRBangCases'*, *auto simp add: pi.inject*)

lemma *BRParCases*[consumes 1, case-names *BRPar*]:

fixes $P :: pi$
and $Q :: pi$
and $Rel :: (pi \times pi) \text{ set}$
and $F :: pi \Rightarrow bool$

assumes $(P, Q \parallel !Q) \in \text{bangRel } Rel$
and $\bigwedge P R. [(P, Q) \in Rel; (R, !Q) \in \text{bangRel } Rel] \Longrightarrow F (P \parallel R)$

shows $F P$
using *assms*
by(*induct rule: BRParCases'*, *auto simp add: pi.inject*)

lemma *bangRelSubset*:

fixes $Rel :: (pi \times pi) \text{ set}$
and $Rel' :: (pi \times pi) \text{ set}$

assumes $(P, Q) \in \text{bangRel } Rel$
and $\bigwedge P Q. (P, Q) \in Rel \Longrightarrow (P, Q) \in Rel'$

shows $(P, Q) \in \text{bangRel } Rel'$
using *assms*
by(*induct rule: bangRel.induct*) (*auto intro: BRBang BRPar BRRes*)

lemma *bangRelSymetric*:

fixes $P :: pi$
and $Q :: pi$
and $Rel :: (pi \times pi) \text{ set}$

assumes $A: (P, Q) \in \text{bangRel } Rel$
and $Sym: \bigwedge P Q. (P, Q) \in Rel \Longrightarrow (Q, P) \in Rel$

shows $(Q, P) \in \text{bangRel } Rel$
proof –

```

from A show ?thesis
proof(induct)
  fix P Q
  assume  $(P, Q) \in \text{Rel}$ 
  hence  $(Q, P) \in \text{Rel}$  by(rule Sym)
  thus  $(!Q, !P) \in \text{bangRel Rel}$  by(rule BRBang)
next
  fix P Q R T
  assume RRelT:  $(R, T) \in \text{Rel}$ 
  assume IH:  $(Q, P) \in \text{bangRel Rel}$ 
  from RRelT have  $(T, R) \in \text{Rel}$  by(rule Sym)
  thus  $(T \parallel Q, R \parallel P) \in \text{bangRel Rel}$  using IH by(rule BRPar)
next
  fix P Q a
  assume  $(Q, P) \in \text{bangRel Rel}$ 
  thus  $(\langle \nu a \rangle Q, \langle \nu a \rangle P) \in \text{bangRel Rel}$  by(rule BRRes)
qed
qed

primrec resChain :: name list  $\Rightarrow$  pi  $\Rightarrow$  pi where
  base: resChain [] P = P
  | step: resChain (x#xs) P =  $\langle \nu x \rangle(\text{resChain } xs \ iP)$ 

lemma resChainPerm[simp]:
  fixes perm :: name prm
  and lst :: name list
  and P :: pi

  shows  $\text{perm} \cdot (\text{resChain } lst \ iP) = \text{resChain } (\text{perm} \cdot lst) (\text{perm} \cdot P)$ 
by(induct-tac lst, auto)

lemma resChainFresh:
  fixes a :: name
  and lst :: name list
  and P :: pi

  assumes  $a \# (lst, P)$ 

  shows  $a \# (\text{resChain } lst \ iP)$ 
using assms apply(induct-tac lst)
apply(simp add: fresh-prod)
by(simp add: fresh-prod name-fresh-abs)

end

theory Strong-Late-Sim
  imports Late-Semantics1 Rel
begin

```

definition *derivative* :: *pi* ⇒ *pi* ⇒ *subject* ⇒ *name* ⇒ (*pi* × *pi*) *set* ⇒ *bool* **where**
derivative *P Q a x Rel* ≡ *case a of InputS b* ⇒ (∀ *u*. (*P[x::=u]*, *Q[x::=u]*) ∈ *Rel*)
| *BoundOutputS b* ⇒ (*P*, *Q*) ∈ *Rel*

definition *simulation* :: *pi* ⇒ (*pi* × *pi*) *set* ⇒ *pi* ⇒ *bool* (⟨- ∼[-] -⟩ [80, 80, 80] 80) **where**

P ∼[*Rel*] *Q* ≡ (∀ *a x Q'*. *Q* ⟶*a*⟨⟨*x*⟩⟩ < *Q'* ∧ *x* ‡ *P* ⟶ (∃ *P'*. *P* ⟶*a*⟨⟨*x*⟩⟩ < *P'*
∧ *derivative P' Q' a x Rel*)) ∧
(∀ *α Q'*. *Q* ⟶*α* < *Q'* ⟶ (∃ *P'*. *P* ⟶*α* < *P'* ∧ (*P'*, *Q'*) ∈ *Rel*))

lemma *monotonic*:

fixes *A* :: (*pi* × *pi*) *set*
and *B* :: (*pi* × *pi*) *set*
and *P* :: *pi*
and *P'* :: *pi*

assumes *P* ∼[*A*] *P'*
and *A* ⊆ *B*

shows *P* ∼[*B*] *P'*

using *assms*

apply(*auto simp add: simulation-def derivative-def*)

by(*case-tac a fastforce+*)

lemma *derivativeMonotonic*:

fixes *A* :: (*pi* × *pi*) *set*
and *B* :: (*pi* × *pi*) *set*
and *P* :: *pi*
and *Q* :: *pi*
and *a* :: *subject*
and *x* :: *name*

assumes *derivative P Q a x A*
and *A* ⊆ *B*

shows *derivative P Q a x B*

using *assms*

by(*case-tac a, auto simp add: derivative-def*)

lemma *derivativeEqvtI*:

fixes *P* :: *pi*
and *Q* :: *pi*
and *a* :: *subject*
and *x* :: *name*
and *Rel* :: (*pi* × *pi*) *set*
and *perm* :: *name prm*

assumes *Der: derivative P Q a x Rel*
and *Eqvt: eqvt Rel*

```

shows derivative (perm · P) (perm · Q) (perm · a) (perm · x) Rel
using assms
apply(case-tac a, auto simp add: derivative-def)
apply(erule-tac x=rev perm · u in allE)
apply(drule-tac perm=perm in eqtRelI)
apply(blast)
apply(force simp add: eqt-subs name-per-rev)
by(force simp add: eqt-def)

```

```

lemma derivativeEqvtI2:
  fixes P :: pi
  and Q :: pi
  and a :: subject
  and x :: name
  and Rel :: (pi × pi) set
  and perm :: name prm

```

```

assumes Der: derivative P Q a x Rel
and Eqvt: eqt Rel

```

```

shows derivative (perm · P) (perm · Q) a (perm · x) Rel
using assms
apply(case-tac a, auto simp add: derivative-def)
apply(erule-tac x=rev perm · u in allE)
apply(drule-tac perm=perm in eqtRelI)
apply(blast)
apply(force simp add: eqt-subs name-per-rev)
by(force simp add: eqt-def)

```

```

lemma freshUnit[simp]:
  fixes y :: name

```

```

shows y # ()
by(auto simp add: fresh-def supp-unit)

```

```

lemma simCasesCont[consumes 1, case-names Bound Free]:

```

```

  fixes P :: pi
  and Q :: pi
  and Rel :: (pi × pi) set
  and C :: 'a::fs-name

```

```

assumes Eqvt: eqt Rel
and Bound:  $\bigwedge a x Q'. \llbracket Q \mapsto a \langle x \rangle \prec Q'; x \# P; x \# Q; x \# a; x \# C \rrbracket \implies$ 
 $\exists P'. P \mapsto a \langle x \rangle \prec P' \wedge \text{derivative } P' Q' a x \text{ Rel}$ 
and Free:  $\bigwedge \alpha Q'. Q \mapsto \alpha \prec Q' \implies \exists P'. P \mapsto \alpha \prec P' \wedge (P', Q') \in \text{Rel}$ 

```

```

shows P  $\rightsquigarrow$ [Rel] Q
using Free

```

```

proof(auto simp add: simulation-def)
  fix  $a\ x\ Q'$ 
  assume  $(x::name) \# P$ 
  assume  $Trans: Q \mapsto a\langle\langle x \rangle\rangle \prec Q'$ 

  obtain  $y::name$  where  $y \# P$  and  $y \# Q$  and  $y \# a$  and  $y \# C$  and  $y \# Q'$  and
 $y \neq x$ 
    by(generate-fresh name) auto

  from  $Trans \langle y \# Q' \rangle$  have  $Q \mapsto a\langle\langle y \rangle\rangle \prec [(x, y)] \cdot Q'$  by(simp add: alphaBound-Residual)
  hence  $\exists P'. P \mapsto a\langle\langle y \rangle\rangle \prec P' \wedge derivative\ P' \ ([ (x, y) ] \cdot Q')\ a\ y\ Rel$ 
    using  $\langle y \# P \rangle \langle y \# Q \rangle \langle y \# a \rangle \langle y \# C \rangle$ 
    by(rule Bound)
  then obtain  $P'$  where  $PTrans: P \mapsto a\langle\langle y \rangle\rangle \prec P'$  and  $PDer: derivative\ P' \ ([ (x, y) ] \cdot Q')\ a\ y\ Rel$ 
    by blast

  from  $PTrans \langle x \# P \rangle \langle y \neq x \rangle$  have  $x \# P'$  by(force intro: freshBoundDerivative)
  with  $PTrans$  have  $P \mapsto a\langle\langle x \rangle\rangle \prec [(x, y)] \cdot P'$  by(simp add: alphaBoundResidual name-swap)
  moreover have  $derivative \ ([ (x, y) ] \cdot P')\ Q'\ a\ x\ Rel$ 
  proof –
    from  $PDer\ Eqvt$  have  $derivative \ ([ (x, y) ] \cdot P') \ ([ (x, y) ] \cdot [(x, y)] \cdot Q')\ a \ ([ (x, y) ] \cdot y)\ Rel$ 
      by(rule derivativeEqvtI2)
    with  $\langle y \neq x \rangle$  show ?thesis by(simp add: name-calc)
  qed
  ultimately show  $\exists P'. P \mapsto a\langle\langle x \rangle\rangle \prec P' \wedge derivative\ P'\ Q'\ a\ x\ Rel$  by blast
qed

lemma simCases[case-names Bound Free]:
  fixes  $P \ ::\ pi$ 
  and  $Q \ ::\ pi$ 
  and  $Rel \ ::\ (pi \times pi)\ set$ 

  assumes  $Bound: \bigwedge a\ y\ Q'. \llbracket Q \mapsto a\langle\langle y \rangle\rangle \prec Q'; y \# P \rrbracket \implies \exists P'. P \mapsto a\langle\langle y \rangle\rangle \prec P' \wedge derivative\ P'\ Q'\ a\ y\ Rel$ 
  and  $Free: \bigwedge \alpha\ Q'. Q \mapsto \alpha \prec Q' \implies \exists P'. P \mapsto \alpha \prec P' \wedge (P', Q') \in Rel$ 

  shows  $P \rightsquigarrow_{[Rel]} Q$ 
using assms
by(auto simp add: simulation-def)

lemma resSimCases[consumes 1, case-names BoundOutput BoundR FreeR]:
  fixes  $x \ ::\ name$ 
  and  $P \ ::\ pi$ 
  and  $Rel \ ::\ (pi \times pi)\ set$ 
  and  $Q \ ::\ pi$ 

```

and $C :: 'a::fs\text{-name}$

assumes $Eqvt$: $eqvt\ Rel$

and $BoundO$: $\bigwedge Q' a. \llbracket Q \mapsto a[x] \prec Q'; a \neq x \rrbracket \implies \exists P'. P \mapsto a \langle \nu x \rangle \prec P' \wedge (P', Q') \in Rel$

and BR : $\bigwedge Q' a y. \llbracket Q \mapsto a \langle y \rangle \prec Q'; x \# a; x \neq y; y \# C \rrbracket \implies \exists P'. P \mapsto a \langle y \rangle \prec P' \wedge derivative\ P' (\langle \nu x \rangle Q')\ a\ y\ Rel$

and BF : $\bigwedge Q' \alpha. \llbracket Q \mapsto \alpha \prec Q'; x \# \alpha \rrbracket \implies \exists P'. P \mapsto \alpha \prec P' \wedge (P', \langle \nu x \rangle Q') \in Rel$

shows $P \rightsquigarrow [Rel] \langle \nu x \rangle Q$

using $Eqvt$

proof(*induct rule: simCasesCont*[**where** $C=(C, x, Q)$])

case($Bound\ a\ y\ Q'$)

have $y \# (C, x, Q)$ **by** *fact*

hence $yFreshC$: $y \# C$ **and** $yineqx$: $y \neq x$ **and** $y \# Q$

by(*simp add: fresh-prod*)**+**

have $\langle \nu x \rangle Q \mapsto a \langle y \rangle \prec Q'$ **by** *fact*

thus $?case$ **using** $yineqx \langle y \# Q \rangle$

proof(*induct rule: resCasesB*)

case($cOpen\ a'\ Q'$)

have $Q \mapsto a'[x] \prec Q'$ **and** $a' \neq x$ **by** *fact***+**

then obtain P' **where** $PTrans$: $P \mapsto a' \langle \nu x \rangle \prec P'$ **and** $P'RelQ'$: $(P', Q') \in Rel$ **by**(*force dest: BoundO*)

from $PTrans \langle y \# P \rangle$ $yineqx$ **have** $y \# P'$ **by**(*force dest: freshBoundDerivative*)

with $PTrans$ **have** $P \mapsto a' \langle \nu y \rangle \prec ([(x, y)] \cdot P')$ **by**(*simp add: alphaBoundResidual*)

moreover from $P'RelQ' Eqvt$ **have** $([(x, y)] \cdot P', [(x, y)] \cdot Q') \in Rel$ **by**(*auto simp add: eqvt-def*)

ultimately show $?case$ **by**(*force simp add: derivative-def name-swap*)

next

case($cRes\ Q'$)

have $Q \mapsto a \langle y \rangle \prec Q'$ **and** $x \# a$ **by** *fact***+**

with $yineqx\ yFreshC$ **show** $?case$ **by**(*force dest: BR*)

qed

next

case($Free\ \alpha\ Q'$)

have $\langle \nu x \rangle Q \mapsto \alpha \prec Q'$ **by** *fact*

thus $?case$

proof(*induct rule: resCasesF*)

case($cRes\ Q'$)

have $Q \mapsto \alpha \prec Q'$ **and** $x \# \alpha$ **by** *fact***+**

thus $?case$ **by**(*rule BF*)

qed

qed

lemma $simE$:

fixes $P :: pi$

```

and Rel :: (pi × pi) set
and Q  :: pi
and a  :: subject
and x  :: name
and Q' :: pi

assumes P ~>[Rel] Q

shows Q ⟶ a«x» < Q' ⟹ x # P ⟹ ∃ P'. P ⟶ a«x» < P' ∧ (derivative P'
Q' a x Rel)
and Q ⟶ α < Q' ⟹ ∃ P'. P ⟶ α < P' ∧ (P', Q') ∈ Rel
using assms by(simp add: simulation-def)+

lemma eqvtI:
fixes P  :: pi
and Q  :: pi
and Rel :: (pi × pi) set
and perm :: name prm

assumes Sim: P ~>[Rel] Q
and RelRel': Rel ⊆ Rel'
and EqvtRel': eqvt Rel'

shows (perm · P) ~>[Rel'] (perm · Q)
proof(induct rule: simCases)
case(Bound a y Q')
have QTrans: (perm · Q) ⟶ a«y» < Q' and yFreshP: y # perm · P by fact+

from QTrans have (rev perm · (perm · Q)) ⟶ rev perm · (a«y» < Q')
by(rule transitions.eqvt)
hence Q ⟶ (rev perm · a)«(rev perm · y)» < (rev perm · Q')
by(simp add: name-rev-per)
moreover from yFreshP have (rev perm · y) # P by(simp add: name-fresh-left)
ultimately have ∃ P'. P ⟶ (rev perm · a)«rev perm · y» < P' ∧ derivative
P' (rev perm · Q') (rev perm · a) (rev perm · y) Rel using Sim
by(force intro: simE)
then obtain P' where PTrans: P ⟶ (rev perm · a)«rev perm · y» < P' and
Pderivative: derivative P' (rev perm · Q') (rev perm · a) (rev perm · y) Rel by
blast

from PTrans have (perm · P) ⟶ perm · ((rev perm · a)«rev perm · y» < P')
by(rule transitions.eqvt)
hence L1: (perm · P) ⟶ a«y» < (perm · P') by(simp add: name-per-rev)
from Pderivative RelRel' have derivative P' (rev perm · Q') (rev perm · a) (rev
perm · y) Rel'
by(rule derivativeMonotonic)
hence derivative (perm · P') (perm · (rev perm · Q')) (perm · (rev perm · a))
(perm · (rev perm · y)) Rel' using EqvtRel'
by(rule derivativeEqvtI)

```

```

hence derivative ( $\text{perm} \cdot P'$ )  $Q'$   $a$   $y$   $Rel'$  by(simp add: name-per-rev)
with  $L1$  show ?case by blast
next
case(Free  $\alpha$   $Q'$ )
have ( $\text{perm} \cdot Q$ )  $\mapsto$   $\alpha \prec Q'$  by fact

hence ( $\text{rev perm} \cdot (\text{perm} \cdot Q)$ )  $\mapsto$   $\text{rev perm} \cdot (\alpha \prec Q')$ 
by(rule transitions.eqvt)
hence  $Q \mapsto (\text{rev perm} \cdot \alpha) \prec (\text{rev perm} \cdot Q')$ 
by(simp add: name-rev-per)
with Sim have  $\exists P'. P \mapsto (\text{rev perm} \cdot \alpha) \prec P' \wedge (P', (\text{rev perm} \cdot Q')) \in Rel$ 
by(force intro: simE)
then obtain  $P'$  where  $PTrans: P \mapsto (\text{rev perm} \cdot \alpha) \prec P'$  and  $PRel: (P', (\text{rev perm} \cdot Q')) \in Rel$ 
by blast

from  $PTrans$  have ( $\text{perm} \cdot P$ )  $\mapsto$   $\text{perm} \cdot ((\text{rev perm} \cdot \alpha) \prec P')$  by(rule transitions.eqvt)
hence  $L1: (\text{perm} \cdot P) \mapsto \alpha \prec (\text{perm} \cdot P')$  by(simp add: name-per-rev)
from  $PRel$   $EqvtRel'$   $RelRel'$  have  $((\text{perm} \cdot P'), (\text{perm} \cdot (\text{rev perm} \cdot Q')) \in Rel'$ 
by(force intro: eqvtRelI)
hence  $((\text{perm} \cdot P'), Q') \in Rel'$  by(simp add: name-per-rev)
with  $L1$  show  $\exists P'. (\text{perm} \cdot P) \mapsto \alpha \prec P' \wedge (P', Q') \in Rel'$  by blast
qed

```

lemma *derivativeReflexive:*

```

fixes  $P$   $:: pi$ 
and  $a$   $:: subject$ 
and  $x$   $:: name$ 
and  $Rel$   $:: (pi \times pi)$  set

```

assumes $Id \subseteq Rel$

shows *derivative* P P a x Rel

using *assms*

apply(*cases a*)

by(*auto simp add: derivative-def*)

lemma *reflexive:*

```

fixes  $P$   $:: pi$ 
and  $Rel$   $:: (pi \times pi)$  set

```

assumes $Id \subseteq Rel$

shows $P \rightsquigarrow[Rel] P$

using *assms*

by(*auto simp add: simulation-def derivativeReflexive*)

```

lemma transitive:
  fixes  $P$     ::  $pi$ 
  and    $Q$      ::  $pi$ 
  and    $R$      ::  $pi$ 
  and    $Rel$   ::  $(pi \times pi)$  set
  and    $Rel'$  ::  $(pi \times pi)$  set
  and    $Rel''$  ::  $(pi \times pi)$  set

  assumes  $PSimQ$ :  $P \rightsquigarrow[Rel] Q$ 
  and      $QSimR$ :  $Q \rightsquigarrow[Rel'] R$ 
  and      $Eqvt'$ : eqvt  $Rel''$ 
  and      $Trans$ :  $Rel \circ Rel' \subseteq Rel''$ 

  shows  $P \rightsquigarrow[Rel''] R$ 
using  $Eqvt'$ 
proof(induct rule: simCasesCont[where  $C=Q$ ])
  case(Bound  $a\ x\ R'$ )
  have  $RTrans$ :  $R \mapsto a\langle x \rangle \prec R'$  by fact

  from  $\langle x \# Q \rangle\ QSimR\ RTrans$  obtain  $Q'$  where  $QTrans$ :  $Q \mapsto a\langle x \rangle \prec Q'$ 
  and  $QDer$ : derivative  $Q'\ R'\ a\ x\ Rel'$ 

  by(blast dest: simE)
  with  $QTrans\ \langle x \# P \rangle\ PSimQ$  obtain  $P'$  where  $PTrans$ :  $P \mapsto a\langle x \rangle \prec P'$ 
  and  $PDer$ : derivative  $P'\ Q'\ a\ x\ Rel$ 

  by(blast dest: simE)
  moreover from  $PDer\ QDer\ Trans$  have derivative  $P'\ R'\ a\ x\ Rel''$ 
  by(cases a) (auto simp add: derivative-def)
  ultimately show ?case by blast
next
  case(Free  $\alpha\ R'$ )
  have  $RTrans$ :  $R \mapsto \alpha \prec R'$  by fact
  with  $QSimR$  obtain  $Q'$  where  $QTrans$ :  $Q \mapsto \alpha \prec Q'$ 
  and  $Q'RelR'$ :  $(Q', R') \in Rel'$ 

  by(blast dest: simE)
  from  $QTrans\ PSimQ$  obtain  $P'$  where  $PTrans$ :  $P \mapsto \alpha \prec P'$ 
  and  $P'RelQ'$ :  $(P', Q') \in Rel$ 

  by(blast dest: simE)
  from  $P'RelQ'\ Q'RelR'\ Trans$  have  $(P', R') \in Rel''$  by blast
  with  $PTrans$  show ?case by blast
qed

end

theory Strong-Late-Bisim
  imports Strong-Late-Sim
begin

lemma monoAux:  $A \subseteq B \implies P \rightsquigarrow[A] Q \longrightarrow P \rightsquigarrow[B] Q$ 

```

by(*auto intro: Strong-Late-Sim.monotonic*)

coinductive-set *bisim* :: (*pi* × *pi*) *set*

where

step: $\llbracket P \rightsquigarrow[bisim] Q; (Q, P) \in bisim \rrbracket \implies (P, Q) \in bisim$

monos *monoAux*

abbreviation

strongBisimJudge (**infixr** $\langle \sim \rangle$ 65) **where** $P \sim Q \equiv (P, Q) \in bisim$

lemma *monotonic'*: *mono*($\lambda S. \{(P, Q) \mid P \rightsquigarrow[S] Q \wedge Q \rightsquigarrow[S] P\}$)

apply(*rule monoI*)

by(*auto dest: monoAux*)

lemma *monotonic*: *mono*($\lambda p \ x1 \ x2.$

$\exists P \ Q. \ x1 = P \wedge$

$\ x2 = Q \wedge P \rightsquigarrow[\{(xa, x). p \ xa \ x\}] Q \wedge Q \rightsquigarrow[\{(xa, x). p \ xa \ x\}] P$)

apply(*rule monoI*)

by(*auto intro: Strong-Late-Sim.monotonic*)

lemma *bisimCoinduct*[*case-names cSim cSym* , *consumes 1*]:

assumes *p*: $(P, Q) \in X$

and *rSim*: $\bigwedge R \ S. (R, S) \in X \implies R \rightsquigarrow[(X \cup bisim)] S$

and *rSym*: $\bigwedge R \ S. (R, S) \in X \implies (S, R) \in X$

shows $P \sim Q$

proof –

have *aux*: $X \cup bisim = \{(P, Q). (P, Q) \in X \vee P \sim Q\}$ **by** *blast*

from *p* **show** *?thesis*

apply(*coinduct, auto*)

apply(*fastforce dest: rSim simp add: aux*)

by(*fastforce dest: rSym*)

qed

lemma *bisimE*:

fixes *P* :: *pi*

and *Q* :: *pi*

assumes $P \sim Q$

shows $P \rightsquigarrow[bisim] Q$

using *assms*

by(*auto intro: bisim.cases*)

lemma *bisimI*:

fixes *P* :: *pi*

and *Q* :: *pi*

```

assumes  $P \rightsquigarrow[bisim] Q$ 
and  $Q \sim P$ 

shows  $P \sim Q$ 
using assms
by(rule bisim.intros)

definition old-bisim ::  $(pi \times pi) \text{ set} \Rightarrow \text{bool}$  where
  old-bisim Rel  $\equiv \forall (P, Q) \in \text{Rel}. P \rightsquigarrow[\text{Rel}] Q \wedge (Q, P) \in \text{Rel}$ 

lemma oldBisimBisimEq:
  shows  $(\bigcup \{ \text{Rel}. (\text{old-bisim } \text{Rel}) \}) = \text{bisim}$  (is ?LHS = ?RHS)
proof
  show ?LHS  $\subseteq$  ?RHS
  proof auto
    fix  $P Q \text{ Rel}$ 
    assume  $(P, Q) \in \text{Rel}$  and old-bisim Rel
    thus  $P \sim Q$ 
    proof(coinduct rule: bisimCoinduct)
      case(cSim  $P Q$ )
      with  $\langle \text{old-bisim } \text{Rel} \rangle$  show ?case
      by(fastforce simp add: old-bisim-def intro: Strong-Late-Sim.monotonic)
    next
      case(cSym  $P Q$ )
      with  $\langle \text{old-bisim } \text{Rel} \rangle$  show ?case
      by(auto simp add: old-bisim-def)
    qed
  qed
next
  show ?RHS  $\subseteq$  ?LHS
  proof auto
    fix  $P Q$ 
    assume  $P \sim Q$ 
    moreover hence old-bisim bisim
    by(auto simp add: old-bisim-def dest: bisim.cases)
    ultimately show  $\exists \text{Rel}. \text{old-bisim } \text{Rel} \wedge (P, Q) \in \text{Rel}$ 
    by blast
  qed
qed

lemma reflexive:
  fixes  $P :: pi$ 

  shows  $P \sim P$ 
proof –
  have  $(P, P) \in \text{Id}$  by simp
  thus ?thesis
  by(coinduct rule: bisimCoinduct, auto intro: Strong-Late-Sim.reflexive)

```

qed

lemma *symmetric*:

fixes $P :: pi$

and $Q :: pi$

assumes $P \sim Q$

shows $Q \sim P$

using *assms*

by(*auto dest: bisim.cases*)

lemma *bisimClosed*:

fixes $P :: pi$

and $Q :: pi$

and $p :: name prm$

assumes $P \sim Q$

shows $(p \cdot P) \sim (p \cdot Q)$

proof –

let $?X = \{(p \cdot P, p \cdot Q) \mid P \sim Q \text{ (} p::name \text{ } prm). P \sim Q\}$

from $\langle P \sim Q \rangle$ have $(p \cdot P, p \cdot Q) \in ?X$ by *blast*

thus *?thesis*

proof(*coinduct rule: bisimCoinduct*)

case(*cSim pP pQ*)

from $\langle (pP, pQ) \in ?X \rangle$ obtain $P \sim Q$ *p* where $P \sim Q$ and $pP = (p::name \text{ } prm)$

• P and $pQ = p \cdot Q$

by *auto*

from $\langle P \sim Q \rangle$ have $P \rightsquigarrow[bisim] Q$ by(*rule bisimE*)

moreover have $bisim \subseteq ?X$

proof

fix x

assume $x \in bisim$

moreover have $x = (([]::name \text{ } prm) \cdot x)$ by *auto*

ultimately show $x \in ?X$

apply(*case-tac x*)

by(*clarify, simp only: eqvts*) *metis*

qed

moreover have *eqvt* $?X$

proof(*auto simp add: eqvt-def*)

fix $P \sim Q$

fix $perm1::name \text{ } prm$

fix $perm2::name \text{ } prm$

assume $P \sim Q$

moreover have $perm1 \cdot perm2 \cdot P = (perm1 @ perm2) \cdot P$ by(*simp add:*

pt2[OF pt-name-inst])

moreover have $perm1 \cdot perm2 \cdot Q = (perm1 @ perm2) \cdot Q$ by(*simp add:*

pt2[*OF pt-name-inst*]

ultimately show $\exists P' Q'. (\exists (\text{perm}::\text{name prm}). \text{perm1} \cdot \text{perm2} \cdot P = \text{perm} \cdot P' \wedge \text{perm1} \cdot \text{perm2} \cdot Q = \text{perm} \cdot Q') \wedge P' \sim Q'$

by *blast*
qed
ultimately have $(p \cdot P) \rightsquigarrow[?X] (p \cdot Q)$
by(*rule Strong-Late-Sim.eqtI*)
with $\langle pP = p \cdot P \rangle \langle pQ = p \cdot Q \rangle$ **show** *?case*
by(*force intro: Strong-Late-Sim.monotonic*)
next
case(*cSym P Q*)
thus *?case* **by**(*auto intro: symmetric*)
qed
qed

lemma *bisimEqt[simp]*:
shows *eqt bisim*
by(*auto simp add: eqt-def bisimClosed*)

lemma *transitive*:
fixes $P :: pi$
and $Q :: pi$
and $R :: pi$

assumes $P \sim Q$
and $Q \sim R$

shows $P \sim R$

proof –
let $?X = \text{bisim } O \text{ bisim}$
from *assms* **have** $(P, R) \in ?X$ **by** *blast*
thus *?thesis*
proof(*coinduct rule: bisimCoinduct*)
case(*cSim P R*)
thus *?case*
by(*fastforce intro: Strong-Late-Sim.transitive dest: bisimE simp add: eqtTrans*)
next
case(*cSym P R*)
thus *?case*
by(*auto dest: symmetric*)
qed
qed

lemma *bisimTransitiveCoinduct*[*case-names cSim cSym, case-conclusion bisim step, consumes 2*]:
assumes $(P, Q) \in X$
and *eqt X*

```

and rSim:  $\bigwedge R S. (R, S) \in X \implies R \rightsquigarrow [(bisim\ O\ (X \cup bisim)\ O\ bisim)] S$ 
and rSym:  $\bigwedge R S. (R, S) \in X \implies (S, R) \in bisim\ O\ (X \cup bisim)\ O\ bisim$ 

shows  $P \sim Q$ 
proof –
let  $?X = bisim\ O\ (X \cup bisim)\ O\ bisim$ 
from  $\langle (P, Q) \in X \rangle$  have  $(P, Q) \in ?X$  by (auto intro: reflexive)
thus ?thesis
proof (coinduct rule: bisimCoinduct)
  case (cSim P Q)
  {
    fix  $P P' Q' Q$ 

    assume  $P \sim P'$  and  $(P', Q') \in X \cup bisim$  and  $Q' \sim Q$ 
    have  $P \rightsquigarrow [(?X \cup bisim)] Q$ 
    proof (cases (P', Q') \in X)
      case True
      from  $\langle P \sim P' \rangle$  have  $P \rightsquigarrow [bisim] P'$  by (rule bisimE)
      moreover from  $\langle (P', Q') \in X \rangle$  have  $P' \rightsquigarrow [(?X)] Q'$  by (rule rSim)
      moreover from  $\langle eqvt\ X \rangle$  bisimEqvt have  $eqvt(?X \cup bisim)$  by blast
      moreover have  $bisim\ O\ ?X \subseteq ?X \cup bisim$  by (auto dest: transitive)
      ultimately have  $P \rightsquigarrow [(?X \cup bisim)] Q'$  by (rule Strong-Late-Sim.transitive)
      moreover from  $\langle Q' \sim Q \rangle$  have  $Q' \rightsquigarrow [bisim] Q$  by (rule bisimE)
      moreover note  $\langle eqvt(?X \cup bisim) \rangle$ 
      moreover have  $(?X \cup bisim)\ O\ bisim \subseteq ?X \cup bisim$ 
      by auto (blast dest: transitive)+
      ultimately show ?thesis by (rule Strong-Late-Sim.transitive)
    next
      case False
      from  $\langle (P', Q') \notin X \rangle$   $\langle (P', Q') \in X \cup bisim \rangle$  have  $P' \sim Q'$  by simp
      with  $\langle P \sim P' \rangle$   $\langle Q' \sim Q \rangle$  have  $P \sim Q$  by (blast dest: transitive)
      hence  $P \rightsquigarrow [bisim] Q$  by (rule bisimE)
      moreover have  $bisim \subseteq ?X \cup bisim$  by auto
      ultimately show ?thesis by (rule Strong-Late-Sim.monotonic)
    qed
  }
with  $\langle (P, Q) \in ?X \rangle$  show ?case by auto
case (cSym P Q)
  {
    fix  $P P' Q' Q$ 

    assume  $P \sim P'$  and  $(P', Q') \in X \cup bisim$  and  $Q' \sim Q$ 
    have  $(Q, P) \in bisim\ O\ (X \cup bisim)\ O\ bisim$ 
    proof (cases (P', Q') \in X)
      case True
      from  $\langle (P', Q') \in X \rangle$  have  $(Q', P') \in ?X$  by (rule rSym)
      then obtain  $Q'' P''$  where  $Q' \sim Q''$  and  $(Q'', P'') \in X \cup bisim$  and  $P''$ 
       $\sim P'$ 
      by auto

```

```

    from  $\langle Q' \sim Q \rangle \langle Q' \sim Q'' \rangle$  have  $Q \sim Q''$  by(metis transitive symmetric)
    moreover from  $\langle P \sim P' \rangle \langle P'' \sim P' \rangle$  have  $P'' \sim P$  by(metis transitive symmetric)
    ultimately show  $?thesis$  using  $\langle (Q'', P'') \in X \cup \text{bisim} \rangle$  by blast
  next
  case False
  from  $\langle (P', Q') \notin X \rangle \langle (P', Q') \in X \cup \text{bisim} \rangle$  have  $P' \sim Q'$  by simp
  with  $\langle P \sim P' \rangle \langle Q' \sim Q \rangle$  have  $Q \sim P$  by(metis transitive symmetric)
  thus  $?thesis$  by(blast intro: reflexive)
qed
}
with  $\langle (P, Q) \in ?X \rangle$  show  $?case$  by blast
qed
qed

end

```

```

theory Strong-Late-Bisim-Subst
  imports Strong-Late-Bisim
begin

```

abbreviation

StrongEqJudge (**infixr** $\langle \sim^s \rangle$ 65) **where** $P \sim^s Q \equiv (P, Q) \in (\text{substClosed bisim})$

lemma *congBisim*:

fixes $P :: pi$

and $Q :: pi$

assumes $P \sim^s Q$

shows $P \sim Q$

using *assms substClosedSubset* by *blast*

lemma *eqvt*:

shows *eqvt* (*substClosed bisim*)

by(*rule eqvtSubstClosed[OF Strong-Late-Bisim.bisimEqvt]*)

lemma *eqClosed*:

fixes $P :: pi$

and $Q :: pi$

and $perm :: \text{name prm}$

assumes $P \sim^s Q$

shows $(perm \cdot P) \sim^s (perm \cdot Q)$

using *assms*

by(*rule eqvtRelI[OF eqvt]*)

```

lemma reflexive:
  fixes  $P :: pi$ 

  shows  $P \sim^s P$ 
by(force simp add: substClosed-def intro: Strong-Late-Bisim.reflexive)

lemma symmetric:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 

  assumes  $P \sim^s Q$ 

  shows  $Q \sim^s P$ 
using assms
by(force simp add: substClosed-def intro: Strong-Late-Bisim.symmetric)

lemma transitive:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $R :: pi$ 

  assumes  $P \sim^s Q$ 
  and  $Q \sim^s R$ 

  shows  $P \sim^s R$ 
using assms
by(force simp add: substClosed-def intro: Strong-Late-Bisim.transitive)

end

theory Strong-Late-Sim-Pres
  imports Strong-Late-Sim
begin

lemma tauPres:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $Rel :: (pi \times pi) set$ 
  and  $Rel' :: (pi \times pi) set$ 

  assumes  $PRelQ: (P, Q) \in Rel$ 

  shows  $\tau.(P) \rightsquigarrow[Rel] \tau.(Q)$ 
proof –
  show  $\tau.(P) \rightsquigarrow[Rel] \tau.(Q)$ 
proof(induct rule: simCases)
  case(Bound a x Q')
  have  $\tau.(Q) \mapsto a \ll x \gg \prec Q'$  by fact
  hence False by auto

```

```

    thus ?case by simp
next
case(Free  $\alpha$   $Q'$ )
have  $\tau.(Q) \mapsto \alpha \prec Q'$  by fact
thus ?case
proof(induct rule: tauCases)
  case cTau
  have  $\tau.(P) \mapsto \tau \prec P$  by(rule Late-Semantics.Tau)
  with PRelQ show ?case by blast
qed
qed
qed

lemma inputPres:
  fixes  $P$     ::  $pi$ 
  and  $x$       ::  $name$ 
  and  $Q$      ::  $pi$ 
  and  $a$      ::  $name$ 
  and  $Rel$   ::  $(pi \times pi)$  set

  assumes PRelQ:  $\forall y. (P[x::=y], Q[x::=y]) \in Rel$ 
  and      Eqvt: eqvt Rel

  shows  $a \langle x \rangle . P \rightsquigarrow [Rel] a \langle x \rangle . Q$ 
using Eqvt
proof(induct rule: simCasesCont[where  $C=(x, a, P, Q)$ ])
  case(Bound  $b$   $y$   $Q'$ )
  from  $\langle y \# (x, a, P, Q) \rangle$  have  $y \neq x$   $y \neq a$   $y \# P$   $y \# Q$  by simp+
  from  $\langle a \langle x \rangle . Q \mapsto b \langle y \rangle \prec Q' \rangle$   $\langle y \neq a \rangle$   $\langle y \neq x \rangle$   $\langle y \# Q \rangle$  show ?case
  proof(induct rule: inputCases)
    case cInput

    have  $a \langle x \rangle . P \mapsto a \langle x \rangle \prec P$  by(rule Input)
    hence  $a \langle x \rangle . P \mapsto a \langle y \rangle \prec ([ (x, y) ] \cdot P)$  using  $\langle y \# P \rangle$ 
      by(simp add: alphaBoundResidual)

    moreover have derivative  $([ (x, y) ] \cdot P)$   $([ (x, y) ] \cdot Q)$   $(InputS a) y Rel$ 
  proof(auto simp add: derivative-def)
    fix  $u$ 
    show  $([ (x, y) ] \cdot P)[y::=u], ([ (x, y) ] \cdot Q)[y::=u] \in Rel$ 
  proof(cases  $y=u$ )
    assume  $y = u$ 
    moreover have  $([ (y, x) ] \cdot P, [ (y, x) ] \cdot Q) \in Rel$ 
  proof -
    from PRelQ have  $(P[x::=x], Q[x::=x]) \in Rel$  by blast
    hence  $(P, Q) \in Rel$  by simp
    with Eqvt show ?thesis by(rule eqvtRelI)
  qed
  ultimately show ?thesis by simp

```

```

next
  assume yinequ:  $y \neq u$ 
  show ?thesis
  proof(cases  $x = u$ )
    assume  $x = u$ 
    moreover have  $([(y, x)] \cdot P)[y::=x], [(y, x)] \cdot Q[y::=x] \in Rel$ 
    proof -
      from PRelQ have  $(P[x::=y], Q[x::=y]) \in Rel$  by blast
      with Eqvt have  $([(y, x)] \cdot (P[x::=y]), [(y, x)] \cdot (Q[x::=y])) \in Rel$ 
        by(rule eqvtRelI)
      with  $\langle y \neq x \rangle$  show ?thesis
        by(simp add: eqvt-subs name-calc)
    qed
  ultimately show ?thesis by simp
next
  assume xinequ:  $x \neq u$ 
  hence  $([(y, x)] \cdot P)[y::=u], [(y, x)] \cdot Q[y::=u] \in Rel$ 
  proof -
    from PRelQ have  $(P[x::=u], Q[x::=u]) \in Rel$  by blast
    with Eqvt have  $([(y, x)] \cdot (P[x::=u]), [(y, x)] \cdot (Q[x::=u])) \in Rel$ 
      by(rule eqvtRelI)
    with  $\langle y \neq x \rangle$  xinequ yinequ show ?thesis
      by(simp add: eqvt-subs name-calc)
    qed
  thus ?thesis by simp
qed
qed
qed
ultimately show ?case by blast
qed
next
  case(Free  $\alpha$   $Q'$ )
  have  $a \langle x \rangle . Q \mapsto \alpha \prec Q'$  by fact
  hence False by auto
  thus ?case by blast
qed

lemma outputPres:
  fixes P :: pi
  and Q :: pi
  and a :: name
  and b :: name
  and Rel ::  $(pi \times pi)$  set
  and Rel' ::  $(pi \times pi)$  set

  assumes PRelQ:  $(P, Q) \in Rel$ 

  shows  $a\{b\}.P \rightsquigarrow[Rel] a\{b\}.Q$ 

```

```

proof –
  show ?thesis
  proof(induct rule: simCases)
    case(Bound c x Q')
    have  $a\{b\}.Q \mapsto c\langle x \rangle \prec Q'$  by fact
    hence False by auto
    thus ?case by simp
  next
    case(Free  $\alpha$  Q')
    have  $a\{b\}.Q \mapsto \alpha \prec Q'$  by fact
    thus ?case
    proof(induct rule: outputCases)
      case cOutput
      have  $a\{b\}.P \mapsto a[b] \prec P$  by(rule Late-Semantics.Output)
      with PRelQ show ?case by blast
    qed
  qed
qed

```

lemma *matchPres*:

```

fixes  $P$  :: pi
and  $Q$  :: pi
and  $a$  :: name
and  $b$  :: name
and  $Rel$  :: (pi  $\times$  pi) set
and  $Rel'$  :: (pi  $\times$  pi) set

```

```

assumes PSimQ:  $P \rightsquigarrow[Rel] Q$ 
and  $Rel \subseteq Rel'$ 

```

shows $[a \frown b]P \rightsquigarrow[Rel'] [a \frown b]Q$

proof –

```

show ?thesis
proof(induct rule: simCases)
  case(Bound c x Q')
  have  $x \# [a \frown b]P$  by fact
  hence  $xFreshP$ :  $x \# P$  by simp
  have  $[a \frown b]Q \mapsto c\langle x \rangle \prec Q'$  by fact
  thus ?case
  proof(induct rule: matchCases)
    case cMatch
    have  $Q \mapsto c\langle x \rangle \prec Q'$  by fact
    with PSimQ  $xFreshP$  obtain  $P'$  where  $PTrans$ :  $P \mapsto c\langle x \rangle \prec P'$ 
      and  $Pderivative$ :  $derivative\ P'\ Q'\ c\ x\ Rel$ 
    by(blast dest: simE)
  
```

```

from  $PTrans$  have  $[a \frown a]P \mapsto c\langle x \rangle \prec P'$  by(rule Late-Semantics.Match)
moreover from  $Pderivative$   $\langle Rel \subseteq Rel' \rangle$  have  $derivative\ P'\ Q'\ c\ x\ Rel'$ 
  by(cases c) (auto simp add: derivative-def)

```

```

    ultimately show ?case by blast
  qed
next
case(Free  $\alpha$   $Q'$ )
have  $[a \frown b]Q \mapsto \alpha \prec Q'$  by fact
thus ?case
proof(induct rule: matchCases)
  case cMatch
  have  $Q \mapsto \alpha \prec Q'$  by fact
  with PSimQ obtain  $P'$  where PTrans:  $P \mapsto \alpha \prec P'$ 
    and PRel:  $(P', Q') \in Rel$ 
    by(blast dest: simE)
  from PTrans have  $[a \frown a]P \mapsto \alpha \prec P'$  by(rule Late-Semantics.Match)
  with PRel  $\langle Rel \subseteq Rel' \rangle$  show ?case by blast
qed
qed
qed

lemma mismatchPres:
  fixes  $P$     ::  $pi$ 
  and  $Q$      ::  $pi$ 
  and  $a$      ::  $name$ 
  and  $b$      ::  $name$ 
  and  $Rel$   ::  $(pi \times pi)$  set
  and  $Rel'$  ::  $(pi \times pi)$  set

  assumes PSimQ:  $P \rightsquigarrow[Rel] Q$ 
  and       $Rel \subseteq Rel'$ 

  shows  $[a \neq b]P \rightsquigarrow[Rel'] [a \neq b]Q$ 
proof(induct rule: simCases)
  case(Bound  $c$   $x$   $Q'$ )
  have  $x \sharp [a \neq b]P$  by fact
  hence xFreshP:  $x \sharp P$  by simp
  from  $\langle [a \neq b]Q \mapsto c \langle x \rangle \prec Q' \rangle$  show ?case
  proof(induct rule: mismatchCases)
    case cMismatch
    have  $Q \mapsto c \langle x \rangle \prec Q'$  by fact
    with PSimQ xFreshP obtain  $P'$  where PTrans:  $P \mapsto c \langle x \rangle \prec P'$ 
      and Pderivative: derivative  $P' Q' c x Rel$ 
    by(blast dest: simE)

    from PTrans  $\langle a \neq b \rangle$  have  $[a \neq b]P \mapsto c \langle x \rangle \prec P'$  by(rule Late-Semantics.Mismatch)
    moreover from Pderivative  $\langle Rel \subseteq Rel' \rangle$  have derivative  $P' Q' c x Rel'$ 
      by(cases  $c$ ) (auto simp add: derivative-def)
    ultimately show ?case by blast
  qed
qed
next
case(Free  $\alpha$   $Q'$ )

```

```

have  $[a \neq b]Q \mapsto \alpha \prec Q'$  by fact
thus ?case
proof(induct rule: mismatchCases)
  case cMismatch
  have  $Q \mapsto \alpha \prec Q'$  by fact
  with PSimQ obtain  $P'$  where PTrans:  $P \mapsto \alpha \prec P'$ 
    and PRel:  $(P', Q') \in Rel$ 
  by(blast dest: simE)
  from PTrans  $\langle a \neq b \rangle$  have  $[a \neq b]P \mapsto \alpha \prec P'$  by(rule Late-Semantics.Mismatch)
  with PRel  $\langle Rel \subseteq Rel' \rangle$  show ?case by blast
qed
qed

```

lemma sumPres:

```

fixes  $P :: pi$ 
and  $Q :: pi$ 
and  $R :: pi$ 

```

```

assumes PSimQ:  $P \rightsquigarrow[Rel] Q$ 
and Id  $\subseteq Rel'$ 
and Rel  $\subseteq Rel'$ 

```

shows $P \oplus R \rightsquigarrow[Rel'] Q \oplus R$

proof –

show ?thesis

proof(induct rule: simCases)

case(Bound $a\ x\ QR$)

have $x \# P \oplus R$ by fact

hence xFreshP: $x \# P$ and xFreshR: $x \# R$ by simp+

have $Q \oplus R \mapsto a \langle x \rangle \prec QR$ by fact

thus ?case

proof(induct rule: sumCases)

case cSum1

have $Q \mapsto a \langle x \rangle \prec QR$ by fact

with xFreshP PSimQ obtain P' where PTrans: $P \mapsto a \langle x \rangle \prec P'$

and Pderivative: derivative $P' QR\ a\ x\ Rel$

by(blast dest: simE)

from PTrans have $P \oplus R \mapsto a \langle x \rangle \prec P'$ by(rule Late-Semantics.Sum1)

moreover from Pderivative $\langle Rel \subseteq Rel' \rangle$ have derivative $P' QR\ a\ x\ Rel'$

by(cases a) (auto simp add: derivative-def)

ultimately show ?case by blast

next

case cSum2

from $\langle R \mapsto a \langle x \rangle \prec QR \rangle$ have $P \oplus R \mapsto a \langle x \rangle \prec QR$ by(rule Sum2)

thus ?case using $\langle Id \subseteq Rel' \rangle$ by(blast intro: derivativeReflexive)

qed

next

case(Free $\alpha\ QR$)

```

have  $Q \oplus R \mapsto \alpha \prec QR$  by fact
thus ?case
proof(induct rule: sumCases)
  case cSum1
  have  $Q \mapsto \alpha \prec QR$  by fact
  with PSimQ obtain  $P'$  where  $PTrans: P \mapsto \alpha \prec P'$  and  $PRel: (P', QR)$ 
 $\in Rel$ 
  by(blast dest: simE)
  from PTrans have  $P \oplus R \mapsto \alpha \prec P'$  by(rule Late-Semantics.Sum1)
  with PRel  $\langle Rel \subseteq Rel' \rangle$  show ?case by blast
next
case cSum2
from  $\langle R \mapsto \alpha \prec QR \rangle$  have  $P \oplus R \mapsto \alpha \prec QR$  by(rule Sum2)
thus ?case using  $\langle Id \subseteq Rel' \rangle$  by(blast intro: derivativeReflexive)
qed
qed
qed

```

lemma parCompose:

```

fixes  $P$     ::  $pi$ 
and     $Q$     ::  $pi$ 
and     $R$     ::  $pi$ 
and     $T$     ::  $pi$ 
and     $Rel$   ::  $(pi \times pi)$  set
and     $Rel'$  ::  $(pi \times pi)$  set
and     $Rel''$  ::  $(pi \times pi)$  set

assumes PSimQ:  $P \rightsquigarrow[Rel] Q$ 
and    RSimT:  $R \rightsquigarrow[Rel'] T$ 
and    PRelQ:  $(P, Q) \in Rel$ 
and    RRel'T:  $(R, T) \in Rel'$ 
and    Par:  $\bigwedge P Q R T. \llbracket (P, Q) \in Rel; (R, T) \in Rel' \rrbracket \implies (P \parallel R, Q \parallel T)$ 
 $\in Rel''$ 
and    Res:  $\bigwedge P Q a. (P, Q) \in Rel'' \implies (\langle \nu a \rangle P, \langle \nu a \rangle Q) \in Rel''$ 
and    EqvtRel: eqvt Rel
and    EqvtRel': eqvt Rel'
and    EqvtRel'': eqvt Rel''

```

shows $P \parallel R \rightsquigarrow[Rel''] Q \parallel T$

using EqvtRel''

proof(induct rule: simCasesCont[where $C=()$])

case(Bound $a x Q'$)

have $x \# P \parallel R$ and $x \# Q \parallel T$ by fact+

hence $xFreshP: x \# P$ and $xFreshR: x \# R$ and $x \# Q$ and $x \# T$ by simp+

have $QTTrans: Q \parallel T \mapsto a \langle x \rangle \prec Q'$ by fact

thus ?case using $\langle x \# Q \rangle \langle x \# T \rangle$

proof(induct rule: parCasesB)

case(cPar1 Q')

have $QTrans: Q \mapsto a \langle x \rangle \prec Q'$ and $xFreshT: x \# T$ by fact+

```

from  $xFreshP$   $PSimQ$   $QTrans$  obtain  $P'$  where  $PTrans:P \mapsto a\langle x \rangle \prec P'$ 
and  $Pderivative: derivative\ P'\ Q'\ a\ x\ Rel$ 

  by(blast dest: simE)
from  $PTrans$   $xFreshR$  have  $P \parallel R \mapsto a\langle x \rangle \prec P' \parallel R$  by(rule Late-Semantics.Par1B)
moreover from  $Pderivative$   $xFreshR$   $xFreshT$   $RRel'T$  have  $derivative\ (P' \parallel$ 
 $R)\ (Q' \parallel T)\ a\ x\ Rel''$ 
  by(cases a, auto intro: Par simp add: derivative-def forget)
ultimately show ?case by blast
next
  case(cPar2 T')
  have  $TTrans: T \mapsto a\langle x \rangle \prec T'$  and  $xFreshQ: x \# Q$  by fact+

  from  $xFreshR$   $RSimT$   $TTrans$  obtain  $R'$  where  $RTrans:R \mapsto a\langle x \rangle \prec R'$ 
and  $Rderivative: derivative\ R'\ T'\ a\ x\ Rel'$ 

  by(blast dest: simE)
  from  $RTrans$   $xFreshP$  have  $ParTrans: P \parallel R \mapsto a\langle x \rangle \prec P \parallel R'$  by(rule
Late-Semantics.Par2B)
  moreover from  $Rderivative$   $xFreshP$   $xFreshQ$   $PRelQ$  have  $derivative\ (P \parallel R')$ 
 $(Q \parallel T')\ a\ x\ Rel''$ 
  by(cases a, auto intro: Par simp add: derivative-def forget)
  ultimately show ?case by blast
qed
next
  case(Free  $\alpha$  QT')
  have  $QTTrans: Q \parallel T \mapsto \alpha \prec QT'$  by fact
  thus ?case using  $PSimQ$   $RSimT$   $PRelQ$   $RRel'T$ 
  proof(induct rule: parCasesF[where C=(P, R)])
  case(cPar1 Q')
  have  $RRel'T: (R, T) \in Rel'$  by fact
  have  $P \rightsquigarrow[Rel] Q$  and  $Q \mapsto \alpha \prec Q'$  by fact+
  then obtain  $P'$  where  $PTrans: P \mapsto \alpha \prec P'$  and  $PRel: (P', Q') \in Rel$ 
  by(blast dest: simE)
  from  $PTrans$  have  $Trans: P \parallel R \mapsto \alpha \prec P' \parallel R$  by(rule Late-Semantics.Par1F)
  moreover from  $PRel$   $RRel'T$  have  $(P' \parallel R, Q' \parallel T) \in Rel''$  by(blast intro:
 $Par$ )
  ultimately show ?case by blast
  next
  case(cPar2 T')
  have  $PRelQ: (P, Q) \in Rel$  by fact
  have  $R \rightsquigarrow[Rel'] T$  and  $T \mapsto \alpha \prec T'$  by fact+
  then obtain  $R'$  where  $RTrans: R \mapsto \alpha \prec R'$  and  $RRel: (R', T') \in Rel'$ 
  by(blast dest: simE)
  from  $RTrans$  have  $Trans: P \parallel R \mapsto \alpha \prec P \parallel R'$  by(rule Late-Semantics.Par2F)
  moreover from  $PRelQ$   $RRel$  have  $(P \parallel R', Q \parallel T') \in Rel''$  by(blast intro:
 $Par$ )
  ultimately show ?case by blast
  next
  case(cComm1 Q' T' a b x)

```

from $\langle x \# (P, R) \rangle$ **have** $x \# P$ **by** *simp*
with $\langle P \rightsquigarrow_{[Rel]} Q \rangle \langle Q \mapsto a \langle x \rangle \prec Q' \rangle \langle x \# P \rangle$
obtain P' **where** $PTrans: P \mapsto a \langle x \rangle \prec P'$
and $Pderivative: derivative\ P'\ Q'\ (InputS\ a)\ x\ Rel$
by(*blast dest: simE*)
from $Pderivative$ **have** $PRel: (P'[x::=b], Q'[x::=b]) \in Rel$ **by**(*simp add: derivative-def*)

have $R \rightsquigarrow_{[Rel']} T$ **and** $T \mapsto a[b] \prec T'$ **by** *fact+*
then obtain R' **where** $RTrans: R \mapsto a[b] \prec R'$ **and** $RRel: (R', T') \in Rel'$
by(*blast dest: simE*)

from $PTrans\ RTrans$ **have** $P \parallel R \mapsto \tau \prec P'[x::=b] \parallel R'$ **by**(*rule Late-Semantics.Comm1*)
moreover from $PRel\ RRel$ **have** $(P'[x::=b] \parallel R', Q'[x::=b] \parallel T') \in Rel''$
by(*blast intro: Par*)
ultimately show *?case* **by** *blast*
next
case(*cComm2 Q' T' a b x*)
have $P \rightsquigarrow_{[Rel]} Q$ **and** $Q \mapsto a[b] \prec Q'$ **by** *fact+*
then obtain P' **where** $PTrans: P \mapsto a[b] \prec P'$ **and** $PRel: (P', Q') \in Rel$
by(*blast dest: simE*)

from $\langle x \# (P, R) \rangle$ **have** $x \# R$ **by** *simp*
with $\langle R \rightsquigarrow_{[Rel']} T \rangle \langle T \mapsto a \langle x \rangle \prec T' \rangle$
obtain R' **where** $RTrans: R \mapsto a \langle x \rangle \prec R'$
and $Rderivative: derivative\ R'\ T'\ (InputS\ a)\ x\ Rel'$
by(*blast dest: simE*)
from $Rderivative$ **have** $RRel: (R'[x::=b], T'[x::=b]) \in Rel'$ **by**(*simp add: derivative-def*)

from $PTrans\ RTrans$ **have** $P \parallel R \mapsto \tau \prec P' \parallel R'[x::=b]$ **by**(*rule Late-Semantics.Comm2*)
moreover from $PRel\ RRel$ **have** $(P' \parallel R'[x::=b], Q' \parallel T'[x::=b]) \in Rel''$
by(*blast intro: Par*)
ultimately show $\exists P'. P \parallel R \mapsto \tau \prec P' \wedge (P', Q' \parallel T'[x::=b]) \in Rel''$ **by**
blast
next
case(*cClose1 Q' T' a x y*)
from $\langle x \# (P, R) \rangle$ **have** $x \# P$ **by** *simp*
with $\langle P \rightsquigarrow_{[Rel]} Q \rangle \langle Q \mapsto a \langle x \rangle \prec Q' \rangle$
obtain P' **where** $PTrans: P \mapsto a \langle x \rangle \prec P'$
and $Pderivative: derivative\ P'\ Q'\ (InputS\ a)\ x\ Rel$
by(*blast dest: simE*)
from $Pderivative$ **have** $PRel: (P'[x::=y], Q'[x::=y]) \in Rel$ **by**(*simp add: derivative-def*)

from $\langle y \# (P, R) \rangle$ **have** $y \# R$ **and** $y \# P$ **by** *simp+*
from $\langle R \rightsquigarrow_{[Rel']} T \rangle \langle T \mapsto a \langle \nu y \rangle \prec T' \rangle \langle y \# R \rangle$
obtain R' **where** $RTrans: R \mapsto a \langle \nu y \rangle \prec R'$
and $Rderivative: derivative\ R'\ T'\ (BoundOutputS\ a)\ y\ Rel'$

by(*blast dest: simE*)
from *Rderivative* **have** $RRel: (R', T') \in Rel'$ **by**(*simp add: derivative-def*)

from *PTrans RTrans* $\langle y \# P \rangle$ **have** $Trans: P \parallel R \mapsto \tau \prec \langle \nu y \rangle (P'[x::=y] \parallel R')$
by(*rule Late-Semantics.Close1*)
moreover from *PRel RRel* **have** $(\langle \nu y \rangle (P'[x::=y] \parallel R'), \langle \nu y \rangle (Q'[x::=y] \parallel T')) \in Rel''$
by(*blast intro: Par Res*)
ultimately show *?case* **by** *blast*
next
case(*cClose2 Q' T' a x y*)
from $\langle y \# (P, R) \rangle$ **have** $y \# P$ **and** $y \# R$ **by** *simp+*
from $\langle P \rightsquigarrow[Rel] Q \rangle$ $\langle Q \mapsto a \langle \nu y \rangle \prec Q' \rangle$ $\langle y \# P \rangle$
obtain P' **where** $PTrans: P \mapsto a \langle \nu y \rangle \prec P'$ **and** $P'RelQ': (P', Q') \in Rel$
by(*force dest: simE simp add: derivative-def*)

from $\langle x \# (P, R) \rangle$ **have** $x \# R$ **by** *simp+*
with $\langle R \rightsquigarrow[Rel'] T \rangle$ $\langle T \mapsto a \langle x \rangle \prec T' \rangle$
obtain R' **where** $RTrans: R \mapsto a \langle x \rangle \prec R'$
and $R'Rel'T': (R'[x::=y], T'[x::=y]) \in Rel'$
by(*force dest: simE simp add: derivative-def*)

from *PTrans RTrans* $\langle y \# R \rangle$ **have** $Trans: P \parallel R \mapsto \tau \prec \langle \nu y \rangle (P' \parallel R'[x::=y])$
by(*rule Close2*)
moreover from $P'RelQ' R'Rel'T'$ **have** $(\langle \nu y \rangle (P' \parallel R'[x::=y]), \langle \nu y \rangle (Q' \parallel T'[x::=y])) \in Rel''$
by(*blast intro: Par Res*)
ultimately show *?case* **by** *blast*
qed
qed

lemma *parPres*:
fixes $P :: pi$
and $Q :: pi$
and $R :: pi$
and $a :: name$
and $b :: name$
and $Rel :: (pi \times pi) set$
and $Rel' :: (pi \times pi) set$

assumes $PSimQ: P \rightsquigarrow[Rel] Q$
and $PRelQ: (P, Q) \in Rel$
and $Par: \bigwedge P Q R. (P, Q) \in Rel \implies (P \parallel R, Q \parallel R) \in Rel'$
and $Res: \bigwedge P Q a. (P, Q) \in Rel' \implies (\langle \nu a \rangle P, \langle \nu a \rangle Q) \in Rel'$
and $EqvtRel: eqvt Rel$
and $EqvtRel': eqvt Rel'$

shows $P \parallel R \rightsquigarrow[Rel'] Q \parallel R$

proof –
note $PSimQ$
moreover have $RSimR: R \rightsquigarrow[Id] R$ **by** $(auto\ intro: reflexive)$
moreover note $PRelQ$ **moreover have** $(R, R) \in Id$ **by** $auto$
moreover from Par **have** $\bigwedge P Q R T. [(P, Q) \in Rel; (R, T) \in Id] \implies (P \parallel R, Q \parallel T) \in Rel'$
by $auto$
moreover note $Res \langle eqvt Rel \rangle$
moreover have $eqvt Id$ **by** $(auto\ simp\ add: eqvt-def)$
ultimately show $?thesis$ **using** $EqvtRel'$ **by** $(rule\ parCompose)$
qed

lemma $resDerivative$:

fixes $P \quad :: pi$
and $Q \quad :: pi$
and $a \quad :: subject$
and $x \quad :: name$
and $y \quad :: name$
and $Rel \quad :: (pi \times pi)\ set$
and $Rel' \quad :: (pi \times pi)\ set$

assumes $Der: derivative\ P\ Q\ a\ x\ Rel$

and $Rel: \bigwedge (P::pi)\ (Q::pi)\ (x::name). (P, Q) \in Rel \implies (\langle \nu x \rangle P, \langle \nu x \rangle Q) \in Rel'$

and $Eqv: eqvt\ Rel$

shows $derivative\ (\langle \nu y \rangle P)\ (\langle \nu y \rangle Q)\ a\ x\ Rel'$

proof –

from $Der\ Rel$ **show** $?thesis$

proof $(cases\ a, auto\ simp\ add: derivative-def)$

fix u

assume $A1: \forall u. (P[x::=u], Q[x::=u]) \in Rel$

show $((\langle \nu y \rangle P)[x::=u], (\langle \nu y \rangle Q)[x::=u]) \in Rel'$

proof $(cases\ x=y)$

assume $x=y$

from $A1$ **have** $(P[x::=x], Q[x::=x]) \in Rel$ **by** $blast$

hence $L1: (\langle \nu y \rangle P, \langle \nu y \rangle Q) \in Rel'$ **by** $(force\ intro: Rel)$

have $y \# \langle \nu y \rangle P$ **and** $y \# \langle \nu y \rangle Q$ **by** $(simp\ only: freshRes)+$

hence $(\langle \nu y \rangle P)[y::=u] = \langle \nu y \rangle P$ **and** $(\langle \nu y \rangle Q)[y::=u] = \langle \nu y \rangle Q$ **by** $(simp\ add: forget)+$

with $L1\ x=y$ **show** $?thesis$ **by** $simp$

next

assume $x \neq y$

show $?thesis$

proof $(cases\ y=u)$

```

assume y equ:  $y = u$ 

have  $\exists (c :: \text{name}). c \# (P, Q, x, y)$  by (blast intro: name-exists-fresh)
then obtain c where cFreshP:  $c \# P$  and cFreshQ:  $c \# Q$  and cineqx:  $c \neq x$ 
and cineqy:  $y \neq c$ 
by (force simp add: fresh-prod name-fresh)

from A1 have  $(P[x::=c], Q[x::=c]) \in \text{Rel}$  by blast
with Eqv have  $([(y, c)] \cdot P[x::=c], [(y, c)] \cdot (Q[x::=c])) \in \text{Rel}$  by (rule
eqvtRelI)
with xineqy cineqx cineqy have  $(([(y, c)] \cdot P)[x::=y], ([(y, c)] \cdot Q)[x::=y])$ 
 $\in \text{Rel}$ 
by (simp add: eqvt-subs name-calc)
hence  $\langle \nu c \rangle (([(y, c)] \cdot P)[x::=y]), \langle \nu c \rangle (([(y, c)] \cdot Q)[x::=y]) \in \text{Rel}'$ 
by (rule Rel)
with cineqx cineqy have  $(\langle \nu c \rangle (([(y, c)] \cdot P))[x::=y], \langle \nu c \rangle (([(y, c)] \cdot Q))$ 
 $[x::=y]) \in \text{Rel}'$  by simp
moreover from cFreshP cFreshQ have  $\langle \nu c \rangle (([(y, c)] \cdot P) = \langle \nu y \rangle P$  and
 $\langle \nu c \rangle (([(y, c)] \cdot Q) = \langle \nu y \rangle Q$ 
by (simp add: alphaRes) +
ultimately show ?thesis using yequ by simp
next
assume yineqy:  $y \neq u$ 
from A1 have  $(P[x::=u], Q[x::=u]) \in \text{Rel}$  by blast
hence  $\langle \nu y \rangle (P[x::=u]), \langle \nu y \rangle (Q[x::=u]) \in \text{Rel}'$  by (rule Rel)
with xineqy yineqy show ?thesis by simp
qed
qed
qed
qed

lemma resPres:
fixes P :: pi
and Q :: pi
and Rel ::  $(\text{pi} \times \text{pi}) \text{ set}$ 
and x :: name
and Rel' ::  $(\text{pi} \times \text{pi}) \text{ set}$ 

assumes PSimQ:  $P \rightsquigarrow[\text{Rel}] Q$ 
and ResRel:  $\bigwedge (P :: \text{pi}) (Q :: \text{pi}) (x :: \text{name}). (P, Q) \in \text{Rel} \implies \langle \nu x \rangle P, \langle \nu x \rangle Q$ 
 $\in \text{Rel}'$ 
and RelRel':  $\text{Rel} \subseteq \text{Rel}'$ 
and EqvtRel: eqvt Rel
and EqvtRel': eqvt Rel'

shows  $\langle \nu x \rangle P \rightsquigarrow[\text{Rel}'] \langle \nu x \rangle Q$ 
using EqvtRel'
proof (induct rule: resSimCases[of - - - (P, x)])
case (BoundOutput Q' a)

```

have $QTrans: Q \mapsto a[x] \prec Q'$ **and** $aineqx: a \neq x$ **by** $fact+$

from $PSimQ\ QTrans$ **obtain** P' **where** $PTrans: P \mapsto a[x] \prec P'$
and $P'RelQ': (P', Q') \in Rel$

by($blast\ dest: simE$)

from $PTrans\ aineqx$ **have** $\langle \nu x \rangle P \mapsto a \langle \nu x \rangle \prec P'$ **by**($rule\ Late-Semantics.Open$)

moreover from $P'RelQ'\ RelRel'$ **have** $(P', Q') \in Rel'$ **by** $force$

ultimately show $?case$ **by** $blast$

next

case($BoundR\ Q'\ a\ y$)

have $QTrans: Q \mapsto a \langle y \rangle \prec Q'$ **and** $xFresha: x \# a$ **by** $fact+$

have $y \# (P, x)$ **by** $fact$

hence $yFreshP: y \# P$ **and** $yineqx: y \neq x$ **by**($simp\ add: fresh-prod$)+

from $PSimQ\ yFreshP\ QTrans$ **obtain** P' **where** $PTrans: P \mapsto a \langle y \rangle \prec P'$
and $Pderivative: derivative\ P'\ Q'\ a\ y\ Rel$

by($blast\ dest: simE$)

from $PTrans\ xFresha\ yineqx$ **have** $ResTrans: \langle \nu x \rangle P \mapsto a \langle y \rangle \prec \langle \nu x \rangle P'$
by($blast\ intro: Late-Semantics.ResB$)

moreover from $Pderivative\ ResRel\ EqvtRel$ **have** $derivative\ (\langle \nu x \rangle P')\ (\langle \nu x \rangle Q')$
 $a\ y\ Rel'$

by($rule\ resDerivative$)

ultimately show $?case$ **by** $blast$

next

case($FreeR\ Q'\ \alpha$)

have $QTrans: Q \mapsto \alpha \prec Q'$ **and** $xFreshAlpha: (x::name) \# \alpha$ **by** $fact+$

from $QTrans\ PSimQ$ **obtain** P' **where** $PTrans: P \mapsto \alpha \prec P'$
and $P'RelQ': (P', Q') \in Rel$

by($blast\ dest: simE$)

from $PTrans\ xFreshAlpha$ **have** $\langle \nu x \rangle P \mapsto \alpha \prec \langle \nu x \rangle P'$ **by**($rule\ Late-Semantics.ResF$)

moreover from $P'RelQ'$ **have** $(\langle \nu x \rangle P', \langle \nu x \rangle Q') \in Rel'$ **by**($rule\ ResRel$)

ultimately show $?case$ **by** $blast$

qed

lemma $resChainI$:

fixes $P :: pi$

and $Q :: pi$

and $Rel :: (pi \times pi)\ set$

and $xs :: name\ list$

assumes $PRelQ: P \rightsquigarrow[Rel] Q$

and $eqvtRel: eqvt\ Rel$

and $Res: \bigwedge P\ Q\ x. (P, Q) \in Rel \implies (\langle \nu x \rangle P, \langle \nu x \rangle Q) \in Rel$

shows $(resChain\ xs)\ P \rightsquigarrow[Rel] (resChain\ xs)\ Q$

```

proof(induct xs)
  from PRelQ show resChain [] P  $\rightsquigarrow$ [Rel] resChain [] Q by simp
next
  fix x xs
  assume IH: (resChain xs P)  $\rightsquigarrow$ [Rel] (resChain xs Q)
  moreover note Res
  moreover have Rel  $\subseteq$  Rel by simp
  ultimately have  $\langle \nu x \rangle$ (resChain xs P)  $\rightsquigarrow$ [Rel]  $\langle \nu x \rangle$ (resChain xs Q) using
eqvtRel
  by(rule-tac resPres)

  thus resChain (x # xs) P  $\rightsquigarrow$ [Rel] resChain (x # xs) Q
  by simp
qed

```

lemma *bangPres*:

```

fixes P :: pi
and Q :: pi
and Rel :: (pi  $\times$  pi) set

```

```

assumes PRelQ: (P, Q)  $\in$  Rel
and Sim:  $\bigwedge P Q. (P, Q) \in Rel \implies P \rightsquigarrow[Rel] Q$ 
and eqvtRel: eqvt Rel

```

shows $!P \rightsquigarrow[bangRel\ Rel] !Q$

proof –

```

let ?Sim =  $\lambda P Rs. (\forall a x Q'. Rs = a \ll x \gg \prec Q' \longrightarrow x \# P \longrightarrow (\exists P'. P \longmapsto a \ll x \gg$ 
 $\prec P' \wedge \text{derivative } P' Q' a x (bangRel\ Rel))) \wedge$ 
 $(\forall \alpha Q'. Rs = \alpha \prec Q' \longrightarrow (\exists P'. P \longmapsto \alpha \prec P' \wedge (P', Q') \in$ 
bangRel Rel))

```

from *eqvtRel* **have** *EqvtBangRel*: *eqvt*(*bangRel Rel*) **by**(*rule eqvtBangRel*)

```

{
  fix Pa Rs
  assume  $!Q \longmapsto Rs$  and (Pa,  $!Q$ )  $\in$  bangRel Rel
  hence ?Sim Pa Rs using PRelQ
  proof(nominal-induct avoiding: Pa P rule: bangInduct)
    case(cPar1B a x Q' Pa P)
    have QTrans:  $Q \longmapsto a \ll x \gg \prec Q'$  by fact
    have (Pa,  $Q \parallel !Q$ )  $\in$  bangRel Rel and  $x \# Pa$  by fact+
    thus ?Sim Pa (a \ll x \gg \prec (Q' \parallel !Q))
    proof(induct rule: BRParCases)
      case(BRPar P R)
      have PRelQ: (P, Q)  $\in$  Rel by fact
      have PBRQ: (R,  $!Q$ )  $\in$  bangRel Rel by fact
      have  $x \# P \parallel R$  by fact
      hence xFreshP:  $x \# P$  and xFreshR:  $x \# R$  by simp+
      show ?case
      proof(auto simp add: residual.inject alpha')

```

from $P\text{Rel}Q$ **have** $P \rightsquigarrow[\text{Rel}] Q$ **by** (*rule Sim*)

with $Q\text{Trans } x\text{Fresh}P$ **obtain** P' **where** $P\text{Trans}: P \mapsto a\langle x \rangle \prec P'$ **and**
 $P'\text{Rel}Q'$: *derivative* $P' Q' a x \text{Rel}$
by (*blast dest: simE*)

from $P\text{Trans } x\text{Fresh}R$ **have** $P \parallel R \mapsto a\langle x \rangle \prec (P' \parallel R)$
by (*force intro: Late-Semantics.Par1B*)

moreover from $P'\text{Rel}Q' PBRQ \langle x \# Q \rangle \langle x \# R \rangle$ **have** *derivative* $(P' \parallel R) (Q' \parallel !Q) a x$ (*bangRel Rel*)
by (*cases a*) (*auto simp add: derivative-def forget intro: Rel.BRPar*)

ultimately show $\exists P'. P \parallel R \mapsto a\langle x \rangle \prec P' \wedge$ *derivative* $P' (Q' \parallel !Q) a x$ (*bangRel Rel*) **by** *blast*

next

fix y

assume $(y::\text{name}) \# Q'$ **and** $y \# P$ **and** $y \# R$ **and** $y \# Q$

from $Q\text{Trans } \langle y \# Q' \rangle$ **have** $Q \mapsto a\langle y \rangle \prec ((x, y) \cdot Q')$
by (*simp add: alphaBoundResidual*)

moreover from $P\text{Rel}Q$ **have** $P \rightsquigarrow[\text{Rel}] Q$ **by** (*rule Sim*)

ultimately obtain P' **where** $P\text{Trans}: P \mapsto a\langle y \rangle \prec P'$ **and** $P'\text{Rel}Q'$:
derivative $P' ((x, y) \cdot Q') a y \text{Rel}$

using $\langle y \# P \rangle$

by (*blast dest: simE*)

from $P\text{Trans } \langle y \# R \rangle$ **have** $P \parallel R \mapsto a\langle y \rangle \prec (P' \parallel R)$ **by** (*force intro: Late-Semantics.Par1B*)

moreover from $P'\text{Rel}Q' PBRQ \langle y \# Q \rangle \langle y \# R \rangle$ **have** *derivative* $(P' \parallel R) (((x, y) \cdot Q') \parallel !Q) a y$ (*bangRel Rel*)
by (*cases a*) (*auto simp add: derivative-def forget intro: Rel.BRPar*)

with $\langle x \# Q \rangle \langle y \# Q \rangle$ **have** *derivative* $(P' \parallel R) (((y, x) \cdot Q') \parallel !((y, x) \cdot Q)) a y$ (*bangRel Rel*)
by (*simp add: name-fresh-fresh name-swap*)

ultimately show $\exists P'. P \parallel R \mapsto a\langle y \rangle \prec P' \wedge$ *derivative* $P' (((y, x) \cdot Q') \parallel !((y, x) \cdot Q)) a y$ (*bangRel Rel*)
by *blast*

qed

qed

next

case ($c\text{Par}1F \alpha Q' Pa P$)

have $Q\text{Trans}: Q \mapsto \alpha \prec Q'$ **by** *fact*

have $(Pa, Q \parallel !Q) \in \text{bangRel Rel}$ **by** *fact*

thus *?case*

proof (*induct rule: BRParCases*)

case ($BR\text{Par } P R$)

have $P\text{Rel}Q: (P, Q) \in \text{Rel}$ **and** $BR: (R, !Q) \in \text{bangRel Rel}$ **by** *fact+*

show *?case*

proof (*auto simp add: residual.inject*)

from $P\text{Rel}Q$ **have** $P \rightsquigarrow[\text{Rel}] Q$ **by** (*rule Sim*)

with $Q\text{Trans}$ **obtain** P' **where** $P\text{Trans}: P \mapsto \alpha \prec P'$ **and** $R\text{Rel}: (P', Q') \in \text{Rel}$

by(blast dest: simE)

from *PTrans* **have** $P \parallel R \mapsto \alpha \prec P' \parallel R$ **by**(rule *Par1F*)

moreover from *RRel BR* **have** $(P' \parallel R, Q' \parallel !Q) \in \text{bangRel Rel}$ **by**(rule *Rel.BRPar*)

ultimately show $\exists P'. P \parallel R \mapsto \alpha \prec P' \wedge (P', Q' \parallel !Q) \in \text{bangRel Rel}$

by blast

qed

qed

next

case(*cPar2B a x Q' Pa P*)

hence *IH*: $\bigwedge Pa. (Pa, !Q) \in \text{bangRel Rel} \implies ?\text{Sim Pa } (a\langle x \rangle \prec Q')$ **by** *simp*

have $(Pa, Q \parallel !Q) \in \text{bangRel Rel}$ **and** $x \# Pa$ **by** *fact+*

thus $?\text{Sim Pa } (a\langle x \rangle \prec (Q \parallel Q'))$

proof(*induct rule: BRParCases*)

case(*BRPar P R*)

have *PRelQ*: $(P, Q) \in \text{Rel}$ **and** *RBRQ*: $(R, !Q) \in \text{bangRel Rel}$ **by** *fact+*

have $x \# P \parallel R$ **by** *fact*

hence *xFreshP*: $x \# P$ **and** *xFreshR*: $x \# R$ **by** *simp+*

from *EqvtBangRel* $\langle x \# Q \rangle$ **show** $?\text{Sim } (P \parallel R) (a\langle x \rangle \prec (Q \parallel Q'))$

proof(*auto simp add: residual.inject alpha' name-fresh-fresh*)

from *RBRQ* **have** $?\text{Sim } R (a\langle x \rangle \prec Q')$ **by**(rule *IH*)

with *xFreshR* **obtain** R' **where** *RTrans*: $R \mapsto a\langle x \rangle \prec R'$ **and** *R'BRQ'*:

derivative $R' Q' a x$ (*bangRel Rel*)

by(*auto simp add: residual.inject*)

from *RTrans* *xFreshP* **have** $P \parallel R \mapsto a\langle x \rangle \prec (P \parallel R')$ **by**(*auto intro:*

Par2B)

moreover from *PRelQ R'BRQ'* $\langle x \# Q \rangle \langle x \# P \rangle$ **have** *derivative* $(P \parallel R')$

$(Q \parallel Q') a x$ (*bangRel Rel*)

by(*cases a*) (*auto simp add: derivative-def forget intro: Rel.BRPar*)

ultimately show $\exists P'. P \parallel R \mapsto a\langle x \rangle \prec P' \wedge \text{derivative } P' (Q \parallel Q') a$

x (*bangRel Rel*) **by** blast

next

fix y

assume $(y::\text{name}) \# Q$ **and** $y \# Q'$ **and** $y \# P$ **and** $y \# R$

from *RBRQ* **have** $?\text{Sim } R (a\langle x \rangle \prec Q')$ **by**(rule *IH*)

with $\langle y \# Q' \rangle$ **have** $?\text{Sim } R (a\langle y \rangle \prec ((x, y) \cdot Q'))$ **by**(*simp add:*

alphaBoundResidual)

with $\langle y \# R \rangle$ **obtain** R' **where** *RTrans*: $R \mapsto a\langle y \rangle \prec R'$ **and** *R'BRQ'*:

derivative $R' ((x, y) \cdot Q') a y$ (*bangRel Rel*)

by(*auto simp add: residual.inject*)

from *RTrans* $\langle y \# P \rangle$ **have** $P \parallel R \mapsto a\langle y \rangle \prec (P \parallel R')$ **by**(*auto intro:*

Par2B)

moreover from *PRelQ R'BRQ'* $\langle y \# P \rangle \langle y \# Q \rangle$ **have** *derivative* $(P \parallel R')$

$(Q \parallel ((x, y) \cdot Q')) a y$ (*bangRel Rel*)

by(*cases a*) (*auto simp add: derivative-def forget intro: Rel.BRPar*)

hence *derivative* $(P \parallel R') (Q \parallel ((y, x) \cdot Q')) a y$ (*bangRel Rel*)

by(*simp add: name-swap*)

ultimately show $\exists P'. P \parallel R \mapsto a \llbracket y \rrbracket \prec P' \wedge \text{derivative } P' (Q \parallel ((y, x) \cdot Q'))$ *a y (bangRel Rel) by blast*
qed
qed
next
case(*cPar2F* α Q' Pa P)
hence *IH*: $\bigwedge Pa. (Pa, !Q) \in \text{bangRel Rel} \implies ?\text{Sim } Pa (\alpha \prec Q')$ **by** *simp*
have $(Pa, Q \parallel !Q) \in \text{bangRel Rel}$ **by** *fact*
thus *?case*
proof(*induct rule: BRParCases*)
case(*BRPar* P R)
have *PRelQ*: $(P, Q) \in \text{Rel}$ **and** *RBRQ*: $(R, !Q) \in \text{bangRel Rel}$ **by** *fact+*
show *?case*
proof(*auto simp add: residual.inject*)
from *RBRQ IH* **have** $\exists R'. R \mapsto \alpha \prec R' \wedge (R', Q') \in \text{bangRel Rel}$
by(*metis simE*)
then obtain R' **where** *RTrans*: $R \mapsto \alpha \prec R'$ **and** *R'RelQ'*: $(R', Q') \in \text{bangRel Rel}$
by *blast*

from *RTrans* **have** $P \parallel R \mapsto \alpha \prec P \parallel R'$ **by**(*rule Par2F*)
moreover from *PRelQ R'RelQ'* **have** $(P \parallel R', Q \parallel Q') \in \text{bangRel Rel}$
by(*rule Rel.BRPar*)
ultimately show $\exists P'. P \parallel R \mapsto \alpha \prec P' \wedge (P', Q \parallel Q') \in \text{bangRel Rel}$
by *blast*
qed
qed
next
case(*cComm1* a x Q' b Q'' Pa P)
hence *IH*: $\bigwedge Pa. (Pa, !Q) \in \text{bangRel Rel} \implies ?\text{Sim } Pa (a[b] \prec Q'')$ **by** *simp*
have *QTrans*: $Q \mapsto a \langle x \rangle \prec Q'$ **by** *fact*
have $(Pa, Q \parallel !Q) \in \text{bangRel Rel}$ **by** *fact*
thus *?case using* $\langle x \# Pa \rangle$
proof(*induct rule: BRParCases*)
case(*BRPar* P R)
have *PRelQ*: $(P, Q) \in \text{Rel}$ **and** *RBRQ*: $(R, !Q) \in \text{bangRel Rel}$ **by** *fact+*
from $\langle x \# P \parallel R \rangle$ **have** $x \# P$ **and** $x \# R$ **by** *simp+*
show *?case*
proof(*auto simp add: residual.inject*)
from *PRelQ* **have** $P \rightsquigarrow[\text{Rel}] Q$ **by**(*rule Sim*)
with *QTrans* $\langle x \# P \rangle$ **obtain** P' **where** *PTrans*: $P \mapsto a \langle x \rangle \prec P'$ **and**
P'RelQ': $(P'[x::=b], Q'[x::=b]) \in \text{Rel}$
by(*drule-tac simE*) (*auto simp add: derivative-def*)

from *IH RBRQ* **have** *RTrans*: $\exists R'. R \mapsto a[b] \prec R' \wedge (R', Q'') \in \text{bangRel Rel}$
by(*auto simp add: derivative-def*)
then obtain R' **where** *RTrans*: $R \mapsto a[b] \prec R'$ **and** *R'RelQ''*: $(R', Q'') \in \text{bangRel Rel}$
by *blast*

by blast
from $PTrans$ $RTrans$ **have** $P \parallel R \mapsto \tau \prec P'[x::=b] \parallel R'$ **by**(rule $Comm1$)
moreover from $P'RelQ'$ $R'RelQ''$ **have** $(P'[x::=b] \parallel R', Q'[x::=b] \parallel Q'')$
 \in $bangRel\ Rel$ **by**(rule $Rel.BRPar$)
ultimately show $\exists P'. P \parallel R \mapsto \tau \prec P' \wedge (P', Q'[x::=b] \parallel Q'') \in$
 $bangRel\ Rel$ **by blast**
qed
qed
next
case($cComm2\ a\ b\ Q'\ x\ Q''\ Pa\ P$)
hence IH: $\bigwedge Pa. (Pa, !Q) \in bangRel\ Rel \implies ?Sim\ Pa\ (a\langle x \rangle \prec Q'')$ **by simp**
have $QTrans: Q \mapsto a[b] \prec Q'$ **by fact**
have $(Pa, Q \parallel !Q) \in bangRel\ Rel$ **by fact**
thus ?case using $\langle x \# Pa \rangle$
proof(*induct rule: BRParCases*)
case($BRPar\ P\ R$)
have $PRelQ: (P, Q) \in Rel$ **and** $RBRQ: (R, !Q) \in bangRel\ Rel$ **by fact+**
from $\langle x \# P \parallel R \rangle$ **have** $x \# P$ **and** $x \# R$ **by simp+**
show ?case
proof(*auto simp add: residual.inject*)
from $PRelQ$ **have** $P \rightsquigarrow[Rel] Q$ **by**(rule Sim)
with $QTrans$ **obtain** P' **where** $PTrans: P \mapsto a[b] \prec P'$ **and** $P'RelQ':$
 $(P', Q') \in Rel$
by(*blast dest: simE*)
from IH $RBRQ\ \langle x \# R \rangle$ **have** $RTrans: \exists R'. R \mapsto a\langle x \rangle \prec R' \wedge (R'[x::=b],$
 $Q''[x::=b]) \in bangRel\ Rel$
by(*fastforce simp add: derivative-def residual.inject*)
then obtain R' **where** $RTrans: R \mapsto a\langle x \rangle \prec R'$ **and** $R'RelQ'':$
 $(R'[x::=b], Q''[x::=b]) \in bangRel\ Rel$
by blast
from $PTrans$ $RTrans$ **have** $P \parallel R \mapsto \tau \prec P' \parallel R'[x::=b]$ **by**(rule $Comm2$)
moreover from $P'RelQ'$ $R'RelQ''$ **have** $(P' \parallel R'[x::=b], Q' \parallel Q''[x::=b])$
 \in $bangRel\ Rel$ **by**(rule $Rel.BRPar$)
ultimately show $\exists P'. P \parallel R \mapsto \tau \prec P' \wedge (P', Q' \parallel (Q''[x::=b])) \in$
 $bangRel\ Rel$ **by blast**
qed
qed
next
case($cClose1\ a\ x\ Q'\ y\ Q''\ Pa\ P$)
hence IH: $\bigwedge Pa. (Pa, !Q) \in bangRel\ Rel \implies ?Sim\ Pa\ (a\langle \nu y \rangle \prec Q'')$ **by**
simp
have $QTrans: Q \mapsto a\langle x \rangle \prec Q'$ **by fact**
have $(Pa, Q \parallel !Q) \in bangRel\ Rel$ **by fact**
moreover have $xFreshPa: x \# Pa$ **by fact**
ultimately show ?case using $\langle y \# Pa \rangle$
proof(*induct rule: BRParCases*)

case(*BRPar P R*)
have $P\text{Rel}Q: (P, Q) \in \text{Rel}$ **and** $R\text{BR}Q: (R, !Q) \in \text{bangRel Rel}$ **by** *fact+*
have $x \# P \parallel R$ **and** $y \# P \parallel R$ **by** *fact+*
hence $x\text{Fresh}P: x \# P$ **and** $x\text{Fresh}R: x \# R$ **and** $y \# R$ **and** $y \# P$ **by** *simp+*
show *?case*
proof(*auto simp add: residual.inject*)
from $P\text{Rel}Q$ **have** $P \rightsquigarrow[\text{Rel}] Q$ **by**(*rule Sim*)
with $Q\text{Trans } x\text{Fresh}P$ **obtain** P' **where** $P\text{Trans}: P \mapsto a \langle x \rangle \prec P'$ **and**
 $P'\text{Rel}Q': (P'[x::=y], Q'[x::=y]) \in \text{Rel}$
by(*fastforce dest: simE simp add: derivative-def*)

from $R\text{BR}Q \langle y \# R \rangle$ *IH* **have** $\exists R'. R \mapsto a \langle \nu y \rangle \prec R' \wedge (R', Q') \in$
 bangRel Rel
by(*auto simp add: residual.inject derivative-def*)
then obtain R' **where** $R\text{Trans}: R \mapsto a \langle \nu y \rangle \prec R'$ **and** $R'\text{Rel}Q'': (R',$
 $Q'') \in \text{bangRel Rel}$
by *blast*

from $P\text{Trans } R\text{Trans} \langle y \# P \rangle$ **have** $P \parallel R \mapsto \tau \prec \langle \nu y \rangle (P'[x::=y] \parallel R')$
by(*rule Close1*)
moreover from $P'\text{Rel}Q' \ R'\text{Rel}Q''$ **have** $\langle \nu y \rangle (P'[x::=y] \parallel R'),$
 $\langle \nu y \rangle (Q'[x::=y] \parallel Q'') \in \text{bangRel Rel}$
by(*force intro: Rel.BRPar BRRes*)
ultimately show $\exists P'. P \parallel R \mapsto \tau \prec P' \wedge (P', \langle \nu y \rangle (Q'[x::=y] \parallel Q''))$
 $\in \text{bangRel Rel}$ **by** *blast*
qed
qed
next
case(*cClose2 a x Q' y Q'' Pa P*)
hence *IH*: $\bigwedge Pa. (Pa, !Q) \in \text{bangRel Rel} \implies ?\text{Sim } Pa \ (a \langle y \rangle \prec Q'')$ **by** *simp*
have $Q\text{Trans}: Q \mapsto a \langle \nu x \rangle \prec Q'$ **by** *fact*
have $(Pa, Q \parallel !Q) \in \text{bangRel Rel}$ **and** $x \# Pa$ **and** $y \# Pa$ **by** *fact+*
thus *?case*
proof(*induct rule: BRParCases*)
case(*BRPar P R*)
have $P\text{Rel}Q: (P, Q) \in \text{Rel}$ **and** $R\text{BR}Q: (R, !Q) \in \text{bangRel Rel}$ **by** *fact+*
have $x \# P \parallel R$ **and** $y \# P \parallel R$ **by** *fact+*
hence $x\text{Fresh}P: x \# P$ **and** $x\text{Fresh}R: x \# R$ **and** $y \# R$ **by** *simp+*
show *?case*
proof(*auto simp add: residual.inject*)
from $P\text{Rel}Q$ **have** $P \rightsquigarrow[\text{Rel}] Q$ **by**(*rule Sim*)
with $Q\text{Trans } x\text{Fresh}P$ **obtain** P' **where** $P\text{Trans}: P \mapsto a \langle \nu x \rangle \prec P'$ **and**
 $P'\text{Rel}Q': (P', Q') \in \text{Rel}$
by(*fastforce dest: simE simp add: derivative-def*)

from $R\text{BR}Q \text{ IH } \langle y \# R \rangle$ **have** $\exists R'. R \mapsto a \langle y \rangle \prec R' \wedge (R'[y::=x],$
 $Q''[y::=x]) \in \text{bangRel Rel}$
by(*fastforce simp add: derivative-def residual.inject*)
then obtain R' **where** $R\text{Trans}: R \mapsto a \langle y \rangle \prec R'$ **and** $R'\text{Rel}Q'': (R'[y::=x],$

```

Q''[y::=x]) ∈ bangRel Rel
  by blast

  from PTrans RTrans xFreshR have P ∥ R ⟶ τ < νx > (P' ∥ R'[y::=x])
    by (rule Close2)
  moreover from P'RelQ' R'RelQ'' have (< νx > (P' ∥ R'[y::=x]), < νx > (Q'
∥ Q''[y::=x])) ∈ bangRel Rel
    by (force intro: Rel.BRPar BRRes)
  ultimately show ∃ P'. P ∥ R ⟶ τ < P' ∧ (P', < νx > (Q' ∥ Q''[y::=x]))
∈ bangRel Rel by blast
  qed
  qed
  next
  case (cBang Rs Pa P)
  hence IH: ∧ Pa. (Pa, Q ∥ !Q) ∈ bangRel Rel ⟹ ?Sim Pa Rs by simp
  have (Pa, !Q) ∈ bangRel Rel by fact
  thus ?case
  proof (induct rule: BRBangCases)
  case (BRBang P)
  have PRelQ: (P, Q) ∈ Rel by fact
  hence (!P, !Q) ∈ bangRel Rel by (rule Rel.BRBang)
  with PRelQ have (P ∥ !P, Q ∥ !Q) ∈ bangRel Rel by (rule BRPar)
  with IH have ?Sim (P ∥ !P) Rs by simp
  thus ?case by (force intro: Bang)
  qed
  qed
}

  moreover from PRelQ have (!P, !Q) ∈ bangRel Rel by (rule BRBang)
  ultimately show ?thesis by (auto simp add: simulation-def)
qed

end

theory Strong-Late-Bisim-Pres
  imports Strong-Late-Bisim Strong-Late-Sim-Pres
begin

lemma tauPres:
  fixes P :: pi
  and Q :: pi

  assumes P ~ Q

  shows τ.(P) ~ τ.(Q)
proof -
  let ?X = {(τ.(P), τ.(Q)), (τ.(Q), τ.(P))}
  have (τ.(P), τ.(Q)) ∈ ?X by auto
  thus ?thesis using ⟨P ~ Q⟩

```

```

    by(coinduct rule: bisimCoinduct)
      (auto intro: Strong-Late-Sim-Pres.tauPres dest: symmetric)
  qed

lemma inputPres:
  fixes P :: pi
  and Q :: pi
  and a :: name
  and x :: name

  assumes PSimQ:  $\forall y. P[x::=y] \sim Q[x::=y]$ 

  shows  $a\langle x \rangle.P \sim a\langle x \rangle.Q$ 
proof -
  let ?X = {(a<x>.P, a<x>.Q) | a x P Q.  $\forall y. P[x::=y] \sim Q[x::=y]}$ 
  {
    fix axP axQ p
    assume (axP, axQ) ∈ ?X
    then obtain a x P Q where A:  $\forall y. P[x::=y] \sim Q[x::=y]$  and B:  $axP = a\langle x \rangle.P$  and C:  $axQ = a\langle x \rangle.Q$ 
    by auto
    have  $\bigwedge y. ((p::name prm) \cdot P)[(p \cdot x)::=y] \sim (p \cdot Q)[(p \cdot x)::=y]$ 
    proof -
      fix y
      from A have  $P[x::=(rev\ p \cdot y)] \sim Q[x::=(rev\ p \cdot y)]$ 
      by blast
      hence  $(p \cdot (P[x::=(rev\ p \cdot y)])) \sim p \cdot (Q[x::=(rev\ p \cdot y)])$ 
      by(rule bisimClosed)
      thus  $(p \cdot P)[(p \cdot x)::=y] \sim (p \cdot Q)[(p \cdot x)::=y]$ 
      by(simp add: eqvts pt-pi-rev[OF pt-name-inst, OF at-name-inst])
    qed
    hence  $((p::name prm) \cdot axP, p \cdot axQ) \in ?X$  using B C
    by auto
  }
  hence eqvt ?X by(simp add: eqvt-def)

  from PSimQ have  $(a\langle x \rangle.P, a\langle x \rangle.Q) \in ?X$  by auto
  thus ?thesis
proof(coinduct rule: bisimCoinduct)
  case(cSim P Q)
  thus ?case using <eqvt ?X>
  by(force intro: inputPres)
next
  case(cSym P Q)
  thus ?case
  by(blast dest: symmetric)
qed
qed

```

```

lemma outputPres:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $a :: name$ 
  and  $b :: name$ 

  assumes  $P \sim Q$ 

  shows  $a\{b\}.P \sim a\{b\}.Q$ 
proof –
  let  $?X = \{(a\{b\}.P, a\{b\}.Q), (a\{b\}.Q, a\{b\}.P)\}$ 
  have  $(a\{b\}.P, a\{b\}.Q) \in ?X$  by auto
  thus ?thesis using  $\langle P \sim Q \rangle$ 
    by(coinduct rule: bisimCoinduct)
    (auto intro: Strong-Late-Sim-Pres.outputPres dest: symmetric)
qed

lemma matchPres:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $a :: name$ 
  and  $b :: name$ 

  assumes  $P \sim Q$ 

  shows  $[a\frown b]P \sim [a\frown b]Q$ 
proof –
  let  $?X = \{([a\frown b]P, [a\frown b]Q), ([a\frown b]Q, [a\frown b]P)\}$ 
  have  $([a\frown b]P, [a\frown b]Q) \in ?X$  by auto
  thus ?thesis using  $\langle P \sim Q \rangle$ 
    by(coinduct rule: bisimCoinduct)
    (auto intro: Strong-Late-Sim-Pres.matchPres dest: symmetric bisimE)
qed

lemma mismatchPres:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $a :: name$ 
  and  $b :: name$ 

  assumes  $P \sim Q$ 

  shows  $[a\neq b]P \sim [a\neq b]Q$ 
proof –
  let  $?X = \{([a\neq b]P, [a\neq b]Q), ([a\neq b]Q, [a\neq b]P)\}$ 
  have  $([a\neq b]P, [a\neq b]Q) \in ?X$  by auto
  thus ?thesis using  $\langle P \sim Q \rangle$ 
    by(coinduct rule: bisimCoinduct)
    (auto intro: Strong-Late-Sim-Pres.mismatchPres dest: symmetric bisimE)

```

qed

lemma *sumPres*:

fixes $P :: pi$

and $Q :: pi$

and $R :: pi$

assumes $P \sim Q$

shows $P \oplus R \sim Q \oplus R$

proof –

let $?X = \{(P \oplus R, Q \oplus R), (Q \oplus R, P \oplus R)\}$

have $(P \oplus R, Q \oplus R) \in ?X$ by *auto*

thus *?thesis* using $\langle P \sim Q \rangle$

by(*coinduct rule: bisimCoinduct*)

(*auto intro: Strong-Late-Sim-Pres.sumPres reflexive dest: symmetric bisimE*)

qed

lemma *resPres*:

fixes $P :: pi$

and $Q :: pi$

and $x :: name$

assumes $P \sim Q$

shows $\langle \nu x \rangle P \sim \langle \nu x \rangle Q$

proof –

let $?X = \{x. \exists P Q. P \sim Q \wedge (\exists a. x = (\langle \nu a \rangle P, \langle \nu a \rangle Q))\}$

from $\langle P \sim Q \rangle$ have $(\langle \nu x \rangle P, \langle \nu x \rangle Q) \in ?X$ by *blast*

thus *?thesis*

proof(*coinduct rule: bisimCoinduct*)

case(*cSim xP xQ*)

{

fix $P Q a$

assume $PSimQ: P \rightsquigarrow[bisim] Q$

moreover have $\bigwedge P Q a. P \sim Q \implies (\langle \nu a \rangle P, \langle \nu a \rangle Q) \in ?X \cup bisim$ by

blast

moreover have $bisim \subseteq ?X \cup bisim$ by *blast*

moreover have *eqvt bisim* by *simp*

moreover have *eqvt ?X*

by(*auto simp add: eqvt-def*) (*blast intro: bisimClosed*)

hence *eqvt (?X \cup bisim)* by *auto*

ultimately have $\langle \nu a \rangle P \rightsquigarrow[?X \cup bisim] \langle \nu a \rangle Q$

by(*rule Strong-Late-Sim-Pres.resPres*)

}

with $\langle (xP, xQ) \in ?X \rangle$ show *?case*

by(*auto dest: bisimE*)

next

case(*cSym xP xQ*)

```

    thus ?case by(auto dest: symmetric)
  qed
qed

lemma parPres:
  fixes P :: pi
  and Q :: pi
  and R :: pi

  assumes P ~ Q

  shows P || R ~ Q || R
proof -
  let ?X = {(resChain lst (P || R), resChain lst (Q || R)) | lst P R Q. P ~ Q}
  have EmptyChain:  $\bigwedge P Q. P || Q = resChain [] (P || Q)$  by auto
  with <P ~ Q> have (P || R, Q || R) ∈ ?X by blast
  thus ?thesis
proof(coinduct rule: bisimCoinduct)
  case(cSim PR QR)
  {
    fix P Q R lst

    assume P ~ Q

    hence P  $\rightsquigarrow$ [bisim] Q by(rule bisimE)
    moreover note <P ~ Q>
    moreover have  $\bigwedge P Q R. P \sim Q \implies (P || R, Q || R) \in ?X$ 
      by auto (blast intro: EmptyChain)
    moreover
    {
      fix xP xQ x
      assume (xP, xQ) ∈ ?X
      then obtain P Q R lst
        where P ~ Q and xP = resChain lst (P || R) and xQeq: xQ = resChain
lst (Q || R)
      by auto
      moreover hence (resChain (x#lst) (P || R), resChain (x#lst) (Q || R)) ∈
?X
      by blast
      ultimately have (<νx>xP, <νx>xQ) ∈ ?X by auto
    }
    note ResPres = this
    moreover have eqvt bisim by simp
    moreover have eqvt ?X
      by(auto simp add: eqvt-def) (blast intro: bisimClosed)
    ultimately have P || R  $\rightsquigarrow$ [?X] Q || R by(rule parPres)
    hence resChain lst (P || R)  $\rightsquigarrow$ [?X] (resChain lst (Q || R)) using <eqvt ?X>
ResPres
      by(rule resChainI)
  }

```

```

    hence resChain lst (P || R)  $\rightsquigarrow$ [(?X  $\cup$  bisim)] (resChain lst (Q || R))
    by(force intro: Strong-Late-Sim.monotonic)
  }
  with  $\langle (PR, QR) \in ?X \rangle$  show ?case
  by auto
next
  case(cSym PR QR)
  thus ?case by(blast dest: symmetric)
qed
qed

```

lemma bangPres:

```

  fixes P :: pi
  and Q :: pi

```

```

  assumes PBiSimQ: P  $\sim$  Q

```

```

  shows !P  $\sim$  !Q

```

proof –

```

  let ?X = bangRel bisim

```

```

  from PBiSimQ have (!P, !Q)  $\in$  ?X by(rule Rel.BRBang)

```

```

  thus ?thesis

```

```

  proof(coinduct rule: bisimCoinduct)

```

```

    case(cSim bP bQ)

```

```

    {

```

```

      fix P Q

```

```

      assume (P, Q)  $\in$  ?X

```

```

      hence P  $\rightsquigarrow$ [?X] Q

```

```

      proof(induct)

```

```

        fix P Q

```

```

        assume P  $\sim$  Q

```

```

        thus !P  $\rightsquigarrow$ [?X] !Q using bisimE bisimEqvt

```

```

          by(rule Strong-Late-Sim-Pres.bangPres)

```

```

      next

```

```

        fix P Q R T

```

```

        assume RBiSimT: R  $\sim$  T

```

```

        assume PBangRelQ: (P, Q)  $\in$  ?X

```

```

        assume PSimQ: P  $\rightsquigarrow$ [?X] Q

```

```

        from RBiSimT have R  $\rightsquigarrow$ [bisim] T by(blast dest: bisimE)

```

```

        thus R || P  $\rightsquigarrow$ [?X] T || Q using PSimQ RBiSimT PBangRelQ Rel.BRPar

```

```

        Rel.BRRes bisimEqvt eqvtBangRel

```

```

        by(blast intro: Strong-Late-Sim-Pres.parCompose)

```

```

      next

```

```

        fix P Q a

```

```

        assume P  $\rightsquigarrow$ [?X] Q

```

```

        moreover from eqvtBangRel bisimEqvt have eqvt ?X by blast

```

```

        ultimately show  $\langle \nu a \rangle P \rightsquigarrow$ [?X]  $\langle \nu a \rangle Q$  using Rel.BRRes by(blast intro:

```

```

        Strong-Late-Sim-Pres.resPres)

```

```

      qed
      hence  $P \rightsquigarrow [((\text{bangRel } \text{bisim}) \cup \text{bisim})] Q$  by(rule-tac Strong-Late-Sim.monotonic)
    auto
  }
  with  $\langle (bP, bQ) \in ?X \rangle$  show ?case by auto
next
  case(cSym bP bQ)
  thus ?case by(metis bangRelSymetric symmetric)
qed
qed
end

```

```

theory Strong-Late-Bisim-Subst-Pres
  imports Strong-Late-Bisim-Subst Strong-Late-Bisim-Pres
begin

```

```

lemma tauPres:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 

  assumes  $P \sim^s Q$ 

  shows  $\tau.(P) \sim^s \tau.(Q)$ 
using assms
by(force simp add: substClosed-def intro: Strong-Late-Bisim-Pres.tauPres)

```

```

lemma inputPres:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $a :: name$ 
  and  $x :: name$ 

  assumes  $P \sim^s Q$ 

  shows  $a\langle x \rangle.P \sim^s a\langle x \rangle.Q$ 
proof(auto simp add: substClosed-def)
  fix  $\sigma :: (name \times name) \text{ list}$ 
  {
    fix  $P Q a x \sigma$ 
    assume  $P \sim^s Q$ 
    then have  $P[\langle \sigma \rangle] \sim^s Q[\langle \sigma \rangle]$  by(rule partUnfold)
    then have  $\forall y. (P[\langle \sigma \rangle])[x::=y] \sim (Q[\langle \sigma \rangle])[x::=y]$ 
      apply(auto simp add: substClosed-def)
      by(erule-tac x=[(x, y)] in allE) auto
    moreover assume  $x \# \sigma$ 
    ultimately have  $(a\langle x \rangle.P)[\langle \sigma \rangle] \sim (a\langle x \rangle.Q)[\langle \sigma \rangle]$  using bisimEqvt
      by(force intro: Strong-Late-Bisim-Pres.inputPres)
  }

```

```

note Goal = this

obtain y::name where y # P and y # Q and y # σ
  by(generate-fresh name) auto
from  $\langle P \sim^s Q \rangle$  have  $([(x, y)] \cdot P) \sim^s ([(x, y)] \cdot Q)$  by(rule eqClosed)
hence  $(a\langle y \rangle.([(x, y)] \cdot P))[\langle \sigma \rangle] \sim (a\langle y \rangle.([(x, y)] \cdot Q))[\langle \sigma \rangle]$  using  $\langle y \# \sigma \rangle$ 
by(rule Goal)
  moreover from  $\langle y \# P \rangle$   $\langle y \# Q \rangle$  have  $a\langle x \rangle.P = a\langle y \rangle.([(x, y)] \cdot P)$  and
 $a\langle x \rangle.Q = a\langle y \rangle.([(x, y)] \cdot Q)$ 
  by(simp add: alphaInput)+

  ultimately show  $(a\langle x \rangle.P)[\langle \sigma \rangle] \sim (a\langle x \rangle.Q)[\langle \sigma \rangle]$  by simp
qed

lemma outputPres:
  fixes P :: pi
  and Q :: pi

  assumes  $P \sim^s Q$ 

  shows  $a\{b\}.P \sim^s a\{b\}.Q$ 
using assms
by(force simp add: substClosed-def intro: Strong-Late-Bisim-Pres.outputPres)

lemma matchPres:
  fixes P :: pi
  and Q :: pi
  and a :: name
  and b :: name

  assumes  $P \sim^s Q$ 

  shows  $[a \frown b]P \sim^s [a \frown b]Q$ 
using assms
by(force simp add: substClosed-def intro: Strong-Late-Bisim-Pres.matchPres)

lemma mismatchPres:
  fixes P :: pi
  and Q :: pi
  and a :: name
  and b :: name

  assumes  $P \sim^s Q$ 

  shows  $[a \neq b]P \sim^s [a \neq b]Q$ 
using assms
by(force simp add: substClosed-def intro: Strong-Late-Bisim-Pres.mismatchPres)

lemma sumPres:

```

```

fixes  $P :: pi$ 
and  $Q :: pi$ 
and  $R :: pi$ 

assumes  $P \sim^s Q$ 

shows  $P \oplus R \sim^s Q \oplus R$ 
using assms
by(force simp add: substClosed-def intro: Strong-Late-Bisim-Pres.sumPres)

lemma parPres:
fixes  $P :: pi$ 
and  $Q :: pi$ 
and  $R :: pi$ 

assumes  $P \sim^s Q$ 

shows  $P \parallel R \sim^s Q \parallel R$ 
using assms
by(force simp add: substClosed-def intro: Strong-Late-Bisim-Pres.parPres)

lemma resPres:
fixes  $P :: pi$ 
and  $Q :: pi$ 
and  $x :: name$ 

assumes PeqQ:  $P \sim^s Q$ 

shows  $\langle \nu x \rangle P \sim^s \langle \nu x \rangle Q$ 
proof(auto simp add: substClosed-def)
fix  $s :: (name \times name) list$ 

have Res:  $\bigwedge P Q x s. \llbracket P[\langle s \rangle] \sim Q[\langle s \rangle]; x \# s \rrbracket \implies (\langle \nu x \rangle P)[\langle s \rangle] \sim (\langle \nu x \rangle Q)[\langle s \rangle]$ 
by(force intro: Strong-Late-Bisim-Pres.resPres)

have  $\exists c :: name. c \# (P, Q, s)$  by(blast intro: name-exists-fresh)
then obtain  $c :: name$  where cFreshP:  $c \# P$  and cFreshQ:  $c \# Q$  and cFreshs:
 $c \# s$ 
by(force simp add: fresh-prod)

from PeqQ have  $P[\langle ((x, c) \cdot s) \rangle] \sim Q[\langle ((x, c) \cdot s) \rangle]$  by(simp add: substClosed-def)
hence  $((x, c) \cdot P[\langle ((x, c) \cdot s) \rangle]) \sim ((x, c) \cdot Q[\langle ((x, c) \cdot s) \rangle])$  by(rule bisimClosed)
hence  $((x, c) \cdot P)[\langle s \rangle] \sim ((x, c) \cdot Q)[\langle s \rangle]$  by simp
hence  $\langle \nu c \rangle(((x, c) \cdot P))[\langle s \rangle] \sim \langle \nu c \rangle(((x, c) \cdot Q))[\langle s \rangle]$  using cFreshs
by(rule Res)

moreover from cFreshP cFreshQ have  $\langle \nu x \rangle P = \langle \nu c \rangle(((x, c) \cdot P)$  and

```

```

<νx>Q = <νc>([(x, c)] · Q)
  by(simp add: alphaRes)+

  ultimately show (<νx>P)[<s>] ~ (<νx>Q)[<s>] by simp
qed

lemma bangPres:
  fixes P :: pi
  and Q :: pi

  assumes P ~s Q

  shows !P ~s !Q
using assms
by(force simp add: substClosed-def intro: Strong-Late-Bisim-Pres.bangPres)

end

theory Late-Tau-Chain
  imports Late-Semantics1
begin

abbreviation tauChain-judge :: pi ⇒ pi ⇒ bool (λ<- ⇒τ > [80, 80] 80)
where P ⇒τ P' ≡ (P, P') ∈ {(P, P') | P P'. P ↦τ < P'}∧*

lemma singleTauChain:
  fixes P :: pi
  and P' :: pi

  assumes P ↦τ < P'

  shows P ⇒τ P'
using assms by(simp add: r-into-rtrancl)

lemma tauChainAddTau[dest]:
  fixes P :: pi
  and P' :: pi
  and P'' :: pi

  shows P ⇒τ P' ⇒ P' ↦τ < P'' ⇒ P ⇒τ P''
  and P ↦τ < P' ⇒ P' ⇒τ P'' ⇒ P ⇒τ P''
by(auto dest: singleTauChain)

lemma tauChainInduct[consumes 1, case-names id ih]:
  fixes P :: pi
  and P' :: pi

  assumes P ⇒τ P'
  and F P

```

and $\bigwedge P' P''. \llbracket P \Longrightarrow_{\tau} P'; P' \longmapsto_{\tau} \prec P''; F P' \rrbracket \Longrightarrow F P''$

shows $F P'$

using *assms*

by(*drule-tac rtrancl-induct*) *auto*

lemma *eqvtChainI*[*eqvt*]:

fixes $P \quad :: pi$

and $P' \quad :: pi$

and $perm \quad :: name prm$

assumes $P \Longrightarrow_{\tau} P'$

shows $(perm \cdot P) \Longrightarrow_{\tau} (perm \cdot P')$

using *assms*

proof(*induct rule: tauChainInduct*)

case *id*

thus *?case* **by** *simp*

next

case(*ih P'' P'''*)

have $P \Longrightarrow_{\tau} P''$ **and** $P'' \longmapsto_{\tau} \prec P'''$ **by** *fact+*

hence $(perm \cdot P'') \longmapsto_{\tau} \prec (perm \cdot P''')$ **by**(*force dest: transitions.eqvt*)

moreover **have** $(perm \cdot P) \Longrightarrow_{\tau} (perm \cdot P'')$ **by** *fact*

ultimately show *?case* **by** *auto*

qed

lemma *eqvtChainE*:

fixes $perm \quad :: name prm$

and $P \quad :: pi$

and $P' \quad :: pi$

assumes *Trans*: $(perm \cdot P) \Longrightarrow_{\tau} (perm \cdot P')$

shows $P \Longrightarrow_{\tau} P'$

proof –

have $rev perm \cdot (perm \cdot P) = P$ **by**(*simp add: pt-rev-pi[OF pt-name-inst, OF at-name-inst]*)

moreover **have** $rev perm \cdot (perm \cdot P') = P'$ **by**(*simp add: pt-rev-pi[OF pt-name-inst, OF at-name-inst]*)

ultimately show *?thesis* **using** *assms*

by(*drule-tac perm=rev perm in eqvtChainI, simp*)

qed

lemma *eqvtChainEq*:

fixes $P \quad :: pi$

and $P' \quad :: pi$

and $perm \quad :: name prm$

shows $P \Longrightarrow_{\tau} P' = (\text{perm} \cdot P) \Longrightarrow_{\tau} (\text{perm} \cdot P')$
by(blast intro: eqvtChainE eqvtChainI)

lemma *freshChain*:

fixes $P :: pi$
and $P' :: pi$
and $x :: name$

assumes $P \Longrightarrow_{\tau} P'$
and $x \# P$

shows $x \# P'$

using *assms*

proof(induct rule: tauChainInduct)

case *id*

thus ?case **by** *simp*

next

case(ih $P' P''$)

have $x \# P$ **and** $x \# P \Longrightarrow x \# P'$ **by** *fact+*

hence $x \# P'$ **by** *simp*

moreover **have** $P' \mapsto_{\tau} P''$ **by** *fact*

ultimately show ?case **by**(force intro: freshFreeDerivative)

qed

lemma *matchChain*:

fixes $b :: name$
and $P :: pi$
and $P' :: pi$

assumes $P \Longrightarrow_{\tau} P'$
and $P \neq P'$

shows $[b \frown b]P \Longrightarrow_{\tau} P'$

using *assms*

proof(induct rule: tauChainInduct)

case *id*

thus ?case **by** *simp*

next

case(ih $P'' P'''$)

have $P'' \text{Trans} P'''$: $P'' \mapsto_{\tau} P'''$ **by** *fact*

show $[b \frown b]P \Longrightarrow_{\tau} P'''$

proof(cases $P = P''$)

assume $P = P''$

moreover with $P'' \text{Trans} P'''$ **have** $[b \frown b]P \mapsto_{\tau} P'''$ **by**(force intro: Match)

thus $[b \frown b]P \Longrightarrow_{\tau} P'''$ **by**(rule singleTauChain)

next

assume $P \neq P''$

moreover **have** $P \neq P'' \Longrightarrow [b \frown b]P \Longrightarrow_{\tau} P''$ **by** *fact*

ultimately show $[b \frown b]P \Longrightarrow_{\tau} P'''$ **using** $P'' \text{Trans} P'''$ **by**(blast)

qed
qed

lemma *mismatchChain*:

fixes $a :: name$
and $b :: name$
and $P :: pi$
and $P' :: pi$

assumes $PChain: P \Rightarrow_{\tau} P'$
and $aineqb: a \neq b$
and $PineqP': P \neq P'$

shows $[a \neq b]P \Rightarrow_{\tau} P'$
using $PChain PineqP'$
proof(*induct rule: tauChainInduct*)
 case *id*
 thus ?*case* by *simp*
next
 case(*ih* $P'' P'''$)
 have $P''TransP''': P'' \mapsto_{\tau} < P'''$ by *fact*
 show $[a \neq b]P \Rightarrow_{\tau} P'''$
 proof(*cases* $P = P''$)
 assume $P = P''$
 moreover with $aineqb P''TransP'''$ have $[a \neq b]P \mapsto_{\tau} < P'''$ by(*force intro: Mismatch*)
 thus $[a \neq b]P \Rightarrow_{\tau} P'''$ by(*rule singleTauChain*)
 next
 assume $P \neq P''$
 moreover have $P \neq P'' \Rightarrow [a \neq b]P \Rightarrow_{\tau} P''$ by *fact+*
 ultimately show $[a \neq b]P \Rightarrow_{\tau} P'''$ using $P''TransP'''$ by(*blast*)
qed
qed

lemma *sum1Chain*[*rule-format*]:

fixes $P :: pi$
and $P' :: pi$
and $Q :: pi$

assumes $P \Rightarrow_{\tau} P'$
and $P \neq P'$

shows $P \oplus Q \Rightarrow_{\tau} P'$
using *assms*
proof(*induct rule: tauChainInduct*)
 case *id*
 thus ?*case* by *simp*
next
 case(*ih* $P'' P'''$)

```

have P''TransP''': P''  $\mapsto_{\tau}$  < P''' by fact
show P  $\oplus$  Q  $\implies_{\tau}$  P'''
proof(cases P = P'')
  assume P=P''
  moreover with P''TransP''' have P  $\oplus$  Q  $\mapsto_{\tau}$  < P''' by(force intro: Sum1)
  thus P  $\oplus$  Q  $\implies_{\tau}$  P''' by(force intro: singleTauChain)
next
  assume P  $\neq$  P''
  moreover have P  $\neq$  P''  $\implies$  P  $\oplus$  Q  $\implies_{\tau}$  P'' by fact
  ultimately show P  $\oplus$  Q  $\implies_{\tau}$  P''' using P''TransP''' by(force)
qed
qed

```

lemma *sum2Chain*[rule-format]:

```

fixes P :: pi
and Q :: pi
and Q' :: pi

assumes Q  $\implies_{\tau}$  Q'
and Q  $\neq$  Q'

shows P  $\oplus$  Q  $\implies_{\tau}$  Q'
using assms
proof(induct rule: tauChainInduct)
  case id
  thus ?case by simp
next
  case(ih Q'' Q''')
  have Q''TransQ''': Q''  $\mapsto_{\tau}$  < Q''' by fact
  show P  $\oplus$  Q  $\implies_{\tau}$  Q'''
  proof(cases Q = Q'')
    assume Q=Q''
    moreover with Q''TransQ''' have P  $\oplus$  Q  $\mapsto_{\tau}$  < Q''' by(force intro: Sum2)
    thus P  $\oplus$  Q  $\implies_{\tau}$  Q''' by(force intro: singleTauChain)
  next
    assume Q  $\neq$  Q''
    moreover have Q  $\neq$  Q''  $\implies$  P  $\oplus$  Q  $\implies_{\tau}$  Q'' by fact
    ultimately show P  $\oplus$  Q  $\implies_{\tau}$  Q''' using Q''TransQ''' by blast
  qed
qed
qed

```

lemma *Par1Chain*:

```

fixes P :: pi
and P' :: pi
and Q :: pi

assumes P  $\implies_{\tau}$  P'

```

```

  shows  $P \parallel Q \Longrightarrow_{\tau} P' \parallel Q$ 
using assms
proof(induct rule: tauChainInduct)
  case id
  thus ?case by simp
next
  case(ih  $P'' P'$ )
  have  $P'' \text{Trans} P'$ :  $P'' \longmapsto_{\tau} \prec P'$  by fact
  have IH:  $P \parallel Q \Longrightarrow_{\tau} P'' \parallel Q$  by fact

  have  $P'' \parallel Q \longmapsto_{\tau} \prec P' \parallel Q$  using  $P'' \text{Trans} P'$  by(force intro: Par1F)
  thus  $P \parallel Q \Longrightarrow_{\tau} P' \parallel Q$  using IH by(force)
qed

```

lemma *Par2Chain*:

```

fixes  $P :: pi$ 
and  $Q :: pi$ 
and  $Q' :: pi$ 

```

```

assumes  $Q \Longrightarrow_{\tau} Q'$ 

```

```

  shows  $P \parallel Q \Longrightarrow_{\tau} P \parallel Q'$ 
using assms
proof(induct rule: tauChainInduct)
  case id
  thus ?case by simp
next
  case(ih  $Q'' Q'$ )
  have  $Q'' \text{Trans} Q'$ :  $Q'' \longmapsto_{\tau} \prec Q'$  by fact
  have IH:  $P \parallel Q \Longrightarrow_{\tau} P \parallel Q''$  by fact

  have  $P \parallel Q'' \longmapsto_{\tau} \prec P \parallel Q'$  using  $Q'' \text{Trans} Q'$  by(force intro: Par2F)
  thus  $P \parallel Q \Longrightarrow_{\tau} P \parallel Q'$  using IH by(force)
qed

```

lemma *chainPar*:

```

fixes  $P :: pi$ 
and  $P' :: pi$ 
and  $Q :: pi$ 
and  $Q' :: pi$ 

```

```

assumes  $P \Longrightarrow_{\tau} P'$ 
and  $Q \Longrightarrow_{\tau} Q'$ 

```

```

  shows  $P \parallel Q \Longrightarrow_{\tau} P' \parallel Q'$ 
proof -
  from  $\langle P \Longrightarrow_{\tau} P' \rangle$  have  $P \parallel Q \Longrightarrow_{\tau} P' \parallel Q$  by(rule Par1Chain)
  moreover from  $\langle Q \Longrightarrow_{\tau} Q' \rangle$  have  $P' \parallel Q \Longrightarrow_{\tau} P' \parallel Q'$  by(rule Par2Chain)
  ultimately show ?thesis by auto

```

qed

lemma *ResChain*:

fixes $P :: pi$
and $P' :: pi$
and $a :: name$

assumes $P \Longrightarrow_{\tau} P'$

shows $\langle \nu a \rangle P \Longrightarrow_{\tau} \langle \nu a \rangle P'$

using *assms*

proof(*induct rule: tauChainInduct*)

case *id*

thus *?case* **by** *simp*

next

case(*ih P'' P'''*)

have $P'' \mapsto_{\tau} \prec P'''$ **by** *fact*

hence $\langle \nu a \rangle P'' \mapsto_{\tau} \prec \langle \nu a \rangle P'''$ **by**(*force intro: ResF*)

moreover **have** $\langle \nu a \rangle P \Longrightarrow_{\tau} \langle \nu a \rangle P''$ **by** *fact*

ultimately show *?case* **by** *force*

qed

lemma *substChain*:

fixes $P :: pi$
and $x :: name$
and $b :: name$
and $P' :: pi$

assumes $PTrans: P[x::=b] \Longrightarrow_{\tau} P'$

shows $P[x::=b] \Longrightarrow_{\tau} P'[x::=b]$

proof(*cases x=b*)

assume $x = b$

with $PTrans$ **show** *?thesis* **by** *simp*

next

assume $x \neq b$

hence $x \# P[x::=b]$ **by**(*simp add: fresh-fact2*)

with $PTrans$ **have** $x \# P'$ **by**(*force intro: freshChain*)

hence $P' = P'[x::=b]$ **by**(*simp add: forget*)

with $PTrans$ **show** *?thesis* **by** *simp*

qed

lemma *bangChain*:

fixes $P :: pi$
and $P' :: pi$

assumes $PTrans: P \parallel !P \Longrightarrow_{\tau} P'$

and $P'ineq: P' \neq P \parallel !P$

```

  shows  $!P \Rightarrow_{\tau} P'$ 
using assms
proof(induct rule: tauChainInduct)
  case id
  thus ?case by simp
next
  case(ih P' P'')
  show ?case
  proof(cases P' = P || !P)
    case True
    from  $\langle P' \mapsto_{\tau} \prec P'' \rangle \langle P' = P || !P \rangle$  have  $!P \mapsto_{\tau} \prec P''$  by(blast intro: Bang)
    thus ?thesis by auto
  next
    case False
    from  $\langle P' \neq P || !P \rangle$  have  $!P \Rightarrow_{\tau} P'$  by(rule ih)
    with  $\langle P' \mapsto_{\tau} \prec P'' \rangle$  show ?thesis by auto
  qed
qed
end

```

```

theory Weak-Late-Step-Semantics
  imports Late-Tau-Chain
begin

```

```

definition inputTransition ::  $pi \Rightarrow name \Rightarrow pi \Rightarrow name \Rightarrow name \Rightarrow pi \Rightarrow bool$  ( $\prec$ -  

 $\Rightarrow_l$ - in  $\rightarrow$ - $\prec$ - $\rightarrow$   $\prec$   $\rightarrow$  [80, 80, 80, 80, 80] 80)
where  $P \Rightarrow_l u$  in  $P'' \rightarrow a \langle x \rangle \prec P' \equiv \exists P''' . P \Rightarrow_{\tau} P''' \wedge P''' \mapsto a \langle x \rangle \prec P''$   

 $\wedge P''[x ::= u] \Rightarrow_{\tau} P'$ 

```

```

definition transition ::  $(pi \times LateSemantics.residual)$  set where
  transition  $\equiv \{x . \exists P P' \alpha P'' P''' . P \Rightarrow_{\tau} P' \wedge P' \mapsto_{\alpha} \prec P'' \wedge P'' \Rightarrow_{\tau} P'''$   

 $\wedge x = (P, \alpha \prec P''')\} \cup$   

 $\{x . \exists P P' a y P'' P''' . P \Rightarrow_{\tau} P' \wedge (P' \mapsto (a \langle \nu y \rangle \prec P'')) \wedge P''$   

 $\Rightarrow_{\tau} P''' \wedge x = (P, (a \langle \nu y \rangle \prec P'''))\}$ 

```

```

abbreviation weakTransition-judge ::  $pi \Rightarrow LateSemantics.residual \Rightarrow bool$  ( $\prec$ -  

 $\Rightarrow_l$   $\rightarrow$  [80, 80] 80)
where  $P \Rightarrow_l Rs \equiv (P, Rs) \in transition$ 

```

```

lemma weakNonInput[dest]:

```

```

  fixes  $P :: pi$ 
  and  $a :: name$ 
  and  $x :: name$ 
  and  $P' :: pi$ 

```

```

  assumes  $P \Rightarrow_l a \langle x \rangle \prec P'$ 

```

```

  shows False

```

using *assms*
 by(*auto simp add: transition-def residual.inject*)

lemma *transitionI*:

fixes $P :: pi$
 and $P''' :: pi$
 and $\alpha :: freeRes$
 and $P'' :: pi$
 and $P' :: pi$
 and $a :: name$
 and $x :: name$
 and $u :: name$

shows $\llbracket P \Rightarrow_{\tau} P'''; P''' \mapsto_{\alpha} \prec P''; P'' \Rightarrow_{\tau} P' \rrbracket \Rightarrow P \Rightarrow_l \alpha \prec P'$
 and $\llbracket P \Rightarrow_{\tau} P'''; P''' \mapsto_{a\langle \nu x \rangle} \prec P''; P'' \Rightarrow_{\tau} P' \rrbracket \Rightarrow P \Rightarrow_l a\langle \nu x \rangle \prec P'$
 and $\llbracket P \Rightarrow_{\tau} P'''; P''' \mapsto_{a\langle x \rangle} \prec P''; P''[x::=u] \Rightarrow_{\tau} P' \rrbracket \Rightarrow P \Rightarrow_l u \text{ in } P'' \rightarrow a\langle x \rangle \prec P'$

proof -

assume $P \Rightarrow_{\tau} P'''$ and $P''' \mapsto_{\alpha} \prec P''$ and $P'' \Rightarrow_{\tau} P'$

thus $P \Rightarrow_l \alpha \prec P'$

by(*force simp add: transition-def*)

next

assume $P \Rightarrow_{\tau} P'''$ and $P''' \mapsto_{a\langle \nu x \rangle} \prec P''$ and $P'' \Rightarrow_{\tau} P'$

thus $P \Rightarrow_l a\langle \nu x \rangle \prec P'$

by(*force simp add: transition-def*)

next

assume $P \Rightarrow_{\tau} P'''$ and $P''' \mapsto_{a\langle x \rangle} \prec P''$ and $P''[x::=u] \Rightarrow_{\tau} P'$

thus $P \Rightarrow_l u \text{ in } P'' \rightarrow a\langle x \rangle \prec P'$

by(*force simp add: inputTransition-def*)

qed

lemma *transitionE*:

fixes $P :: pi$
 and $\alpha :: freeRes$
 and $P' :: pi$
 and $P'' :: pi$
 and $a :: name$
 and $u :: name$
 and $x :: name$

shows $P \Rightarrow_l \alpha \prec P' \Rightarrow \exists P'' P'''. P \Rightarrow_{\tau} P'' \wedge P'' \mapsto_{\alpha} \prec P''' \wedge P''' \Rightarrow_{\tau} P'$
 (*is - => ?thesis1*)

and $\llbracket P \Rightarrow_l a\langle \nu x \rangle \prec P'; x \# P \rrbracket \Rightarrow \exists P'' P'''. P \Rightarrow_{\tau} P'' \wedge P'' \mapsto_{a\langle \nu x \rangle} \prec P''' \wedge P''' \Rightarrow_{\tau} P'$

and $\llbracket P \Rightarrow_l u \text{ in } P'' \rightarrow a\langle x \rangle \prec P' \rrbracket \Rightarrow \exists P'''. P \Rightarrow_{\tau} P'' \wedge P'' \mapsto_{a\langle x \rangle} \prec P''' \wedge P'''[x::=u] \Rightarrow_{\tau} P'$

proof -

assume $P \Rightarrow_l \alpha \prec P'$

```

thus ?thesis1 by(auto simp add: transition-def residual.inject)
next
assume  $P \Longrightarrow_l a < \nu x > \prec P'$  and  $x \# P$ 
thus  $\exists P'' P''' . P \Longrightarrow_\tau P''' \wedge P''' \mapsto a < \nu x > \prec P'' \wedge P'' \Longrightarrow_\tau P'$ 
using [hypsubst-thin = true]
apply(auto simp add: transition-def residualInject name-abs-eq)
apply(rule-tac x=[(x, y)] \cdot P'' in exI)
apply(rule-tac x=P' in exI)
apply(clarsimp)
apply(auto)
apply(subgoal-tac x \# P'')
apply(simp add: alphaBoundResidual name-swap)
using freshChain
apply(force dest: freshBoundDerivative)
using eqvtChainI
by simp
next
assume  $PTrans: P \Longrightarrow_l u \text{ in } P'' \rightarrow a < x > \prec P'$ 
thus  $\exists P''' . P \Longrightarrow_\tau P''' \wedge P''' \mapsto a < x > \prec P'' \wedge P'''[x::=u] \Longrightarrow_\tau P'$ 
by(auto simp add: inputTransition-def)
qed

lemma alphaInput:
fixes  $P :: pi$ 
and  $u :: name$ 
and  $P'' :: pi$ 
and  $a :: name$ 
and  $x :: name$ 
and  $P' :: pi$ 
and  $y :: name$ 

assumes  $PTrans: P \Longrightarrow_l u \text{ in } P'' \rightarrow a < x > \prec P'$ 
and  $yFreshP: y \# P$ 

shows  $P \Longrightarrow_l u \text{ in } [(x, y)] \cdot P'' \rightarrow a < y > \prec P'$ 
proof(cases x=y)
assume  $x=y$ 
with  $PTrans$  show ?thesis by simp
next
assume  $x \neq y$ 
from  $PTrans$  obtain  $P'''$  where  $PChain: P \Longrightarrow_\tau P'''$ 
and  $P'''Trans: P''' \mapsto a < x > \prec P''$ 
and  $P''Chain: P''[x::=u] \Longrightarrow_\tau P'$ 
by(blast dest: transitionE)

from  $PChain$   $yFreshP$  have  $y \# P'''$  by(rule freshChain)
with  $P'''Trans$   $x \neq y$  have  $yFreshP'': y \# P''$  by(blast dest: freshBoundDerivative)

```

with $P''' \text{Trans}$ **have** $P''' \mapsto a \langle y \rangle \prec [(x, y)] \cdot P''$ **by** (*simp add: alphaBound-Residual*)
moreover from $P'' \text{Chain } y \text{Fresh} P''$ **have** $([(x, y)] \cdot P'')[y::=u] \Longrightarrow_{\tau} P'$
by (*simp add: renaming name-swap*)
ultimately show *?thesis* **using** $P \text{Chain}$ **by** (*blast intro: transitionI*)
qed

lemma *tauActionChain*:

fixes $P :: pi$
and $P' :: pi$

shows $P \Longrightarrow_{l\tau} \prec P' \Longrightarrow P \Longrightarrow_{\tau} P'$
and $P \neq P' \Longrightarrow P \Longrightarrow_{\tau} P' \Longrightarrow P \Longrightarrow_{l\tau} \prec P'$

proof –

assume $P \Longrightarrow_{l\tau} \prec P'$
then obtain $P'' P'''$ **where** $P \Longrightarrow_{\tau} P''$
and $P'' \mapsto_{\tau} \prec P'''$
and $P''' \Longrightarrow_{\tau} P'$

by (*blast dest: transitionE*)

thus $P \Longrightarrow_{\tau} P'$ **by** *auto*

next

assume $P \Longrightarrow_{\tau} P'$ **and** $P \neq P'$

thus $P \Longrightarrow_{l\tau} \prec P'$

proof (*induct rule: tauChainInduct*)

case *id*

thus *?case* **by** *simp*

next

case (*ih* $P'' P'''$)

have $P \Longrightarrow_{\tau} P''$ **and** $P'' \mapsto_{\tau} \prec P'''$ **by** *fact+*

moreover have $P''' \Longrightarrow_{\tau} P'$ **by** *simp*

ultimately show *?case* **by** (*rule transitionI*)

qed

qed

lemma *singleActionChain*:

fixes $P :: pi$
and $a :: name$
and $x :: name$
and $\alpha :: freeRes$
and $u :: name$

shows $P \mapsto a \langle \nu x \rangle \prec P' \Longrightarrow P \Longrightarrow_{l\alpha} \prec P'$

and $\llbracket P \mapsto a \langle x \rangle \prec P' \rrbracket \Longrightarrow P \Longrightarrow_{lu} \text{ in } P' \mapsto a \langle x \rangle \prec P'[x::=u]$

and $P \mapsto \alpha \prec P' \Longrightarrow P \Longrightarrow_{l\alpha} \prec P'$

proof –

assume $P \mapsto a \langle \nu x \rangle \prec P'$

moreover have $P \Longrightarrow_{\tau} P$ **by** *simp*

moreover have $P' \Longrightarrow_{\tau} P'$ **by** *simp*

ultimately show $P \Longrightarrow_{l\alpha} \prec P'$ **by** (*blast intro: transitionI*)

next
 assume $P \mapsto_a \langle x \rangle \prec P'$
 moreover have $P \Longrightarrow_\tau P$ **by** *simp*
 moreover have $P'[x::=u] \Longrightarrow_\tau P'[x::=u]$ **by** *simp*
 ultimately show $P \Longrightarrow_{\iota u} \text{in } P' \mapsto_a \langle x \rangle \prec P'[x::=u]$ **by**(*blast intro: transitionI*)
next
 assume $P \mapsto_\alpha \prec P'$
 moreover have $P \Longrightarrow_\tau P$ **by** *simp*
 moreover have $P' \Longrightarrow_\tau P'$ **by** *simp*
 ultimately show $P \Longrightarrow_{\iota \alpha} \prec P'$ **by**(*blast intro: transitionI*)
qed

lemma *Tau*:
 fixes $P :: pi$

 shows $\tau.(P) \Longrightarrow_{\iota} \tau \prec P$
proof –
 have $\tau.(P) \Longrightarrow_\tau \tau.(P)$ **by** *simp*
 moreover have $\tau.(P) \mapsto_\tau \prec P$ **by**(*rule transitions.Tau*)
 moreover have $P \Longrightarrow_\tau P$ **by** *simp*
 ultimately show *?thesis* **by**(*rule transitionI*)
qed

lemma *Input*:
 fixes $a :: name$
 and $x :: name$
 and $u :: name$
 and $P :: pi$

 shows $a \langle x \rangle . P \Longrightarrow_{\iota u} \text{in } P \mapsto_a \langle x \rangle \prec P[x::=u]$
proof –
 have $a \langle x \rangle . P \Longrightarrow_\tau a \langle x \rangle . P$ **by** *simp*
 moreover have $a \langle x \rangle . P \mapsto_a \langle x \rangle \prec P$ **by**(*rule Input*)
 moreover have $P[x::=u] \Longrightarrow_\tau P[x::=u]$ **by** *simp*
 ultimately show *?thesis* **by**(*rule transitionI*)
qed

lemma *Output*:
 fixes $a :: name$
 and $b :: name$
 and $P :: pi$

 shows $a\{b\}.P \Longrightarrow_{\iota} a[b] \prec P$
proof –
 have $a\{b\}.P \Longrightarrow_\tau a\{b\}.P$ **by** *simp*
 moreover have $a\{b\}.P \mapsto_a [b] \prec P$ **by**(*rule transitions.Output*)
 moreover have $P \Longrightarrow_\tau P$ **by** *simp*
 ultimately show *?thesis* **by**(*rule transitionI*)
qed

```

lemma Match:
  fixes P :: pi
  and Rs :: residual
  and a :: name
  and u :: name
  and b :: name
  and x :: name
  and P' :: pi

  shows P  $\Rightarrow_l$  Rs  $\Rightarrow$  [a $\frown$ a]P  $\Rightarrow_l$  Rs
  and P  $\Rightarrow_l$  u in P'' $\rightarrow$ b<x> < P'  $\Rightarrow$  [a $\frown$ a]P  $\Rightarrow_l$  u in P'' $\rightarrow$ b<x> < P'
proof -
  assume PTrans: P  $\Rightarrow_l$  Rs
  thus [a $\frown$ a]P  $\Rightarrow_l$  Rs
  proof (nominal-induct avoiding: P rule: residual.strong-inducts)
    case (BoundR b x P')
    have PTrans: P  $\Rightarrow_l$  b<x> < P' and xFreshP: x  $\#$  P by fact+
    from PTrans obtain b' where beq: b = BoundOutputS b' by (cases b) auto
    with PTrans xFreshP obtain P'' P''' where PTrans: P  $\Rightarrow_\tau$  P''
      and P''Trans: P''  $\mapsto$  b'< $\nu$ x> < P'''
      and P'''Trans: P'''  $\Rightarrow_\tau$  P'

    by (blast dest: transitionE)
    show ?case
    proof (cases P = P'')
      assume P = P''
      moreover have [a $\frown$ a]P  $\Rightarrow_\tau$  [a $\frown$ a]P by simp
      moreover from P''Trans have [a $\frown$ a]P''  $\mapsto$  b'< $\nu$ x> < P''' by (rule Match)
      ultimately show ?thesis using beq P'''Trans by (blast intro: transitionI)
    next
      assume P  $\neq$  P''
      with PTrans have [a $\frown$ a]P  $\Rightarrow_\tau$  P'' by (rule matchChain)
      thus ?thesis using beq P''Trans P'''Trans by (blast intro: transitionI)
    qed
  next
    case (FreeR  $\alpha$  P')
    have P  $\Rightarrow_l$   $\alpha$  < P' by fact

    then obtain P'' P''' where PTrans: P  $\Rightarrow_\tau$  P''
      and P''Trans: P''  $\mapsto$   $\alpha$  < P'''
      and P'''Trans: P'''  $\Rightarrow_\tau$  P'

    by (blast dest: transitionE)
    show ?case
    proof (cases P = P'')
      assume P = P''
      moreover have [a $\frown$ a]P  $\Rightarrow_\tau$  [a $\frown$ a]P by simp
      moreover from P''Trans have [a $\frown$ a]P''  $\mapsto$   $\alpha$  < P''' by (rule transitions.Match)
      ultimately show ?thesis using P'''Trans by (blast intro: transitionI)
    end
  end
end

```

```

next
  assume  $P \neq P''$ 
  with  $PTrans$  have  $[a \frown a]P \Longrightarrow_{\tau} P''$  by(rule matchChain)
  thus ?thesis using  $P''Trans$   $P'''Trans$  by(rule transitionI)
qed
qed
next
  assume  $P \Longrightarrow_{lu} in P'' \rightarrow b \langle x \rangle \prec P'$ 
  then obtain  $P'''$  where  $PChain: P \Longrightarrow_{\tau} P'''$ 
    and  $P'''Trans: P''' \mapsto b \langle x \rangle \prec P''$ 
    and  $P''Chain: P''[x::=u] \Longrightarrow_{\tau} P'$ 
  by(blast dest: transitionE)
  show  $[a \frown a]P \Longrightarrow_{lu} in P'' \rightarrow b \langle x \rangle \prec P'$ 
  proof(cases  $P=P'''$ )
    assume  $P=P'''$ 
    moreover have  $[a \frown a]P \Longrightarrow_{\tau} [a \frown a]P$  by simp
    moreover from  $P'''Trans$  have  $[a \frown a]P''' \mapsto b \langle x \rangle \prec P''$  by(rule Late-Semantics.Match)
    ultimately show ?thesis using  $P''Chain$  by(blast intro: transitionI)
  next
    assume  $P \neq P'''$ 
    with  $PChain$  have  $[a \frown a]P \Longrightarrow_{\tau} P'''$  by(rule matchChain)
    thus ?thesis using  $P'''Trans$   $P''Chain$  by(rule transitionI)
  qed
qed

```

lemma *Mismatch*:

```

fixes  $P :: pi$ 
and  $Rs :: residual$ 
and  $a :: name$ 
and  $c :: name$ 
and  $u :: name$ 
and  $b :: name$ 
and  $x :: name$ 
and  $P' :: pi$ 

```

shows $\llbracket P \Longrightarrow_l Rs; a \neq c \rrbracket \Longrightarrow [a \neq c]P \Longrightarrow_l Rs$

and $\llbracket P \Longrightarrow_{lu} in P'' \rightarrow b \langle x \rangle \prec P'; a \neq c \rrbracket \Longrightarrow [a \neq c]P \Longrightarrow_{lu} in P'' \rightarrow b \langle x \rangle \prec P'$

proof –

assume $PTrans: P \Longrightarrow_l Rs$

and $aineqc: a \neq c$

thus $[a \neq c]P \Longrightarrow_l Rs$

proof(nominal-induct avoiding: P rule: residual.strong-inducts)

case(BoundR $b x P'$)

have $PTrans: P \Longrightarrow_l b \langle x \rangle \prec P'$ and $xFreshP: x \# P$ by fact+

from $PTrans$ obtain b' where $beg: b = BoundOutputS b'$ by(cases b , auto)

with $PTrans$ $xFreshP$ obtain $P'' P'''$ where $PTrans: P \Longrightarrow_{\tau} P''$

and $P''Trans: P'' \mapsto b' \langle \nu x \rangle \prec P'''$

and $P'''Trans: P''' \Longrightarrow_{\tau} P'$

```

    by(blast dest: transitionE)
  show ?case
  proof(cases P = P'')
    assume P = P''
    moreover have [a≠c]P ⇒τ [a≠c]P by simp
    moreover from P''Trans aineqc have [a≠c]P'' ↦ b'<νx> < P''' by(rule
transitions.Mismatch)
    ultimately show ?thesis using beq P'''Trans by(blast intro: transitionI)
  next
    assume P ≠ P''
    with PTrans aineqc have [a≠c]P ⇒τ P'' by(rule mismatchChain)
    thus ?thesis using beq P''Trans P'''Trans by(blast intro: transitionI)
  qed
next
case(FreeR α P')
have P ⇒l α < P' by fact

  then obtain P'' P''' where PTrans: P ⇒τ P''
    and P''Trans: P'' ↦ α < P'''
    and P'''Trans: P''' ⇒τ P'

  by(blast dest: transitionE)
  show ?case
  proof(cases P = P'')
    assume P = P''
    moreover have [a≠c]P ⇒τ [a≠c]P by simp
    moreover from P''Trans ‹a ≠ c› have [a≠c]P'' ↦ α < P''' by(rule
transitions.Mismatch)
    ultimately show ?thesis using P'''Trans by(blast intro: transitionI)
  next
    assume P ≠ P''
    with PTrans aineqc have [a≠c]P ⇒τ P'' by(rule mismatchChain)
    thus ?thesis using P''Trans P'''Trans by(rule transitionI)
  qed
qed
next
assume aineqc: a ≠ c
assume P ⇒lu in P'' → b<x> < P'
then obtain P''' where PChain: P ⇒τ P'''
  and P'''Trans: P''' ↦ b<x> < P''
  and P''Chain: P''[x::=u] ⇒τ P'

  by(blast dest: transitionE)
  show [a≠c]P ⇒lu in P'' → b<x> < P'
  proof(cases P=P''')
    assume P=P'''
    moreover have [a≠c]P ⇒τ [a≠c]P by simp
    moreover from P'''Trans aineqc have [a≠c]P''' ↦ b<x> < P'' by(rule
Late-Semantics.Mismatch)
    ultimately show ?thesis using P''Chain by(blast intro: transitionI)
  next

```

assume $P \neq P'''$
with $PChain\ aineqc$ **have** $[a \neq c]P \Longrightarrow_{\tau} P'''$ **by**(*rule mismatchChain*)
thus $?thesis$ **using** $P'''Trans\ P''Chain$ **by**(*rule transitionI*)
qed
qed

lemma *Open*:

fixes $P :: pi$
and $a :: name$
and $b :: name$
and $P' :: pi$

assumes $Trans: P \Longrightarrow_l a[b] \prec P'$
and $aInEqb: a \neq b$

shows $\langle \nu b \rangle P \Longrightarrow_l a \langle \nu b \rangle \prec P'$

proof –

from $Trans$ **obtain** $P''\ P'''$ **where** $A: P \Longrightarrow_{\tau} P''$
and $B: P'' \mapsto_a [b] \prec P'''$
and $C: P''' \Longrightarrow_{\tau} P'$

by(*force dest: transitionE*)

from A **have** $\langle \nu b \rangle P \Longrightarrow_{\tau} \langle \nu b \rangle P''$ **by**(*rule ResChain*)

moreover from $B\ aInEqb$ **have** $\langle \nu b \rangle P'' \mapsto_a \langle \nu b \rangle \prec P'''$ **by**(*rule Open*)

ultimately show $?thesis$ **using** C **by**(*force simp add: transition-def*)

qed

lemma *Sum1*:

fixes $P :: pi$
and $Rs :: residual$
and $Q :: pi$
and $u :: name$
and $P'' :: pi$
and $a :: name$
and $x :: name$
and $P' :: pi$

shows $P \Longrightarrow_l Rs \Longrightarrow P \oplus Q \Longrightarrow_l Rs$

and $P \Longrightarrow_l u$ **in** $P'' \rightarrow_a \langle x \rangle \prec P' \Longrightarrow P \oplus Q \Longrightarrow_l u$ **in** $P'' \rightarrow_a \langle x \rangle \prec P'$

proof –

assume $P \Longrightarrow_l Rs$

thus $P \oplus Q \Longrightarrow_l Rs$

proof(*nominal-induct avoiding: P rule: residual.strong-inducts*)

case($BoundR\ a\ x\ P'\ P$)

have $PTrans: P \Longrightarrow_l a \langle x \rangle \prec P'$

and $xFreshP: x \# P$ **by** *fact+*

from $PTrans$ **obtain** a' **where** $aeq: a = BoundOutputS\ a'$ **by**(*cases a, auto*)

with $PTrans\ xFreshP$ **obtain** $P''\ P'''$ **where** $PTrans: P \Longrightarrow_{\tau} P''$

and $P''Trans: P'' \mapsto_{a'} \langle \nu x \rangle \prec P'''$

and $P'''Trans: P''' \Longrightarrow_{\tau} P'$

```

    by(blast dest: transitionE)
  show ?case
  proof(cases P = P'')
    assume P = P''
    moreover have P ⊕ Q ⇒τ P ⊕ Q by simp
    moreover from P''Trans have P'' ⊕ Q ↦τ a'⟨νx⟩ < P''' by(rule transi-
tions.Sum1)
    ultimately show ?thesis using P'''Trans aeq by(blast intro: transitionI)
  next
    assume P ≠ P''
    with PTrans have P ⊕ Q ⇒τ P'' by(rule sum1Chain)
    thus ?thesis using P''Trans P'''Trans aeq by(blast intro: transitionI)
  qed
next
case(FreeR α P')
have P ⇒l α < P' by fact

  then obtain P'' P''' where PTrans: P ⇒τ P''
    and P''Trans: P'' ↦τ α < P'''
    and P'''Trans: P''' ⇒τ P'
  by(blast dest: transitionE)
  show ?case
  proof(cases P = P'')
    assume P = P''
    moreover have P ⊕ Q ⇒τ P ⊕ Q by simp
    moreover from P''Trans have P'' ⊕ Q ↦τ α < P''' by(rule transi-
tions.Sum1)
    ultimately show ?thesis using P'''Trans by(blast intro: transitionI)
  next
    assume P ≠ P''
    with PTrans have P ⊕ Q ⇒τ P'' by(rule sum1Chain)
    thus ?thesis using P''Trans P'''Trans by(rule transitionI)
  qed
  qed
next
assume P ⇒lu in P'' → a⟨x⟩ < P'
then obtain P''' where PChain: P ⇒τ P'''
  and P'''Trans: P''' ↦τ a⟨x⟩ < P''
  and P''Chain: P''[x::=u] ⇒τ P'
  by(blast dest: transitionE)
show P ⊕ Q ⇒lu in P'' → a⟨x⟩ < P'
proof(cases P = P''')
  assume P = P'''
  moreover have P ⊕ Q ⇒τ P ⊕ Q by simp
  moreover from P'''Trans have P''' ⊕ Q ↦τ a⟨x⟩ < P'' by(rule transi-
tions.Sum1)
  ultimately show ?thesis using P''Chain by(blast intro: transitionI)
next
  assume P ≠ P'''

```

```

    with PChain have  $P \oplus Q \Longrightarrow_{\tau} P'''$  by(rule sum1Chain)
    thus ?thesis using P'''Trans P''Chain by(blast intro: transitionI)
  qed
qed

lemma Sum2:
  fixes Q :: pi
  and Rs :: residual
  and P :: pi
  and u :: name
  and Q'' :: pi
  and a :: name
  and x :: name
  and Q' :: pi

  shows  $Q \Longrightarrow_l Rs \Longrightarrow P \oplus Q \Longrightarrow_l Rs$ 
  and  $Q \Longrightarrow_l u$  in  $Q'' \rightarrow a \langle x \rangle \prec Q' \Longrightarrow P \oplus Q \Longrightarrow_l u$  in  $Q'' \rightarrow a \langle x \rangle \prec Q'$ 
proof -
  assume  $Q \Longrightarrow_l Rs$ 
  thus  $P \oplus Q \Longrightarrow_l Rs$ 
  proof(nominal-induct avoiding: Q rule: residual.strong-inducts)
    case(BoundR a x Q' Q)
    have QTrans:  $Q \Longrightarrow_l a \langle x \rangle \prec Q'$ 
    and xFreshQ:  $x \# Q$  by fact+
    from QTrans obtain a' where aeq:  $a = \text{BoundOutputS } a'$  by(cases a, auto)
    with QTrans xFreshQ obtain Q'' Q''' where QTrans:  $Q \Longrightarrow_{\tau} Q''$ 
    and Q''Trans:  $Q'' \mapsto a' \langle \nu x \rangle \prec Q'''$ 
    and Q'''Trans:  $Q''' \Longrightarrow_{\tau} Q'$ 

    by(blast dest: transitionE)
  show ?case
  proof(cases Q = Q'')
    assume Q = Q''
    moreover have  $P \oplus Q \Longrightarrow_{\tau} P \oplus Q$  by simp
    moreover from Q''Trans have  $P \oplus Q'' \mapsto a' \langle \nu x \rangle \prec Q'''$  by(rule transitions.Sum2)
    ultimately show ?thesis using Q'''Trans aeq by(blast intro: transitionI)
  next
    assume  $Q \neq Q''$ 
    with QTrans have  $P \oplus Q \Longrightarrow_{\tau} Q''$  by(rule sum2Chain)
    thus ?thesis using Q''Trans Q'''Trans aeq by(blast intro: transitionI)
  qed
  next
  case(FreeR  $\alpha$  Q')
  have  $Q \Longrightarrow_l \alpha \prec Q'$  by fact

  then obtain Q'' Q''' where QTrans:  $Q \Longrightarrow_{\tau} Q''$ 
    and Q''Trans:  $Q'' \mapsto \alpha \prec Q'''$ 
    and Q'''Trans:  $Q''' \Longrightarrow_{\tau} Q'$ 
  by(blast dest: transitionE)

```

```

show ?case
proof(cases Q = Q'')
  assume Q = Q''
  moreover have P ⊕ Q ⇒τ P ⊕ Q by simp
  moreover from Q''Trans have P ⊕ Q'' ↦ α < Q''' by(rule transi-
tions.Sum2)
  ultimately show ?thesis using Q'''Trans by(blast intro: transitionI)
next
assume Q ≠ Q''
with QTrans have P ⊕ Q ⇒τ Q'' by(rule sum2Chain)
thus ?thesis using Q''Trans Q'''Trans by(rule transitionI)
qed
qed
next
assume Q ⇒lu in Q'' → a < x > < Q'
then obtain Q''' where QChain: Q ⇒τ Q'''
      and Q'''Trans: Q''' ↦ a < x > < Q''
      and Q''Chain: Q''[x::=u] ⇒τ Q'
  by(blast dest: transitionE)
show P ⊕ Q ⇒lu in Q'' → a < x > < Q'
proof(cases Q = Q''')
  assume Q = Q'''
  moreover have P ⊕ Q ⇒τ P ⊕ Q by simp
  moreover from Q'''Trans have P ⊕ Q''' ↦ a < x > < Q'' by(rule transi-
tions.Sum2)
  ultimately show ?thesis using Q''Chain by(blast intro: transitionI)
next
assume Q ≠ Q'''
with QChain have P ⊕ Q ⇒τ Q''' by(rule sum2Chain)
thus ?thesis using Q'''Trans Q''Chain by(blast intro: transitionI)
qed
qed

```

lemma Par1B:

```

fixes P :: pi
and a :: name
and x :: name
and P' :: pi
and u :: name
and P'' :: pi

```

```

shows [P ⇒la < ν x > < P'; x # Q] ⇒ P || Q ⇒la < ν x > < (P' || Q)
and [P ⇒lu in P'' → a < x > < P'; x # Q] ⇒ P || Q ⇒lu in (P'' || Q) → a < x >
< P' || Q

```

proof –

```

assume PTrans: P ⇒l a < ν x > < P'
assume xFreshQ: x # Q

```

```

have Goal: ∧ P a x P' Q. [P ⇒la < ν x > < P'; x # P; x # Q] ⇒ P || Q

```

$\Longrightarrow_I a \langle \nu x \rangle \prec (P' \parallel Q)$
proof –
fix $P \ a \ x \ P' \ Q$
assume $PTrans: P \Longrightarrow_I a \langle \nu x \rangle \prec P'$
assume $xFreshP: x \# P$
assume $xFreshQ: x \# (Q::pi)$

from $PTrans \ xFreshP$ **obtain** $P'' \ P'''$ **where** $PTrans: P \Longrightarrow_\tau P''$
and $P''Trans: P'' \mapsto a \langle \nu x \rangle \prec P'''$
and $P'''Trans: P''' \Longrightarrow_\tau P'$

by(*blast dest: transitionE*)
from $PTrans$ **have** $P \parallel Q \Longrightarrow_\tau P'' \parallel Q$ **by**(*rule Par1Chain*)
moreover from $P''Trans \ xFreshQ$ **have** $P'' \parallel Q \mapsto a \langle \nu x \rangle \prec (P''' \parallel Q)$
by(*rule Par1B*)
moreover from $P'''Trans$ **have** $P''' \parallel Q \Longrightarrow_\tau P' \parallel Q$ **by**(*rule Par1Chain*)
ultimately show $P \parallel Q \Longrightarrow_I a \langle \nu x \rangle \prec (P' \parallel Q)$ **by**(*rule transitionI*)
qed

have $\exists c::name. c \# (P, P', Q)$ **by**(*blast intro: name-exists-fresh*)
then obtain $c::name$ **where** $cFreshP: c \# P$ **and** $cFreshP': c \# P'$ **and** $cFreshQ:$
 $c \# Q$
by(*force simp add: fresh-prod*)

from $cFreshP'$ **have** $a \langle \nu x \rangle \prec P' = a \langle \nu c \rangle \prec ([x, c] \cdot P')$ **by**(*rule alphaBound-Residual*)
moreover have $a \langle \nu x \rangle \prec (P' \parallel Q) = a \langle \nu c \rangle \prec ([x, c] \cdot P' \parallel Q)$
proof –
from $cFreshP' \ cFreshQ$ **have** $c \# P' \parallel Q$ **by** *simp*
hence $a \langle \nu x \rangle \prec (P' \parallel Q) = a \langle \nu c \rangle \prec ([x, c] \cdot (P' \parallel Q))$ **by**(*rule alphaBound-Residual*)
with $cFreshQ \ xFreshQ$ **show** *?thesis* **by**(*simp add: name-fresh-fresh*)
qed

ultimately show $P \parallel Q \Longrightarrow_I a \langle \nu x \rangle \prec P' \parallel Q$ **using** $PTrans \ cFreshP \ cFreshQ$
by(*force intro: Goal*)
next
assume $PTrans: P \Longrightarrow_I u$ **in** $P'' \mapsto a \langle x \rangle \prec P'$
and $xFreshQ: x \# Q$
from $PTrans$ **obtain** P''' **where** $PChain: P \Longrightarrow_\tau P'''$
and $P'''Trans: P''' \mapsto a \langle x \rangle \prec P''$
and $P''Chain: P''[x::=u] \Longrightarrow_\tau P'$

by(*blast dest: transitionE*)
from $PChain$ **have** $P \parallel Q \Longrightarrow_\tau P''' \parallel Q$ **by**(*rule Par1Chain*)
moreover from $P'''Trans \ xFreshQ$ **have** $P''' \parallel Q \mapsto a \langle x \rangle \prec (P'' \parallel Q)$ **by**(*rule Par1B*)
moreover have $(P'' \parallel Q)[x::=u] \Longrightarrow_\tau P' \parallel Q$
proof –
from $P''Chain$ **have** $P''[x::=u] \parallel Q \Longrightarrow_\tau P' \parallel Q$ **by**(*rule Par1Chain*)
with $xFreshQ$ **show** *?thesis* **by**(*simp add: forget*)
qed

ultimately show $P \parallel Q \Longrightarrow_{\iota u} (P'' \parallel Q) \rightarrow a \langle x \rangle \prec (P' \parallel Q)$ by(*rule transitionI*)

qed

lemma *Par1F*:

fixes $P :: pi$

and $\alpha :: freeRes$

and $P' :: pi$

assumes $PTrans: P \Longrightarrow_{\iota} \alpha \prec P'$

shows $P \parallel Q \Longrightarrow_{\iota} \alpha \prec (P' \parallel Q)$

proof –

from $PTrans$ obtain $P'' P'''$ where $PTrans: P \Longrightarrow_{\tau} P''$

and $P''Trans: P'' \mapsto_{\alpha} \prec P'''$

and $P'''Trans: P''' \Longrightarrow_{\tau} P'$

by(*blast dest: transitionE*)

from $PTrans$ have $P \parallel Q \Longrightarrow_{\tau} P'' \parallel Q$ by(*rule Par1Chain*)

moreover from $P''Trans$ have $P'' \parallel Q \mapsto_{\alpha} \prec (P''' \parallel Q)$ by(*rule transitions.Par1F*)

moreover from $P'''Trans$ have $P''' \parallel Q \Longrightarrow_{\tau} P' \parallel Q$ by(*rule Par1Chain*)

ultimately show *?thesis* by(*rule transitionI*)

qed

lemma *Par2B*:

fixes $Q :: pi$

and $a :: name$

and $x :: name$

and $Q' :: pi$

and $P :: pi$

and $u :: name$

and $Q'' :: pi$

shows $Q \Longrightarrow_{\iota} a \langle \nu x \rangle \prec Q' \Longrightarrow x \# P \Longrightarrow P \parallel Q \Longrightarrow_{\iota} a \langle \nu x \rangle \prec (P \parallel Q')$

and $Q \Longrightarrow_{\iota u} (Q'' \rightarrow a \langle x \rangle \prec Q' \Longrightarrow x \# P \Longrightarrow P \parallel Q \Longrightarrow_{\iota u} (P \parallel Q'') \rightarrow a \langle x \rangle \prec P \parallel Q')$

proof –

assume $QTrans: Q \Longrightarrow_{\iota} a \langle \nu x \rangle \prec Q'$

assume $xFreshP: x \# P$

have *Goal*: $\bigwedge Q a x Q' P. \llbracket Q \Longrightarrow_{\iota} a \langle \nu x \rangle \prec Q'; x \# Q; x \# P \rrbracket \Longrightarrow P \parallel Q \Longrightarrow_{\iota} a \langle \nu x \rangle \prec (P \parallel Q')$

proof –

fix $Q a x Q' P$

assume $QTrans: Q \Longrightarrow_{\iota} a \langle \nu x \rangle \prec Q'$

assume $xFreshQ: x \# Q$

assume $xFreshP: x \# (P :: pi)$

from $QTrans$ $xFreshQ$ obtain $Q'' Q'''$ where $QTrans: Q \Longrightarrow_{\tau} Q''$

and $Q''\text{Trans}: Q'' \mapsto a\langle \nu x \rangle \prec Q'''$
and $Q'''\text{Trans}: Q''' \Longrightarrow_{\tau} Q'$
by(*blast dest: transitionE*)
from $Q\text{Trans}$ **have** $P \parallel Q \Longrightarrow_{\tau} P \parallel Q''$ **by**(*rule Par2Chain*)
moreover from $Q''\text{Trans}$ $x\text{Fresh}P$ **have** $P \parallel Q'' \mapsto a\langle \nu x \rangle \prec (P \parallel Q''')$
by(*rule Par2B*)
moreover from $Q'''\text{Trans}$ **have** $P \parallel Q''' \Longrightarrow_{\tau} P \parallel Q'$ **by**(*rule Par2Chain*)
ultimately show $P \parallel Q \Longrightarrow_l a\langle \nu x \rangle \prec (P \parallel Q')$ **by**(*rule transitionI*)
qed

have $\exists c::\text{name}. c \# (Q, Q', P)$ **by**(*blast intro: name-exists-fresh*)
then obtain $c::\text{name}$ **where** $c\text{Fresh}Q: c \# Q$ **and** $c\text{Fresh}Q': c \# Q'$ **and** $c\text{Fresh}P:$
 $c \# P$
by(*force simp add: fresh-prod*)

from $c\text{Fresh}Q'$ **have** $a\langle \nu x \rangle \prec Q' = a\langle \nu c \rangle \prec ([(x, c)] \cdot Q')$ **by**(*rule alphaBoundResidual*)
moreover have $a\langle \nu x \rangle \prec (P \parallel Q') = a\langle \nu c \rangle \prec (P \parallel ([(x, c)] \cdot Q'))$
proof –
from $c\text{Fresh}Q'$ $c\text{Fresh}P$ **have** $c \# P \parallel Q'$ **by** *simp*
hence $a\langle \nu x \rangle \prec (P \parallel Q') = a\langle \nu c \rangle \prec ([(x, c)] \cdot (P \parallel Q'))$ **by**(*rule alphaBoundResidual*)
with $c\text{Fresh}P$ $x\text{Fresh}P$ **show** *?thesis* **by**(*simp add: name-fresh-fresh*)
qed
ultimately show $P \parallel Q \Longrightarrow_l a\langle \nu x \rangle \prec P \parallel Q'$ **using** $Q\text{Trans}$ $c\text{Fresh}Q$ $c\text{Fresh}P$
by(*force intro: Goal*)
next
assume $Q\text{Trans}: Q \Longrightarrow_l u$ **in** $Q'' \mapsto a\langle x \rangle \prec Q'$
and $x\text{Fresh}P: x \# P$
from $Q\text{Trans}$ **obtain** Q''' **where** $Q\text{Chain}: Q \Longrightarrow_{\tau} Q'''$
and $Q'''\text{Trans}: Q''' \mapsto a\langle x \rangle \prec Q''$
and $Q''\text{Chain}: Q''[x::=u] \Longrightarrow_{\tau} Q'$
by(*blast dest: transitionE*)
from $Q\text{Chain}$ **have** $P \parallel Q \Longrightarrow_{\tau} P \parallel Q'''$ **by**(*rule Par2Chain*)
moreover from $Q'''\text{Trans}$ $x\text{Fresh}P$ **have** $P \parallel Q''' \mapsto a\langle x \rangle \prec (P \parallel Q'')$ **by**(*rule Par2B*)
moreover have $(P \parallel Q'')[x::=u] \Longrightarrow_{\tau} P \parallel Q'$
proof –
from $Q''\text{Chain}$ **have** $P \parallel (Q''[x::=u]) \Longrightarrow_{\tau} P \parallel Q'$ **by**(*rule Par2Chain*)
with $x\text{Fresh}P$ **show** *?thesis* **by**(*simp add: forget*)
qed
ultimately show $P \parallel Q \Longrightarrow_l u$ **in** $(P \parallel Q'') \mapsto a\langle x \rangle \prec (P \parallel Q')$ **by**(*rule transitionI*)
qed

lemma *Par2F*:
fixes $Q :: pi$
and $\alpha :: \text{freeRes}$
and $Q' :: pi$


```

fixes  $P :: pi$ 
and  $a :: name$ 
and  $b :: name$ 
and  $P' :: pi$ 
and  $Q :: pi$ 
and  $x :: name$ 
and  $Q'' :: pi$ 
and  $Q' :: pi$ 

assumes  $PTrans: P \Rightarrow_l a[b] \prec P'$ 
and  $QTrans: Q \Rightarrow_l b \text{ in } Q'' \rightarrow a \langle x \rangle \prec Q'$ 

shows  $P \parallel Q \Rightarrow_{l\tau} \prec P' \parallel Q'$ 
proof –
from  $PTrans$  obtain  $P'' P'''$  where  $PChain: P \Rightarrow_\tau P'''$ 
and  $P'''Trans: P''' \mapsto_a [b] \prec P''$ 
and  $P''Chain: P'' \Rightarrow_\tau P'$ 
by(blast dest: transitionE)
from  $QTrans$  obtain  $Q'''$  where  $QChain: Q \Rightarrow_\tau Q'''$ 
and  $Q'''Trans: Q''' \mapsto_a \langle x \rangle \prec Q''$ 
and  $Q''Chain: Q''[x::=b] \Rightarrow_\tau Q'$ 
by(blast dest: transitionE)

from  $PChain$   $QChain$  have  $P \parallel Q \Rightarrow_\tau P''' \parallel Q'''$  by(rule chainPar)
moreover from  $P'''Trans$   $Q'''Trans$  have  $P''' \parallel Q''' \mapsto_\tau \prec P'' \parallel (Q''[x::=b])$ 
by(rule Comm2)
moreover from  $P''Chain$   $Q''Chain$  have  $P'' \parallel (Q''[x::=b]) \Rightarrow_\tau P' \parallel Q'$  by(rule chainPar)
ultimately show ?thesis by(rule transitionI)
qed

```

lemma *Close1*:

```

fixes  $P :: pi$ 
and  $y :: name$ 
and  $P'' :: pi$ 
and  $a :: name$ 
and  $x :: name$ 
and  $P' :: pi$ 
and  $Q :: pi$ 
and  $Q' :: pi$ 

assumes  $PTrans: P \Rightarrow_l y \text{ in } P'' \rightarrow a \langle x \rangle \prec P'$ 
and  $QTrans: Q \Rightarrow_l a \langle \nu y \rangle \prec Q'$ 
and  $yFreshP: y \# P$ 
and  $yFreshQ: y \# Q$ 

```

shows $P \parallel Q \Rightarrow_{l\tau} \prec \langle \nu y \rangle (P' \parallel Q')$

proof –

```

from  $PTrans$  obtain  $P'''$  where  $PChain: P \Rightarrow_\tau P'''$ 

```

and $P'''Trans: P''' \mapsto a \langle x \rangle \prec P''$
and $P''Chain: P''[x::=y] \Rightarrow_{\tau} P'$
by(blast dest: transitionE)
from $QTrans$ $yFreshQ$ **obtain** Q'' Q''' **where** $QChain: Q \Rightarrow_{\tau} Q'''$
and $Q'''Trans: Q''' \mapsto a \langle \nu y \rangle \prec Q''$
and $Q''Chain: Q'' \Rightarrow_{\tau} Q'$
by(blast dest: transitionE)
from $PChain$ $yFreshP$ **have** $yFreshP''': y \# P'''$ **by**(rule freshChain)
from $PChain$ $QChain$ **have** $P \parallel Q \Rightarrow_{\tau} P''' \parallel Q'''$ **by**(rule chainPar)
moreover from $P'''Trans$ $Q'''Trans$ $yFreshP'''$ **have** $P''' \parallel Q''' \mapsto_{\tau} \prec \langle \nu y \rangle (P''[x::=y])$
 $\parallel Q''$
by(rule Close1)
moreover have $\langle \nu y \rangle (P''[x::=y] \parallel Q'') \Rightarrow_{\tau} \langle \nu y \rangle (P' \parallel Q')$
proof –
from $P''Chain$ $Q''Chain$ **have** $P''[x::=y] \parallel Q'' \Rightarrow_{\tau} P' \parallel Q'$ **by**(rule chainPar)
thus ?thesis **by**(rule ResChain)
qed
ultimately show $P \parallel Q \Rightarrow_{\iota\tau} \prec \langle \nu y \rangle (P' \parallel Q')$ **by**(rule transitionI)
qed

lemma Close2:

fixes $P :: pi$
and $y :: name$
and $a :: name$
and $x :: name$
and $P' :: pi$
and $Q :: pi$
and $Q'' :: pi$
and $Q' :: pi$

assumes $PTrans: P \Rightarrow_{\iota} a \langle \nu y \rangle \prec P'$
and $QTrans: Q \Rightarrow_{\iota} y$ in $Q'' \rightarrow a \langle x \rangle \prec Q'$
and $yFreshP: y \# P$
and $yFreshQ: y \# Q$

shows $P \parallel Q \Rightarrow_{\iota\tau} \prec \langle \nu y \rangle (P' \parallel Q')$

proof –

from $PTrans$ $yFreshP$ **obtain** P'' P''' **where** $PChain: P \Rightarrow_{\tau} P'''$
and $P'''Trans: P''' \mapsto a \langle \nu y \rangle \prec P''$
and $P''Chain: P'' \Rightarrow_{\tau} P'$
by(blast dest: transitionE)

from $QTrans$ **obtain** Q''' **where** $QChain: Q \Rightarrow_{\tau} Q'''$
and $Q'''Trans: Q''' \mapsto a \langle x \rangle \prec Q''$
and $Q''Chain: Q''[x::=y] \Rightarrow_{\tau} Q'$
by(blast dest: transitionE)

from $QChain$ $yFreshQ$ **have** $yFreshQ'''$: $y \# Q'''$ **by**(rule *freshChain*)
from $PChain$ $QChain$ **have** $P \parallel Q \Longrightarrow_{\tau} P''' \parallel Q'''$ **by**(rule *chainPar*)
moreover from $P'''Trans$ $Q'''Trans$ $yFreshQ'''$ **have** $P''' \parallel Q''' \longmapsto_{\tau} \prec \langle \nu y \rangle (P'' \parallel (Q''[x::=y]))$
by(rule *Close2*)
moreover have $\langle \nu y \rangle (P'' \parallel (Q''[x::=y])) \Longrightarrow_{\tau} \langle \nu y \rangle (P' \parallel Q')$
proof –
from $P''Chain$ $Q''Chain$ **have** $P'' \parallel (Q''[x::=y]) \Longrightarrow_{\tau} P' \parallel Q'$ **by**(rule *chain-Par*)
thus *?thesis* **by**(rule *ResChain*)
qed
ultimately show $P \parallel Q \Longrightarrow_{\tau} \prec \langle \nu y \rangle (P' \parallel Q')$ **by**(rule *transitionI*)
qed

lemma *ResF*:

fixes $P :: pi$
and $\alpha :: freeRes$
and $P' :: pi$
and $x :: name$

assumes $PTrans$: $P \Longrightarrow_{\tau} \alpha \prec P'$
and $xFreshAlpha$: $x \# \alpha$

shows $\langle \nu x \rangle P \Longrightarrow_{\tau} \alpha \prec \langle \nu x \rangle P'$

proof –

from $PTrans$ **obtain** $P'' P'''$ **where** $PChain$: $P \Longrightarrow_{\tau} P''$
and $P''Trans$: $P'' \longmapsto_{\alpha} \prec P'''$
and $P'''Chain$: $P''' \Longrightarrow_{\tau} P'$
by(blast dest: *transitionE*)

from $PChain$ **have** $\langle \nu x \rangle P \Longrightarrow_{\tau} \langle \nu x \rangle P''$ **by**(rule *ResChain*)
moreover from $P''Trans$ $xFreshAlpha$ **have** $\langle \nu x \rangle P'' \longmapsto_{\alpha} \prec \langle \nu x \rangle P'''$
by(rule *transitions.ResF*)
moreover from $P'''Chain$ **have** $\langle \nu x \rangle P''' \Longrightarrow_{\tau} \langle \nu x \rangle P'$ **by**(rule *ResChain*)
ultimately show *?thesis* **by**(rule *transitionI*)
qed

lemma *ResB*:

fixes $P :: pi$
and $a :: name$
and $x :: name$
and $P' :: pi$
and $y :: name$
and $u :: name$
and $P'' :: pi$

shows $\llbracket P \Longrightarrow_{\tau} \alpha \langle \nu x \rangle \prec P'; y \neq a; y \neq x; x \# P \rrbracket \Longrightarrow \langle \nu y \rangle P \Longrightarrow_{\tau} \alpha \langle \nu x \rangle \prec (\langle \nu y \rangle P')$

and $\llbracket P \Longrightarrow_{\iota u} \text{in } P'' \rightarrow a \langle x \rangle \prec P'; y \neq a; y \neq x; y \neq u \rrbracket \Longrightarrow \langle \nu y \rangle P \Longrightarrow_{\iota u} \text{in } (\langle \nu y \rangle P'') \rightarrow a \langle x \rangle \prec (\langle \nu y \rangle P')$

proof –

assume $PTrans: P \Longrightarrow_{\iota a} \langle \nu x \rangle \prec P'$
 and $yineqa: y \neq a$
 and $yineqx: y \neq x$
 and $xFreshP: x \# P$

from $PTrans$ $xFreshP$ obtain $P'' P'''$ where $PChain: P \Longrightarrow_{\tau} P''$
 and $P''Trans: P'' \mapsto a \langle \nu x \rangle \prec P'''$
 and $P'''Chain: P''' \Longrightarrow_{\tau} P'$

by(*blast dest: transitionE*)

from $PChain$ have $\langle \nu y \rangle P \Longrightarrow_{\tau} \langle \nu y \rangle P''$ by(*rule ResChain*)
 moreover from $P''Trans$ $yineqa$ $yineqx$ have $\langle \nu y \rangle P'' \mapsto a \langle \nu x \rangle \prec (\langle \nu y \rangle P''')$
 by(*force intro: ResB*)
 moreover from $P'''Chain$ have $\langle \nu y \rangle P''' \Longrightarrow_{\tau} \langle \nu y \rangle P'$ by(*rule ResChain*)
 ultimately show $\langle \nu y \rangle P \Longrightarrow_{\iota} a \langle \nu x \rangle \prec \langle \nu y \rangle P'$ by(*rule transitionI*)

next

assume $PTrans: P \Longrightarrow_{\iota u} \text{in } P'' \rightarrow a \langle x \rangle \prec P'$
 and $yineqa: y \neq a$
 and $yineqx: y \neq x$
 and $yinequ: y \neq u$

from $PTrans$ obtain P''' where $PChain: P \Longrightarrow_{\tau} P'''$
 and $P'''Trans: P''' \mapsto a \langle x \rangle \prec P''$
 and $P''Chain: P''[x::=u] \Longrightarrow_{\tau} P'$

by(*blast dest: transitionE*)

from $PChain$ have $\langle \nu y \rangle P \Longrightarrow_{\tau} \langle \nu y \rangle P'''$ by(*rule ResChain*)
 moreover from $P'''Trans$ $yineqa$ $yineqx$ have $\langle \nu y \rangle P''' \mapsto a \langle x \rangle \prec (\langle \nu y \rangle P'')$
 by(*force intro: ResB*)
 moreover have $(\langle \nu y \rangle P'')[x::=u] \Longrightarrow_{\tau} \langle \nu y \rangle P'$

proof –

from $P''Chain$ have $\langle \nu y \rangle (P''[x::=u]) \Longrightarrow_{\tau} \langle \nu y \rangle P'$ by(*rule ResChain*)
 with $yineqx$ $yinequ$ show *?thesis* by(*simp add: eqvt-subs[THEN sym]*)

qed

ultimately show $\langle \nu y \rangle P \Longrightarrow_{\iota u} \text{in } (\langle \nu y \rangle P'') \rightarrow a \langle x \rangle \prec \langle \nu y \rangle P'$ by(*rule transitionI*)

qed

lemma *Bang*:

fixes $P :: pi$
 and $Rs :: residual$
 and $u :: name$
 and $P'' :: pi$
 and $a :: name$
 and $x :: name$
 and $P' :: pi$

shows $P \parallel !P \Rightarrow_l Rs \Rightarrow !P \Rightarrow_l Rs$
and $P \parallel !P \Rightarrow_l u$ in $P'' \rightarrow a \langle x \rangle \prec P' \Rightarrow !P \Rightarrow_l u$ in $P'' \rightarrow a \langle x \rangle \prec P'$
proof –
assume $P \parallel !P \Rightarrow_l Rs$
thus $!P \Rightarrow_l Rs$
proof(*nominal-induct avoiding: P rule: residual.strong-inducts*)
case(*BoundR a x P' P*)
assume $xFreshP: x \# P$
assume $PTrans: P \parallel !P \Rightarrow_l a \langle x \rangle \prec P'$
from $PTrans$ **obtain** a' **where** $aeq: a = BoundOutputS a'$ **by**(*cases a, auto*)

with $PTrans$ $xFreshP$ **obtain** $P'' P'''$ **where** $PChain: P \parallel !P \Rightarrow_\tau P''$
and $P''Trans: P'' \mapsto a' \langle \nu x \rangle \prec P'''$
and $P'''Chain: P''' \Rightarrow_\tau P'$
by(*force dest: transitionE*)

show $!P \Rightarrow_l a \langle x \rangle \prec P'$
proof(*cases P'' = P \parallel !P*)
assume $P'' = P \parallel !P$
moreover with $P''Trans$ **have** $!P \mapsto a' \langle \nu x \rangle \prec P'''$ **by**(*blast intro: transitions.Bang*)
ultimately show *?thesis* **using** $PChain$ $P'''Chain$ aeq **by**(*simp, rule-tac transitionI, auto*)
next
assume $P'' \neq P \parallel !P$
with $PChain$ **have** $!P \Rightarrow_\tau P''$ **by**(*rule bangChain*)
with $P''Trans$ $P'''Chain$ aeq **show** *?thesis* **by**(*blast intro: transitionI*)
qed
next
fix α $P' P$
assume $P \parallel !P \Rightarrow_l \alpha \prec P'$

then obtain $P'' P'''$ **where** $PChain: P \parallel !P \Rightarrow_\tau P''$
and $P''Trans: P'' \mapsto \alpha \prec P'''$
and $P'''Chain: P''' \Rightarrow_\tau P'$
by(*force dest: transitionE*)

show $!P \Rightarrow_l \alpha \prec P'$
proof(*cases P'' = P \parallel !P*)
assume $P'' = P \parallel !P$
moreover with $P''Trans$ **have** $!P \mapsto \alpha \prec P'''$ **by**(*blast intro: transitions.Bang*)
ultimately show *?thesis* **using** $PChain$ $P'''Chain$ **by**(*rule-tac transitionI, auto*)
next
assume $P'' \neq P \parallel !P$
with $PChain$ **have** $!P \Rightarrow_\tau P''$ **by**(*rule bangChain*)

```

    with P''Trans P'''Chain show ?thesis by(blast intro: transitionI)
  qed
next
assume P || !P ==>_l u in P''->a<x> < P'

then obtain P''' where PChain: P || !P ==>_tau P'''
  and P'''Trans: P''' ->a<x> < P''
  and P''Chain: P''[x::=u] ==>_tau P'
  by(force dest: transitionE)

show !P ==>_l u in P''->a<x> < P'
proof(cases P''' = P || !P)
  assume P''' = P || !P
  moreover with P'''Trans have !P ->a<x> < P'' by(blast intro: transi-
tions.Bang)
  ultimately show ?thesis using PChain P''Chain by(rule-tac transitionI, auto)
next
assume P''' ≠ P || !P
with PChain have !P ==>_tau P''' by(rule bangChain)
with P'''Trans P''Chain show ?thesis by(blast intro: transitionI)
qed
qed

```

lemma *tauTransitionChain*:

```

fixes P :: pi
and P' :: pi

```

```

assumes P ==>_l tau < P'

```

```

shows P ==>_tau P'

```

using *assms*

by(*auto simp add: transition-def residualInject*)

lemma *chainTransitionAppend*:

```

fixes P :: pi
and P' :: pi
and Rs :: residual
and a :: name
and x :: name
and P'' :: pi
and u :: name
and P''' :: pi
and alpha :: freeRes

```

```

shows P ==>_tau P' ==> P' ==>_l Rs ==> P ==>_l Rs

```

```

and P ==>_tau P'' ==> P'' ==>_l u in P'''->a<x> < P' ==> P ==>_l u in P'''->a<x>
< P'

```

```

and P ==>_l a<vx> < P'' ==> P'' ==>_tau P' ==> x # P ==> P ==>_l a<vx> < P'

```

and $P \Rightarrow_l u$ in $P''' \rightarrow a \langle x \rangle \prec P'' \Rightarrow P'' \Rightarrow_\tau P' \Rightarrow P \Rightarrow_l u$ in $P''' \rightarrow a \langle x \rangle \prec P'$
and $P \Rightarrow_l \alpha \prec P'' \Rightarrow P'' \Rightarrow_\tau P' \Rightarrow P \Rightarrow_l \alpha \prec P'$
proof –
assume $P \Rightarrow_\tau P'$ **and** $P' \Rightarrow_l Rs$
thus $P \Rightarrow_l Rs$
by(*auto simp add: transition-def residualInject*) (*blast dest: rtrancl-trans*)+
next
assume $P \Rightarrow_\tau P''$ **and** $P'' \Rightarrow_l u$ in $P''' \rightarrow a \langle x \rangle \prec P'$
thus $P \Rightarrow_l u$ in $P''' \rightarrow a \langle x \rangle \prec P'$
apply(*auto simp add: inputTransition-def residualInject*)
by(*blast dest: rtrancl-trans*)+
next
assume $PTrans: P \Rightarrow_l a \langle \nu x \rangle \prec P''$
assume $P''Chain: P'' \Rightarrow_\tau P'$
assume $xFreshP: x \# P$

from $PTrans$ $xFreshP$ **obtain** $P''' P''''$ **where** $PChain: P \Rightarrow_\tau P'''$
and $P'''Trans: P''' \mapsto a \langle \nu x \rangle \prec P''''$
and $P''''Chain: P'''' \Rightarrow_\tau P''$

by(*blast dest: transitionE*)

from $P''''Chain$ $P''Chain$ **have** $P'''' \Rightarrow_\tau P'$ **by** *auto*
with $PChain$ $P'''Trans$ **show** $P \Rightarrow_l a \langle \nu x \rangle \prec P'$ **by**(*rule transitionI*)
next
assume $PTrans: P \Rightarrow_l u$ in $P''' \rightarrow a \langle x \rangle \prec P''$
assume $P''Chain: P'' \Rightarrow_\tau P'$

from $PTrans$ **obtain** P'''' **where** $PChain: P \Rightarrow_\tau P''''$
and $P''''Trans: P'''' \mapsto a \langle x \rangle \prec P'''$
and $P'''Chain: P'''[x::=u] \Rightarrow_\tau P''$

by(*blast dest: transitionE*)

from $P'''Chain$ $P''Chain$ **have** $P'''[x::=u] \Rightarrow_\tau P'$ **by** *auto*
with $PChain$ $P''''Trans$ **show** $P \Rightarrow_l u$ in $P''' \rightarrow a \langle x \rangle \prec P'$ **by**(*blast intro: transitionI*)
next
assume $PTrans: P \Rightarrow_l \alpha \prec P''$
assume $P''Chain: P'' \Rightarrow_\tau P'$

from $PTrans$ **obtain** $P''' P''''$ **where** $PChain: P \Rightarrow_\tau P'''$
and $P'''Trans: P''' \mapsto \alpha \prec P''''$
and $P''''Chain: P'''' \Rightarrow_\tau P''$

by(*blast dest: transitionE*)

from $P''''Chain$ $P''Chain$ **have** $P'''' \Rightarrow_\tau P'$ **by** *auto*
with $PChain$ $P'''Trans$ **show** $P \Rightarrow_l \alpha \prec P'$ **by**(*rule transitionI*)
qed

lemma *freshInputTransition*:

fixes $P :: pi$
and $a :: name$
and $x :: name$
and $u :: name$
and $P'' :: pi$
and $P' :: pi$
and $c :: name$

assumes $PTrans: P \Longrightarrow_l u \text{ in } P'' \rightarrow a \langle x \rangle \prec P'$
and $cFreshP: c \# P$
and $cinequ: c \neq u$

shows $c \# P'$

proof –

from $PTrans$ **obtain** P''' **where** $PChain: P \Longrightarrow_\tau P'''$
and $P'''Trans: P''' \mapsto a \langle x \rangle \prec P''$
and $P''Chain: P''[x::=u] \Longrightarrow_\tau P'$
by(blast dest: transitionE)

from $PChain$ $cFreshP$ **have** $cFreshP''': c \# P'''$ **by**(rule freshChain)
show $c \# P'$

proof(cases $x=c$)

assume $xeqc: x=c$

from $cinequ$ **have** $c \# P''[c::=u]$ **apply** – **by**(rule fresh-fact2)

with $P''Chain$ $xeqc$ **show** $?thesis$ **by**(force intro: freshChain)

next

assume $xineqc: x \neq c$

with $P'''Trans$ $cFreshP'''$ **have** $c \# P''$ **by**(blast dest: freshBoundDerivative)

with $cinequ$ **have** $c \# P''[x::=u]$

apply –

apply(rule fresh-fact1)

by simp

with $P''Chain$ **show** $?thesis$ **by**(rule freshChain)

qed

qed

lemma *freshBoundOutputTransition*:

fixes $P :: pi$
and $a :: name$
and $x :: name$
and $P' :: pi$
and $c :: name$

assumes $PTrans: P \Longrightarrow_l a \langle \nu x \rangle \prec P'$
and $cFreshP: c \# P$
and $cineqx: c \neq x$

shows $c \# P'$
proof –
have *Goal*: $\bigwedge P a x P' c. \llbracket P \Longrightarrow_l a \langle \nu x \rangle \prec P'; x \# P; c \# P; c \neq x \rrbracket \Longrightarrow c \# P'$
proof –
fix $P a x P' c$
assume $PTrans: P \Longrightarrow_l a \langle \nu x \rangle \prec P'$
assume $xFreshP: x \# P$
assume $cFreshP: (c::name) \# P$
assume $cineqx: c \neq x$

from $PTrans$ $xFreshP$ **obtain** $P'' P'''$ **where** $PTrans: P \Longrightarrow_\tau P''$
and $P''Trans: P'' \longmapsto a \langle \nu x \rangle \prec P'''$
and $P'''Trans: P''' \Longrightarrow_\tau P'$
by(*blast dest: transitionE*)

from $PTrans$ $cFreshP$ **have** $c \# P''$ **by**(*rule freshChain*)
with $P''Trans$ $cineqx$ **have** $c \# P'''$ **by**(*blast dest: Late-Semantics.freshBoundDerivative*)
with $P'''Trans$ **show** $c \# P'$ **by**(*rule freshChain*)
qed

have $\exists d::name. d \# (P, P', c)$ **by**(*blast intro: name-exists-fresh*)
then obtain $d::name$ **where** $dFreshP: d \# P$ **and** $dFreshP': d \# P'$ **and** $cineqd: c \neq d$
by(*force simp add: fresh-prod*)

from $PTrans$ $dFreshP'$ **have** $P \Longrightarrow_l a \langle \nu d \rangle \prec ((x, d) \cdot P')$ **by**(*simp add: alphaBoundResidual*)
hence $c \# ((x, d) \cdot P')$ **using** $dFreshP$ $cFreshP$ $cineqd$ **by**(*rule Goal*)
with $cineqd$ $cineqx$ **show** *?thesis* **by**(*simp add: name-fresh-left name-calc*)
qed

lemma *freshTauTransition*:
fixes $P :: pi$
and $c :: name$

assumes $PTrans: P \Longrightarrow_l \tau \prec P'$
and $cFreshP: c \# P$

shows $c \# P'$
proof –
from $PTrans$ **have** $P \Longrightarrow_\tau P'$ **by**(*rule tauTransitionChain*)
thus *?thesis* **using** $cFreshP$ **by**(*rule freshChain*)
qed

lemma *freshOutputTransition*:
fixes $P :: pi$
and $a :: name$
and $b :: name$
and $P' :: pi$

and $c :: \text{name}$

assumes $P\text{Trans}: P \Longrightarrow_l a[b] \prec P'$
and $c\text{Fresh}P: c \# P$

shows $c \# P'$

proof –

from $P\text{Trans}$ **obtain** $P'' P'''$ **where** $P\text{Trans}: P \Longrightarrow_\tau P''$
and $P''\text{Trans}: P'' \mapsto a[b] \prec P'''$
and $P'''\text{Trans}: P''' \Longrightarrow_\tau P'$

by(*blast dest: transitionE*)

from $P\text{Trans}$ $c\text{Fresh}P$ **have** $c \# P''$ **by**(*rule freshChain*)
with $P''\text{Trans}$ **have** $c \# P'''$ **by**(*blast dest: Late-Semantics.freshFreeDerivative*)
with $P'''\text{Trans}$ **show** $?thesis$ **by**(*rule freshChain*)

qed

lemma *eqvtI*:

fixes $P :: pi$
and $Rs :: \text{residual}$
and $perm :: \text{name prm}$
and $u :: \text{name}$
and $P'' :: pi$
and $a :: \text{name}$
and $x :: \text{name}$
and $P' :: pi$

shows $P \Longrightarrow_l Rs \Longrightarrow (perm \cdot P) \Longrightarrow_l (perm \cdot Rs)$
and $P \Longrightarrow_l u$ **in** $P'' \mapsto a \langle x \rangle \prec P' \Longrightarrow (perm \cdot P) \Longrightarrow_l (perm \cdot u)$ **in** $(perm \cdot P'' \mapsto (perm \cdot a) \langle perm \cdot x \rangle \prec (perm \cdot P'))$

proof –

assume $P \Longrightarrow_l Rs$
thus $(perm \cdot P) \Longrightarrow_l (perm \cdot Rs)$
proof(*nominal-induct Rs avoiding: P rule: residual.strong-inducts*)
case(*BoundR a x P' P*)
have $P\text{Trans}: P \Longrightarrow_l a \langle x \rangle \prec P'$ **by** *fact*
moreover then obtain b **where** $aeqb: a = \text{BoundOutputS } b$ **by**(*cases a, auto*)
moreover have $x \# P$ **by** *fact*
ultimately obtain $P'' P'''$ **where** $P\text{Trans}: P \Longrightarrow_\tau P''$
and $P''\text{Trans}: P'' \mapsto b \langle \nu x \rangle \prec P'''$
and $P'''\text{Trans}: P''' \Longrightarrow_\tau P'$

by(*blast dest: transitionE*)

from $P\text{Trans}$ **have** $(perm \cdot P) \Longrightarrow_\tau (perm \cdot P'')$ **by**(*rule eqvtChainI*)
moreover from $P''\text{Trans}$ **have** $(perm \cdot P'') \mapsto (perm \cdot (b \langle \nu x \rangle \prec P'''))$
by(*rule eqvts*)
moreover from $P'''\text{Trans}$ **have** $(perm \cdot P''') \Longrightarrow_\tau (perm \cdot P')$ **by**(*rule eqvtChainI*)
ultimately show $?case$ **using** $aeqb$ **by**(*force intro: transitionI*)

```

next
  case(FreeR  $\alpha$   $P'$   $P$ )
  have  $P \Rightarrow_l \alpha \prec P'$  by fact
  then obtain  $P''$   $P'''$  where  $PTrans: P \Rightarrow_\tau P''$ 
    and  $P''Trans: P'' \mapsto \alpha \prec P'''$ 
    and  $P'''Trans: P''' \Rightarrow_\tau P'$ 
  by(blast dest: transitionE)

  from  $PTrans$  have  $(perm \cdot P) \Rightarrow_\tau (perm \cdot P'')$  by(rule eqvtChainI)
  moreover from  $P''Trans$  have  $(perm \cdot P'') \mapsto (perm \cdot (\alpha \prec P'''))$ 
  by(rule eqvts)
  moreover from  $P'''Trans$  have  $(perm \cdot P''') \Rightarrow_\tau (perm \cdot P')$  by(rule
eqvtChainI)
  ultimately show  $?case$  by(force intro: transitionI)
  qed
next
  assume  $P \Rightarrow_l u$  in  $P'' \rightarrow a \langle x \rangle \prec P'$ 

  then obtain  $P'''$  where  $PChain: P \Rightarrow_\tau P'''$ 
    and  $P'''Trans: P''' \mapsto a \langle x \rangle \prec P''$ 
    and  $P''Chain: P''[x::=u] \Rightarrow_\tau P'$ 
  by(blast dest: transitionE)

  from  $PChain$  have  $(perm \cdot P) \Rightarrow_\tau (perm \cdot P''')$  by(rule eqvtChainI)
  moreover from  $P'''Trans$  have  $(perm \cdot P''') \mapsto (perm \cdot (a \langle x \rangle \prec P''))$ 
  by(rule eqvts)
  moreover from  $P''Chain$  have  $(perm \cdot P''[x::=u]) \Rightarrow_\tau (perm \cdot P')$  by(rule
eqvtChainI)
  ultimately show  $(perm \cdot P) \Rightarrow_l (perm \cdot u)$  in  $(perm \cdot P'') \rightarrow (perm \cdot a) \langle (perm$ 
   $\cdot x) \rangle \prec (perm \cdot P')$ 
  by(force intro: transitionI simp add: eqvt-subs[THEN sym] perm-bij)
  qed

lemmas freshTransition = freshBoundOutputTransition freshOutputTransition
freshInputTransition freshTauTransition

end

theory Weak-Late-Semantics
  imports Weak-Late-Step-Semantics
  begin

  definition weakTransition :: (pi  $\times$  residual) set
    where weakTransition  $\equiv$  Weak-Late-Step-Semantics.transition  $\cup$   $\{x. \exists P. x =$ 
     $(P, \tau \prec P)\}$ 

  abbreviation weakLateTransition-judge :: pi  $\Rightarrow$  residual  $\Rightarrow$  bool ( $\prec - \Rightarrow_l \hat{-} \rightarrow$  [80,
  80] 80)

```

where $P \Rightarrow_l \hat{R}s \equiv (P, R_s) \in \text{weakTransition}$

lemma *transitionI*:

fixes $P :: pi$
and $R_s :: residual$
and $P' :: pi$

shows $P \Rightarrow_l R_s \Rightarrow P \Rightarrow_l \hat{R}s$
and $P \Rightarrow_l \hat{\tau} \prec P$

proof –
assume $P \Rightarrow_l R_s$
thus $P \Rightarrow_l \hat{R}s$ **by**(*simp add: weakTransition-def*)
next
show $P \Rightarrow_l \hat{\tau} \prec P$ **by**(*simp add: weakTransition-def*)
qed

lemma *transitionCases*[*consumes 1, case-names Step Stay*]:

fixes $P :: pi$
and $R_s :: residual$
and $P' :: pi$

assumes $P \Rightarrow_l \hat{R}s$
and $P \Rightarrow_l R_s \Rightarrow F R_s$
and $R_s = \tau \prec P \Rightarrow F (\tau \prec P)$

shows $F R_s$

using *assms*
by(*auto simp add: weakTransition-def*)

lemma *singleActionChain*:

fixes $P :: pi$
and $\alpha :: freeRes$
and $P' :: pi$

assumes $P \mapsto \alpha \prec P'$

shows $P \Rightarrow_l \hat{(\alpha \prec P')}$

using *assms*
by(*auto intro: Weak-Late-Step-Semantics.singleActionChain simp add: weakTransition-def*)

lemma *Tau*:

fixes $P :: pi$

shows $\tau.(P) \Rightarrow_l \hat{\tau} \prec P$

by(*auto intro: Weak-Late-Step-Semantics.Tau simp add: weakTransition-def*)

lemma *Output*:

```

fixes a :: name
and b :: name
and P :: pi

shows a{b}.P  $\Longrightarrow_i \hat{a}[b] \prec P$ 
by(auto intro: Weak-Late-Step-Semantics.Output
    simp add: weakTransition-def)

lemma Match:
fixes a :: name
and P :: pi
and b :: name
and x :: name
and P' :: pi
and  $\alpha$  :: freeRes

shows P  $\Longrightarrow_i \hat{b}\langle\nu x\rangle \prec P' \Longrightarrow [a\hat{\ }a]P \Longrightarrow_i \hat{b}\langle\nu x\rangle \prec P'$ 
and P  $\Longrightarrow_i \hat{\alpha} \prec P' \Longrightarrow P \neq P' \Longrightarrow [a\hat{\ }a]P \Longrightarrow_i \hat{\alpha} \prec P'$ 
by(auto simp add: residual.inject weakTransition-def intro: Weak-Late-Step-Semantics.Match)

lemma Mismatch:
fixes a :: name
and c :: name
and P :: pi
and b :: name
and x :: name
and P' :: pi
and  $\alpha$  :: freeRes

shows  $\llbracket P \Longrightarrow_i \hat{b}\langle\nu x\rangle \prec P'; a \neq c \rrbracket \Longrightarrow [a\neq c]P \Longrightarrow_i \hat{b}\langle\nu x\rangle \prec P'$ 
and P  $\Longrightarrow_i \hat{\alpha} \prec P' \Longrightarrow P \neq P' \Longrightarrow a \neq c \Longrightarrow [a\neq c]P \Longrightarrow_i \hat{\alpha} \prec P'$ 
by(auto simp add: residual.inject weakTransition-def intro: Weak-Late-Step-Semantics.Mismatch)

lemma Open:
fixes P :: pi
and a :: name
and b :: name
and P' :: pi

assumes Trans: P  $\Longrightarrow_i \hat{a}[b] \prec P'$ 
and aInEqb: a  $\neq$  b

shows  $\langle\nu b\rangle P \Longrightarrow_i \hat{a}\langle\nu b\rangle \prec P'$ 
using assms
by(auto intro: Weak-Late-Step-Semantics.Open
    simp add: weakTransition-def residual.inject)

lemma Par1B:
fixes P :: pi

```

```

and a :: name
and x :: name
and P' :: pi

assumes PTrans: P  $\Longrightarrow_l \hat{a} \langle \nu x \rangle \prec P'$ 
and xFreshQ: x  $\#$  Q

shows P  $\parallel$  Q  $\Longrightarrow_l \hat{a} \langle \nu x \rangle \prec (P' \parallel Q)$ 
using assms
by(auto intro: Weak-Late-Step-Semantics.Par1B
  simp add: weakTransition-def residual.inject)

lemma Par1F:
  fixes P :: pi
  and  $\alpha$  :: freeRes
  and P' :: pi

  assumes PTrans: P  $\Longrightarrow_l \hat{\alpha} \prec P'$ 

  shows P  $\parallel$  Q  $\Longrightarrow_l \hat{\alpha} \prec (P' \parallel Q)$ 
using assms
by(auto intro: Weak-Late-Step-Semantics.Par1F
  simp add: weakTransition-def residual.inject)

lemma Par2B:
  fixes Q :: pi
  and a :: name
  and x :: name
  and Q' :: pi

  assumes QTrans: Q  $\Longrightarrow_l \hat{a} \langle \nu x \rangle \prec Q'$ 
  and xFreshP: x  $\#$  P

  shows P  $\parallel$  Q  $\Longrightarrow_l \hat{a} \langle \nu x \rangle \prec (P \parallel Q')$ 
using assms
by(auto intro: Weak-Late-Step-Semantics.Par2B
  simp add: weakTransition-def residual.inject)

lemma Par2F:
  fixes Q :: pi
  and  $\alpha$  :: freeRes
  and Q' :: pi

  assumes QTrans: Q  $\Longrightarrow_l \hat{\alpha} \prec Q'$ 

  shows P  $\parallel$  Q  $\Longrightarrow_l \hat{\alpha} \prec (P \parallel Q')$ 
using assms
by(auto intro: Weak-Late-Step-Semantics.Par2F
  simp add: weakTransition-def residual.inject)

```

```

lemma Comm1:
  fixes  $P :: pi$ 
  and  $a :: name$ 
  and  $b :: name$ 
  and  $P'' :: pi$ 
  and  $P' :: pi$ 
  and  $Q :: pi$ 
  and  $Q' :: pi$ 

  assumes  $PTrans: P \Longrightarrow_l b \text{ in } P'' \rightarrow a \langle x \rangle \prec P'$ 
  and  $QTrans: Q \Longrightarrow_l \hat{a}[b] \prec Q'$ 

  shows  $P \parallel Q \Longrightarrow_l \hat{\tau} \prec P' \parallel Q'$ 
using assms
by(auto intro: Weak-Late-Step-Semantics.Comm1
  simp add: weakTransition-def residual.inject)

```

```

lemma Comm2:
  fixes  $P :: pi$ 
  and  $a :: name$ 
  and  $b :: name$ 
  and  $Q'' :: pi$ 
  and  $P' :: pi$ 
  and  $Q :: pi$ 
  and  $Q' :: pi$ 

  assumes  $PTrans: P \Longrightarrow_l \hat{a}[b] \prec P'$ 
  and  $QTrans: Q \Longrightarrow_l b \text{ in } Q'' \rightarrow a \langle x \rangle \prec Q'$ 

  shows  $P \parallel Q \Longrightarrow_l \hat{\tau} \prec P' \parallel Q'$ 
using assms
by(auto intro: Weak-Late-Step-Semantics.Comm2
  simp add: weakTransition-def residual.inject)

```

```

lemma Close1:
  fixes  $P :: pi$ 
  and  $y :: name$ 
  and  $P'' :: pi$ 
  and  $a :: name$ 
  and  $x :: name$ 
  and  $P' :: pi$ 
  and  $Q :: pi$ 
  and  $Q' :: pi$ 

  assumes  $PTrans: P \Longrightarrow_l y \text{ in } P'' \rightarrow a \langle x \rangle \prec P'$ 
  and  $QTrans: Q \Longrightarrow_l \hat{a} \langle \nu y \rangle \prec Q'$ 
  and  $xFreshP: y \# P$ 
  and  $xFreshQ: y \# Q$ 

```

shows $P \parallel Q \Longrightarrow_l \hat{\tau} \prec \langle \nu y \rangle (P' \parallel Q')$
using *assms*
by(*auto intro: Weak-Late-Step-Semantics.Close1*
simp add: weakTransition-def residual.inject)

lemma *Close2*:

fixes $P :: pi$
and $a :: name$
and $x :: name$
and $P' :: pi$
and $Q :: pi$
and $y :: name$
and $Q'' :: pi$
and $Q' :: pi$

assumes $PTrans: P \Longrightarrow_l \hat{a} \langle \nu y \rangle \prec P'$
and $QTrans: Q \Longrightarrow_l y \text{ in } Q'' \rightarrow a \langle x \rangle \prec Q'$
and $xFreshP: y \# P$
and $xFreshQ: y \# Q$

shows $P \parallel Q \Longrightarrow_l \hat{\tau} \prec \langle \nu y \rangle (P' \parallel Q')$
using *assms*
by(*auto intro: Weak-Late-Step-Semantics.Close2*
simp add: weakTransition-def residual.inject)

lemma *ResF*:

fixes $P :: pi$
and $\alpha :: freeRes$
and $P' :: pi$
and $x :: name$

assumes $PTrans: P \Longrightarrow_l \hat{\alpha} \prec P'$
and $xFreshAlpha: x \# \alpha$

shows $\langle \nu x \rangle P \Longrightarrow_l \hat{\alpha} \prec \langle \nu x \rangle P'$
using *assms*
by(*auto intro: Weak-Late-Step-Semantics.ResF*
simp add: weakTransition-def residual.inject)

lemma *ResB*:

fixes $P :: pi$
and $a :: name$
and $x :: name$
and $P' :: pi$
and $y :: name$

assumes $PTrans: P \Longrightarrow_l \hat{a} \langle \nu x \rangle \prec P'$
and $yineqa: y \neq a$

```

and   yineqx:  $y \neq x$ 
and   xFreshP:  $x \# P$ 

shows  $\langle \nu y \rangle P \Longrightarrow_l \hat{a} \langle \nu x \rangle \prec (\langle \nu y \rangle P')$ 
using assms
by(auto intro: Weak-Late-Step-Semantics.ResB
    simp add: weakTransition-def residual.inject)

lemma Bang:
  fixes  $P :: pi$ 
  and    $Rs :: residual$ 

  assumes  $P \parallel !P \Longrightarrow_l \hat{Rs}$ 
  and      $Rs \neq \tau \prec P \parallel !P$ 

  shows  $!P \Longrightarrow_l \hat{Rs}$ 
using assms
by(auto intro: Weak-Late-Step-Semantics.Bang
    simp add: weakTransition-def residual.inject)

lemma tauTransitionChain:
  fixes  $P :: pi$ 
  and    $P' :: pi$ 

  assumes  $P \Longrightarrow_l \hat{\tau} \prec P'$ 

  shows  $P \Longrightarrow_\tau P'$ 
using assms
by(auto intro: Weak-Late-Step-Semantics.tauTransitionChain
    simp add: weakTransition-def residual.inject transition-def)

lemma chainTransitionAppend:
  fixes  $P :: pi$ 
  and    $P' :: pi$ 
  and    $Rs :: residual$ 
  and    $a :: name$ 
  and    $x :: name$ 
  and    $P'' :: pi$ 
  and    $\alpha :: freeRes$ 

  shows  $P \Longrightarrow_\tau P' \Longrightarrow P' \Longrightarrow_l \hat{Rs} \Longrightarrow P \Longrightarrow_l \hat{Rs}$ 
  and    $P \Longrightarrow_l \hat{a} \langle \nu x \rangle \prec P'' \Longrightarrow P'' \Longrightarrow_\tau P' \Longrightarrow x \# P \Longrightarrow P \Longrightarrow_l \hat{a} \langle \nu x \rangle \prec$ 
   $P'$ 
  and    $P \Longrightarrow_l \hat{\alpha} \prec P'' \Longrightarrow P'' \Longrightarrow_\tau P' \Longrightarrow P \Longrightarrow_l \hat{\alpha} \prec P'$ 
proof –
  assume  $P \Longrightarrow_\tau P'$  and  $P' \Longrightarrow_l \hat{Rs}$ 
  thus  $P \Longrightarrow_l \hat{Rs}$ 
  by(auto intro: Weak-Late-Step-Semantics.chainTransitionAppend
    Weak-Late-Step-Semantics.tauActionChain)

```

```

      simp add: weakTransition-def residual.inject)
next
  assume  $P \Longrightarrow_l \hat{a} \langle \nu x \rangle \prec P''$  and  $P'' \Longrightarrow_\tau P'$  and  $x \# P$ 
  thus  $P \Longrightarrow_l \hat{a} \langle \nu x \rangle \prec P'$ 
  by(auto intro: Weak-Late-Step-Semantics.chainTransitionAppend
      simp add: weakTransition-def residual.inject)
next
  assume  $P \Longrightarrow_l \hat{\alpha} \prec P''$  and  $P'' \Longrightarrow_\tau P'$ 
  thus  $P \Longrightarrow_l \hat{\alpha} \prec P'$ 
  apply(case-tac  $P''=P'$ )
  by(auto dest: Weak-Late-Step-Semantics.chainTransitionAppend
      Weak-Late-Step-Semantics.tauActionChain
      simp add: weakTransition-def residual.inject)
qed

lemma weakEqWeakTransitionAppend:
  fixes  $P :: pi$ 
  and  $P' :: pi$ 
  and  $\alpha :: freeRes$ 
  and  $P'' :: pi$ 

  assumes  $PTrans: P \Longrightarrow_l \tau \prec P'$ 
  and  $P'Trans: P' \Longrightarrow_l \hat{\alpha} \prec P''$ 

  shows  $P \Longrightarrow_l \hat{\alpha} \prec P''$ 
proof(cases  $\alpha=\tau$ )
  assume alphaEqTau:  $\alpha = \tau$ 
  with  $P'Trans$  have  $P' \Longrightarrow_\tau P''$  by(blast intro: tauTransitionChain)
  with  $PTrans$  alphaEqTau show ?thesis
  by(blast intro: Weak-Late-Step-Semantics.chainTransitionAppend)
next
  assume alphaIneqTau:  $\alpha \neq \tau$ 
  from  $PTrans$  have  $P \Longrightarrow_\tau P'$  by(rule Weak-Late-Step-Semantics.tauTransitionChain)
  moreover from  $P'Trans$  alphaIneqTau have  $P' \Longrightarrow_l \hat{\alpha} \prec P''$ 
  by(auto simp add: weakTransition-def residual.inject)
  ultimately show ?thesis
  by(rule Weak-Late-Step-Semantics.chainTransitionAppend)
qed

lemma freshBoundOutputTransition:
  fixes  $P :: pi$ 
  and  $a :: name$ 
  and  $x :: name$ 
  and  $P' :: pi$ 
  and  $c :: name$ 

  assumes  $PTrans: P \Longrightarrow_l \hat{a} \langle \nu x \rangle \prec P'$ 
  and  $cFreshP: c \# P$ 
  and  $cineqx: c \neq x$ 

```

shows $c \# P'$
using *assms*
by(*auto intro: Weak-Late-Step-Semantics.freshBoundOutputTransition*
simp add: weakTransition-def residual.inject)

lemma *freshTauTransition:*

fixes $P :: pi$
and $c :: name$

assumes $PTrans: P \Longrightarrow_l \hat{\tau} \prec P'$
and $cFreshP: c \# P$

shows $c \# P'$
using *assms*
by(*auto intro: Weak-Late-Step-Semantics.freshTauTransition*
simp add: weakTransition-def residual.inject)

lemma *freshOutputTransition:*

fixes $P :: pi$
and $a :: name$
and $b :: name$
and $P' :: pi$
and $c :: name$

assumes $PTrans: P \Longrightarrow_l \hat{a}[b] \prec P'$
and $cFreshP: c \# P$

shows $c \# P'$
using *assms*
by(*auto intro: Weak-Late-Step-Semantics.freshOutputTransition*
simp add: weakTransition-def residual.inject)

lemma *eqvI:*

fixes $P :: pi$
and $Rs :: residual$
and $perm :: name prm$

assumes $P \Longrightarrow_l \hat{Rs}$

shows $(perm \cdot P) \Longrightarrow_l \hat{(perm \cdot Rs)}$
using *assms*
by(*auto intro: Weak-Late-Step-Semantics.eqvI*
simp add: weakTransition-def residual.inject)

lemma *freshInputTransition:*

fixes $P :: pi$
and $a :: name$
and $b :: name$

```

and P' :: pi
and c :: name

assumes PTrans: P  $\Longrightarrow_l \hat{a} \langle b \rangle \prec P'$ 
and cFreshP: c  $\# P$ 
and cineqb: c  $\neq b$ 

shows c  $\# P'$ 
using assms
by(auto intro: Weak-Late-Step-Semantics.freshInputTransition
    simp add: weakTransition-def residual.inject)

lemmas freshTransition = freshBoundOutputTransition freshOutputTransition
    freshInputTransition freshTauTransition

end

theory Weak-Late-Sim
imports Weak-Late-Semantics Strong-Late-Sim
begin

definition weakSimAct :: pi  $\Rightarrow$  residual  $\Rightarrow$  ('a::fs-name)  $\Rightarrow$  (pi  $\times$  pi) set  $\Rightarrow$  bool
where
    weakSimAct P Rs C Rel  $\equiv$  ( $\forall Q' a x. Rs = a \langle \nu x \rangle \prec Q' \longrightarrow x \# C \longrightarrow (\exists P'. P \Longrightarrow_l \hat{a} \langle \nu x \rangle \prec P' \wedge (P', Q') \in Rel)$ )  $\wedge$ 
    ( $\forall Q' a x. Rs = a \langle x \rangle \prec Q' \longrightarrow x \# C \longrightarrow (\exists P''. \forall u. \exists P'. P \Longrightarrow_l u \text{ in } P'' \rightarrow a \langle x \rangle \prec P' \wedge (P', Q'[x::=u]) \in Rel)$ )  $\wedge$ 
    ( $\forall Q' \alpha. Rs = \alpha \prec Q' \longrightarrow (\exists P'. P \Longrightarrow_l \hat{\alpha} \prec P' \wedge (P', Q') \in Rel)$ )

definition weakSimAux :: pi  $\Rightarrow$  (pi  $\times$  pi) set  $\Rightarrow$  pi  $\Rightarrow$  bool where
    weakSimAux P Rel Q  $\equiv$  ( $\forall Q' a x. (Q \mapsto a \langle \nu x \rangle \prec Q' \wedge x \# P) \longrightarrow (\exists P'. P \Longrightarrow_l \hat{a} \langle \nu x \rangle \prec P' \wedge (P', Q') \in Rel)$ )  $\wedge$ 
    ( $\forall Q' a x. (Q \mapsto a \langle x \rangle \prec Q' \wedge x \# P) \longrightarrow (\exists P''. \forall u. \exists P'. P \Longrightarrow_l u \text{ in } P'' \rightarrow a \langle x \rangle \prec P' \wedge (P', Q'[x::=u]) \in Rel)$ )  $\wedge$ 
    ( $\forall Q' \alpha. Q \mapsto \alpha \prec Q' \longrightarrow (\exists P'. P \Longrightarrow_l \hat{\alpha} \prec P' \wedge (P', Q') \in Rel)$ )

definition weakSimulation :: pi  $\Rightarrow$  (pi  $\times$  pi) set  $\Rightarrow$  pi  $\Rightarrow$  bool ( $\langle \prec \rightsquigarrow \hat{\langle - \rangle} \prec - \rangle \rightarrow [80, 80, 80] 80$ ) where
    P  $\rightsquigarrow \hat{\langle Rel \rangle} Q \equiv (\forall Rs. Q \mapsto Rs \longrightarrow weakSimAct P Rs P Rel)$ 

lemmas simDef = weakSimAct-def weakSimulation-def

lemma weakSimAux P Rel Q = weakSimulation P Rel Q
by(auto simp add: weakSimAux-def simDef)

lemma monotonic:
fixes A :: (pi  $\times$  pi) set

```

```

and B :: (pi × pi) set
and P :: pi
and P' :: pi

assumes P ~>^ <A> P'
and A ⊆ B

shows P ~>^ <B> P'
using assms
apply(auto simp add: simDef)
apply blast
apply(erule-tac x=a<x> < Q' in allE)
apply(clarsimp)
apply(rotate-tac 4)
apply(erule-tac x=Q' in allE)
apply(erule-tac x=a in allE)
apply(erule-tac x=x in allE)
by blast+

lemma simCasesCont[consumes 1, case-names Bound Input Free]:
  fixes P :: pi
  and Q :: pi
  and Rel :: (pi × pi) set
  and C :: 'a::fs-name

  assumes Eqvt: eqvt Rel
  and Bound:  $\bigwedge Q' a x. \llbracket x \# C; Q \mapsto a \langle \nu x \rangle \prec Q' \rrbracket \implies \exists P'. P \implies_l a \langle \nu x \rangle \prec P' \wedge (P', Q') \in \text{Rel}$ 
  and Input:  $\bigwedge Q' a x. \llbracket x \# C; Q \mapsto a \langle x \rangle \prec Q' \rrbracket \implies \exists P''. \forall u. \exists P'. P \implies_l u \text{ in } P'' \rightarrow a \langle x \rangle \prec P' \wedge (P', Q'[x::=u]) \in \text{Rel}$ 
  and Free:  $\bigwedge Q' \alpha. Q \mapsto \alpha \prec Q' \implies (\exists P'. P \implies_l \alpha \prec P' \wedge (P', Q') \in \text{Rel})$ 

  shows P ~>^ <Rel> Q
using Free
proof(auto simp add: simDef)
  fix Q' a x
  assume xFreshP: (x::name) # P
  assume Trans: Q ↦ a⟨νx⟩ ≺ Q'
  have ∃ c::name. c # (P, Q', x, C) by(blast intro: name-exists-fresh)
  then obtain c::name where cFreshP: c # P and cFreshQ': c # Q' and cFreshC: c # C

  and cineqx: c ≠ x
  by(force simp add: fresh-prod)

  from Trans cFreshQ' have Q ↦ a⟨νc⟩ ≺ (([x, c]) · Q') by(simp add: algebraBoundResidual)
  with cFreshC have ∃ P'. P ⇒l a⟨νc⟩ ≺ P' ∧ (P', [[x, c]] · Q') ∈ Rel
  by(rule Bound)

```

then obtain P' **where** $PTrans: P \Longrightarrow_l \hat{a} \langle \nu c \rangle \prec P'$ **and** $P'RelQ': (P', [(x, c)] \cdot Q') \in Rel$
by *blast*

from $PTrans$ $xFreshP$ *cineqx* **have** $xFreshP': x \# P'$ **by**(*force dest: freshTransition*)

with $PTrans$ **have** $P \Longrightarrow_l \hat{a} \langle \nu x \rangle \prec ([(x, c)] \cdot P')$ **by**(*simp add: alphaBoundResidual name-swap*)

moreover have $([(x, c)] \cdot P', Q') \in Rel$ (**is** *?goal*)

proof –

from $Eqvt$ $P'RelQ'$ **have** $([(x, c)] \cdot P', [(x, c)] \cdot [(x, c)] \cdot Q') \in Rel$
by(*rule eqvtRelI*)

with *cineqx* **show** *?goal* **by**(*simp add: name-calc*)

qed

ultimately show $\exists P'. P \Longrightarrow_l \hat{a} \langle \nu x \rangle \prec P' \wedge (P', Q') \in Rel$ **by** *blast*

next

fix $Q' a x u$

assume $QTrans: Q \mapsto a \langle x \rangle \prec (Q'::pi)$ **and** $xFreshP: x \# P$

have $\exists c::name. c \# (P, Q', C, x)$ **by**(*blast intro: name-exists-fresh*)

then obtain $c::name$ **where** $cFreshP: c \# P$ **and** $cFreshQ': c \# Q'$ **and** $cFreshC: c \# C$

and *cineqx: c ≠ x*

by(*force simp add: fresh-prod*)

from $QTrans$ $cFreshQ'$ **have** $Q \mapsto a \langle c \rangle \prec ([(x, c)] \cdot Q')$ **by**(*simp add: alphaBoundResidual*)

with $cFreshC$ **have** $\exists P''. \forall u. \exists P'. P \Longrightarrow_l u$ **in** $P'' \rightarrow a \langle c \rangle \prec P' \wedge (P', [(x, c)] \cdot Q')[c::=u] \in Rel$
by(*rule Input*)

then obtain P'' **where** $L1: \forall u. \exists P'. P \Longrightarrow_l u$ **in** $P'' \rightarrow a \langle c \rangle \prec P' \wedge (P', [(x, c)] \cdot Q')[c::=u] \in Rel$ **by** *blast*

have $\forall u. \exists P'. P \Longrightarrow_l u$ **in** $([(c, x)] \cdot P'') \rightarrow a \langle x \rangle \prec P' \wedge (P', Q'[x::=u]) \in Rel$

proof(*auto*)

fix u

from $L1$ **obtain** P' **where** $PTrans: P \Longrightarrow_l u$ **in** $P'' \rightarrow a \langle c \rangle \prec P'$ **and** $P'RelQ': (P', [(x, c)] \cdot Q')[c::=u] \in Rel$
by *blast*

from $PTrans$ $xFreshP$ **have** $P \Longrightarrow_l u$ **in** $([(c, x)] \cdot P'') \rightarrow a \langle x \rangle \prec P'$ **by**(*rule alphaInput*)

moreover from $P'RelQ'$ $cFreshQ'$ **have** $(P', Q'[x::=u]) \in Rel$ **by**(*simp add: renaming[THEN sym] name-swap*)

ultimately show $\exists P'. P \Longrightarrow_l u$ **in** $([(c, x)] \cdot P'') \rightarrow a \langle x \rangle \prec P' \wedge (P', Q'[x::=u]) \in Rel$ **by** *blast*

qed

thus $\exists P''. \forall u. \exists P'. P \Longrightarrow_l u$ in $P'' \rightarrow a \langle x \rangle \prec P' \wedge (P', Q'[x::=u]) \in \text{Rel}$ **by**
blast
qed

lemma *simCases*[*case-names Bound Input Free*]:

fixes $P :: pi$
and $Q :: pi$
and $\text{Rel} :: (pi \times pi)$ *set*
and $C :: 'a::fs\text{-name}$

assumes *Bound*: $\bigwedge Q' a x. \llbracket Q \mapsto a \langle \nu x \rangle \prec Q'; x \# P \rrbracket \Longrightarrow \exists P'. P \Longrightarrow_l \hat{a} \langle \nu x \rangle \prec P' \wedge (P', Q') \in \text{Rel}$
and *Input*: $\bigwedge Q' a x. \llbracket Q \mapsto a \langle x \rangle \prec Q'; x \# P \rrbracket \Longrightarrow \exists P''. \forall u. \exists P'. P \Longrightarrow_l u$ in $P'' \rightarrow a \langle x \rangle \prec P' \wedge (P', Q'[x::=u]) \in \text{Rel}$
and *Free*: $\bigwedge Q' \alpha. Q \mapsto \alpha \prec Q' \Longrightarrow (\exists P'. P \Longrightarrow_l \hat{\alpha} \prec P' \wedge (P', Q') \in \text{Rel})$

shows $P \rightsquigarrow^{\hat{\langle \text{Rel} \rangle}} Q$
using *assms*
by(*auto simp add: simDef*)

lemma *simActBoundCases*[*consumes 1, case-names Input BoundOutput*]:

fixes $P :: pi$
and $a :: \text{subject}$
and $x :: \text{name}$
and $Q' :: pi$
and $C :: 'a::fs\text{-name}$
and $\text{Rel} :: (pi \times pi)$ *set*

assumes *EqvtRel*: *eqvt Rel*
and *DerInput*: $\bigwedge b. a = \text{InputS } b \Longrightarrow (\exists P''. \forall u. \exists P'. (P \Longrightarrow_l u$ in $P'' \rightarrow b \langle x \rangle \prec P') \wedge (P', Q'[x::=u]) \in \text{Rel})$
and *DerBoundOutput*: $\bigwedge b. a = \text{BoundOutputS } b \Longrightarrow (\exists P'. (P \Longrightarrow_l \hat{b} \langle \nu x \rangle \prec P') \wedge (P', Q') \in \text{Rel})$

shows *weakSimAct* $P (a \langle x \rangle \prec Q') P \text{ Rel}$
proof(*simp add: weakSimAct-def fresh-prod, auto*)
fix $Q'' b y$
assume *Eq*: $a \langle x \rangle \prec Q' = b \langle \nu y \rangle \prec Q''$
assume *yFreshP*: $y \# P$

from *Eq* **have** $a = \text{BoundOutputS } b$ **by**(*simp add: residual.inject*)

from *yFreshP DerBoundOutput*[*OF this*] *Eq* **show** $\exists P'. P \Longrightarrow_l \hat{b} \langle \nu y \rangle \prec P' \wedge (P', Q'') \in \text{Rel}$

proof(*cases x=y, auto simp add: residual.inject name-abs-eq*)
fix P'
assume *PTrans*: $P \Longrightarrow_l \hat{b} \langle \nu x \rangle \prec P'$

assume $P'RelQ'$: $(P', [(x, y)] \cdot Q'') \in Rel$
assume $xineq$: $x \neq y$

with $PTrans$ $yFreshP$ **have** $yFreshP'$: $y \# P'$
by(*force intro: freshTransition*)

hence $b \langle \nu x \rangle \prec P' = b \langle \nu y \rangle \prec [(x, y)] \cdot P'$ **by**(*rule alphaBoundResidual*)
moreover have $([(x, y)] \cdot P', Q'') \in Rel$

proof –
from $EqvtRel$ $P'RelQ'$ **have** $([(x, y)] \cdot P', [(x, y)] \cdot ((x, y)] \cdot Q'') \in Rel$
by(*rule eqvtRelI*)
thus $?thesis$ **by**(*simp add: name-calc*)
qed

ultimately show $\exists P'. P \Longrightarrow_i \hat{b} \langle \nu y \rangle \prec P' \wedge (P', Q'') \in Rel$ **using** $PTrans$
by *auto*
qed
next

fix Q'' b y u
assume Eq : $a \langle x \rangle \prec Q' = b \langle y \rangle \prec Q''$
assume $yFreshP$: $y \# P$

from Eq **have** $a = InputS$ b **by**(*simp add: residual.inject*)
from $DerInput[OF this]$ **obtain** P'' **where** $L1$: $\forall u. \exists P'. P \Longrightarrow_l u$ *in* $P'' \rightarrow b \langle x \rangle \prec P' \wedge$
 $(P', Q'[x::=u]) \in Rel$

by *blast*
have $\forall u. \exists P'. P \Longrightarrow_l u$ *in* $([(x, y)] \cdot P'') \rightarrow b \langle y \rangle \prec P' \wedge (P', Q''[y::=u]) \in Rel$
proof(*rule allI*)
fix u
from $L1$ Eq **show** $\exists P'. P \Longrightarrow_l u$ *in* $([(x, y)] \cdot P'') \rightarrow b \langle y \rangle \prec P' \wedge (P',$
 $Q''[y::=u]) \in Rel$
proof(*cases x=y, auto simp add: residual.inject name-abs-eq*)
assume Der : $\forall u. \exists P'. P \Longrightarrow_l u$ *in* $P'' \rightarrow b \langle x \rangle \prec P' \wedge (P', ((x, y)] \cdot$
 $Q'')[x::=u]) \in Rel$
assume $xFreshQ''$: $x \# Q''$

from Der **obtain** P' **where** $PTrans$: $P \Longrightarrow_l u$ *in* $P'' \rightarrow b \langle x \rangle \prec P'$
and $P'RelQ'$: $(P', ((x, y)] \cdot Q'')[x::=u]) \in Rel$
by *force*

from $PTrans$ $yFreshP$ **have** $P \Longrightarrow_l u$ *in* $([(x, y)] \cdot P'') \rightarrow b \langle y \rangle \prec P'$ **by**(*rule*
alphaInput)
moreover from $xFreshQ''$ $P'RelQ'$ **have** $(P', Q''[y::=u]) \in Rel$
by(*simp add: renaming*)
ultimately show $?thesis$ **by** *force*
qed
qed
thus $\exists P''. \forall u. \exists P'. P \Longrightarrow_l u$ *in* $P'' \rightarrow b \langle y \rangle \prec P' \wedge (P', Q''[y::=u]) \in Rel$

by *blast*
qed

lemma *simActFreeCases*[*consumes 0, case-names Der*]:

fixes $P :: pi$
and $\alpha :: freeRes$
and $Q' :: pi$
and $Rel :: (pi \times pi) set$

assumes $\exists P'. (P \Longrightarrow_l \hat{\alpha} \prec P') \wedge (P', Q') \in Rel$

shows *weakSimAct* $P (\alpha \prec Q') P Rel$

using *assms*

by(*simp add: residual.inject weakSimAct-def fresh-prod*)

lemma *simE*:

fixes $P :: pi$
and $Rel :: (pi \times pi) set$
and $Q :: pi$
and $a :: name$
and $x :: name$
and $u :: name$
and $Q' :: pi$

assumes $P \rightsquigarrow^{\hat{}} \langle Rel \rangle Q$

shows $Q \mapsto_a \langle \nu x \rangle \prec Q' \Longrightarrow x \# P \Longrightarrow \exists P'. P \Longrightarrow_l \hat{a} \langle \nu x \rangle \prec P' \wedge (P', Q') \in Rel$

and $Q \mapsto_a \langle x \rangle \prec Q' \Longrightarrow x \# P \Longrightarrow \exists P''. \forall u. \exists P'. P \Longrightarrow_l u \text{ in } P'' \rightarrow a \langle x \rangle \prec P' \wedge (P', Q'[x::=u]) \in Rel$

and $Q \mapsto_\alpha \prec Q' \Longrightarrow (\exists P'. P \Longrightarrow_l \hat{\alpha} \prec P' \wedge (P', Q') \in Rel)$

using *assms* by(*simp add: simDef*)+

lemma *weakSimTauChain*:

fixes $P :: pi$
and $Rel :: (pi \times pi) set$
and $Q :: pi$
and $Q' :: pi$

assumes *QChain*: $Q \Longrightarrow_\tau Q'$

and *PRelQ*: $(P, Q) \in Rel$

and *Sim*: $\bigwedge P Q. (P, Q) \in Rel \Longrightarrow P \rightsquigarrow^{\hat{}} \langle Rel \rangle Q$

shows $\exists P'. P \Longrightarrow_\tau P' \wedge (P', Q') \in Rel$

proof –

from *QChain* show *?thesis*

proof(*induct rule: tauChainInduct*)

case *id*

have $P \Longrightarrow_\tau P$ by *simp*

with $P\text{Rel}Q$ **show** $?case$ **by** $blast$
next
case(ih $Q' Q''$)
have $IH: \exists P'. P \Longrightarrow_{\tau} P' \wedge (P', Q') \in Rel$ **by** $fact$
then obtain P' **where** $PChain: P \Longrightarrow_{\tau} P'$ **and** $P'\text{Rel}Q': (P', Q') \in Rel$ **by**
 $blast$
from $P'\text{Rel}Q'$ **have** $P' \rightsquigarrow \hat{\langle Rel \rangle} Q'$ **by**($rule$ Sim)
moreover have $Q'\text{Trans}: Q' \mapsto_{\tau} \prec Q''$ **by** $fact$
ultimately have $\exists P''. P' \Longrightarrow_{\hat{l}} \tau \prec P'' \wedge (P'', Q'') \in Rel$ **by**($rule$ $simE$)
then obtain P'' **where** $P'\text{Trans}: P' \Longrightarrow_{\hat{l}} \tau \prec P''$ **and** $P''\text{Rel}Q'': (P'', Q'') \in$
 Rel **by** $blast$
from $P'\text{Trans}$ **have** $P' \Longrightarrow_{\tau} P''$ **by**($rule$ $tauTransitionChain$)
with $PChain$ **have** $P \Longrightarrow_{\tau} P''$ **by** $auto$
with $P''\text{Rel}Q''$ **show** $?case$ **by** $blast$
qed
qed

lemma $simE2$:

fixes $P :: pi$
and $Rel :: (pi \times pi)$ set
and $Q :: pi$
and $a :: name$
and $x :: name$
and $Q' :: pi$

assumes $PSimQ: P \rightsquigarrow \hat{\langle Rel \rangle} Q$
and $Sim: \bigwedge P Q. (P, Q) \in Rel \Longrightarrow P \rightsquigarrow \hat{\langle Rel \rangle} Q$
and $Eqvt: eqvt\ Rel$
and $P\text{Rel}Q: (P, Q) \in Rel$

shows $Q \Longrightarrow_{\hat{l}} a \langle \nu x \rangle \prec Q' \Longrightarrow x \# P \Longrightarrow \exists P'. P \Longrightarrow_{\hat{l}} a \langle \nu x \rangle \prec P' \wedge (P', Q') \in Rel$

and $Q \Longrightarrow_{\hat{l}} \alpha \prec Q' \Longrightarrow \exists P'. P \Longrightarrow_{\hat{l}} \alpha \prec P' \wedge (P', Q') \in Rel$

proof –

assume $Q\text{Trans}: Q \Longrightarrow_{\hat{l}} a \langle \nu x \rangle \prec Q'$

assume $x\text{Fresh}P: x \# P$

have $Goal: \bigwedge P Q a x Q'. \llbracket P \rightsquigarrow \hat{\langle Rel \rangle} Q; Q \Longrightarrow_{\hat{l}} a \langle \nu x \rangle \prec Q'; x \# P; x \# Q; (P, Q) \in Rel \rrbracket \Longrightarrow$

$\exists P'. P \Longrightarrow_{\hat{l}} a \langle \nu x \rangle \prec P' \wedge (P', Q') \in Rel$

proof –

fix $P Q a x Q'$

assume $PSimQ: P \rightsquigarrow \hat{\langle Rel \rangle} Q$

assume $Q\text{Trans}: Q \Longrightarrow_{\hat{l}} a \langle \nu x \rangle \prec Q'$

assume $x\text{Fresh}P: x \# P$

assume $x\text{Fresh}Q: x \# Q$

assume $P\text{Rel}Q: (P, Q) \in Rel$

from $Q\text{Trans}$ $x\text{Fresh}Q$ **obtain** $Q'' Q'''$ **where** $QChain: Q \Longrightarrow_{\tau} Q''$

and $Q''\text{Trans}: Q'' \mapsto a \langle \nu x \rangle \prec Q'''$

and $Q'''Chain: Q''' \Longrightarrow_{\tau} Q'$

by(force dest: *Weak-Late-Step-Semantics.transitionE simp add: weakTransition-def*)

from $QChain PRelQ Sim$ **have** $\exists P''. P \Longrightarrow_{\tau} P'' \wedge (P'', Q'') \in Rel$
by(rule *weakSimTauChain*)

then obtain P'' **where** $PChain: P \Longrightarrow_{\tau} P''$ **and** $P''RelQ'': (P'', Q'') \in Rel$

by *blast*

from $PChain xFreshP$ **have** $xFreshP'': x \# P''$ **by**(rule *freshChain*)

from $P''RelQ''$ **have** $P'' \rightsquigarrow^{\hat{}} \langle Rel \rangle Q''$ **by**(rule *Sim*)

hence $\exists P'''. P'' \Longrightarrow_{\hat{l}} a \langle \nu x \rangle \prec P''' \wedge (P''', Q''') \in Rel$ **using** $Q''Trans$
 $xFreshP''$

by(rule *simE*)

then obtain P''' **where** $P''Trans: P'' \Longrightarrow_{\hat{l}} a \langle \nu x \rangle \prec P'''$ **and** $P'''RelQ''': (P''', Q''') \in Rel$

by *blast*

from $P'''RelQ'''$ **have** $P''' \rightsquigarrow^{\hat{}} \langle Rel \rangle Q'''$ **by**(rule *Sim*)

have $\exists P'. P''' \Longrightarrow_{\tau} P' \wedge (P', Q') \in Rel$ **using** $Q'''Chain P'''RelQ''' Sim$
by(rule *weakSimTauChain*)

then obtain P' **where** $P'''Chain: P''' \Longrightarrow_{\tau} P'$ **and** $P'RelQ': (P', Q') \in Rel$

by *blast*

from $PChain P''Trans P'''Chain xFreshP''$ **have** $P \Longrightarrow_{\hat{l}} a \langle \nu x \rangle \prec P'$
by(blast dest: *chainTransitionAppend*)

with $P'RelQ'$ **show** $\exists P'. P \Longrightarrow_{\hat{l}} a \langle \nu x \rangle \prec P' \wedge (P', Q') \in Rel$ **by** *blast*

qed

have $\exists c::name. c \# (Q, Q', P, x)$ **by**(blast intro: *name-exists-fresh*)

then obtain $c::name$ **where** $cFreshQ: c \# Q$ **and** $cFreshQ': c \# Q'$ **and** $cFreshP: c \# P$

and $xineqc: x \neq c$

by(force *simp add: fresh-prod*)

from $QTrans cFreshQ'$ **have** $Q \Longrightarrow_{\hat{l}} a \langle \nu c \rangle \prec ((x, c) \cdot Q')$ **by**(*simp add: alphaBoundResidual*)

with $PSimQ$ **have** $\exists P'. P \Longrightarrow_{\hat{l}} a \langle \nu c \rangle \prec P' \wedge (P', [(x, c) \cdot Q']) \in Rel$ **using**
 $cFreshP cFreshQ \langle (P, Q) \in Rel \rangle$

by(rule *Goal*)

then obtain P' **where** $PTrans: P \Longrightarrow_{\hat{l}} a \langle \nu c \rangle \prec P'$ **and** $P'RelQ': (P', [(x, c) \cdot Q']) \in Rel$

by *force*

have $P \Longrightarrow_{\hat{l}} a \langle \nu x \rangle \prec ((x, c) \cdot P')$

proof –

from $PTrans xFreshP xineqc$ **have** $x \# P'$ **by**(rule *freshTransition*)

with $PTrans$ **show** *?thesis* **by**(*simp add: alphaBoundResidual name-swap*)

qed

moreover have $((x, c) \cdot P', Q') \in Rel$

proof –
from $\text{Eqvt } P' \text{Rel} Q'$ **have** $[(x, c)] \cdot P', [(x, c)] \cdot [(x, c)] \cdot Q' \in \text{Rel}$
by (*rule eqvtRelI*)
thus *?thesis* **by** *simp*
qed

ultimately show $\exists P'. P \Longrightarrow_l \hat{a} \langle \nu x \rangle \prec P' \wedge (P', Q') \in \text{Rel}$ **by** *blast*
next
assume $Q \text{Trans}: Q \Longrightarrow_l \hat{\alpha} \prec Q'$
thus $\exists P'. P \Longrightarrow_l \hat{\alpha} \prec P' \wedge (P', Q') \in \text{Rel}$
proof (*induct rule: transitionCases*)
case *Step*
have $Q \Longrightarrow_l \hat{\alpha} \prec Q'$ **by** *fact*
then obtain $Q'' Q'''$ **where** $Q \text{Chain}: Q \Longrightarrow_\tau Q''$
and $Q'' \text{Trans}: Q'' \mapsto_\alpha \prec Q'''$
and $Q''' \text{Chain}: Q''' \Longrightarrow_\tau Q'$
by (*blast dest: Weak-Late-Step-Semantics.transitionE*)

from $Q \text{Chain } P \text{Rel} Q \text{ Sim}$ **have** $\exists P''. P \Longrightarrow_\tau P'' \wedge (P'', Q'') \in \text{Rel}$
by (*rule weakSimTauChain*)
then obtain P'' **where** $P \text{Chain}: P \Longrightarrow_\tau P''$ **and** $P'' \text{Rel} Q'': (P'', Q'') \in \text{Rel}$
by *blast*
from $P'' \text{Rel} Q''$ **have** $P'' \rightsquigarrow \hat{\langle \text{Rel} \rangle} Q''$ **by** (*rule Sim*)
hence $\exists P'''. P'' \Longrightarrow_l \hat{\alpha} \prec P''' \wedge (P''', Q''') \in \text{Rel}$ **using** $Q'' \text{Trans}$
by (*rule simE*)
then obtain P''' **where** $P'' \text{Trans}: P'' \Longrightarrow_l \hat{\alpha} \prec P'''$ **and** $P''' \text{Rel} Q''': (P''', Q''') \in \text{Rel}$
by *blast*

from $P''' \text{Rel} Q'''$ **have** $P''' \rightsquigarrow \hat{\langle \text{Rel} \rangle} Q'''$ **by** (*rule Sim*)
have $\exists P'. P''' \Longrightarrow_\tau P' \wedge (P', Q') \in \text{Rel}$ **using** $Q''' \text{Chain } P''' \text{Rel} Q''' \text{ Sim}$
by (*rule weakSimTauChain*)
then obtain P' **where** $P''' \text{Chain}: P''' \Longrightarrow_\tau P'$ **and** $P' \text{Rel} Q': (P', Q') \in \text{Rel}$
by *blast*

from $P \text{Chain } P'' \text{Trans } P''' \text{Chain}$ **have** $P \Longrightarrow_l \hat{\alpha} \prec P'$
by (*blast dest: chainTransitionAppend*)
with $P' \text{Rel} Q'$ **show** *?case* **by** *blast*
next
case *Stay*
have $\alpha \prec Q' = \tau \prec Q$ **by** *fact*
hence $Q = Q'$ **and** $\alpha = \tau$ **by** (*simp add: residual.inject*)
moreover **have** $P \Longrightarrow_l \hat{\tau} \prec P$ **by** (*simp add: weakTransition-def*)
ultimately show *?case* **using** $P \text{Rel} Q$ **by** *blast*
qed
qed

lemma *eqvtI*:
fixes P **::** pi

and $Q :: pi$
and $Rel :: (pi \times pi) \text{ set}$
and $perm :: \text{name prm}$

assumes $Sim: P \rightsquigarrow^{\hat{}} \langle Rel \rangle Q$
and $RelRel': Rel \subseteq Rel'$
and $EqvtRel': \text{eqvt } Rel'$

shows $(perm \cdot P) \rightsquigarrow^{\hat{}} \langle Rel' \rangle (perm \cdot Q)$

proof –

from $EqvtRel'$ **show** $?thesis$
proof(*induct rule: simCasesCont[of - (perm \cdot P)]*)
case($Bound\ Q'\ a\ x$)
have $Trans: (perm \cdot Q) \mapsto a \langle \nu x \rangle \prec Q'$ **and** $xFreshP: x \# perm \cdot P$ **by** $fact+$

from $Trans$ **have** $(rev\ perm \cdot (perm \cdot Q)) \mapsto rev\ perm \cdot (a \langle \nu x \rangle \prec Q')$
by(*rule eqvts*)
hence $Q \mapsto (rev\ perm \cdot a) \langle \nu (rev\ perm \cdot x) \rangle \prec (rev\ perm \cdot Q')$
by(*simp add: name-rev-per*)
moreover from $xFreshP$ **have** $(rev\ perm \cdot x) \# P$ **by**(*simp add: name-fresh-left*)
ultimately have $\exists P'. P \Longrightarrow_i^{\hat{}} (rev\ perm \cdot a) \langle \nu (rev\ perm \cdot x) \rangle \prec P' \wedge (P',$
 $rev\ perm \cdot Q') \in Rel$ **using** Sim
by(*force intro: simE*)
then obtain P' **where** $PTrans: P \Longrightarrow_i^{\hat{}} (rev\ perm \cdot a) \langle \nu (rev\ perm \cdot x) \rangle \prec$
 P' **and** $P'RelQ': (P', rev\ perm \cdot Q') \in Rel$ **by** $blast$

from $PTrans$ **have** $(perm \cdot P) \Longrightarrow_i^{\hat{}} perm \cdot ((rev\ perm \cdot a) \langle \nu (rev\ perm \cdot x) \rangle$
 $\prec P')$
by(*rule Weak-Late-Semantics.eqvtI*)
hence $L1: (perm \cdot P) \Longrightarrow_i^{\hat{}} a \langle \nu x \rangle \prec (perm \cdot P')$ **by**(*simp add: name-per-rev*)
from $P'RelQ'\ RelRel'$ **have** $(P', rev\ perm \cdot Q') \in Rel'$ **by** $blast$
with $EqvtRel'$ **have** $(perm \cdot P', perm \cdot (rev\ perm \cdot Q')) \in Rel'$
by(*rule eqvtRelI*)
hence $(perm \cdot P', Q') \in Rel'$ **by**(*simp add: name-per-rev*)
with $L1$ **show** $?case$ **by** $blast$

next
case($Input\ Q'\ a\ x$)
have $Trans: (perm \cdot Q) \mapsto a \langle x \rangle \prec Q'$ **and** $xFreshP: x \# perm \cdot P$ **by** $fact+$

from $Trans$ **have** $(rev\ perm \cdot (perm \cdot Q)) \mapsto rev\ perm \cdot (a \langle x \rangle \prec Q')$
by(*rule eqvts*)
hence $Q \mapsto (rev\ perm \cdot a) \langle (rev\ perm \cdot x) \rangle \prec (rev\ perm \cdot Q')$
by(*simp add: name-rev-per*)
moreover from $xFreshP$ **have** $xFreshP: (rev\ perm \cdot x) \# P$ **by**(*simp add:*
 $name-fresh-left$)
ultimately have $\exists P''. \forall u. \exists P'. P \Longrightarrow_l u \text{ in } P'' \rightarrow (rev\ perm \cdot a) \langle (rev\ perm \cdot$
 $x) \rangle \prec P' \wedge (P', (rev\ perm \cdot Q'))[(rev\ perm \cdot x) ::= u] \in Rel$ **using** Sim
by(*force intro: simE*)
then obtain P'' **where** $L1: \forall u. \exists P'. P \Longrightarrow_l u \text{ in } P'' \rightarrow (rev\ perm \cdot a) \langle (rev$

$perm \cdot x \rangle \prec P' \wedge (P', (rev \text{ perm} \cdot Q')[(rev \text{ perm} \cdot x)::=u]) \in Rel$
by *blast*
have $\forall u. \exists P'. (perm \cdot P) \Longrightarrow_l u \text{ in } (perm \cdot P'') \rightarrow a \langle x \rangle \prec P' \wedge (P', Q'[x::=u])$
 $\in Rel'$
proof(*rule allI*)
fix u
from $L1$ **obtain** P' **where** $PTrans: P \Longrightarrow_l (rev \text{ perm} \cdot u) \text{ in } P'' \rightarrow (rev \text{ perm} \cdot a) \langle (rev \text{ perm} \cdot x) \rangle \prec P'$
and $P'RelQ': (P', (rev \text{ perm} \cdot Q')[(rev \text{ perm} \cdot x)::=(rev \text{ perm} \cdot u)]) \in Rel$ **by** *blast*
from $PTrans$ **have** $(perm \cdot P) \Longrightarrow_l (perm \cdot (rev \text{ perm} \cdot u)) \text{ in } (perm \cdot P'') \rightarrow (perm \cdot rev \text{ perm} \cdot a) \langle (perm \cdot rev \text{ perm} \cdot x) \rangle \prec (perm \cdot P')$
by(*rule-tac Weak-Late-Step-Semantics.eqtI, auto*)
hence $L2: (perm \cdot P) \Longrightarrow_l u \text{ in } (perm \cdot P'') \rightarrow a \langle x \rangle \prec (perm \cdot P')$ **by**(*simp add: name-per-rev*)
from $P'RelQ' RelRel'$ **have** $(P', (rev \text{ perm} \cdot Q')[(rev \text{ perm} \cdot x)::=(rev \text{ perm} \cdot u)]) \in Rel'$ **by** *blast*
with $EqvRel'$ **have** $(perm \cdot P', perm \cdot ((rev \text{ perm} \cdot Q')[(rev \text{ perm} \cdot x)::=(rev \text{ perm} \cdot u)])) \in Rel'$
by(*rule eqvRelI*)
hence $(perm \cdot P', Q'[x::=u]) \in Rel'$ **by**(*simp add: name-per-rev eqv-subst[THEN sym] name-calc*)
with $L2$ **show** $\exists P'. (perm \cdot P) \Longrightarrow_l u \text{ in } (perm \cdot P'') \rightarrow a \langle x \rangle \prec P' \wedge (P', Q'[x::=u]) \in Rel'$ **by** *blast*
qed

thus *?case* **by** *blast*
next
case(*Free Q' α*)
have $Trans: (perm \cdot Q) \mapsto \alpha \prec Q'$ **by** *fact*

from $Trans$ **have** $(rev \text{ perm} \cdot (perm \cdot Q)) \mapsto rev \text{ perm} \cdot (\alpha \prec Q')$
by(*rule eqvts*)
hence $Q \mapsto (rev \text{ perm} \cdot \alpha) \prec (rev \text{ perm} \cdot Q')$
by(*simp add: name-rev-per*)
with Sim **have** $(\exists P'. P \Longrightarrow_l \hat{} (rev \text{ perm} \cdot \alpha) \prec P' \wedge (P', (rev \text{ perm} \cdot Q')) \in Rel)$
by(*rule simE*)
then obtain P' **where** $PTrans: P \Longrightarrow_l \hat{} (rev \text{ perm} \cdot \alpha) \prec P'$ **and** $PRel: (P', (rev \text{ perm} \cdot Q')) \in Rel$ **by** *blast*
from $PTrans$ **have** $(perm \cdot P) \Longrightarrow_l \hat{} perm \cdot ((rev \text{ perm} \cdot \alpha) \prec P')$
by(*rule Weak-Late-Semantics.eqtI*)
hence $L1: (perm \cdot P) \Longrightarrow_l \hat{} \alpha \prec (perm \cdot P')$ **by**(*simp add: name-per-rev*)
from $PRel EqvRel' RelRel'$ **have** $((perm \cdot P'), (perm \cdot (rev \text{ perm} \cdot Q'))) \in Rel'$
by(*force intro: eqvRelI*)
hence $((perm \cdot P'), Q') \in Rel'$ **by**(*simp add: name-per-rev*)
with $L1$ **show** *?case* **by** *blast*
qed

qed

lemma *reflexive*:

fixes $P :: pi$

and $Rel :: (pi \times pi) \text{ set}$

assumes $Id \subseteq Rel$

shows $P \rightsquigarrow^{\wedge} \langle Rel \rangle P$

using *assms*

by(*auto intro: Weak-Late-Step-Semantics.singleActionChain*
simp add: simDef weakTransition-def)

lemma *transitive*:

fixes $P :: pi$

and $Q :: pi$

and $R :: pi$

and $Rel :: (pi \times pi) \text{ set}$

and $Rel' :: (pi \times pi) \text{ set}$

and $Rel'' :: (pi \times pi) \text{ set}$

assumes $QSimR: Q \rightsquigarrow^{\wedge} \langle Rel' \rangle R$

and $Eqt: eqvt Rel$

and $Eqt': eqvt Rel''$

and $Trans: Rel \circ Rel' \subseteq Rel''$

and $Sim: \bigwedge P Q. (P, Q) \in Rel \implies P \rightsquigarrow^{\wedge} \langle Rel \rangle Q$

and $PRelQ: (P, Q) \in Rel$

shows $P \rightsquigarrow^{\wedge} \langle Rel'' \rangle R$

proof –

from $PRelQ$ **have** $PSimQ: P \rightsquigarrow^{\wedge} \langle Rel \rangle Q$ **by**(*rule Sim*)

from Eqt' **show** *?thesis*

proof(*induct rule: simCasesCont[of - (P, Q)]*)

case(*Bound R' a x*)

have $RTrans: R \mapsto a \langle \nu x \rangle \prec R'$ **by** *fact*

have $x \# (P, Q)$ **by** *fact*

hence $xFreshP: x \# P$ **and** $xFreshQ: x \# Q$ **by**(*simp add: fresh-prod*)+

from $QSimR RTrans xFreshQ$ **have** $\exists Q'. Q \implies_i^{\wedge} a \langle \nu x \rangle \prec Q' \wedge (Q', R') \in Rel'$

by(*rule simE*)

then obtain Q' **where** $QTrans: Q \implies_i^{\wedge} a \langle \nu x \rangle \prec Q'$ **and** $Q'RelR': (Q', R') \in Rel'$ **by** *blast*

from $PSimQ Sim Eqt PRelQ QTrans xFreshP$ **have** $\exists P'. P \implies_i^{\wedge} a \langle \nu x \rangle \prec P' \wedge (P', Q') \in Rel$

by(*rule simE2*)

then obtain P' **where** $PTrans: P \implies_i^{\wedge} a \langle \nu x \rangle \prec P'$ **and** $P'RelQ': (P', Q')$

$\in \text{Rel}$ **by blast**
moreover from $P' \text{Rel} Q' Q' \text{Rel} R'$ **Trans have** $(P', R') \in \text{Rel}''$ **by blast**
ultimately show *?case by blast*
next
case(*Input R' a x*)
have $R \text{Trans}: R \mapsto a \langle x \rangle \prec R'$ **by fact**
have $x \# (P, Q)$ **by fact**
hence $x \text{Fresh} P: x \# P$ **and** $x \text{Fresh} Q: x \# Q$ **by**(*simp add: fresh-prod*)

from $Q \text{Sim} R R \text{Trans} x \text{Fresh} Q$ **obtain** Q'' **where** $\forall u. \exists Q'. Q \Rightarrow_{lu} \text{in}$
 $Q' \rightarrow a \langle x \rangle \prec Q' \wedge (Q', R'[x::=u]) \in \text{Rel}'$
by(*blast dest: simE*)
hence $\exists Q'''. Q \Rightarrow_{\tau} Q''' \wedge Q''' \mapsto a \langle x \rangle \prec Q'' \wedge (\forall u. \exists Q'. Q''[x::=u] \Rightarrow_{\tau}$
 $Q' \wedge (Q', R'[x::=u]) \in \text{Rel}')$
by(*simp add: inputTransition-def, blast*)
then obtain Q''' **where** $Q \text{Chain}: Q \Rightarrow_{\tau} Q'''$
and $Q''' \text{Trans}: Q''' \mapsto a \langle x \rangle \prec Q''$
and $L1: \forall u. \exists Q'. Q''[x::=u] \Rightarrow_{\tau} Q' \wedge (Q', R'[x::=u]) \in \text{Rel}'$
by blast
from $Q \text{Chain} P \text{Rel} Q \text{Sim}$ **have** $\exists P'''. P \Rightarrow_{\tau} P''' \wedge (P''', Q''') \in \text{Rel}$
by(*rule weakSimTauChain*)
then obtain P''' **where** $P \text{Chain}: P \Rightarrow_{\tau} P'''$ **and** $P''' \text{Rel} Q''': (P''', Q''') \in$
 Rel **by blast**
from $P \text{Chain} x \text{Fresh} P$ **have** $x \text{Fresh} P''': x \# P'''$ **by**(*rule freshChain*)
from $P''' \text{Rel} Q'''$ **have** $P''' \rightsquigarrow \langle \text{Rel} \rangle Q'''$ **by**(*rule Sim*)
hence $\exists P'''. \forall u. \exists P''. P''' \Rightarrow_{lu} \text{in } P'' \rightarrow a \langle x \rangle \prec P'' \wedge (P'', Q''[x::=u]) \in$
 Rel **using** $Q''' \text{Trans} x \text{Fresh} P'''$
by(*rule simE*)
then obtain P'''' **where** $L2: \forall u. \exists P''. P''' \Rightarrow_{lu} \text{in } P'''' \rightarrow a \langle x \rangle \prec P'' \wedge$
 $(P'', Q''[x::=u]) \in \text{Rel}$
by blast
have $\forall u. \exists P' Q'. P \Rightarrow_{lu} \text{in } P'''' \rightarrow a \langle x \rangle \prec P' \wedge (P', R'[x::=u]) \in \text{Rel}''$
proof(*rule allI*)
fix u
from $L1$ **obtain** Q' **where** $Q'' \text{Chain}: Q''[x::=u] \Rightarrow_{\tau} Q'$ **and** $Q' \text{Rel} R': (Q',$
 $R'[x::=u]) \in \text{Rel}'$
by blast
from $L2$ **obtain** P'' **where** $P''' \text{Trans}: P''' \Rightarrow_{lu} \text{in } P'''' \rightarrow a \langle x \rangle \prec P''$
and $P'' \text{Rel} Q'': (P'', Q''[x::=u]) \in \text{Rel}$

by blast
from $P'' \text{Rel} Q''$ **have** $P'' \rightsquigarrow \langle \text{Rel} \rangle Q''[x::=u]$ **by**(*rule Sim*)
have $\exists P'. P'' \Rightarrow_{\tau} P' \wedge (P', Q') \in \text{Rel}$ **using** $Q'' \text{Chain} P'' \text{Rel} Q'' \text{Sim}$
by(*rule weakSimTauChain*)
then obtain P' **where** $P'' \text{Chain}: P'' \Rightarrow_{\tau} P'$ **and** $P' \text{Rel} Q': (P', Q') \in \text{Rel}$
by blast
from $P \text{Chain} P''' \text{Trans} P'' \text{Chain}$ **have** $P \Rightarrow_{lu} \text{in } P'''' \rightarrow a \langle x \rangle \prec P'$
by(*blast dest: Weak-Late-Step-Semantics.chainTransitionAppend*)
moreover from $P' \text{Rel} Q' Q' \text{Rel} R'$ **have** $(P', R'[x::=u]) \in \text{Rel}''$ **by**(*insert*
 Trans, auto)

ultimately show $\exists P' Q'. P \Longrightarrow_{\iota} u$ in $P'''' \rightarrow a \langle x \rangle \prec P' \wedge (P', R'[x::=u]) \in \text{Rel}''$ **by** *blast*
qed
thus *?case by force*
next
case(*Free R' α*)
have $R\text{Trans}: R \mapsto \alpha \prec R'$ **by** *fact*
with $Q\text{Sim}R$ **have** $\exists Q'. Q \Longrightarrow_{\hat{\iota}} \alpha \prec Q' \wedge (Q', R') \in \text{Rel}'$ **by**(*rule simE*)
then obtain Q' **where** $Q\text{Trans}: Q \Longrightarrow_{\hat{\iota}} \alpha \prec Q'$ **and** $Q'\text{Rel}R': (Q', R') \in \text{Rel}'$
by *blast*
from $PSimQ$ Sim $Eqvt$ $P\text{Rel}Q$ $Q\text{Trans}$ **have** $\exists P'. P \Longrightarrow_{\hat{\iota}} \alpha \prec P' \wedge (P', Q') \in \text{Rel}$ **by**(*rule simE2*)
then obtain P' **where** $P\text{Trans}: P \Longrightarrow_{\hat{\iota}} \alpha \prec P'$ **and** $P'\text{Rel}Q': (P', Q') \in \text{Rel}$
by *blast*
from $P'\text{Rel}Q'$ $Q'\text{Rel}R'$ $Trans$ **have** $(P', R') \in \text{Rel}''$ **by** *blast*
with $P\text{Trans}$ **show** $\exists P'. P \Longrightarrow_{\hat{\iota}} \alpha \prec P' \wedge (P', R') \in \text{Rel}''$ **by** *blast*
qed
qed

lemma *strongSimWeakSim:*

fixes $P :: pi$
and $Q :: pi$
and $Rel :: (pi \times pi)$ *set*

assumes $PSimQ: P \rightsquigarrow[Rel] Q$

shows $P \rightsquigarrow^{\wedge} \langle Rel \rangle Q$

proof(*induct rule: simCases*)

case(*Bound Q' a x*)

have $Q \mapsto a \langle \nu x \rangle \prec Q'$ **and** $x \# P$ **by** *fact+*

with $PSimQ$ **obtain** P' **where** $P\text{Trans}: P \mapsto a \langle \nu x \rangle \prec P'$ **and** $P'\text{Rel}Q': (P', Q') \in \text{Rel}$

by(*force dest: Strong-Late-Sim.simE simp add: derivative-def*)

from $P\text{Trans}$ **have** $P \Longrightarrow_{\hat{\iota}} a \langle \nu x \rangle \prec P'$

by(*force intro: Weak-Late-Step-Semantics.singleActionChain simp add: weak-Transition-def*)

with $P'\text{Rel}Q'$ **show** *?case by blast*

next

case(*Input Q' a x*)

assume $Q \mapsto a \langle x \rangle \prec Q'$ **and** $x \# P$

with $PSimQ$ **obtain** P' **where** $P\text{Trans}: P \mapsto a \langle x \rangle \prec P'$ **and** $P\text{Der}: \text{derivative } P' Q' (\text{Input} S a) x \text{ Rel}$

by(*blast dest: Strong-Late-Sim.simE*)

have $\forall u. \exists P''. P \Longrightarrow_{\iota} u$ in $P' \rightarrow a \langle x \rangle \prec P'' \wedge (P'', Q'[x::=u]) \in \text{Rel}$

proof(*rule allI*)

fix u

from $P\text{Trans}$ **have** $P \Longrightarrow_{\iota} u$ in $P' \rightarrow a \langle x \rangle \prec P'[x::=u]$ **by**(*blast intro: Weak-Late-Step-Semantics.singleActio*)

moreover from $PDer$ **have** $(P'[x::=u], Q'[x::=u]) \in Rel$ **by** $(force\ simp\ add: derivative-def)$
ultimately show $\exists P''. P \Longrightarrow_l u$ in $P' \rightarrow a \langle x \rangle \prec P'' \wedge (P'', Q'[x::=u]) \in Rel$
by $auto$
qed
thus $?case$ **by** $blast$
next
case $(Free\ Q'\ \alpha)$
have $Q \mapsto \alpha \prec Q'$ **by** $fact$
with $PSimQ$ **obtain** P' **where** $PTrans: P \mapsto \alpha \prec P'$ **and** $P'RelQ': (P', Q') \in Rel$
by $(blast\ dest: Strong-Late-Sim.simE)$
from $PTrans$ **have** $P \Longrightarrow_l \hat{\alpha} \prec P'$ **by** $(rule\ Weak-Late-Semantics.singleActionChain)$
with $P'RelQ'$ **show** $?case$ **by** $blast$
qed

lemma $strongAppend:$

fixes $P \quad :: pi$
and $Q \quad :: pi$
and $R \quad :: pi$
and $Rel \quad :: (pi \times pi)\ set$
and $Rel' \quad :: (pi \times pi)\ set$
and $Rel'' \quad :: (pi \times pi)\ set$

assumes $PSimQ: P \rightsquigarrow^{\hat{}} \langle Rel \rangle Q$
and $QSimR: Q \rightsquigarrow [Rel] R$
and $Eqvt'': eqvt\ Rel''$
and $Trans: Rel\ O\ Rel' \subseteq Rel''$

shows $P \rightsquigarrow^{\hat{}} \langle Rel'' \rangle R$

proof –

from $Eqvt''$ **show** $?thesis$

proof $(induct\ rule: simCasesCont[of - (P, Q)])$

case $(Bound\ R'\ a\ x)$

have $x \# (P, Q)$ **by** $fact$

hence $xFreshP: x \# P$ **and** $xFreshQ: x \# Q$ **by** $(simp\ add: fresh-prod)+$

have $RTrans: R \mapsto a \langle \nu x \rangle \prec R'$ **by** $fact$

from $xFreshQ\ QSimR\ RTrans$ **obtain** Q' **where** $QTrans: Q \mapsto a \langle \nu x \rangle \prec Q'$
and $Q'Rel'R': (Q', R') \in Rel'$

by $(force\ dest: Strong-Late-Sim.simE\ simp\ add: derivative-def)$

with $PSimQ\ QTrans\ xFreshP$ **have** $\exists P'. P \Longrightarrow_l \hat{a} \langle \nu x \rangle \prec P' \wedge (P', Q') \in Rel$

by $(blast\ intro: simE)$

then obtain P' **where** $PTrans: P \Longrightarrow_l \hat{a} \langle \nu x \rangle \prec P'$ **and** $P'RelQ': (P', Q') \in Rel$ **by** $blast$

moreover from $P'RelQ'\ Q'Rel'R'\ Trans$ **have** $(P', R') \in Rel''$ **by** $blast$

ultimately show $?case$ **by** $blast$

next

```

case(Input  $R' a x$ )
have  $RTrans: R \mapsto a \langle x \rangle \prec R'$  by fact
have  $x \# (P, Q)$  by fact
hence  $xFreshP: x \# P$  and  $xFreshQ: x \# Q$  by(simp add: fresh-prod)+

from  $QSimR RTrans xFreshQ$  obtain  $Q'$  where  $QTrans: Q \mapsto a \langle x \rangle \prec Q'$ 
and  $Q'Der: derivative\ Q'\ R'\ (InputS\ a)\ x\ Rel'$ 
by(blast dest: Strong-Late-Sim.simE)
from  $QTrans PSimQ xFreshP$  obtain  $P''$  where  $L2: \forall u. \exists P'. P \Longrightarrow_{!u} in$ 
 $P'' \rightarrow a \langle x \rangle \prec P' \wedge (P', Q'[x::=u]) \in Rel$ 
by(blast dest: simE)
have  $\forall u. \exists P'. P \Longrightarrow_{!u} in P'' \rightarrow a \langle x \rangle \prec P' \wedge (P', R'[x::=u]) \in Rel''$ 
proof(rule allI)
fix  $u$ 
from  $L2$  obtain  $P'$  where  $PTrans: P \Longrightarrow_{!u} in P'' \rightarrow a \langle x \rangle \prec P'$ 
and  $P'RelQ': (P', Q'[x::=u]) \in Rel$ 

by blast
moreover from  $Q'Der$  have  $(Q'[x::=u], R'[x::=u]) \in Rel'$  by(simp add:
derivative-def)
ultimately show  $\exists P'. P \Longrightarrow_{!u} in P'' \rightarrow a \langle x \rangle \prec P' \wedge (P', R'[x::=u]) \in Rel''$ 
using  $Trans$  by blast
qed
thus ?case by force
next
case(Free  $R' \alpha$ )
have  $RTrans: R \mapsto \alpha \prec R'$  by fact
with  $QSimR$  obtain  $Q'$  where  $QTrans: Q \mapsto \alpha \prec Q'$  and  $Q'RelR': (Q', R')$ 
 $\in Rel'$ 
by(blast dest: Strong-Late-Sim.simE)
from  $PSimQ QTrans$  have  $\exists P'. P \Longrightarrow_{!} \hat{\alpha} \prec P' \wedge (P', Q') \in Rel$ 
by(blast intro: simE)
then obtain  $P'$  where  $PTrans: P \Longrightarrow_{!} \hat{\alpha} \prec P'$  and  $P'RelQ': (P', Q') \in Rel$ 
by blast
from  $P'RelQ' Q'RelR' Trans$  have  $(P', R') \in Rel''$  by blast
with  $PTrans$  show ?case by blast
qed
qed

end

theory Weak-Late-Bisim
imports Weak-Late-Sim Strong-Late-Bisim
begin

lemma monoAux:  $A \subseteq B \Longrightarrow P \rightsquigarrow^{\hat{A}} Q \longrightarrow P \rightsquigarrow^{\hat{B}} Q$ 
by(auto intro: Weak-Late-Sim.monotonic)

coinductive-set weakBisim ::  $(pi \times pi)$  set
where

```

step: $\llbracket P \rightsquigarrow^{\wedge} \langle \text{weakBisim} \rangle Q; (Q, P) \in \text{weakBisim} \rrbracket \implies (P, Q) \in \text{weakBisim}$
monos *monoAux*

abbreviation

weakBisimJudge (**infixr** $\langle \approx \rangle$ 65) **where** $P \approx Q \equiv (P, Q) \in \text{weakBisim}$

lemma *weakBisimCoinductAux*[*case-names weakBisim, case-conclusion weakBisim step, consumes 1*]:

assumes $p: (P, Q) \in X$
and *step*: $\bigwedge P Q. (P, Q) \in X \implies P \rightsquigarrow^{\wedge} \langle (X \cup \text{weakBisim}) \rangle Q \wedge ((Q, P) \in X \vee Q \approx P)$

shows $P \approx Q$

proof –

have *aux*: $X \cup \text{weakBisim} = \{(P, Q). (P, Q) \in X \vee P \approx Q\}$ **by** *blast*

from p **show** *?thesis*

by(*coinduct, force dest: step simp add: aux*)

qed

lemma *weakBisimCoinduct*[*consumes 1, case-names cSim cSym*]:

fixes $P :: pi$

and $Q :: pi$

assumes $(P, Q) \in X$

and $\bigwedge P Q. (P, Q) \in X \implies P \rightsquigarrow^{\wedge} \langle (X \cup \text{weakBisim}) \rangle Q$

and $\bigwedge P Q. (P, Q) \in X \implies (Q, P) \in X$

shows $P \approx Q$

using *assms*

by(*coinduct rule: weakBisimCoinductAux*) *auto*

lemma *weak-coinduct*[*case-names weakBisim, case-conclusion weakBisim step, consumes 1*]:

assumes $p: (P, Q) \in X$

and *step*: $\bigwedge P Q. (P, Q) \in X \implies P \rightsquigarrow^{\wedge} \langle X \rangle Q \wedge (Q, P) \in X$

shows $P \approx Q$

using p

proof(*coinduct rule: weakBisimCoinductAux*)

case (*weakBisim P Q*)

from *step[OF this]* **show** *?case* **using** *Weak-Late-Sim.monotonic* **by** *blast*

qed

lemma *weakBisimWeakCoinduct*[*consumes 1, case-names cSim cSym*]:

fixes $P :: pi$

and $Q :: pi$

assumes $(P, Q) \in X$

```

and  $\bigwedge P Q. (P, Q) \in X \implies P \rightsquigarrow^{\hat{}} \langle X \rangle Q$ 
and  $\bigwedge P Q. (P, Q) \in X \implies (Q, P) \in X$ 

shows  $P \approx Q$ 
using assms
by(coinduct rule: weak-coinduct) auto
lemma monotonic:  $\text{mono}(\lambda p x1 x2. \exists P Q. x1 = P \wedge x2 = Q \wedge P \rightsquigarrow^{\hat{}} \langle \{(xa, x). p \ x a \ x\} \rangle Q \wedge Q \rightsquigarrow^{\hat{}} \langle \{(xa, x). p \ x a \ x\} \rangle P)$ 
by(auto intro: monoI Weak-Late-Sim.monotonic)

lemma unfoldE:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 

  assumes  $P \approx Q$ 

  shows  $P \rightsquigarrow^{\hat{}} \langle \text{weakBisim} \rangle Q$ 
  and  $Q \approx P$ 
using assms
by(auto intro: weakBisim.cases)

lemma unfoldI:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 

  assumes  $P \rightsquigarrow^{\hat{}} \langle \text{weakBisim} \rangle Q$ 
  and  $Q \approx P$ 

  shows  $P \approx Q$ 
using assms
by(auto intro: weakBisim.cases)

lemma eqvt:
  shows eqvt weakBisim
proof(auto simp add: eqvt-def)
  let  $?X = \{x. \exists P Q (\text{perm}::\text{name prm}). P \approx Q \wedge x = (\text{perm} \cdot P, \text{perm} \cdot Q)\}$ 
  fix  $P Q$ 
  fix  $\text{perm}::\text{name prm}$ 
  assume  $PBiSimQ: P \approx Q$ 

  hence  $(\text{perm} \cdot P, \text{perm} \cdot Q) \in ?X$  by blast
  moreover have  $\bigwedge P Q \text{perm}::\text{name prm}. \llbracket P \rightsquigarrow^{\hat{}} \langle \text{weakBisim} \rangle Q \rrbracket \implies (\text{perm} \cdot P) \rightsquigarrow^{\hat{}} \langle ?X \rangle (\text{perm} \cdot Q)$ 
  proof –
    fix  $P Q$ 
    fix  $\text{perm}::\text{name prm}$ 
    assume  $P \rightsquigarrow^{\hat{}} \langle \text{weakBisim} \rangle Q$ 

    moreover have  $\text{weakBisim} \subseteq ?X$ 

```

```

proof(auto)
  fix  $P Q$ 
  assume  $P \approx Q$ 
  moreover have  $P = ([\!:\!:]::\text{name } prm) \cdot P$  and  $Q = ([\!:\!:]::\text{name } prm) \cdot Q$  by
auto
  ultimately show  $\exists P' Q'. P' \approx Q' \wedge (\exists (\text{perm}::\text{name } prm). P = \text{perm} \cdot P'$ 
 $\wedge Q = \text{perm} \cdot Q')$ 
    by blast
  qed

moreover have eqvt ?X
proof(auto simp add: eqvt-def)
  fix  $P Q$ 
  fix  $\text{perm1}::\text{name } prm$ 
  fix  $\text{perm2}::\text{name } prm$ 

  assume  $P \approx Q$ 
  moreover have  $\text{perm1} \cdot \text{perm2} \cdot P = (\text{perm1} @ \text{perm2}) \cdot P$  by(simp add:
pt2[OF pt-name-inst])
  moreover have  $\text{perm1} \cdot \text{perm2} \cdot Q = (\text{perm1} @ \text{perm2}) \cdot Q$  by(simp add:
pt2[OF pt-name-inst])

  ultimately show  $\exists P' Q'. P' \approx Q' \wedge (\exists (\text{perm}::\text{name } prm). \text{perm1} \cdot \text{perm2}$ 
 $\cdot P = \text{perm} \cdot P' \wedge$ 
 $\text{perm1} \cdot \text{perm2} \cdot Q = \text{perm} \cdot Q')$ 
    by blast
  qed

ultimately show  $(\text{perm} \cdot P) \rightsquigarrow^{\langle ?X \rangle} (\text{perm} \cdot Q)$ 
  by(rule Weak-Late-Sim.eqvtI)
qed

ultimately show  $(\text{perm} \cdot P) \approx (\text{perm} \cdot Q)$  by(coinduct rule: weak-coinduct,
blast dest: unfoldE)
qed

lemma eqvtI:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $\text{perm} :: \text{name } prm$ 

  assumes  $P \approx Q$ 

  shows  $(\text{perm} \cdot P) \approx (\text{perm} \cdot Q)$ 
using assms
by(rule eqvtRelI[OF eqvt])

lemma weakBisimEqvt[simp]:
  shows eqvt weakBisim

```

by(*auto simp add: eqvt-def eqvtI*)

lemma *strongBisimWeakBisim*:

fixes $P :: pi$

and $Q :: pi$

assumes $PSimQ: P \sim Q$

shows $P \approx Q$

proof –

have $\bigwedge P Q. P \rightsquigarrow[bisim] Q \implies P \rightsquigarrow^{\langle bisim \cup weakBisim \rangle} Q$

proof –

fix $P Q$

assume $P \rightsquigarrow[bisim] Q$

hence $P \rightsquigarrow^{\langle bisim \rangle} Q$ **by**(*rule strongSimWeakSim*)

thus $P \rightsquigarrow^{\langle bisim \cup weakBisim \rangle} Q$

by(*blast intro: Weak-Late-Sim.monotonic*)

qed

with $PSimQ$ **show** *?thesis*

by(*coinduct rule: weakBisimCoinductAux, force dest: Strong-Late-Bisim.bisimE symmetric*)

qed

lemma *reflexive*:

fixes $P :: pi$

shows $P \approx P$

proof –

have $(P, P) \in Id$ **by** *simp*

then show *?thesis*

proof (*coinduct rule: weak-coinduct*)

case (*weakBisim P Q*)

have $(P, Q) \in Id$ **by** *fact*

thus *?case* **by**(*auto intro: Weak-Late-Sim.reflexive*)

qed

qed

lemma *symmetric*:

fixes $P :: pi$

and $Q :: pi$

assumes $P \approx Q$

shows $Q \approx P$

using *assms*

by(*auto dest: unfoldE intro: unfoldI*)

```

lemma transitive:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $R :: pi$ 

  assumes  $PBiSimQ: P \approx Q$ 
  and  $QBiSimR: Q \approx R$ 

  shows  $P \approx R$ 
proof -
  let  $?X = weakBisim\ O\ weakBisim$ 
  from assms have  $(P, R) \in ?X$  by blast
  moreover have  $\bigwedge P\ Q\ R. \llbracket Q \rightsquigarrow^{<weakBisim>} R; P \approx Q \rrbracket \implies$ 
     $P \rightsquigarrow^{<(?X \cup weakBisim)>} R$ 

  proof -
    fix  $P\ Q\ R$ 
    assume  $PBiSimQ: P \approx Q$ 
    assume  $Q \rightsquigarrow^{<weakBisim>} R$ 
    moreover have eqvt weakBisim by (rule eqvt)
    moreover from eqvt have eqvt  $(?X \cup weakBisim)$  by (auto simp add: eqvt-Trans)
    moreover have weakBisim O weakBisim  $\subseteq ?X \cup weakBisim$  by auto
    moreover have  $\bigwedge P\ Q. P \approx Q \implies P \rightsquigarrow^{<weakBisim>} Q$  by (rule unfoldE)

    ultimately show  $P \rightsquigarrow^{<(?X \cup weakBisim)>} R$  using  $PBiSimQ$ 
      by (rule Weak-Late-Sim.transitive)
  qed

  ultimately show ?thesis
    apply (coinduct rule: weakBisimCoinduct, auto)
    by (blast dest: unfoldE symmetric)+
qed

lemma transitive-coinduct-weak [case-names WeakBisimEarly, case-conclusion WeakBisimEarly step, consumes 2]:
  assumes  $p: (P, Q) \in X$ 
  and Eqvt: eqvt X
  and step:  $\bigwedge P\ Q. (P, Q) \in X \implies P \rightsquigarrow^{<(bisim\ O\ X\ O\ bisim)>} Q \wedge (Q, P) \in X$ 

  shows  $P \approx Q$ 
proof -
  let  $?X = bisim\ O\ X\ O\ bisim$ 

  have Sim:  $\bigwedge P\ P'\ Q'\ Q. \llbracket P \sim P'; P' \rightsquigarrow^{<?X>} Q'; Q' \rightsquigarrow[bisim] Q \rrbracket \implies$ 
     $P \rightsquigarrow^{<?X>} Q$ 

  proof -
    fix  $P\ P'\ Q'\ Q$ 

```

```

assume  $PBisimP'$ :  $P \sim P'$ 
assume  $P'SimQ'$ :  $P' \rightsquigarrow^{\langle ?X \rangle} Q'$ 
assume  $Q'SimQ$ :  $Q' \rightsquigarrow[bisim] Q$ 

show  $P \rightsquigarrow^{\langle ?X \rangle} Q$ 
proof –
  have  $P' \rightsquigarrow^{\langle ?X \rangle} Q$ 
  proof –
    have  $?X \ O \ bisim \subseteq ?X$  by(blast intro: Strong-Late-Bisim.transitive)
    moreover from Strong-Late-Bisim.bisimEqvt Eqvt have eqvt  $?X$  by blast
    ultimately show  $?thesis$  using  $P'SimQ' \ Q'SimQ$  by(blast intro: strongApend)
  qed
  moreover have eqvt bisim by(rule Strong-Late-Bisim.bisimEqvt)
  moreover from Strong-Late-Bisim.bisimEqvt Eqvt have eqvt  $?X$  by blast
  moreover have bisim  $O \ ?X \subseteq ?X$  by(blast intro: Strong-Late-Bisim.transitive)
  moreover have  $\bigwedge P \ Q. P \sim Q \implies P \rightsquigarrow^{\langle bisim \rangle} Q$  by(blast dest: Strong-Late-Bisim.bisimE
strongSimWeakSim)
  ultimately show  $?thesis$  using  $PBisimP'$  by(rule Weak-Late-Sim.transitive)
  qed
qed

from  $p$  have  $(P, Q) \in ?X$  by(blast intro: Strong-Late-Bisim.reflexive)
moreover from step Sim have  $\bigwedge P \ Q. (P, Q) \in ?X \implies P \rightsquigarrow^{\langle ?X \rangle} Q \wedge (Q, P) \in ?X$ 
by(blast dest: Strong-Late-Bisim.bisimE Strong-Late-Bisim.symmetric)

ultimately show  $?thesis$  by(rule weak-coinduct)
qed

lemma weakBisimTransitiveCoinduct[case-names cSim cSym, consumes 2]:
assumes  $p$ :  $(P, Q) \in X$ 
and Eqvt: eqvt  $X$ 
and rSim:  $\bigwedge P \ Q. (P, Q) \in X \implies P \rightsquigarrow^{\langle (bisim \ O \ X \ O \ bisim) \rangle} Q$ 
and rSym:  $\bigwedge P \ Q. (P, Q) \in X \implies (Q, P) \in X$ 

shows  $P \approx Q$ 
using  $assms$ 
by(coinduct rule: transitive-coinduct-weak) auto

end

theory Weak-Late-Step-Sim
imports Weak-Late-Step-Semantics Weak-Late-Sim Strong-Late-Sim
begin

definition weakStepSimAct ::  $pi \Rightarrow residual \Rightarrow ('a::fs-name) \Rightarrow (pi \times pi) \ set \Rightarrow bool$  where
  weakStepSimAct  $P \ Rs \ C \ Rel \equiv (\forall Q' \ a \ x. Rs = a \langle \nu x \rangle \prec Q' \longrightarrow x \# C \longrightarrow$ 

```

$(\exists P' . P \Rightarrow_{\text{I}} a \langle \nu x \rangle \prec P' \wedge (P', Q') \in \text{Rel}) \wedge$
 $(\forall Q' a x. Rs = a \langle x \rangle \prec Q' \longrightarrow x \# C \longrightarrow (\exists P'' . \forall u. \exists P' .$
 $P \Rightarrow_{\text{I}} u \text{ in } P'' \rightarrow a \langle x \rangle \prec P' \wedge (P', Q'[x::=u]) \in \text{Rel})) \wedge$
 $(\forall Q' \alpha. Rs = \alpha \prec Q' \longrightarrow (\exists P' . P \Rightarrow_{\text{I}} \alpha \prec P' \wedge (P', Q') \in$
 $\text{Rel}))$

definition *weakStepSimAux* :: $pi \Rightarrow (pi \times pi)$ set $\Rightarrow pi \Rightarrow bool$ **where**
 $weakStepSimAux P Rel Q \equiv (\forall Q' a x. (Q \mapsto a \langle \nu x \rangle \prec Q' \wedge x \# P) \longrightarrow (\exists P' .$
 $P \Rightarrow_{\text{I}} a \langle \nu x \rangle \prec P' \wedge (P', Q') \in \text{Rel})) \wedge$
 $(\forall Q' a x. (Q \mapsto a \langle x \rangle \prec Q' \wedge x \# P) \longrightarrow (\exists P'' . \forall u. \exists P' .$
 $P \Rightarrow_{\text{I}} u \text{ in } P'' \rightarrow a \langle x \rangle \prec P' \wedge (P', Q'[x::=u]) \in \text{Rel})) \wedge$
 $(\forall Q' \alpha. Q \mapsto \alpha \prec Q' \longrightarrow (\exists P' . P \Rightarrow_{\text{I}} \alpha \prec P' \wedge (P', Q')$
 $\in \text{Rel}))$

definition *weakStepSim* :: $pi \Rightarrow (pi \times pi)$ set $\Rightarrow pi \Rightarrow bool$ ($\prec \rightsquigarrow \langle - \rangle \rightarrow$ [80, 80, 80] 80) **where**

$P \rightsquigarrow \langle \text{Rel} \rangle Q \equiv (\forall Rs. Q \mapsto Rs \longrightarrow weakStepSimAct P Rs P Rel)$

lemmas *weakStepSimDef* = *weakStepSimAct-def weakStepSim-def*

lemma *weakStepSimAux* $P Rel Q = weakStepSim P Rel Q$

by (*auto simp add: weakStepSimDef weakStepSimAux-def*)

lemma *monotonic*:

fixes $A :: (pi \times pi)$ set
and $B :: (pi \times pi)$ set
and $P :: pi$
and $P' :: pi$

assumes $P \rightsquigarrow \langle A \rangle P'$
and $A \subseteq B$

shows $P \rightsquigarrow \langle B \rangle P'$

using *assms*

apply (*auto simp add: weakStepSimDef*)

apply *blast*

apply (*erule-tac x=a<x> < Q' in allE*)

apply (*clarsimp*)

apply (*rotate-tac 4*)

apply (*erule-tac x=Q' in allE*)

apply (*erule-tac x=a in allE*)

apply (*erule-tac x=x in allE*)

by *blast+*

lemma *simCasesCont*[*consumes 1, case-names Bound Input Free*]:

fixes $P :: pi$
and $Q :: pi$
and $Rel :: (pi \times pi)$ set
and $C :: 'a::fs-name$

assumes *Eqvt*: *eqvt Rel*
and *Bound*: $\bigwedge Q' a x. \llbracket x \# C; Q \mapsto a \langle \nu x \rangle \prec Q \rrbracket \implies \exists P'. P \implies_l a \langle \nu x \rangle \prec P' \wedge (P', Q') \in \text{Rel}$
and *Input*: $\bigwedge Q' a x. \llbracket x \# C; Q \mapsto a \langle x \rangle \prec Q \rrbracket \implies \exists P''. \forall u. \exists P'. P \implies_l u \text{ in } P'' \rightarrow a \langle x \rangle \prec P' \wedge (P', Q'[x::=u]) \in \text{Rel}$
and *Free*: $\bigwedge Q' \alpha. Q \mapsto \alpha \prec Q' \implies (\exists P'. P \implies_l \alpha \prec P' \wedge (P', Q') \in \text{Rel})$

shows $P \rightsquigarrow \langle \text{Rel} \rangle Q$

using *Free*

proof(*auto simp add: weakStepSimDef*)

fix $Q' a x$

assume $x\text{Fresh}P: (x::\text{name}) \# P$

assume *Trans*: $Q \mapsto a \langle \nu x \rangle \prec Q'$

have $\exists c::\text{name}. c \# (P, Q', x, C)$ **by**(*blast intro: name-exists-fresh*)

then obtain $c::\text{name}$ **where** $c\text{Fresh}P: c \# P$ **and** $c\text{Fresh}Q': c \# Q'$ **and** $c\text{Fresh}C: c \# C$

and $c\text{ineq}x: c \neq x$

by(*force simp add: fresh-prod*)

from *Trans* $c\text{Fresh}Q'$ **have** $Q \mapsto a \langle \nu c \rangle \prec ((x, c) \cdot Q')$ **by**(*simp add: alphaBoundResidual*)

with $c\text{Fresh}C$ **have** $\exists P'. P \implies_l a \langle \nu c \rangle \prec P' \wedge (P', [(x, c)] \cdot Q') \in \text{Rel}$

by(*rule Bound*)

then obtain P' **where** $P\text{Trans}: P \implies_l a \langle \nu c \rangle \prec P'$ **and** $P'\text{Rel}Q': (P', [(x, c)] \cdot Q') \in \text{Rel}$

by *blast*

from $P\text{Trans}$ $x\text{Fresh}P$ $c\text{ineq}x$ **have** $x\text{Fresh}P': x \# P'$ **by**(*force dest: Weak-Late-Step-Semantics.freshTransition*)

with $P\text{Trans}$ **have** $P \implies_l a \langle \nu x \rangle \prec ((x, c) \cdot P')$ **by**(*simp add: alphaBoundResidual name-swap*)

moreover have $((x, c) \cdot P', Q') \in \text{Rel}$ (**is** *?goal*)

proof –

from *Eqvt* $P'\text{Rel}Q'$ **have** $((x, c) \cdot P', [(x, c)] \cdot [(x, c)] \cdot Q') \in \text{Rel}$

by(*rule eqvtRelI*)

with $c\text{ineq}x$ **show** *?goal* **by**(*simp add: name-calc*)

qed

ultimately show $\exists P'. P \implies_l a \langle \nu x \rangle \prec P' \wedge (P', Q') \in \text{Rel}$ **by** *blast*

next

fix $Q' a x u$

assume $Q\text{Trans}: Q \mapsto a \langle x \rangle \prec (Q'::\text{pi})$

and $x\text{Fresh}P: x \# P$

have $\exists c::\text{name}. c \# (P, Q', C, x)$ **by**(*blast intro: name-exists-fresh*)

then obtain $c::\text{name}$ **where** $c\text{Fresh}P: c \# P$ **and** $c\text{Fresh}Q': c \# Q'$ **and** $c\text{Fresh}C: c \# C$

and $c\text{ineq}x: c \neq x$

by(*force simp add: fresh-prod*)

from $Q\text{Trans}$ $c\text{Fresh}Q'$ **have** $Q \mapsto a \langle c \rangle \prec ((x, c) \cdot Q')$ **by**(*simp add: al-*

phaBoundResidual

with *cFreshC* **have** $\exists P''. \forall u. \exists P'. P \Longrightarrow_l u \text{ in } P'' \rightarrow a \langle c \rangle \prec P' \wedge (P', ([x, c]) \cdot Q')[c::=u]) \in \text{Rel}$
by(*rule Input*)

then obtain P'' **where** $L1: \forall u. \exists P'. P \Longrightarrow_l u \text{ in } P'' \rightarrow a \langle c \rangle \prec P' \wedge (P', ([x, c]) \cdot Q')[c::=u]) \in \text{Rel}$ **by** *blast*

have $\forall u. \exists P'. P \Longrightarrow_l u \text{ in } ([c, x]) \cdot P'' \rightarrow a \langle x \rangle \prec P' \wedge (P', Q'[x::=u]) \in \text{Rel}$
proof(*auto*)

fix u

from $L1$ **obtain** P' **where** $P\text{Trans}: P \Longrightarrow_l u \text{ in } P'' \rightarrow a \langle c \rangle \prec P'$ **and** $P'\text{Rel}Q': (P', ([x, c]) \cdot Q')[c::=u]) \in \text{Rel}$
by *blast*

from $P\text{Trans}$ $x\text{Fresh}P$ **have** $P \Longrightarrow_l u \text{ in } ([c, x]) \cdot P'' \rightarrow a \langle x \rangle \prec P'$ **by**(*rule alphaInput*)

moreover from $P'\text{Rel}Q'$ $c\text{Fresh}Q'$ **have** $(P', Q'[x::=u]) \in \text{Rel}$ **by**(*simp add: renaming[THEN sym] name-swap*)

ultimately show $\exists P'. P \Longrightarrow_l u \text{ in } ([c, x]) \cdot P'' \rightarrow a \langle x \rangle \prec P' \wedge (P', Q'[x::=u]) \in \text{Rel}$ **by** *blast*
qed

thus $\exists P''. \forall u. \exists P'. P \Longrightarrow_l u \text{ in } P'' \rightarrow a \langle x \rangle \prec P' \wedge (P', Q'[x::=u]) \in \text{Rel}$ **by** *blast*
qed

lemma *simCases[consumes 0, case-names Bound Input Free]*:

fixes $P :: pi$

and $Q :: pi$

and $\text{Rel} :: (pi \times pi)$ *set*

and $C :: 'a::fs\text{-name}$

assumes *Bound*: $\bigwedge Q' a x. \llbracket Q \mapsto a \langle \nu x \rangle \prec Q'; x \# P \rrbracket \Longrightarrow \exists P'. P \Longrightarrow_l a \langle \nu x \rangle \prec P' \wedge (P', Q') \in \text{Rel}$

and *Input*: $\bigwedge Q' a x. \llbracket Q \mapsto a \langle x \rangle \prec Q'; x \# P \rrbracket \Longrightarrow \exists P''. \forall u. \exists P'. P \Longrightarrow_l u \text{ in } P'' \rightarrow a \langle x \rangle \prec P' \wedge (P', Q'[x::=u]) \in \text{Rel}$

and *Free*: $\bigwedge Q' \alpha. Q \mapsto \alpha \prec Q' \Longrightarrow (\exists P'. P \Longrightarrow_l \alpha \prec P' \wedge (P', Q') \in \text{Rel})$

shows $P \rightsquigarrow \langle \text{Rel} \rangle Q$

using *assms*

by(*auto simp add: weakStepSimDef*)

lemma *simActBoundCases[consumes 1, case-names Input BoundOutput]*:

fixes $P :: pi$

and $a :: \text{subject}$

and $x :: \text{name}$

and $Q' :: pi$

```

and C :: 'a::fs-name
and Rel :: (pi × pi) set

assumes EqvtRel: eqvt Rel
and DerInput:  $\bigwedge b. a = \text{InputS } b \implies (\exists P''. \forall u. \exists P'. (P \implies_l u \text{ in } P'' \rightarrow b \langle x \rangle \prec P') \wedge (P', Q'[x::=u]) \in \text{Rel})$ 
and DerBoundOutput:  $\bigwedge b. a = \text{BoundOutputS } b \implies (\exists P'. (P \implies_l b \langle \nu x \rangle \prec P') \wedge (P', Q') \in \text{Rel})$ 

shows weakStepSimAct P (a«x»  $\prec$  Q') P Rel
proof(simp add: weakStepSimAct-def fresh-prod, auto)
  fix Q'' b y
  assume Eq: a«x»  $\prec$  Q' = b  $\langle \nu y \rangle \prec$  Q''
  assume yFreshP: y  $\#$  P

  from Eq have a = BoundOutputS b by(simp add: residual.inject)

  from yFreshP DerBoundOutput[OF this] Eq show  $\exists P'. P \implies_l b \langle \nu y \rangle \prec P' \wedge (P', Q'') \in \text{Rel}$ 
  proof(cases x=y, auto simp add: residual.inject name-abs-eq)
    fix P'
    assume PTrans: P  $\implies_l b \langle \nu x \rangle \prec P'$ 
    assume P'RelQ': (P', ((x, y)  $\cdot$  Q''))  $\in$  Rel
    assume xineqy: x  $\neq$  y

    with PTrans yFreshP have yFreshP': y  $\#$  P'
      by(force intro: Weak-Late-Step-Semantics.freshTransition)

    hence b  $\langle \nu x \rangle \prec P' = b \langle \nu y \rangle \prec [(x, y)  $\cdot$  P']$  by(rule alphaBoundResidual)
    moreover have ((x, y)  $\cdot$  P', Q'')  $\in$  Rel
    proof –
      from EqvtRel P'RelQ' have ((x, y)  $\cdot$  P', [(x, y)  $\cdot$  ((x, y)  $\cdot$  Q'')]  $\in$  Rel
        by(rule eqvtRelI)
      thus ?thesis by(simp add: name-calc)
    qed

    ultimately show  $\exists P'. P \implies_l b \langle \nu y \rangle \prec P' \wedge (P', Q'') \in \text{Rel}$  using PTrans
  by auto
  qed
next
  fix Q'' b y u
  assume Eq: a«x»  $\prec$  Q' = b  $\langle y \rangle \prec$  Q''
  assume yFreshP: y  $\#$  P

  from Eq have a = InputS b by(simp add: residual.inject)
  from DerInput[OF this] obtain P'' where L1:  $\forall u. \exists P'. P \implies_l u \text{ in } P'' \rightarrow b \langle x \rangle \prec P' \wedge (P', Q'[x::=u]) \in \text{Rel}$ 

  by blast

```

have $\forall u. \exists P'. P \Longrightarrow_{lu} \text{in } ((x, y) \cdot P'') \rightarrow b \langle y \rangle \prec P' \wedge (P', Q''[y::=u]) \in \text{Rel}$
proof(*rule allI*)
fix u
from *L1 Eq* **show** $\exists P'. P \Longrightarrow_{lu} \text{in } ((x, y) \cdot P'') \rightarrow b \langle y \rangle \prec P' \wedge (P', Q''[y::=u]) \in \text{Rel}$
proof(*cases x=y, auto simp add: residual.inject name-abs-eq*)
assume $\text{Der}: \forall u. \exists P'. P \Longrightarrow_{lu} \text{in } P'' \rightarrow b \langle x \rangle \prec P' \wedge (P', ((x, y) \cdot Q'')[x::=u]) \in \text{Rel}$
assume $x \text{Fresh} Q'': x \# Q''$

from Der **obtain** P' **where** $P \text{Trans}: P \Longrightarrow_{lu} \text{in } P'' \rightarrow b \langle x \rangle \prec P'$
and $P' \text{Rel} Q': (P', ((x, y) \cdot Q'')[x::=u]) \in \text{Rel}$
by *force*

from $P \text{Trans}$ $y \text{Fresh} P$ **have** $P \Longrightarrow_{lu} \text{in } ((x, y) \cdot P'') \rightarrow b \langle y \rangle \prec P'$ **by**(*rule alphaInput*)
moreover from $x \text{Fresh} Q''$ $P' \text{Rel} Q'$ **have** $(P', Q''[y::=u]) \in \text{Rel}$
by(*simp add: renaming*)
ultimately show *?thesis* **by** *force*
qed
qed
thus $\exists P''. \forall u. \exists P'. P \Longrightarrow_{lu} \text{in } P'' \rightarrow b \langle y \rangle \prec P' \wedge (P', Q''[y::=u]) \in \text{Rel}$
by *blast*
qed

lemma *simActFreeCases*[*consumes 0, case-names Free*]:

fixes $P :: pi$
and $\alpha :: \text{freeRes}$
and $C :: 'a::\text{fs-name}$
and $\text{Rel} :: (pi \times pi) \text{ set}$

assumes $\text{Der}: \exists P'. (P \Longrightarrow_{l\alpha} \prec P') \wedge (P', Q') \in \text{Rel}$

shows $\text{weakStepSimAct } P (\alpha \prec Q') P \text{ Rel}$

using *assms*

by(*simp add: weakStepSimAct-def residual.inject*)

lemma *simE*:

fixes $P :: pi$
and $\text{Rel} :: (pi \times pi) \text{ set}$
and $Q :: pi$
and $a :: \text{name}$
and $x :: \text{name}$
and $u :: \text{name}$
and $Q' :: pi$

assumes $P \rightsquigarrow \langle \text{Rel} \rangle Q$

shows $Q \mapsto a \langle \nu x \rangle \prec Q' \implies x \# P \implies \exists P'. P \implies_{\iota} a \langle \nu x \rangle \prec P' \wedge (P', Q') \in Rel$
and $Q \mapsto a \langle x \rangle \prec Q' \implies x \# P \implies \exists P''. \forall u. \exists P'. P \implies_{\iota} u \text{ in } P'' \rightarrow a \langle x \rangle \prec P' \wedge (P', Q'[x::=u]) \in Rel$
and $Q \mapsto \alpha \prec Q' \implies (\exists P'. P \implies_{\iota} \alpha \prec P' \wedge (P', Q') \in Rel)$
using *assms* **by** (*simp add: weakStepSimDef*)⁺

lemma *weakSimTauChain*:

fixes $P :: pi$
and $Rel :: (pi \times pi) \text{ set}$
and $Q :: pi$
and $Q' :: pi$

assumes $QChain: Q \implies_{\tau} Q'$
and $PRelQ: (P, Q) \in Rel$
and $Sim: \bigwedge P Q. (P, Q) \in Rel \implies P \rightsquigarrow \langle Rel \rangle Q$

shows $\exists P'. P \implies_{\tau} P' \wedge (P', Q') \in Rel$

proof –

from $QChain$ **show** *?thesis*

proof (*induct rule: tauChainInduct*)

case *id*

have $P \implies_{\tau} P$ **by** *simp*

with $PRelQ$ **show** *?case* **by** *blast*

next

case (*ih* $Q' Q''$)

have $IH: \exists P'. P \implies_{\tau} P' \wedge (P', Q') \in Rel$ **by** *fact*

then obtain P' **where** $PChain: P \implies_{\tau} P'$ **and** $P'RelQ': (P', Q') \in Rel$ **by**

blast

from $P'RelQ'$ **have** $P' \rightsquigarrow \langle Rel \rangle Q'$ **by** (*rule Sim*)

moreover have $Q'Trans: Q' \mapsto_{\tau} \prec Q''$ **by** *fact*

ultimately have $\exists P''. P' \implies_{\iota} \tau \prec P'' \wedge (P'', Q'') \in Rel$ **by** (*rule simE*)

then obtain P'' **where** $P'Trans: P' \implies_{\iota} \tau \prec P''$ **and** $P''RelQ'': (P'', Q'') \in$

Rel **by** *blast*

from $P'Trans$ **have** $P' \implies_{\tau} P''$ **by** (*rule Weak-Late-Step-Semantics.tauTransitionChain*)

with $PChain$ **have** $P \implies_{\tau} P''$ **by** *auto*

with $P''RelQ''$ **show** *?case* **by** *blast*

qed

qed

lemma *strongSimWeakEqSim*:

fixes $P :: pi$
and $Q :: pi$
and $Rel :: (pi \times pi) \text{ set}$

assumes $PSimQ: P \rightsquigarrow [Rel] Q$

shows $P \rightsquigarrow \langle Rel \rangle Q$

proof (*auto simp add: weakStepSimDef*)

fix $Q' a x$
assume $Q \mapsto a \langle \nu x \rangle \prec Q'$ **and** $x \# P$
with $PSimQ$ **have** $\exists P'. P \mapsto a \langle \nu x \rangle \prec P' \wedge \text{derivative } P' Q'$ (*BoundOutputS*
a) $x \text{ Rel}$
by(*rule Strong-Late-Sim.simE*)
then obtain P' **where** $PTrans: P \mapsto a \langle \nu x \rangle \prec P'$ **and** $P'RelQ': (P', Q') \in$
 Rel
by(*force simp add: derivative-def*)

from $PTrans$ **have** $P \Longrightarrow_1 a \langle \nu x \rangle \prec P'$ **by**(*rule Weak-Late-Step-Semantics.singleActionChain*)
thus $\exists P'. P \Longrightarrow_1 a \langle \nu x \rangle \prec P' \wedge (P', Q') \in Rel$ **using** $P'RelQ'$ **by** *blast*
next

fix $Q' a x u$
assume $Q \mapsto a \langle x \rangle \prec Q'$ **and** $x \# P$
with $PSimQ$ **have** $L1: \exists P'. P \mapsto a \langle x \rangle \prec P' \wedge \text{derivative } P' Q'$ (*InputS a*) x
 Rel
by(*blast intro: Strong-Late-Sim.simE*)
then obtain P' **where** $PTrans: P \mapsto a \langle x \rangle \prec P'$ **and** $PDer: \text{derivative } P' Q'$
(*InputS a*) $x \text{ Rel}$
by *blast*

have $\forall u. \exists P''. P \Longrightarrow_1 u \text{ in } P' \rightarrow a \langle x \rangle \prec P'' \wedge (P'', Q'[x::=u]) \in Rel$
proof(*rule allI*)

fix u
from $PTrans$ **have** $P \Longrightarrow_1 u \text{ in } P' \rightarrow a \langle x \rangle \prec P'[x::=u]$ **by**(*blast intro: Weak-Late-Step-Semantics.singleActionChain*)
moreover from $PDer$ **have** $(P'[x::=u], Q'[x::=u]) \in Rel$ **by**(*force simp add: derivative-def*)

ultimately show $\exists P''. P \Longrightarrow_1 u \text{ in } P' \rightarrow a \langle x \rangle \prec P'' \wedge (P'', Q'[x::=u]) \in Rel$
by *auto*

qed
thus $\exists P''. \forall u. \exists P'. P \Longrightarrow_1 u \text{ in } P'' \rightarrow a \langle x \rangle \prec P' \wedge (P', Q'[x::=u]) \in Rel$ **by**
blast

next
fix $Q' \alpha$
assume $Q \mapsto \alpha \prec Q'$
with $PSimQ$ **have** $\exists P'. P \mapsto \alpha \prec P' \wedge (P', Q') \in Rel$ **by**(*rule Strong-Late-Sim.simE*)
then obtain P' **where** $PTrans: P \mapsto \alpha \prec P'$ **and** $P'RelQ': (P', Q') \in Rel$ **by**
blast

from $PTrans$ **have** $P \Longrightarrow_1 \alpha \prec P'$ **by**(*rule Weak-Late-Step-Semantics.singleActionChain*)
thus $\exists P'. P \Longrightarrow_1 \alpha \prec P' \wedge (P', Q') \in Rel$ **using** $P'RelQ'$ **by** *blast*

qed

lemma *weakSimWeakEqSim:*

fixes $P \quad :: pi$
and $Q \quad :: pi$
and $Rel \quad :: (pi \times pi) \text{ set}$

assumes $P \rightsquigarrow \langle Rel \rangle Q$

```

  shows  $P \rightsquigarrow^{\wedge} \langle Rel \rangle Q$ 
using assms
by(force simp add: weakStepSimDef simDef weakTransition-def)

lemma eqvtI:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $Rel :: (pi \times pi) set$ 
  and  $perm :: name prm$ 

  assumes  $Sim: P \rightsquigarrow \langle Rel \rangle Q$ 
  and  $RelRel': Rel \subseteq Rel'$ 
  and  $EqvtRel': eqvt Rel'$ 

  shows  $(perm \cdot P) \rightsquigarrow \langle Rel' \rangle (perm \cdot Q)$ 
using EqvtRel'
proof(induct rule: simCasesCont[of - perm \cdot P])
  case(Bound Q' a x)
  have  $QTrans: (perm \cdot Q) \mapsto a \langle \nu x \rangle \prec Q'$  by fact
  have  $xFreshP: x \# perm \cdot P$  by fact

  from  $QTrans$  have  $(rev perm \cdot (perm \cdot Q)) \mapsto rev perm \cdot (a \langle \nu x \rangle \prec Q')$ 
  by(rule eqvts)
  hence  $Q \mapsto (rev perm \cdot a) \langle \nu(rev perm \cdot x) \rangle \prec (rev perm \cdot Q')$ 
  by(simp add: name-rev-per)
  moreover from  $xFreshP$  have  $(rev perm \cdot x) \# P$  by(simp add: name-fresh-left)
  ultimately obtain  $P'$  where  $PTrans: P \Longrightarrow_l (rev perm \cdot a) \langle \nu(rev perm \cdot x) \rangle \prec P'$ 
  and  $P'RelQ': (P', rev perm \cdot Q') \in Rel$  using Sim
  by(blast dest: simE)

  from  $PTrans$  have  $(perm \cdot P) \Longrightarrow_l perm \cdot ((rev perm \cdot a) \langle \nu(rev perm \cdot x) \rangle \prec P')$ 
  by(rule Weak-Late-Step-Semantics.eqvtI)
  hence  $(perm \cdot P) \Longrightarrow_l a \langle \nu x \rangle \prec (perm \cdot P')$  by(simp add: name-per-rev)
  moreover have  $(perm \cdot P', Q') \in Rel'$ 
  proof –
  from  $P'RelQ' RelRel'$  have  $(P', rev perm \cdot Q') \in Rel'$  by blast
  with  $EqvtRel'$  have  $(perm \cdot P', perm \cdot (rev perm \cdot Q')) \in Rel'$ 
  by(rule eqvtRelI)
  thus ?thesis by(simp add: name-per-rev)
qed
  ultimately show ?case by blast
next
  case(Input Q' a x)
  have  $QTrans: (perm \cdot Q) \mapsto a \langle x \rangle \prec Q'$  by fact
  have  $xFreshP: x \# perm \cdot P$  by fact

```

from $QTrans$ **have** $(rev\ perm \cdot (perm \cdot Q)) \mapsto rev\ perm \cdot (a \prec x \prec Q')$
by(*rule eqvts*)
hence $Q \mapsto (rev\ perm \cdot a) \prec (rev\ perm \cdot x) \prec (rev\ perm \cdot Q')$
by(*simp add: name-rev-per*)
moreover from $xFreshP$ **have** $xFreshP: (rev\ perm \cdot x) \# P$ **by**(*simp add: name-fresh-left*)
ultimately obtain P''
where $L1: \forall u. \exists P'. P \Longrightarrow_l u$ in $P'' \rightarrow (rev\ perm \cdot a) \prec (rev\ perm \cdot x) \prec P' \wedge (P', (rev\ perm \cdot Q')[x::=u]) \in Rel$ **using** Sim
by(*blast dest: simE*)
have $\forall u. \exists P'. (perm \cdot P) \Longrightarrow_l u$ in $(perm \cdot P'') \rightarrow a \prec x \prec P' \wedge (P', Q'[x::=u]) \in Rel'$
proof(*rule allI*)
fix u
from $L1$ **obtain** P' **where** $PTrans: P \Longrightarrow_l (rev\ perm \cdot u)$ in $P'' \rightarrow (rev\ perm \cdot a) \prec (rev\ perm \cdot x) \prec P'$
and $P'RelQ': (P', (rev\ perm \cdot Q')[x::=(rev\ perm \cdot u)]) \in Rel$ **by** *blast*
from $PTrans$ **have** $(perm \cdot P) \Longrightarrow_l (perm \cdot (rev\ perm \cdot u))$ in $(perm \cdot P'') \rightarrow (perm \cdot rev\ perm \cdot a) \prec (perm \cdot rev\ perm \cdot x) \prec (perm \cdot P')$
by(*rule-tac Weak-Late-Step-Semantics.eqvtI, auto*)
hence $(perm \cdot P) \Longrightarrow_l u$ in $(perm \cdot P'') \rightarrow a \prec x \prec (perm \cdot P')$ **by**(*simp add: name-per-rev*)
moreover have $(perm \cdot P', Q'[x::=u]) \in Rel'$
proof –
from $P'RelQ'$ $RelRel'$ **have** $(P', (rev\ perm \cdot Q')[x::=(rev\ perm \cdot u)]) \in Rel'$ **by** *blast*
with $EqvRel'$ **have** $(perm \cdot P', perm \cdot ((rev\ perm \cdot Q')[x::=(rev\ perm \cdot u)])) \in Rel'$
by(*rule eqvRelI*)
thus *?thesis* **by**(*simp add: name-per-rev eqvt-subs[THEN sym] name-calc*)
qed
ultimately show $\exists P'. (perm \cdot P) \Longrightarrow_l u$ in $(perm \cdot P'') \rightarrow a \prec x \prec P' \wedge (P', Q'[x::=u]) \in Rel'$ **by** *blast*
qed
thus *?case* **by** *blast*
next
case(*Free Q' α*)
have $QTrans: (perm \cdot Q) \mapsto \alpha \prec Q'$ **by** *fact*

from $QTrans$ **have** $(rev\ perm \cdot (perm \cdot Q)) \mapsto rev\ perm \cdot (\alpha \prec Q')$
by(*rule eqvts*)
hence $Q \mapsto (rev\ perm \cdot \alpha) \prec (rev\ perm \cdot Q')$
by(*simp add: name-rev-per*)
with Sim **obtain** P' **where** $PTrans: P \Longrightarrow_l (rev\ perm \cdot \alpha) \prec P'$ **and** $PRel: (P', (rev\ perm \cdot Q')) \in Rel$
by(*blast dest: simE*)
from $PTrans$ **have** $(perm \cdot P) \Longrightarrow_l perm \cdot ((rev\ perm \cdot \alpha) \prec P')$
by(*rule Weak-Late-Step-Semantics.eqvtI*)

hence $(perm \cdot P) \Longrightarrow_I \alpha \prec (perm \cdot P')$ **by** (*simp add: name-per-rev*)
 moreover **have** $((perm \cdot P'), Q') \in Rel'$
proof –
 from $PRel\ EqvtRel'\ RelRel'$ **have** $((perm \cdot P'), (perm \cdot (rev\ perm \cdot Q')))$ $\in Rel'$
by (*force intro: eqvtRelI*)
thus $?thesis$ **by** (*simp add: name-per-rev*)
qed
 ultimately **show** $?case$ **by** *blast*
qed

lemma *simE2*:

fixes $P :: pi$
and $Rel :: (pi \times pi)\ set$
and $Q :: pi$
and $a :: name$
and $x :: name$
and $Q' :: pi$

assumes $PSimQ: P \rightsquigarrow \langle Rel \rangle Q$
and $Sim: \bigwedge P\ Q. (P, Q) \in Rel \Longrightarrow P \rightsquigarrow^{\hat{}} \langle Rel \rangle Q$
and $Eqvt: eqvt\ Rel$
and $PRelQ: (P, Q) \in Rel$

shows $Q \Longrightarrow_I a \langle \nu x \rangle \prec Q' \Longrightarrow x \# P \Longrightarrow \exists P'. P \Longrightarrow_I a \langle \nu x \rangle \prec P' \wedge (P', Q') \in Rel$

and $Q \Longrightarrow_I \alpha \prec Q' \Longrightarrow \exists P'. P \Longrightarrow_I \alpha \prec P' \wedge (P', Q') \in Rel$

proof –

assume $QTrans: Q \Longrightarrow_I a \langle \nu x \rangle \prec Q'$

assume $xFreshP: x \# P$

have $Goal: \bigwedge P\ Q\ a\ x\ Q'. \llbracket P \rightsquigarrow \langle Rel \rangle Q; Q \Longrightarrow_I a \langle \nu x \rangle \prec Q'; x \# P; x \# Q; (P, Q) \in Rel \rrbracket \Longrightarrow$

$\exists P'. P \Longrightarrow_I a \langle \nu x \rangle \prec P' \wedge (P', Q') \in Rel$

proof –

fix $P\ Q\ a\ x\ Q'$

assume $PSimQ: P \rightsquigarrow \langle Rel \rangle Q$

assume $QTrans: Q \Longrightarrow_I a \langle \nu x \rangle \prec Q'$

assume $xFreshP: x \# P$

assume $xFreshQ: x \# Q$

assume $PRelQ: (P, Q) \in Rel$

from $QTrans\ xFreshQ$ **obtain** $Q''\ Q'''$ **where** $QChain: Q \Longrightarrow_{\tau} Q''$

and $Q''Trans: Q'' \mapsto a \langle \nu x \rangle \prec Q'''$

and $Q'''Chain: Q''' \Longrightarrow_{\tau} Q'$

by (*force dest: transitionE simp add: weakTransition-def*)

from $QChain\ PRelQ\ Sim$ **have** $\exists P''. P \Longrightarrow_{\tau} P'' \wedge (P'', Q') \in Rel$

by (*rule Weak-Late-Sim.weakSimTauChain*)

then obtain P'' **where** $PChain: P \Longrightarrow_{\tau} P''$ **and** $P''RelQ'': (P'', Q') \in Rel$

by *blast*
from $PChain\ xFreshP$ **have** $xFreshP'' : x \# P''$ **by**(*rule freshChain*)

from $P''RelQ''$ **have** $P'' \rightsquigarrow \hat{\ } <Rel> Q''$ **by**(*rule Sim*)
hence $\exists P''' . P'' \Longrightarrow_l a <\nu x> \prec P''' \wedge (P''', Q''') \in Rel$ **using** $Q''Trans$
 $xFreshP''$
by(*rule Weak-Late-Sim.simE*)
then obtain P''' **where** $P''Trans : P'' \Longrightarrow_l a <\nu x> \prec P'''$ **and** $P'''RelQ''' :$
 $(P''', Q''') \in Rel$
by(*force simp add: weakTransition-def*)

have $\exists P' . P''' \Longrightarrow_\tau P' \wedge (P', Q') \in Rel$ **using** $Q'''Chain\ P'''RelQ'''\ Sim$
by(*rule Weak-Late-Sim.weakSimTauChain*)
then obtain P' **where** $P'''Chain : P''' \Longrightarrow_\tau P'$ **and** $P'RelQ' : (P', Q') \in Rel$
by *blast*

from $PChain\ P''Trans\ P'''Chain\ xFreshP''$ **have** $P \Longrightarrow_l a <\nu x> \prec P'$
by(*blast dest: Weak-Late-Step-Semantics.chainTransitionAppend*)
with $P'RelQ'$ **show** $\exists P' . P \Longrightarrow_l a <\nu x> \prec P' \wedge (P', Q') \in Rel$ **by** *blast*
qed

have $\exists c :: name . c \# (Q, Q', P, x)$ **by**(*blast intro: name-exists-fresh*)
then obtain $c :: name$ **where** $cFreshQ : c \# Q$ **and** $cFreshQ' : c \# Q'$ **and** $cFreshP :$
 $c \# P$
and $xineqc : x \neq c$
by(*force simp add: fresh-prod*)

from $QTrans\ cFreshQ'$ **have** $Q \Longrightarrow_l a <\nu c> \prec ((x, c) \cdot Q')$ **by**(*simp add: alphaBoundResidual*)
with $PSimQ$ **have** $\exists P' . P \Longrightarrow_l a <\nu c> \prec P' \wedge (P', [(x, c) \cdot Q']) \in Rel$ **using**
 $cFreshP\ cFreshQ\ PRelQ$
by(*rule Goal*)
then obtain P' **where** $PTrans : P \Longrightarrow_l a <\nu c> \prec P'$ **and** $P'RelQ' : (P', [(x, c)$
 $\cdot Q']) \in Rel$
by *force*
have $P \Longrightarrow_l a <\nu x> \prec ((x, c) \cdot P')$
proof –
from $PTrans\ xFreshP\ xineqc$ **have** $x \# P'$ **by**(*rule Weak-Late-Step-Semantics.freshTransition*)
with $PTrans$ **show** *?thesis* **by**(*simp add: alphaBoundResidual name-swap*)
qed
moreover have $((x, c) \cdot P', Q') \in Rel$
proof –
from $Eqvt\ P'RelQ'$ **have** $((x, c) \cdot P', [(x, c) \cdot [(x, c) \cdot Q']) \in Rel$
by(*rule eqvtRelI*)
thus *?thesis* **by** *simp*
qed

ultimately show $\exists P' . P \Longrightarrow_l a <\nu x> \prec P' \wedge (P', Q') \in Rel$ **by** *blast*
next

```

assume  $QTrans: Q \Rightarrow_l \alpha \prec Q'$ 

then obtain  $Q'' Q'''$  where  $QChain: Q \Rightarrow_\tau Q''$ 
  and  $Q''Trans: Q'' \mapsto_\alpha \prec Q'''$ 
  and  $Q'''Chain: Q''' \Rightarrow_\tau Q'$ 
  by(blast dest: transitionE)

thus  $\exists P'. P \Rightarrow_l \alpha \prec P' \wedge (P', Q') \in Rel$ 
proof(induct arbitrary:  $\alpha Q''' Q'$  rule: tauChainInduct)
  case(id  $\alpha Q'''$ )
  from  $PSimQ \langle Q \mapsto_\alpha \prec Q''' \rangle$  have  $\exists P'. P \Rightarrow_l \alpha \prec P' \wedge (P', Q''') \in Rel$ 
    by(blast dest: simE)
  then obtain  $P'''$  where  $PTrans: P \Rightarrow_l \alpha \prec P'''$  and  $P'RelQ''': (P''', Q''') \in Rel$ 
  by blast

  have  $\exists P'. P''' \Rightarrow_\tau P' \wedge (P', Q') \in Rel$  using  $\langle Q''' \Rightarrow_\tau Q' \rangle P'RelQ''' Sim$ 
    by(rule Weak-Late-Sim.weakSimTauChain)
  then obtain  $P'$  where  $P'''Chain: P''' \Rightarrow_\tau P'$  and  $P'RelQ': (P', Q') \in Rel$ 
  by blast

  from  $P'''Chain PTrans$  have  $P \Rightarrow_l \alpha \prec P'$ 
    by(blast dest: Weak-Late-Step-Semantics.chainTransitionAppend)

  with  $P'RelQ'$  show ?case by blast
  next
  case(ih  $Q'''' Q''' \alpha Q'' Q'$ )
  have  $Q''' \Rightarrow_\tau Q''$  by simp
  with  $\langle Q'''' \mapsto_\tau \prec Q''' \rangle$  obtain  $P''$  where  $PTrans: P \Rightarrow_l \tau \prec P''$  and
 $P''RelQ''': (P''', Q''') \in Rel$ 
    by(drule-tac ih) auto
  from  $P''RelQ'''' \langle Q'''' \mapsto_\alpha \prec Q'' \rangle$  obtain  $P''$  where
 $P''Trans: P'' \Rightarrow_l \alpha \prec P''$  and  $P''RelQ'': (P'', Q'') \in Rel$ 
    by(blast dest: Weak-Late-Sim.simE Sim)
  from  $P''RelQ'' \langle Q'' \Rightarrow_\tau Q' \rangle Sim$  obtain  $P'$  where
 $P''Chain: P'' \Rightarrow_\tau P'$  and  $P'RelQ': (P', Q') \in Rel$ 
    by(drule-tac Weak-Late-Sim.weakSimTauChain) auto

  from  $PTrans P''Trans P''Chain$  have  $P \Rightarrow_l \alpha \prec P'$ 
    apply(auto simp add: weakTransition-def residual.inject)
    apply(drule-tac Weak-Late-Step-Semantics.tauTransitionChain, auto)
    apply(drule-tac Weak-Late-Step-Semantics.chainTransitionAppend, simp)
    apply(rule Weak-Late-Step-Semantics.chainTransitionAppend, auto)
    by(drule-tac Weak-Late-Step-Semantics.chainTransitionAppend, auto)
  with  $\langle (P', Q') \in Rel \rangle$  show ?case by blast
qed
qed

```

```

lemma reflexive:
  fixes  $P :: pi$ 
  and  $Rel :: (pi \times pi)$  set

  assumes  $Id \subseteq Rel$ 

  shows  $P \rightsquigarrow \langle Rel \rangle P$ 
using assms
by(auto intro: Weak-Late-Step-Semantics.singleActionChain simp add: weakStep-SimDef)

lemma transitive:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $R :: pi$ 
  and  $Rel :: (pi \times pi)$  set
  and  $Rel' :: (pi \times pi)$  set
  and  $Rel'' :: (pi \times pi)$  set

  assumes  $PSimQ: P \rightsquigarrow \langle Rel \rangle Q$ 
  and  $QSimR: Q \rightsquigarrow \langle Rel' \rangle R$ 
  and  $Eqvt: eqvt Rel$ 
  and  $Eqvt': eqvt Rel''$ 
  and  $Trans: Rel \circ Rel' \subseteq Rel''$ 
  and  $Sim: \bigwedge P Q. (P, Q) \in Rel \implies P \rightsquigarrow \langle Rel \rangle Q$ 
  and  $PRelQ: (P, Q) \in Rel$ 

  shows  $P \rightsquigarrow \langle Rel'' \rangle R$ 
using  $Eqvt'$ 
proof(induct rule: simCasesCont[of - (P, Q)])
  case(Bound  $R' a x$ )
  have  $RTrans: R \mapsto a \langle \nu x \rangle \prec R'$  by fact
  have  $x \# (P, Q)$  by fact
  hence  $xFreshP: x \# P$  and  $xFreshQ: x \# Q$  by(simp add: fresh-prod)+

  from  $QSimR RTrans xFreshQ$  obtain  $Q'$  where  $QTrans: Q \implies \iota a \langle \nu x \rangle \prec Q'$ 
  and  $Q'RelR': (Q', R') \in Rel'$ 

  by(blast dest: simE)
  from  $PSimQ Sim Eqvt PRelQ QTrans xFreshP$  obtain  $P'$  where  $PTrans: P \implies \iota a \langle \nu x \rangle \prec P'$ 
  and  $P'RelQ': (P', Q') \in Rel$ 

  by(blast dest: simE2)
  moreover from  $P'RelQ' Q'RelR' Trans$  have  $(P', R') \in Rel''$  by blast
  ultimately show ?case by blast
next
  case(Input  $R' a x$ )
  have  $RTrans: R \mapsto a \langle x \rangle \prec R'$  by fact
  have  $x \# (P, Q)$  by fact

```

hence $xFreshP: x \# P$ **and** $xFreshQ: x \# Q$ **by** (*simp add: fresh-prod*)⁺

from $QSimR RTrans xFreshQ$ **obtain** Q''
where $\forall u. \exists Q'. Q \Longrightarrow_{\iota} u$ **in** $Q'' \rightarrow a \langle x \rangle \prec Q' \wedge (Q', R'[x::=u]) \in Rel'$
by (*blast dest: simE*)

hence $\exists Q''' . Q \Longrightarrow_{\tau} Q''' \wedge Q''' \mapsto a \langle x \rangle \prec Q'' \wedge (\forall u. \exists Q'. Q''[x::=u] \Longrightarrow_{\tau} Q' \wedge (Q', R'[x::=u]) \in Rel')$

by (*simp add: inputTransition-def, blast*)

then obtain Q''' **where** $QChain: Q \Longrightarrow_{\tau} Q'''$
and $Q'''Trans: Q''' \mapsto a \langle x \rangle \prec Q''$
and $L1: \forall u. \exists Q'. Q''[x::=u] \Longrightarrow_{\tau} Q' \wedge (Q', R'[x::=u]) \in Rel'$

by *blast*

from $QChain PRelQ Sim$ **obtain** P''' **where** $PChain: P \Longrightarrow_{\tau} P'''$ **and** $P'''RelQ''': (P''', Q''') \in Rel$

by (*drule-tac Weak-Late-Sim.weakSimTauChain*) *auto*

from $PChain xFreshP$ **have** $xFreshP''': x \# P'''$ **by** (*rule freshChain*)

from $P'''RelQ'''$ **have** $P''' \rightsquigarrow \langle Rel \rangle Q'''$ **by** (*rule Sim*)

with $xFreshP''' Q'''Trans$ **obtain** P'''' **where** $L2: \forall u. \exists P''. P''' \Longrightarrow_{\iota} u$ **in** $P'''' \rightarrow a \langle x \rangle \prec P'' \wedge (P'', Q''[x::=u]) \in Rel$

by (*blast dest: Weak-Late-Sim.simE*)

have $\forall u. \exists P' Q'. P \Longrightarrow_{\iota} u$ **in** $P'''' \rightarrow a \langle x \rangle \prec P' \wedge (P', R'[x::=u]) \in Rel''$

proof (*rule allI*)

fix u

from $L1$ **obtain** Q' **where** $Q''Chain: Q''[x::=u] \Longrightarrow_{\tau} Q'$ **and** $Q'RelR': (Q', R'[x::=u]) \in Rel'$

by *blast*

from $L2$ **obtain** P'' **where** $P'''Trans: P''' \Longrightarrow_{\iota} u$ **in** $P'''' \rightarrow a \langle x \rangle \prec P''$
and $P''RelQ'': (P'', Q''[x::=u]) \in Rel$

by *blast*

from $P''RelQ''$ **have** $P'' \rightsquigarrow \langle Rel \rangle Q''[x::=u]$ **by** (*rule Sim*)

have $\exists P'. P'' \Longrightarrow_{\tau} P' \wedge (P', Q') \in Rel$ **using** $Q''Chain P''RelQ'' Sim$

by (*rule Weak-Late-Sim.weakSimTauChain*)

then obtain P' **where** $P''Chain: P'' \Longrightarrow_{\tau} P'$ **and** $P'RelQ': (P', Q') \in Rel$

by *blast*

from $PChain P'''Trans P''Chain$ **have** $P \Longrightarrow_{\iota} u$ **in** $P'''' \rightarrow a \langle x \rangle \prec P'$

by (*blast dest: Weak-Late-Step-Semantics.chainTransitionAppend*)

moreover from $P'RelQ' Q'RelR'$ **have** $(P', R'[x::=u]) \in Rel''$ **by** (*insert Trans, auto*)

ultimately show $\exists P' Q'. P \Longrightarrow_{\iota} u$ **in** $P'''' \rightarrow a \langle x \rangle \prec P' \wedge (P', R'[x::=u]) \in Rel''$ **by** *blast*

qed

thus *?case by force*

next

case (*Free R' α*)

have $RTrans: R \mapsto \alpha \prec R'$ **by** *fact*

with $QSimR$ **obtain** Q' **where** $QTrans: Q \Longrightarrow_{\iota} \alpha \prec Q'$ **and** $Q'RelR': (Q', R') \in Rel'$

by (*blast dest: simE*)

from $PSimQ Sim Eqvt PRelQ QTrans$ **obtain** P' **where** $PTrans: P \Longrightarrow_{\iota} \alpha \prec P'$

```

                                and P'RelQ': (P', Q') ∈ Rel
    by(blast dest: simE2)
    from P'RelQ' Q'RelR' Trans have (P', R') ∈ Rel'' by blast
    with PTrans show ?case by blast
qed

end

theory Weak-Late-Cong
  imports Weak-Late-Bisim Weak-Late-Step-Sim Strong-Late-Bisim
begin

definition congruence :: (pi × pi) set where
  congruence ≡ {(P, Q) | P Q. P ~><weakBisim> Q ∧ Q ~><weakBisim> P}
abbreviation congruenceJudge (infixr ‹≈› 65) where P ≈ Q ≡ (P, Q) ∈ con-
gruence

lemma unfoldE:
  fixes P :: pi
  and Q :: pi
  and s :: (name × name) list

  assumes P ≈ Q

  shows P ~><weakBisim> Q
  and Q ~><weakBisim> P
proof -
  from assms show P ~><weakBisim> Q by(force simp add: congruence-def)
next
  from assms show Q ~><weakBisim> P by(force simp add: congruence-def)
qed

lemma unfoldI:
  fixes P :: pi
  and Q :: pi

  assumes P ~><weakBisim> Q
  and Q ~><weakBisim> P

  shows P ≈ Q
using assms by(force simp add: congruence-def)

lemma eqvt:
  shows eqvt congruence
proof -
  have ∧ P Q (perm::name prm). P ~><weakBisim> Q ⇒ (perm · P) ~><weakBisim>
(perm · Q)
  proof -

```

```

    fix P Q perm
    assume P  $\rightsquigarrow$ <weakBisim> Q
    thus ((perm::name prm) · P)  $\rightsquigarrow$ <weakBisim> (perm · Q)
    apply –
    by(blast intro: Weak-Late-Step-Sim.eqtI Weak-Late-Bisim.eqt)
  qed
  thus ?thesis
  by(simp add: congruence-def eqt-def)
qed

lemma eqtI:
  fixes P :: pi
  and Q :: pi
  and perm :: name prm

  assumes P  $\simeq$  Q

  shows (perm · P)  $\simeq$  (perm · Q)
using assms
by(rule eqtRelI[OF eqt])

lemma strongBisimWeakEq:
  fixes P :: pi
  and Q :: pi

  assumes P  $\sim$  Q

  shows P  $\simeq$  Q
proof –
  have  $\bigwedge P Q. P \rightsquigarrow[bisim] Q \implies P \rightsquigarrow$ <weakBisim> Q
  proof –
    fix P Q
    assume P  $\rightsquigarrow$ [bisim] Q
    hence P  $\rightsquigarrow$ <bisim> Q by(rule strongSimWeakEqSim)
    moreover have bisim  $\subseteq$  weakBisim
    by(auto intro: strongBisimWeakBisim)
    ultimately show P  $\rightsquigarrow$ <weakBisim> Q by(rule Weak-Late-Step-Sim.monotonic)
  qed
  with assms show ?thesis
  by(blast intro: unfoldI dest: Strong-Late-Bisim.bisimE Strong-Late-Bisim.symmetric)
qed

lemma congruenceWeakBisim:
  fixes P :: pi
  and Q :: pi

  assumes P  $\simeq$  Q

  shows P  $\approx$  Q

```

```

proof –
  let ?X = {(P, Q) | P Q. P ≈ Q}
  from assms have (P, Q) ∈ ?X by auto
  thus ?thesis
  proof(coinduct rule: weakBisimCoinduct)
    case(cSim P Q)
    {
      fix P Q
      assume P ≈ Q
      hence P ≈<weakBisim> Q by(simp add: congruence-def)
      hence P ≈<(?X ∪ weakBisim)> Q by(rule-tac Weak-Late-Step-Sim.monotonic)
    }
  auto
  hence P ≈^<(?X ∪ weakBisim)> Q by(rule Weak-Late-Step-Sim.weakSimWeakEqSim)
  }
  with ⟨(P, Q) ∈ ?X⟩ show ?case by auto
  next
  case(cSym P Q)
  thus ?case by(auto simp add: congruence-def)
  qed
qed

```

```

lemma congruenceSubsetWeakBisim:
  shows congruence ⊆ weakBisim
by(auto intro: congruenceWeakBisim)

```

```

lemma reflexive:
  fixes P :: pi

```

```

  shows P ≈ P

```

```

proof –
  from Weak-Late-Bisim.reflexive have  $\bigwedge P. P \approx \langle \text{weakBisim} \rangle P$ 
  by(blast intro: Weak-Late-Step-Sim.reflexive)
  thus ?thesis
  by(force simp add: substClosed-def congruence-def)
qed

```

```

lemma symetric:

```

```

  fixes P :: pi

```

```

  and Q :: pi

```

```

  assumes P ≈ Q

```

```

  shows Q ≈ P

```

```

using assms

```

```

by(force simp add: substClosed-def congruence-def)

```

```

lemma transitive:

```

```

  fixes P :: pi

```

```

  and Q :: pi

```

```

and R :: pi

assumes P ≈ Q
and Q ≈ R

shows P ≈ R
proof –
  have Goal:  $\bigwedge P Q R. \llbracket P \rightsquigarrow\langle \text{weakBisim} \rangle Q; Q \rightsquigarrow\langle \text{weakBisim} \rangle R; P \approx Q \rrbracket \implies$ 
P  $\rightsquigarrow\langle \text{weakBisim} \rangle R$ 
  using Weak-Late-Bisim.eqvt Weak-Late-Bisim.unfoldE Weak-Late-Bisim.transitive
  by(blast intro: Weak-Late-Step-Sim.transitive)
  from assms show ?thesis
  apply(simp add: congruence-def) using assms
  by(blast intro: Goal dest: congruenceWeakBisim symmetric)
qed

end

theory Weak-Late-Bisim-Subst
  imports Weak-Late-Bisim Strong-Late-Bisim-Subst
begin

consts weakBisimSubst :: (pi × pi) set
abbreviation
  weakBisimSubstJudge (infixr <≈s> 65) where P ≈s Q ≡ (P, Q) ∈ (substClosed
weakBisim)

lemma congBisim:
  fixes P :: pi
  and Q :: pi

  assumes P ≈s Q

  shows P ≈ Q
proof –
  from assms substClosedSubset show ?thesis
  by blast
qed

lemma strongBisimWeakBisim:
  fixes P :: pi
  and Q :: pi

  assumes P ≈s Q

  shows P ≈s Q
using assms
by(auto simp add: substClosed-def intro: strongBisimWeakBisim)

```

```

lemma eqvt:
  shows eqvt (substClosed weakBisim)
by(rule eqvtSubstClosed[OF Weak-Late-Bisim.eqvt])

lemma eqvtI:
  fixes P :: pi
  and Q :: pi
  and perm :: name prm

  assumes  $P \approx^s Q$ 

  shows  $(perm \cdot P) \approx^s (perm \cdot Q)$ 
using assms
by(rule-tac eqvtRelI[OF eqvt])

lemma reflexive:
  fixes P :: pi

  shows  $P \approx^s P$ 
by(force simp add: substClosed-def intro: Weak-Late-Bisim.reflexive)

lemma symetric:
  fixes P :: pi
  and Q :: pi

  assumes  $P \approx^s Q$ 

  shows  $Q \approx^s P$ 
using assms
by(force simp add: substClosed-def intro: Weak-Late-Bisim.symmetric)

lemma transitive:
  fixes P :: pi
  and Q :: pi
  and R :: pi

  assumes  $P \approx^s Q$ 
  and  $Q \approx^s R$ 

  shows  $P \approx^s R$ 
using assms
by(force simp add: substClosed-def intro: Weak-Late-Bisim.transitive)

lemma partUnfold:
  fixes P :: pi
  and Q :: pi
  and s :: (name × name) list

```

```

assumes  $P \approx^s Q$ 

shows  $P[\langle s \rangle] \approx^s Q[\langle s \rangle]$ 
using assms
proof(auto simp add: substClosed-def)
  fix  $s'$ 
  assume  $\forall s. P[\langle s \rangle] \approx Q[\langle s \rangle]$ 
  hence  $P[\langle (s@s') \rangle] \approx Q[\langle (s@s') \rangle]$  by blast
  moreover have  $P[\langle (s@s') \rangle] = (P[\langle s \rangle])[\langle s' \rangle]$ 
    by(induct s', auto)
  moreover have  $Q[\langle (s@s') \rangle] = (Q[\langle s \rangle])[\langle s' \rangle]$ 
    by(induct s', auto)

  ultimately show  $(P[\langle s \rangle])[\langle s' \rangle] \approx (Q[\langle s \rangle])[\langle s' \rangle]$ 
    by simp
qed

end

theory Weak-Late-Cong-Subst
  imports Weak-Late-Cong Weak-Late-Bisim-Subst Strong-Late-Bisim-Subst
begin

definition congruenceSubst ::  $pi \Rightarrow pi \Rightarrow bool$  (infix  $\langle \simeq^s \rangle$  65) where
   $P \simeq^s Q \equiv (P, Q) \in (substClosed\ congruence)$ 

lemmas congruenceSubstDef = congruenceSubst-def congruence-def substClosed-def

lemma unfoldE:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $s :: (name \times name) list$ 

  assumes  $P \simeq^s Q$ 

  shows  $P[\langle s \rangle] \rightsquigarrow \langle weakBisim \rangle Q[\langle s \rangle]$ 
  and  $Q[\langle s \rangle] \rightsquigarrow \langle weakBisim \rangle P[\langle s \rangle]$ 
proof –
  from assms show  $P[\langle s \rangle] \rightsquigarrow \langle weakBisim \rangle Q[\langle s \rangle]$  by(force simp add: congruenceSubstDef)
  next
  from assms show  $Q[\langle s \rangle] \rightsquigarrow \langle weakBisim \rangle P[\langle s \rangle]$  by(force simp add: congruenceSubstDef)
qed

lemma unfoldI:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 

```

```

assumes  $\forall s. P[\langle s \rangle] \rightsquigarrow \langle \text{weakBisim} \rangle Q[\langle s \rangle] \wedge Q[\langle s \rangle] \rightsquigarrow \langle \text{weakBisim} \rangle P[\langle s \rangle]$ 

shows  $P \simeq^s Q$ 
proof –
  from assms show ?thesis by(force simp add: congruenceSubstDef)
qed

```

```

lemma weakEqSubset:
  shows substClosed congruence  $\subseteq$  weakBisim
proof(auto simp add: substClosed-def)
  fix P Q
  assume  $\forall s. P[\langle s \rangle] \simeq Q[\langle s \rangle]$ 
  hence  $P[\langle [] \rangle] \simeq Q[\langle [] \rangle]$  by blast
  thus  $P \approx Q$ 
  by(force dest: congruenceWeakBisim intro: Weak-Late-Bisim.unfoldI)
qed

```

```

lemma weakCongWeakEq:
  fixes P :: pi
  and Q :: pi

  assumes  $P \simeq^s Q$ 

  shows  $P \simeq Q$ 
using assms
apply(auto simp add: substClosed-def congruenceSubst-def)
apply(erule-tac x=[] in allE)
by auto

```

```

lemma eqvt:
  shows eqvt (substClosed congruence)
by(rule eqvtSubstClosed[OF Weak-Late-Cong.eqvt])

```

```

lemma eqvtI:
  fixes P :: pi
  and Q :: pi
  and perm :: name prm

  assumes  $P \simeq^s Q$ 

  shows  $(\text{perm} \cdot P) \simeq^s (\text{perm} \cdot Q)$ 
using assms
by(simp add: congruenceSubst-def) (rule eqvtRelI[OF eqvt])

```

```

lemma strongEqWeakCong:
  fixes P :: pi
  and Q :: pi

```

```

assumes  $P \sim^s Q$ 

shows  $P \simeq^s Q$ 
using assms
by(force intro: strongBisimWeakEq simp add: substClosed-def congruenceSubst-def)

lemma congSubstBisimSubst:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 

  assumes  $P \simeq^s Q$ 

  shows  $P \approx^s Q$ 
using assms
by(force simp add: congruenceSubst-def substClosed-def intro: congruenceWeakBisim)

lemma reflexive:
  fixes  $P :: pi$ 

  shows  $P \simeq^s P$ 
proof –
  from Weak-Late-Bisim.reflexive have  $\bigwedge P. P \rightsquigarrow \langle \text{weakBisim} \rangle P$ 
  by(blast intro: Weak-Late-Step-Sim.reflexive)
  thus ?thesis
  by(force simp add: congruenceSubstDef)
qed

lemma symetric:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 

  assumes  $P \simeq^s Q$ 

  shows  $Q \simeq^s P$ 
using assms
by(force simp add: congruenceSubstDef)

lemma transitive:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $R :: pi$ 

  assumes  $P \simeq^s Q$ 
  and  $Q \simeq^s R$ 

  shows  $P \simeq^s R$ 
using assms

```

by(*force simp add: congruenceSubst-def substClosed-def intro: Weak-Late-Cong.transitive*)

lemma *partUnfold*:

fixes $P :: pi$
and $Q :: pi$
and $s :: (name \times name) list$

assumes $P \simeq^s Q$

shows $P[\langle s \rangle] \simeq^s Q[\langle s \rangle]$

using *assms*

proof(*auto simp add: congruenceSubst-def substClosed-def*)

fix s'

assume $\forall s. (P[\langle s \rangle], Q[\langle s \rangle]) \in congruence$

hence $(P[\langle (s@s') \rangle], Q[\langle (s@s') \rangle]) \in congruence$ **by** *blast*

moreover have $P[\langle (s@s') \rangle] = (P[\langle s \rangle])[\langle s' \rangle]$

by(*induct s', auto*)

moreover have $Q[\langle (s@s') \rangle] = (Q[\langle s \rangle])[\langle s' \rangle]$

by(*induct s', auto*)

ultimately show $((P[\langle s \rangle])[\langle s' \rangle], (Q[\langle s \rangle])[\langle s' \rangle]) \in congruence$

by *simp*

qed

end

theory *Strong-Late-Sim-SC*

imports *Strong-Late-Sim*

begin

lemma *nilSim[dest]*:

fixes $a :: name$

and $b :: name$

and $x :: name$

and $P :: pi$

and $Q :: pi$

shows $0 \rightsquigarrow[Rel] \tau.(P) \implies False$

and $0 \rightsquigarrow[Rel] a\langle x \rangle.P \implies False$

and $0 \rightsquigarrow[Rel] a\{b\}.P \implies False$

by(*fastforce simp add: simulation-def intro: Tau Input Output*)**+**

lemma *nilSimRight*:

fixes $P :: pi$

and $Rel :: (pi \times pi) set$

shows $P \rightsquigarrow[Rel] 0$

by(*auto simp add: simulation-def*)

lemma *matchIdLeft*:

fixes $a :: \textit{name}$
and $P :: \textit{pi}$
and $Rel :: (\textit{pi} \times \textit{pi}) \textit{set}$

assumes $Id \subseteq Rel$

shows $[a \frown a]P \rightsquigarrow[Rel] P$

using *assms*

by(*force simp add: simulation-def dest: Match derivativeReflexive*)

lemma *matchIdRight*:

fixes $P :: \textit{pi}$
and $a :: \textit{name}$
and $Rel :: (\textit{pi} \times \textit{pi}) \textit{set}$

assumes *IdRel*: $Id \subseteq Rel$

shows $P \rightsquigarrow[Rel] [a \frown a]P$

using *assms*

by(*fastforce simp add: simulation-def elim: matchCases intro: derivativeReflexive*)

lemma *matchNilLeft*:

fixes $a :: \textit{name}$
and $b :: \textit{name}$
and $P :: \textit{pi}$

assumes $a \neq b$

shows $0 \rightsquigarrow[Rel] [a \frown b]P$

using *assms*

by(*auto simp add: simulation-def*)

lemma *mismatchIdLeft*:

fixes $a :: \textit{name}$
and $b :: \textit{name}$
and $P :: \textit{pi}$
and $Rel :: (\textit{pi} \times \textit{pi}) \textit{set}$

assumes $Id \subseteq Rel$

and $a \neq b$

shows $[a \neq b]P \rightsquigarrow[Rel] P$

```

using assms
by(fastforce simp add: simulation-def intro: Mismatch dest: derivativeReflexive)

lemma mismatchIdRight:
  fixes  $P :: pi$ 
  and  $a :: name$ 
  and  $b :: name$ 
  and  $Rel :: (pi \times pi) set$ 

  assumes  $IdRel: Id \subseteq Rel$ 
  and  $aineqb: a \neq b$ 

  shows  $P \rightsquigarrow[Rel] [a \neq b] P$ 
using assms
by(fastforce simp add: simulation-def elim: mismatchCases intro: derivativeReflexive)

lemma mismatchNilLeft:
  fixes  $a :: name$ 
  and  $P :: pi$ 

  shows  $0 \rightsquigarrow[Rel] [a \neq a] P$ 
by(auto simp add: simulation-def)

lemma sumSym:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $Rel :: (pi \times pi) set$ 

  assumes  $Id: Id \subseteq Rel$ 

  shows  $P \oplus Q \rightsquigarrow[Rel] Q \oplus P$ 
using assms
by(fastforce simp add: simulation-def elim: sumCases intro: Sum1 Sum2 derivativeReflexive)

lemma sumIdempLeft:
  fixes  $P :: pi$ 
  and  $Rel :: (pi \times pi) set$ 

  assumes  $Id \subseteq Rel$ 

  shows  $P \rightsquigarrow[Rel] P \oplus P$ 
using assms
by(fastforce simp add: simulation-def elim: sumCases intro: derivativeReflexive)

lemma sumIdempRight:

```

```

fixes  $P :: pi$ 
and  $Rel :: (pi \times pi) set$ 

assumes  $Id: Id \subseteq Rel$ 

shows  $P \oplus P \rightsquigarrow[Rel] P$ 
using assms
by(fastforce simp add: simulation-def intro: Sum1 derivativeReflexive)

lemma sumAssocLeft:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $R :: pi$ 
  and  $Rel :: (pi \times pi) set$ 

  assumes  $Id: Id \subseteq Rel$ 

  shows  $(P \oplus Q) \oplus R \rightsquigarrow[Rel] P \oplus (Q \oplus R)$ 
using assms
by(fastforce simp add: simulation-def elim: sumCases intro: Sum1 Sum2 derivativeReflexive)

lemma sumAssocRight:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $R :: pi$ 
  and  $Rel :: (pi \times pi) set$ 

  assumes  $Id: Id \subseteq Rel$ 

  shows  $P \oplus (Q \oplus R) \rightsquigarrow[Rel] (P \oplus Q) \oplus R$ 
using assms
by(fastforce simp add: simulation-def elim: sumCases intro: Sum1 Sum2 derivativeReflexive)

lemma sumZeroLeft:
  fixes  $P :: pi$ 
  and  $Rel :: (pi \times pi) set$ 

  assumes  $Id: Id \subseteq Rel$ 

  shows  $P \oplus \mathbf{0} \rightsquigarrow[Rel] P$ 
using assms
by(fastforce simp add: simulation-def intro: Sum1 derivativeReflexive)

lemma sumZeroRight:
  fixes  $P :: pi$ 
  and  $Rel :: (pi \times pi) set$ 

```

```

assumes Id:  $Id \subseteq Rel$ 

shows  $P \rightsquigarrow[Rel] P \oplus \mathbf{0}$ 
using assms
by(fastforce simp add: simulation-def elim: sumCases intro: derivativeReflexive)

lemma sumResLeft:
  fixes x :: name
  and P :: pi
  and Q :: pi

  assumes Id:  $Id \subseteq Rel$ 
  and Eqvt: eqvt Rel

  shows  $(\langle \nu x \rangle P) \oplus (\langle \nu x \rangle Q) \rightsquigarrow[Rel] \langle \nu x \rangle (P \oplus Q)$ 
using Eqvt
proof(induct rule: simCasesCont[where C=(x, P, Q)])
  case(Bound a y PQ)
  from  $\langle y \# (x, P, Q) \rangle$  have  $y \neq x$  and  $y \# P$  and  $y \# Q$  by(simp add: fresh-prod)+
  hence  $y \# P \oplus Q$  by simp
  with  $\langle \langle \nu x \rangle (P \oplus Q) \mapsto a \langle y \rangle \prec PQ \rangle \langle y \neq x \rangle$  show ?case
  proof(induct rule: resCasesB)
    case(cOpen a PQ)
    from  $\langle P \oplus Q \mapsto a[x] \prec PQ \rangle \langle y \# P \rangle \langle y \# Q \rangle$  have  $y \# PQ$  by(force dest: freshFreeDerivative)
    from  $\langle P \oplus Q \mapsto a[x] \prec PQ \rangle$  show ?case
    proof(induct rule: sumCases)
      case cSum1
      from  $\langle P \mapsto a[x] \prec PQ \rangle \langle a \neq x \rangle$  have  $\langle \nu x \rangle P \mapsto a \langle \nu x \rangle \prec PQ$  by(rule Open)
      hence  $(\langle \nu x \rangle P) \oplus (\langle \nu x \rangle Q) \mapsto a \langle \nu x \rangle \prec PQ$  by(rule Sum1)
      with  $\langle y \# PQ \rangle$  have  $(\langle \nu x \rangle P) \oplus (\langle \nu x \rangle Q) \mapsto a \langle \nu y \rangle \prec ((y, x) \cdot PQ)$ 
      by(simp add: alphaBoundResidual)
      moreover from Id have derivative  $((y, x) \cdot PQ) ((y, x) \cdot PQ)$  (BoundOutputS)
    a) y Rel
      by(force simp add: derivative-def)
      ultimately show ?case by blast
    next
    case cSum2
    from  $\langle Q \mapsto a[x] \prec PQ \rangle \langle a \neq x \rangle$  have  $\langle \nu x \rangle Q \mapsto a \langle \nu x \rangle \prec PQ$  by(rule Open)
    hence  $(\langle \nu x \rangle P) \oplus (\langle \nu x \rangle Q) \mapsto a \langle \nu x \rangle \prec PQ$  by(rule Sum2)
    with  $\langle y \# PQ \rangle$  have  $(\langle \nu x \rangle P) \oplus (\langle \nu x \rangle Q) \mapsto a \langle \nu y \rangle \prec ((y, x) \cdot PQ)$ 
    by(simp add: alphaBoundResidual)
    moreover from Id have derivative  $((y, x) \cdot PQ) ((y, x) \cdot PQ)$  (BoundOutputS)
  a) y Rel
    by(force simp add: derivative-def)
    ultimately show ?case by blast
  qed

```

```

next
  case(cRes PQ)
  from  $\langle P \oplus Q \mapsto a \langle y \rangle \prec PQ \rangle$  show ?case
  proof(induct rule: sumCases)
    case cSum1
    from  $\langle P \mapsto a \langle y \rangle \prec PQ \rangle \langle x \# a \rangle \langle y \neq x \rangle$  have  $\langle \nu x \rangle P \mapsto a \langle y \rangle \prec \langle \nu x \rangle PQ$ 
  by(rule-tac ResB) auto
    hence  $\langle \langle \nu x \rangle P \rangle \oplus \langle \langle \nu x \rangle Q \rangle \mapsto a \langle y \rangle \prec \langle \nu x \rangle PQ$  by(rule Sum1)
    moreover from Id have derivative  $\langle \langle \nu x \rangle PQ \rangle \langle \langle \nu x \rangle PQ \rangle a y Rel$ 
      by(cases a) (auto simp add: derivative-def)
    ultimately show ?case by blast
  next
  case cSum2
  from  $\langle Q \mapsto a \langle y \rangle \prec PQ \rangle \langle x \# a \rangle \langle y \neq x \rangle$  have  $\langle \nu x \rangle Q \mapsto a \langle y \rangle \prec \langle \nu x \rangle PQ$ 
  by(rule-tac ResB) auto
    hence  $\langle \langle \nu x \rangle P \rangle \oplus \langle \langle \nu x \rangle Q \rangle \mapsto a \langle y \rangle \prec \langle \nu x \rangle PQ$  by(rule Sum2)
    moreover from Id have derivative  $\langle \langle \nu x \rangle PQ \rangle \langle \langle \nu x \rangle PQ \rangle a y Rel$ 
      by(cases a) (auto simp add: derivative-def)
    ultimately show ?case by blast
  qed
qed
next
  case(Free  $\alpha PQ$ )
  from  $\langle \langle \nu x \rangle (P \oplus Q) \mapsto \alpha \prec PQ \rangle$  show ?case
  proof(induct rule: resCasesF)
    case(cRes PQ)
    from  $\langle P \oplus Q \mapsto \alpha \prec PQ \rangle$  show ?case
    proof(induct rule: sumCases)
      case cSum1
      from  $\langle P \mapsto \alpha \prec PQ \rangle \langle x \# \alpha \rangle$  have  $\langle \nu x \rangle P \mapsto \alpha \prec \langle \nu x \rangle PQ$  by(rule ResF)
      hence  $\langle \langle \nu x \rangle P \rangle \oplus \langle \langle \nu x \rangle Q \rangle \mapsto \alpha \prec \langle \nu x \rangle PQ$  by(rule Sum1)
      with Id show ?case by blast
    next
    case cSum2
    from  $\langle Q \mapsto \alpha \prec PQ \rangle \langle x \# \alpha \rangle$  have  $\langle \nu x \rangle Q \mapsto \alpha \prec \langle \nu x \rangle PQ$  by(rule ResF)
    hence  $\langle \langle \nu x \rangle P \rangle \oplus \langle \langle \nu x \rangle Q \rangle \mapsto \alpha \prec \langle \nu x \rangle PQ$  by(rule Sum2)
    with Id show ?case by blast
  qed
qed
qed

lemma sumResRight:
  fixes x :: name
  and P :: pi
  and Q :: pi

  assumes Id:  $Id \subseteq Rel$ 
  and Eqvt: eqvt Rel

```

```

shows  $\langle \nu x \rangle (P \oplus Q) \rightsquigarrow_{[Rel]} (\langle \nu x \rangle P) \oplus (\langle \nu x \rangle Q)$ 
using  $\langle eqvt Rel \rangle$ 
proof(induct rule: simCasesCont[where  $C=(x, P, Q)$ ])
  case(Bound a y PQ)
  from  $\langle y \# (x, P, Q) \rangle$  have  $y \neq x$  and  $y \# P$  and  $y \# Q$  by(simp add: fresh-prod) +
  from  $\langle (\langle \nu x \rangle P) \oplus (\langle \nu x \rangle Q) \mapsto a \langle y \rangle \prec PQ \rangle$  show ?case
  proof(induct rule: sumCases)
    case cSum1
    from  $\langle \langle \nu x \rangle P \mapsto a \langle y \rangle \prec PQ \rangle$  show ?case using  $\langle y \neq x \rangle \langle y \# P \rangle$ 
    proof(induct rule: resCasesB)
      case(cOpen a P')
      from  $\langle P \mapsto a[x] \prec P' \rangle \langle y \# P \rangle$  have  $y \# P'$  by(rule freshFreeDerivative)

      from  $\langle P \mapsto a[x] \prec P' \rangle$  have  $P \oplus Q \mapsto a[x] \prec P'$  by(rule Sum1)
      hence  $\langle \nu x \rangle (P \oplus Q) \mapsto a \langle \nu x \rangle \prec P'$  using  $\langle a \neq x \rangle$  by(rule Open)
      with  $\langle y \# P' \rangle$  have  $\langle \nu x \rangle (P \oplus Q) \mapsto a \langle \nu y \rangle \prec [(y, x)] \cdot P'$  by(simp add: alphaBoundResidual)
      moreover from Id have derivative  $([(y, x)] \cdot P') ((y, x)] \cdot P')$  (BoundOutputS a) y Rel
      by(force simp add: derivative-def)
      ultimately show ?case by blast
    next
    case(cRes P')
    from  $\langle P \mapsto a \langle y \rangle \prec P' \rangle$  have  $P \oplus Q \mapsto a \langle y \rangle \prec P'$  by(rule Sum1)
    hence  $\langle \nu x \rangle (P \oplus Q) \mapsto a \langle y \rangle \prec \langle \nu x \rangle P'$  using  $\langle x \# a \rangle \langle y \neq x \rangle$  by(rule-tac ResB) auto
    moreover from Id have derivative  $(\langle \nu x \rangle P') (\langle \nu x \rangle P')$  a y Rel
    by(cases a) (auto simp add: derivative-def)
    ultimately show ?case by blast
  qed
  next
  case cSum2
  from  $\langle \langle \nu x \rangle Q \mapsto a \langle y \rangle \prec PQ \rangle$  show ?case using  $\langle y \neq x \rangle \langle y \# Q \rangle$ 
  proof(induct rule: resCasesB)
    case(cOpen a Q')
    from  $\langle Q \mapsto a[x] \prec Q' \rangle \langle y \# Q \rangle$  have  $y \# Q'$  by(rule freshFreeDerivative)

    from  $\langle Q \mapsto a[x] \prec Q' \rangle$  have  $P \oplus Q \mapsto a[x] \prec Q'$  by(rule Sum2)
    hence  $\langle \nu x \rangle (P \oplus Q) \mapsto a \langle \nu x \rangle \prec Q'$  using  $\langle a \neq x \rangle$  by(rule Open)
    with  $\langle y \# Q' \rangle$  have  $\langle \nu x \rangle (P \oplus Q) \mapsto a \langle \nu y \rangle \prec [(y, x)] \cdot Q'$  by(simp add: alphaBoundResidual)
    moreover from Id have derivative  $([(y, x)] \cdot Q') ((y, x)] \cdot Q')$  (BoundOutputS a) y Rel
    by(force simp add: derivative-def)
    ultimately show ?case by blast
  next
  case(cRes Q')
  from  $\langle Q \mapsto a \langle y \rangle \prec Q' \rangle$  have  $P \oplus Q \mapsto a \langle y \rangle \prec Q'$  by(rule Sum2)
  hence  $\langle \nu x \rangle (P \oplus Q) \mapsto a \langle y \rangle \prec \langle \nu x \rangle Q'$  using  $\langle x \# a \rangle \langle y \neq x \rangle$  by(rule-tac)

```

```

ResB) auto
  moreover from Id have derivative ( $\langle \nu x \rangle Q'$ ) ( $\langle \nu x \rangle Q'$ ) a y Rel
    by(cases a) (auto simp add: derivative-def)
  ultimately show ?case by blast
qed
qed
next
case(Free  $\alpha$  PQ)
from ( $\langle \nu x \rangle P \oplus \langle \nu x \rangle Q \mapsto \alpha \prec PQ$ ) show ?case
proof(induct rule: sumCases)
  case cSum1
  from ( $\langle \nu x \rangle P \mapsto \alpha \prec PQ$ ) show ?case
  proof(induct rule: resCasesF)
    case(cRes P')
    from ( $P \mapsto \alpha \prec P'$ ) have  $P \oplus Q \mapsto \alpha \prec P'$  by(rule Sum1)
    hence  $\langle \nu x \rangle (P \oplus Q) \mapsto \alpha \prec \langle \nu x \rangle P'$  using  $\langle x \# \alpha \rangle$  by(rule ResF)
    with Id show ?case by blast
  qed
next
case cSum2
from ( $\langle \nu x \rangle Q \mapsto \alpha \prec PQ$ ) show ?case
proof(induct rule: resCasesF)
  case(cRes Q')
  from ( $Q \mapsto \alpha \prec Q'$ ) have  $P \oplus Q \mapsto \alpha \prec Q'$  by(rule Sum2)
  hence  $\langle \nu x \rangle (P \oplus Q) \mapsto \alpha \prec \langle \nu x \rangle Q'$  using  $\langle x \# \alpha \rangle$  by(rule ResF)
  with Id show ?case by blast
qed
qed
qed

```

lemma parZeroLeft:

```

fixes P :: pi
and Rel :: (pi  $\times$  pi) set

assumes ParZero:  $\bigwedge Q. (Q \parallel \mathbf{0}, Q) \in Rel$ 

shows  $P \parallel \mathbf{0} \rightsquigarrow [Rel] P$ 
proof -
{
  fix P Q a x
  from ParZero have derivative (P  $\parallel \mathbf{0}$ ) P a x Rel
    by(case-tac a) (auto simp add: derivative-def)
}
thus ?thesis using assms
  by(fastforce simp add: simulation-def intro: Par1B Par1F)
qed

```

```

lemma parZeroRight:
  fixes P :: pi
  and Rel :: (pi × pi) set

  assumes ParZero:  $\bigwedge Q. (Q, Q \parallel \mathbf{0}) \in Rel$ 

  shows  $P \rightsquigarrow[Rel] P \parallel \mathbf{0}$ 
proof -
  {
    fix P Q a x
    from ParZero have derivative P (P  $\parallel$   $\mathbf{0}$ ) a x Rel
    by(case-tac a) (auto simp add: derivative-def)
  }
  thus ?thesis using assms
    by(fastforce simp add: simulation-def elim: parCasesF parCasesB)+
qed

lemma parSym:
  fixes P :: pi
  and Q :: pi
  and Rel :: (pi × pi) set

  assumes Sym:  $\bigwedge R S. (R \parallel S, S \parallel R) \in Rel$ 
  and Res:  $\bigwedge R S x. (R, S) \in Rel \implies (\langle \nu x \rangle R, \langle \nu x \rangle S) \in Rel$ 

  shows  $P \parallel Q \rightsquigarrow[Rel] Q \parallel P$ 
proof(induct rule: simCases)
  case(Bound a x QP)
  from  $\langle x \# (P \parallel Q) \rangle$  have  $x \# Q$  and  $x \# P$  by simp+
  with  $\langle Q \parallel P \mapsto a \langle x \rangle \prec QP \rangle$  show ?case
  proof(induct rule: parCasesB)
    case(cPar1 Q')
    from  $\langle Q \mapsto a \langle x \rangle \prec Q' \rangle$  have  $P \parallel Q \mapsto a \langle x \rangle \prec P \parallel Q'$  using  $\langle x \# P \rangle$  by(rule
    Par2B)
    moreover have derivative (P  $\parallel$  Q') (Q'  $\parallel$  P) a x Rel
    by(cases a, auto simp add: derivative-def intro: Sym)
    ultimately show ?case by blast
  next
    case(cPar2 P')
    from  $\langle P \mapsto a \langle x \rangle \prec P' \rangle$  have  $P \parallel Q \mapsto a \langle x \rangle \prec P' \parallel Q$  using  $\langle x \# Q \rangle$  by(rule
    Par1B)
    moreover have derivative (P'  $\parallel$  Q) (Q  $\parallel$  P') a x Rel
    by(cases a, auto simp add: derivative-def intro: Sym)
    ultimately show ?case by blast
  qed
next
  case(Free  $\alpha$  QP)
  from  $\langle Q \parallel P \mapsto \alpha \prec QP \rangle$  show ?case
  proof(induct rule: parCasesF[where C=()])

```

```

    case(cPar1 Q')
    from ⟨Q ⟶ α < Q'⟩ have P ∥ Q ⟶ α < P ∥ Q' by(rule Par2F)
    moreover have (P ∥ Q', Q' ∥ P) ∈ Rel by(rule Sym)
    ultimately show ?case by blast
  next
    case(cPar2 P')
    from ⟨P ⟶ α < P'⟩ have P ∥ Q ⟶ α < P' ∥ Q by(rule Par1F)
    moreover have (P' ∥ Q, Q ∥ P') ∈ Rel by(rule Sym)
    ultimately show ?case by blast
  next
    case(cComm1 Q' P' a b x)
    from ⟨P ⟶ a[b] < P'⟩ ⟨Q ⟶ a<x> < Q'⟩
    have P ∥ Q ⟶ τ < P' ∥ (Q'[x::=b]) by(rule Comm2)
    moreover have (P' ∥ Q'[x::=b], Q'[x::=b] ∥ P') ∈ Rel by(rule Sym)
    ultimately show ?case by blast
  next
    case(cComm2 Q' P' a b x)
    from ⟨P ⟶ a<x> < P'⟩ ⟨Q ⟶ a[b] < Q'⟩
    have P ∥ Q ⟶ τ < (P'[x::=b]) ∥ Q' by(rule Comm1)
    moreover have (P'[x::=b] ∥ Q', Q' ∥ P'[x::=b]) ∈ Rel by(rule Sym)
    ultimately show ?case by blast
  next
    case(cClose1 Q' P' a x y)
    from ⟨P ⟶ a<νy> < P'⟩ ⟨Q ⟶ a<x> < Q'⟩ ⟨y # Q⟩
    have P ∥ Q ⟶ τ < <νy>(P' ∥ (Q'[x::=y])) by(rule Close2)
    moreover have (<νy>(P' ∥ Q'[x::=y]), <νy>(Q'[x::=y] ∥ P')) ∈ Rel by(metis
Res Sym)
    ultimately show ?case by blast
  next
    case(cClose2 Q' P' a x y)
    from ⟨P ⟶ a<x> < P'⟩ ⟨Q ⟶ a<νy> < Q'⟩ ⟨y # P⟩
    have P ∥ Q ⟶ τ < <νy>((P'[x::=y]) ∥ Q') by(rule Close1)
    moreover have (<νy>(P'[x::=y] ∥ Q'), <νy>(Q' ∥ P'[x::=y])) ∈ Rel by(metis
Res Sym)
    ultimately show ?case by blast
qed
qed

```

lemma *parAssocLeft*:

```

  fixes P   :: pi
  and Q     :: pi
  and R     :: pi
  and Rel   :: (pi × pi) set

```

```

  assumes Ass:   ∧ S T U. ((S ∥ T) ∥ U, S ∥ (T ∥ U)) ∈ Rel
  and Res:       ∧ S T x. (S, T) ∈ Rel ⇒ (<νx>S, <νx>T) ∈ Rel
  and FreshExt: ∧ S T U x. x # S ⇒ (<νx>((S ∥ T) ∥ U), S ∥ <νx>(T ∥
U)) ∈ Rel
  and FreshExt': ∧ S T U x. x # U ⇒ ((<νx>(S ∥ T)) ∥ U, <νx>(S ∥ (T ∥

```

$U))) \in Rel$

shows $(P \parallel Q) \parallel R \rightsquigarrow[Rel] P \parallel (Q \parallel R)$
proof(*induct rule: simCases*)
case(*Bound a x PQR*)
from $\langle x \# (P \parallel Q) \parallel R \rangle$ **have** $x \# P$ **and** $x \# Q$ **and** $x \# R$ **by** *simp+*
hence $x \# (Q \parallel R)$ **by** *simp*
with $\langle P \parallel (Q \parallel R) \mapsto a\langle x \rangle \prec PQR \rangle$ $\langle x \# P \rangle$ **show** *?case*
proof(*induct rule: parCasesB*)
case(*cPar1 P'*)
from $\langle P \mapsto a\langle x \rangle \prec P' \rangle$ **have** $P \parallel Q \mapsto a\langle x \rangle \prec P' \parallel Q$ **using** $\langle x \# Q \rangle$
by(*rule Par1B*)
hence $(P \parallel Q) \parallel R \mapsto a\langle x \rangle \prec (P' \parallel Q) \parallel R$ **using** $\langle x \# R \rangle$ **by**(*rule Par1B*)
moreover **have** *derivative* $((P' \parallel Q) \parallel R) (P' \parallel (Q \parallel R))$ $a x Rel$
by(*cases a, auto intro: Ass simp add: derivative-def*)
ultimately show *?case* **by** *blast*
next
case(*cPar2 QR*)
from $\langle Q \parallel R \mapsto a\langle x \rangle \prec QR \rangle$ $\langle x \# Q \rangle$ $\langle x \# R \rangle$ **show** *?case*
proof(*induct rule: parCasesB*)
case(*cPar1 Q'*)
from $\langle Q \mapsto a\langle x \rangle \prec Q' \rangle$ **have** $P \parallel Q \mapsto a\langle x \rangle \prec P \parallel Q'$ **using** $\langle x \# P \rangle$
by(*rule Par2B*)
hence $(P \parallel Q) \parallel R \mapsto a\langle x \rangle \prec (P \parallel Q') \parallel R$ **using** $\langle x \# R \rangle$ **by**(*rule Par1B*)
moreover **have** *derivative* $((P \parallel Q') \parallel R) (P \parallel (Q' \parallel R))$ $a x Rel$
by(*cases a, auto intro: Ass simp add: derivative-def*)
ultimately show *?case* **by** *blast*
next
case(*cPar2 R'*)
from $\langle R \mapsto a\langle x \rangle \prec R' \rangle$ **have** $(P \parallel Q) \parallel R \mapsto a\langle x \rangle \prec (P \parallel Q) \parallel R'$ **using**
 $\langle x \# P \rangle$ $\langle x \# Q \rangle$
by(*rule-tac Par2B*) *auto*
moreover **have** *derivative* $((P \parallel Q) \parallel R') (P \parallel (Q \parallel R'))$ $a x Rel$
by(*cases a, auto intro: Ass simp add: derivative-def*)
ultimately show *?case* **by** *blast*
qed
qed
next
case(*Free α PQR*)
from $\langle P \parallel (Q \parallel R) \mapsto \alpha \prec PQR \rangle$ **show** *?case*
proof(*induct rule: parCasesF[where C=Q]*)
case(*cPar1 P'*)
from $\langle P \mapsto \alpha \prec P' \rangle$ **have** $P \parallel Q \mapsto \alpha \prec P' \parallel Q$ **by**(*rule Par1F*)
hence $(P \parallel Q) \parallel R \mapsto \alpha \prec (P' \parallel Q) \parallel R$ **by**(*rule Par1F*)
moreover **from** *Ass* **have** $((P' \parallel Q) \parallel R, P' \parallel (Q \parallel R)) \in Rel$ **by** *blast*
ultimately show *?case* **by** *blast*
next
case(*cPar2 QR*)
from $\langle Q \parallel R \mapsto \alpha \prec QR \rangle$ **show** *?case*

```

proof(induct rule: parCasesF[where  $C=P$ ])
  case(cPar1  $Q'$ )
    from  $\langle Q \mapsto \alpha \prec Q' \rangle$  have  $(P \parallel Q) \mapsto \alpha \prec P \parallel Q'$  by(rule Par2F)
    hence  $(P \parallel Q) \parallel R \mapsto \alpha \prec (P \parallel Q') \parallel R$  by(rule Par1F)
    moreover from Ass have  $((P \parallel Q') \parallel R, P \parallel (Q' \parallel R)) \in Rel$  by blast
    ultimately show ?case by blast
  next
    case(cPar2  $R'$ )
      from  $\langle R \mapsto \alpha \prec R' \rangle$  have  $(P \parallel Q) \parallel R \mapsto \alpha \prec (P \parallel Q) \parallel R'$  by(rule
Par2F)
      moreover from Ass have  $((P \parallel Q) \parallel R', P \parallel (Q \parallel R')) \in Rel$  by blast
      ultimately show ?case by blast
    next
      case(cComm1  $Q' R' a b x$ )
        from  $\langle Q \mapsto a \langle x \rangle \prec Q' \rangle$   $\langle x \# P \rangle$  have  $P \parallel Q \mapsto a \langle x \rangle \prec P \parallel Q'$  by(rule
Par2B)
        hence  $(P \parallel Q) \parallel R \mapsto \tau \prec (P \parallel Q')[x::=b] \parallel R'$  using  $\langle R \mapsto a[b] \prec R' \rangle$ 
by(rule Comm1)
        with  $\langle x \# P \rangle$  have  $(P \parallel Q) \parallel R \mapsto \tau \prec (P \parallel (Q'[x::=b])) \parallel R'$  by(simp add:
forget)
        moreover from Ass have  $((P \parallel (Q'[x::=b])) \parallel R', P \parallel (Q'[x::=b] \parallel R')) \in$ 
Rel by blast
        ultimately show ?case by blast
      next
        case(cComm2  $Q' R' a b x$ )
          from  $\langle Q \mapsto a[b] \prec Q' \rangle$  have  $P \parallel Q \mapsto a[b] \prec P \parallel Q'$  by(rule Par2F)
          with  $\langle x \# P \rangle$   $\langle x \# Q \rangle$   $\langle R \mapsto a \langle x \rangle \prec R' \rangle$  have  $(P \parallel Q) \parallel R \mapsto \tau \prec (P \parallel$ 
 $Q') \parallel R'[x::=b]$ 
          by(force intro: Comm2)
          moreover from Ass have  $((P \parallel Q') \parallel R'[x::=b], P \parallel (Q' \parallel R'[x::=b])) \in Rel$ 
by blast
          ultimately show ?case by blast
        next
          case(cClose1  $Q' R' a x y$ )
            from  $\langle Q \mapsto a \langle x \rangle \prec Q' \rangle$   $\langle x \# P \rangle$  have  $P \parallel Q \mapsto a \langle x \rangle \prec P \parallel Q'$  by(rule
Par2B)
            with  $\langle y \# P \rangle$   $\langle y \# Q \rangle$   $\langle x \# P \rangle$   $\langle R \mapsto a \langle \nu y \rangle \prec R' \rangle$  have  $(P \parallel Q) \parallel R \mapsto \tau$ 
 $\prec \langle \nu y \rangle ((P \parallel Q')[x::=y] \parallel R')$ 
            by(rule-tac Close1) auto
            with  $\langle x \# P \rangle$  have  $(P \parallel Q) \parallel R \mapsto \tau \prec \langle \nu y \rangle ((P \parallel (Q'[x::=y])) \parallel R')$ 
by(simp add: forget)
            moreover from  $\langle y \# P \rangle$  have  $\langle \nu y \rangle ((P \parallel Q'[x::=y]) \parallel R'), P \parallel \langle \nu y \rangle (Q'[x::=y]$ 
 $\parallel R') \in Rel$ 
            by(rule FreshExt)
            ultimately show ?case by blast
          next
            case(cClose2  $Q' R' a x y$ )
              from  $\langle Q \mapsto a \langle \nu y \rangle \prec Q' \rangle$   $\langle y \# P \rangle$  have  $P \parallel Q \mapsto a \langle \nu y \rangle \prec P \parallel Q'$  by(rule
Par2B)

```

hence *Act*: $(P \parallel Q) \parallel R \mapsto \tau \prec \langle \nu y \rangle ((P \parallel Q') \parallel R'[x::=y])$ **using** $\langle R \mapsto a \langle x \rangle \prec R' \rangle \langle y \# R \rangle$ **by** (*rule Close2*)
moreover from $\langle y \# P \rangle$ **have** $(\langle \nu y \rangle ((P \parallel Q') \parallel R'[x::=y]), P \parallel \langle \nu y \rangle (Q' \parallel R'[x::=y])) \in Rel$
by (*rule FreshExt*)
ultimately show *?case* **by** *blast*
qed
next
case (*cComm1 P' QR a b x*)
from $\langle Q \parallel R \mapsto a[b] \prec QR \rangle$ **show** *?case*
proof (*induct rule: parCasesF[where C=()]*)
case (*cPar1 Q'*)
from $\langle P \mapsto a \langle x \rangle \prec P' \rangle \langle Q \mapsto a[b] \prec Q' \rangle$ **have** $P \parallel Q \mapsto \tau \prec P'[x::=b] \parallel Q'$ **by** (*rule Comm1*)
hence $(P \parallel Q) \parallel R \mapsto \tau \prec (P'[x::=b] \parallel Q') \parallel R$ **by** (*rule Par1F*)
moreover from *Ass* **have** $((P'[x::=b] \parallel Q') \parallel R, P'[x::=b] \parallel (Q' \parallel R)) \in Rel$
by *blast*
ultimately show *?case* **by** *blast*
next
case (*cPar2 R'*)
from $\langle P \mapsto a \langle x \rangle \prec P' \rangle \langle x \# Q \rangle$ **have** $P \parallel Q \mapsto a \langle x \rangle \prec P' \parallel Q$ **by** (*rule Par1B*)
hence $(P \parallel Q) \parallel R \mapsto \tau \prec (P' \parallel Q)[x::=b] \parallel R'$ **using** $\langle R \mapsto a[b] \prec R' \rangle$ **by** (*rule Comm1*)
with $\langle x \# Q \rangle$ **have** $(P \parallel Q) \parallel R \mapsto \tau \prec (P'[x::=b] \parallel Q) \parallel R'$ **by** (*simp add: forget*)
moreover from *Ass* **have** $((P'[x::=b] \parallel Q) \parallel R', P'[x::=b] \parallel (Q \parallel R')) \in Rel$
by *blast*
ultimately show *?case* **by** *blast*
next
case (*cComm1 Q' R'*)
from $\langle a[b] = \tau \rangle$ **have** *False* **by** *simp* **thus** *?case* **by** *simp*
next
case (*cComm2 Q' R'*)
from $\langle a[b] = \tau \rangle$ **have** *False* **by** *simp* **thus** *?case* **by** *simp*
next
case (*cClose1 Q' R'*)
from $\langle a[b] = \tau \rangle$ **have** *False* **by** *simp* **thus** *?case* **by** *simp*
next
case (*cClose2 Q' R'*)
from $\langle a[b] = \tau \rangle$ **have** *False* **by** *simp* **thus** *?case* **by** *simp*
qed
next
case (*cComm2 P' QR a b x*)
from $\langle x \# Q \parallel R \rangle$ **have** $x \# Q$ **and** $x \# R$ **by** *simp+*
with $\langle Q \parallel R \mapsto a \langle x \rangle \prec QR \rangle$ **show** *?case*
proof (*induct rule: parCasesB*)
case (*cPar1 Q'*)
from $\langle P \mapsto a[b] \prec P' \rangle \langle Q \mapsto a \langle x \rangle \prec Q' \rangle$ **have** $P \parallel Q \mapsto \tau \prec P' \parallel Q'$

$(Q'[x::=b])$ **by**(*rule Comm2*)
hence $(P \parallel Q) \parallel R \mapsto \tau \prec (P' \parallel Q'[x::=b]) \parallel R$ **by**(*rule Par1F*)
moreover from *Ass* **have** $((P' \parallel Q'[x::=b]) \parallel R, P' \parallel Q'[x::=b] \parallel R) \in Rel$
by *blast*
with $\langle x \# R \rangle$ **have** $((P' \parallel Q'[x::=b]) \parallel R, P' \parallel (Q' \parallel R)[x::=b]) \in Rel$ **by**(*force simp add: forget*)
ultimately show *?case* **by** *blast*
next
case(*cPar2 R'*)
from $\langle P \mapsto a[b] \prec P' \rangle$ **have** $P \parallel Q \mapsto a[b] \prec P' \parallel Q$ **by**(*rule Par1F*)
hence $(P \parallel Q) \parallel R \mapsto \tau \prec (P' \parallel Q) \parallel (R'[x::=b])$ **using** $\langle R \mapsto a \langle x \rangle \prec R' \rangle$ **by** (*rule Comm2*)
moreover from *Ass* **have** $((P' \parallel Q) \parallel R'[x::=b], P' \parallel Q \parallel (R'[x::=b])) \in Rel$
by *blast*
hence $((P' \parallel Q) \parallel R'[x::=b], P' \parallel (Q \parallel R')[x::=b]) \in Rel$ **using** $\langle x \# Q \rangle$
by(*force simp add: forget*)
ultimately show *?case* **by** *blast*
qed
next
case(*cClose1 P' QR a x y*)
from $\langle x \# Q \parallel R \rangle$ **have** $x \# Q$ **by** *simp*
from $\langle y \# Q \parallel R \rangle$ **have** $y \# Q$ **and** $y \# R$ **by** *simp+*
from $\langle Q \parallel R \mapsto a \langle \nu y \rangle \prec QR \rangle$ $\langle y \# Q \rangle$ $\langle y \# R \rangle$ **show** *?case*
proof(*induct rule: parCasesB*)
case(*cPar1 Q'*)
from $\langle P \mapsto a \langle x \rangle \prec P' \rangle$ $\langle Q \mapsto a \langle \nu y \rangle \prec Q' \rangle$ $\langle y \# P \rangle$ **have** $P \parallel Q \mapsto \tau \prec \langle \nu y \rangle (P'[x::=y] \parallel Q')$ **by**(*rule Close1*)
hence $(P \parallel Q) \parallel R \mapsto \tau \prec \langle \nu y \rangle (P'[x::=y] \parallel Q') \parallel R$ **by**(*rule Par1F*)
moreover from $\langle y \# R \rangle$ **have** $(\langle \nu y \rangle (P'[x::=y] \parallel Q')) \parallel R, \langle \nu y \rangle (P'[x::=y] \parallel Q') \parallel R) \in Rel$
by(*rule FreshExt'*)
ultimately show *?case* **by** *blast*
next
case(*cPar2 R'*)
from $\langle P \mapsto a \langle x \rangle \prec P' \rangle$ $\langle x \# Q \rangle$ **have** $P \parallel Q \mapsto a \langle x \rangle \prec P' \parallel Q$ **by**(*rule Par1B*)
with $\langle R \mapsto a \langle \nu y \rangle \prec R' \rangle$ $\langle y \# P \rangle$ $\langle y \# Q \rangle$ **have** $(P \parallel Q) \parallel R \mapsto \tau \prec \langle \nu y \rangle ((P' \parallel Q)[x::=y] \parallel R')$
by(*rule-tac Close1 auto*)
with $\langle x \# Q \rangle$ **have** $(P \parallel Q) \parallel R \mapsto \tau \prec \langle \nu y \rangle ((P'[x::=y] \parallel Q) \parallel R')$ **by**(*simp add: forget*)
moreover have $(\langle \nu y \rangle ((P'[x::=y] \parallel Q) \parallel R'), \langle \nu y \rangle (P'[x::=y] \parallel (Q \parallel R')))) \in Rel$ **by**(*metis Ass Res*)
ultimately show *?case* **by** *blast*
qed
next
case(*cClose2 P' QR a x y*)
from $\langle y \# Q \parallel R \rangle$ **have** $y \# Q$ **and** $y \# R$ **by** *simp+*
from $\langle x \# Q \parallel R \rangle$ **have** $x \# Q$ **and** $x \# R$ **by** *simp+*

```

with  $\langle Q \parallel R \mapsto a \langle x \rangle \prec QR \rangle$  show ?case
proof (induct rule: parCasesB)
  case (cPar1 Q')
    from  $\langle P \mapsto a \langle \nu y \rangle \prec P' \rangle \langle Q \mapsto a \langle x \rangle \prec Q' \rangle$  have  $P \parallel Q \mapsto \tau \prec \langle \nu y \rangle (P'$ 
 $\parallel Q'[x::=y])$  using  $\langle y \# Q \rangle$ 
    by (rule Close2)
    hence  $(P \parallel Q) \parallel R \mapsto \tau \prec \langle \nu y \rangle (P' \parallel Q'[x::=y]) \parallel R$  by (rule Par1F)
    moreover from  $\langle y \# R \rangle$  have  $((\langle \nu y \rangle (P' \parallel Q'[x::=y]) \parallel R, \langle \nu y \rangle (P' \parallel$ 
 $(Q'[x::=y] \parallel R))) \in Rel$ 
    by (rule FreshExt')
    with  $\langle x \# R \rangle$  have  $((\langle \nu y \rangle (P' \parallel Q'[x::=y]) \parallel R, \langle \nu y \rangle (P' \parallel (Q' \parallel R)[x::=y]))$ 
 $\in Rel$ 
    by (simp add: forget)
    ultimately show ?case by blast
  next
    case (cPar2 R')
    from  $\langle P \mapsto a \langle \nu y \rangle \prec P' \rangle \langle y \# Q \rangle$  have  $P \parallel Q \mapsto a \langle \nu y \rangle \prec P' \parallel Q$  by (rule
 $Par1B$ )
    hence  $(P \parallel Q) \parallel R \mapsto \tau \prec \langle \nu y \rangle ((P' \parallel Q) \parallel R'[x::=y])$  using  $\langle R \mapsto a \langle x \rangle$ 
 $\prec R' \rangle \langle y \# R \rangle$  by (rule Close2)
    moreover have  $((P' \parallel Q) \parallel R'[x::=y], P' \parallel (Q \parallel R'[x::=y])) \in Rel$  by (rule
 $Ass$ )
    hence  $(\langle \nu y \rangle ((P' \parallel Q) \parallel R'[x::=y]), \langle \nu y \rangle (P' \parallel (Q \parallel R'[x::=y]))) \in Rel$ 
by (rule Res)
    hence  $(\langle \nu y \rangle ((P' \parallel Q) \parallel R'[x::=y]), \langle \nu y \rangle (P' \parallel (Q \parallel R')[x::=y])) \in Rel$ 
using  $\langle x \# Q \rangle$ 
    by (simp add: forget)
    ultimately show ?case by blast
  qed
qed
qed

```

lemma *substRes3*:

```

fixes  $a :: name$ 
and  $P :: pi$ 
and  $x :: name$ 

```

shows $(\langle \nu a \rangle P)[x::=a] = \langle \nu x \rangle ([x, a] \cdot P)$

proof –

have $a \# \langle \nu a \rangle P$ **by** (*simp add: name-fresh-abs*)

hence $(\langle \nu a \rangle P)[x::=a] = [x, a] \cdot \langle \nu a \rangle P$ **by** (*rule injPermSubst[THEN sym]*)

thus $(\langle \nu a \rangle P)[x::=a] = \langle \nu x \rangle ([x, a] \cdot P)$ **by** (*simp add: name-calc*)

qed

lemma *scopeExtParLeft*:

```

fixes  $P :: pi$ 
and  $Q :: pi$ 
and  $a :: name$ 
and  $lst :: name list$ 

```

```

and Rel :: (pi × pi) set

assumes x # P
and Id:      Id ⊆ Rel
and EqvtRel: eqvt Rel
and Res:    ∧R S y. y # R ⇒ (<νy>(R || S), R || <νy>S) ∈ Rel
and ScopeExt: ∧R S y z. y # R ⇒ (<νy><νz>(R || S), <νz>(R || <νy>S)) ∈ Rel

shows <νx>(P || Q) ~>[Rel] P || <νx>Q
using <eqvt Rel>
proof(induct rule: simCasesCont[where C=(x, P, Q)])
  case(Bound a y PxQ)
  from <y # (x, P, Q)> have y ≠ x and y # P and y # Q by simp+
  hence y # P and y # <νx>Q by(simp add: abs-fresh)+
  with <P || <νx>Q ↦ a«y» < PxQ> show ?case
  proof(induct rule: parCasesB)
    case(cPar1 P')
    from <P ↦ a«y» < P'> <x # P> <y ≠ x> have x # a and x # P'
      by(force intro: freshBoundDerivative)+

    from <P ↦ a«y» < P'> <y # Q> have P || Q ↦ a«y» < P' || Q> by(rule
Par1B)
    with <x # a> <y ≠ x> have <νx>(P || Q) ↦ a«y» <νx>(P' || Q) by(rule-tac
ResB) auto
    moreover have derivative (<νx>(P' || Q)) (P' || <νx>Q) a y Rel
    proof(cases a, auto simp add: derivative-def)
    fix u

    show ((<νx>(P' || Q))[y::=u], P'[y::=u] || ((<νx>Q)[y::=u])) ∈ Rel
    proof(cases x=u)
      case True
      have (<νx>(P' || Q))[y::=x] = <νy>(((y, x) · P') || ((y, x) · Q))
        by(simp add: substRes3)
      moreover from <x # P'> have P'[y::=x] = [(y, x) · P'] by(rule in-
jPermSubst[THEN sym])
      moreover have (<νx>Q)[y::=x] = <νy>([(y, x) · Q]) by(rule substRes3)
      moreover from <x # P'> <y ≠ x> have y # [(y, x) · P'] by(simp add:
name-fresh-left name-calc)
      ultimately show ?thesis using <x = u>by(force intro: Res)
    next
      case False
      with <y ≠ x> have (<νx>(P' || Q))[y::=u] = <νx>(P'[y::=u] || Q[y::=u])
        by(simp add: fresh-prod name-fresh)
      moreover from <x ≠ u> <y ≠ x> have (<νx>Q)[y::=u] = <νx>(Q[y::=u])
        by(simp add: fresh-prod name-fresh)
      moreover from <x # P'> <x ≠ u> have x # P'[y::=u] by(simp add: fresh-fact1)
      ultimately show ?thesis by(force intro: Res)
    qed

```

```

next
  from  $\langle x \# P' \rangle$  show  $\langle \nu x \rangle (P' \parallel Q), P' \parallel \langle \nu x \rangle Q \in Rel$  by (rule Res)
qed

ultimately show ?case by blast
next
case (cPar2 xQ)
from  $\langle \nu x \rangle Q \mapsto a \langle y \rangle \prec xQ \langle y \neq x \rangle \langle y \# Q \rangle$  show ?case
proof (induct rule: resCasesB)
  case (cOpen a Q')
    from  $\langle Q \mapsto a[x] \prec Q' \rangle \langle y \# Q \rangle$  have  $yFreshQ'$ :  $y \# Q'$  by (force intro:
freshFreeDerivative)

    from  $\langle Q \mapsto a[x] \prec Q' \rangle$  have  $P \parallel Q \mapsto a[x] \prec P \parallel Q'$  by (rule Par2F)
    hence  $\langle \nu x \rangle (P \parallel Q) \mapsto a \langle \nu x \rangle \prec P \parallel Q'$  using  $\langle a \neq x \rangle$  by (rule Open)
    with  $\langle y \# P \rangle \langle y \# Q' \rangle$  have  $\langle \nu x \rangle (P \parallel Q) \mapsto a \langle \nu y \rangle \prec [(x, y)] \cdot (P \parallel Q')$ 
      by (subst alphaBoundResidual[where  $x'=x$ ]) (auto simp add: fresh-left
calc-atm)
    with  $\langle y \# P \rangle \langle x \# P \rangle$  have  $\langle \nu x \rangle (P \parallel Q) \mapsto a \langle \nu y \rangle \prec P \parallel [(x, y)] \cdot Q'$ 
      by (simp add: name-fresh-fresh)

    moreover have derivative  $(P \parallel [(x, y)] \cdot Q')$   $(P \parallel [(y, x)] \cdot Q')$ 
(BoundOutputS a)  $y Rel$  using Id
      by (auto simp add: derivative-def name-swap)

ultimately show ?case by blast
next
case (cRes Q')

from  $\langle Q \mapsto a \langle y \rangle \prec Q' \rangle \langle y \# P \rangle$  have  $P \parallel Q \mapsto a \langle y \rangle \prec P \parallel Q'$  by (rule
Par2B)
hence  $\langle \nu x \rangle (P \parallel Q) \mapsto a \langle y \rangle \prec \langle \nu x \rangle (P \parallel Q')$  using  $\langle x \# a \rangle \langle y \neq x \rangle$ 
by (rule-tac ResB) auto
moreover have derivative  $\langle \nu x \rangle (P \parallel Q')$   $(P \parallel \langle \nu x \rangle Q')$   $a y Rel$ 
proof (cases a, auto simp add: derivative-def)
  fix u
  show  $((\langle \nu x \rangle (P \parallel Q'))[y::=u], P[y::=u] \parallel (\langle \nu x \rangle Q')[y::=u]) \in Rel$ 
proof (cases  $x=u$ )
  case True
    from  $\langle x \# P \rangle \langle y \# P \rangle$  have  $\langle \nu x \rangle (P \parallel Q')[y::=x] = \langle \nu y \rangle (P \parallel [(y, x)]
\cdot Q')$ 
      by (simp add: substRes3 perm-fresh-fresh)
    moreover from  $\langle y \# P \rangle$  have  $P[y::=x] = P$  by (simp add: forget)
  moreover have  $\langle \nu x \rangle Q'[y::=x] = \langle \nu y \rangle ([ (y, x) ] \cdot Q')$  by (rule substRes3)
  ultimately show ?thesis using  $\langle x=u \rangle \langle y \# P \rangle$  by (force intro: Res)
next
case False
  with  $\langle y \neq x \rangle$  have  $\langle \nu x \rangle (P \parallel Q')[y::=u] = \langle \nu x \rangle ((P \parallel Q')[y::=u])$ 
by (simp add: fresh-prod name-fresh)

```

```

moreover from  $\langle y \neq x \rangle \langle x \neq u \rangle$  have  $(\langle \nu x \rangle Q')[y::=u] = \langle \nu x \rangle (Q'[y::=u])$ 
  by(simp add: fresh-prod name-fresh)
moreover from  $\langle x \# P \rangle \langle x \neq u \rangle$  have  $x \# P[y::=u]$  by(force simp add:
fresh-fact1)
  ultimately show ?thesis by(force intro: Res)
qed
next
from  $\langle x \# P \rangle$  show  $(\langle \nu x \rangle (P \parallel Q'), P \parallel \langle \nu x \rangle Q') \in Rel$  by(rule Res)
qed
ultimately show ?case by blast
qed
qed
next
case(Free  $\alpha PxQ$ )
from  $\langle P \parallel \langle \nu x \rangle Q \mapsto \alpha \prec PxQ \rangle$  show ?case
proof(induct rule: parCasesF[where C=x])
  case(cPar1 P')
    from  $\langle P \mapsto \alpha \prec P' \rangle \langle x \# P \rangle$  have  $x \# \alpha$  and  $x \# P'$  by(force intro: freshFreeD-
erivative)+
    from  $\langle P \mapsto \alpha \prec P' \rangle$  have  $P \parallel Q \mapsto \alpha \prec P' \parallel Q$  by(rule Par1F)
    hence  $\langle \nu x \rangle (P \parallel Q) \mapsto \alpha \prec \langle \nu x \rangle (P' \parallel Q)$  using  $\langle x \# \alpha \rangle$  by(rule ResF)
    moreover from  $\langle x \# P' \rangle$  have  $(\langle \nu x \rangle (P' \parallel Q), P' \parallel \langle \nu x \rangle Q) \in Rel$  by(rule
Res)
    ultimately show ?case by blast
  next
    case(cPar2 Q')
      from  $\langle \langle \nu x \rangle Q \mapsto \alpha \prec Q' \rangle$  show ?case
      proof(induct rule: resCasesF)
        case(cRes Q')
          from  $\langle Q \mapsto \alpha \prec Q' \rangle$  have  $P \parallel Q \mapsto \alpha \prec P \parallel Q'$  by(rule Par2F)
          hence  $\langle \nu x \rangle (P \parallel Q) \mapsto \alpha \prec \langle \nu x \rangle (P \parallel Q')$  using  $\langle x \# \alpha \rangle$  by(rule ResF)
          moreover from  $\langle x \# P \rangle$  have  $(\langle \nu x \rangle (P \parallel Q'), P \parallel \langle \nu x \rangle Q') \in Rel$  by(rule
Res)
          ultimately show ?case by blast
        qed
      next
        case(cComm1 P' xQ a b y)
          from  $\langle y \# x \rangle$  have  $y \neq x$  by simp
          from  $\langle P \mapsto a \langle y \rangle \prec P' \rangle \langle x \# P \rangle \langle y \neq x \rangle$  have  $x \# P'$  by(force intro:
freshBoundDerivative)
          from  $\langle \langle \nu x \rangle Q \mapsto a[b] \prec xQ \rangle$  show ?case
          proof(induct rule: resCasesF)
            case(cRes Q')
              from  $\langle x \# a[b] \rangle$  have  $x \neq b$  by simp
              from  $\langle P \mapsto a \langle y \rangle \prec P' \rangle \langle Q \mapsto a[b] \prec Q' \rangle$  have  $P \parallel Q \mapsto \tau \prec P'[y::=b]$ 
parallel  $Q'$  by(rule Comm1)
              hence  $\langle \nu x \rangle (P \parallel Q) \mapsto \tau \prec \langle \nu x \rangle (P'[y::=b] \parallel Q')$  by(rule-tac ResF) auto
              moreover from  $\langle x \# P' \rangle \langle x \neq b \rangle$  have  $x \# P'[y::=b]$  by(force intro: fresh-fact1)
              hence  $(\langle \nu x \rangle (P'[y::=b] \parallel Q'), P'[y::=b] \parallel \langle \nu x \rangle Q') \in Rel$  by(rule Res)
            qed
          qed
        qed
      qed
    qed
  qed

```

```

    ultimately show ?case by blast
  qed
next
case(cComm2 P' xQ a b y)
from ⟨y # x⟩ ⟨y # <νx>Q⟩ have y ≠ x and y # Q by(simp add: abs-fresh)+
with ⟨<νx>Q ⟶ a⟨y> <xQ⟩ show ?case
proof(induct rule: resCasesB)
  case(cOpen b Q')
  from ⟨InputS a = BoundOutputS b⟩ have False by simp
  thus ?case by simp
next
case(cRes Q')
  from ⟨P ⟶ a[b] <P'⟩ ⟨Q ⟶ a⟨y> <Q'⟩ have P || Q ⟶ τ <P' ||
Q'[y::=b] by(rule Comm2)
  hence <νx>(P || Q) ⟶ τ <νx>(P' || Q'[y::=b]) by(rule-tac ResF) auto
  moreover from ⟨P ⟶ a[b] <P'⟩ ⟨x # P⟩ have x # P' and x ≠ b by(force
dest: freshFreeDerivative)+
  from ⟨x # P'⟩ have (<νx>(P' || Q'[y::=b]), P' || <νx>(Q'[y::=b])) ∈ Rel
by(rule Res)
  with ⟨y ≠ x⟩ ⟨x ≠ b⟩ have (<νx>(P' || Q'[y::=b]), P' || (<νx>Q')[y::=b])
∈ Rel by simp
  ultimately show ?case by blast
  qed
next
case(cClose1 P' Q' a y z)
from ⟨y # x⟩ have y ≠ x by simp
from ⟨z # x⟩ ⟨z # <νx>Q⟩ have z # Q and z ≠ x by(simp add: abs-fresh)+
from ⟨P ⟶ a⟨y> <P'⟩ ⟨z # P⟩ have z ≠ a by(force dest: freshBoundDeriva-
tive)
from ⟨<νx>Q ⟶ a⟨νz> <Q'⟩ ⟨z ≠ x⟩ ⟨z # Q⟩ show ?case
proof(induct rule: resCasesB)
  case(cOpen b Q')
  from ⟨BoundOutputS a = BoundOutputS b⟩ have a = b by simp
  with ⟨Q ⟶ b[x] <Q'⟩ have ([[z, x]] · Q) ⟶ [[z, x]] · (a[x] <Q')
  by(rule-tac transitions.eqvt) simp
  with ⟨b ≠ x⟩ ⟨z ≠ a⟩ ⟨a = b⟩ ⟨z ≠ x⟩ have ([[z, x]] · Q) ⟶ a[z] <([[z, x]]
· Q')
  by(simp add: name-calc eqvts)
  with ⟨P ⟶ a⟨y> <P'⟩ have P || ([[z, x]] · Q) ⟶ τ <P'[y::=z] || ([[z, x]]
· Q')
  by(rule Comm1)
  hence <νz>(P || ([[z, x]] · Q)) ⟶ τ <νz>(P'[y::=z] || ([[z, x]] · Q'))
  by(rule-tac ResF) auto
  hence <νx>(P || Q) ⟶ τ <νz>(P'[y::=z] || ([[z, x]] · Q')) using ⟨z #
P⟩ ⟨z # Q⟩ ⟨x # P⟩
  by(subst alphaRes[where c=z]) auto
  with Id show ?case by force
next
case(cRes Q')

```

```

    from  $\langle P \mapsto a \langle y \rangle \prec P' \rangle \langle Q \mapsto a \langle \nu z \rangle \prec Q' \rangle \langle z \# P \rangle$  have  $P \parallel Q \mapsto \tau \prec$ 
 $\langle \nu z \rangle (P'[y::=z] \parallel Q')$ 
    by(rule Close1)
    hence  $\langle \nu x \rangle (P \parallel Q) \mapsto \tau \prec \langle \nu x \rangle \langle \nu z \rangle (P'[y::=z] \parallel Q')$  by(rule-tac ResF)
  auto
  moreover from  $\langle P \mapsto a \langle y \rangle \prec P' \rangle \langle y \neq x \rangle \langle x \# P \rangle$  have  $x \# P'$ 
  by(force dest: freshBoundDerivative)
  with  $\langle z \neq x \rangle$  have  $x \# P'[y::=z]$  by(simp add: fresh-fact1)
  hence  $(\langle \nu x \rangle \langle \nu z \rangle (P'[y::=z] \parallel Q'), \langle \nu z \rangle (P'[y::=z] \parallel \langle \nu x \rangle Q')) \in Rel$ 
  by(rule ScopeExt)
  ultimately show ?case by blast
qed
next
case(cClose2 P' xQ a y z)
from  $\langle z \# x \rangle \langle z \# \langle \nu x \rangle Q \rangle$  have  $z \neq x$  and  $z \# Q$  by(auto simp add: abs-fresh)
from  $\langle y \# x \rangle \langle y \# \langle \nu x \rangle Q \rangle$  have  $y \neq x$  and  $y \# Q$  by(auto simp add: abs-fresh)
with  $\langle \langle \nu x \rangle Q \mapsto a \langle y \rangle \prec xQ \rangle$  show ?case
proof(induct rule: resCasesB)
  case(cOpen b Q')
  from  $\langle InputS a = BoundOutputS b \rangle$  have False by simp
  thus ?case by simp
next
case(cRes Q')
from  $\langle P \mapsto a \langle \nu z \rangle \prec P' \rangle \langle Q \mapsto a \langle y \rangle \prec Q' \rangle \langle z \# Q \rangle$  have  $P \parallel Q \mapsto \tau$ 
 $\prec \langle \nu z \rangle (P' \parallel Q'[y::=z])$ 
  by(rule Close2)
  hence  $\langle \nu x \rangle (P \parallel Q) \mapsto \tau \prec \langle \nu x \rangle \langle \nu z \rangle (P' \parallel (Q'[y::=z]))$ 
  by(rule-tac ResF) auto
  moreover from  $\langle P \mapsto a \langle \nu z \rangle \prec P' \rangle \langle x \# P \rangle \langle z \neq x \rangle$  have  $x \# P'$  by(force
  dest: freshBoundDerivative)
  hence  $(\langle \nu x \rangle \langle \nu z \rangle (P' \parallel (Q'[y::=z])), \langle \nu z \rangle (P' \parallel (\langle \nu x \rangle (Q'[y::=z]))) \in Rel$ 
  by(rule ScopeExt)
  with  $\langle z \neq x \rangle \langle y \neq x \rangle$  have  $(\langle \nu x \rangle \langle \nu z \rangle (P' \parallel (Q'[y::=z])), \langle \nu z \rangle (P' \parallel$ 
 $(\langle \nu x \rangle Q')[y::=z])) \in Rel$ 
  by simp
  ultimately show ?case by blast
qed
qed
qed

```

lemma *scopeExtParRight*:

```

fixes  $P :: pi$ 
and  $Q :: pi$ 
and  $a :: name$ 
and  $Rel :: (pi \times pi) set$ 

```

```

assumes  $x \# P$ 
and  $Id: Id \subseteq Rel$ 
and  $eqvt Rel$ 

```

and *Res*: $\bigwedge R S y. y \# R \implies (R \parallel \langle \nu y \rangle S, \langle \nu y \rangle (R \parallel S)) \in Rel$
and *ScopeExt*: $\bigwedge R S y z. y \# R \implies (\langle \nu z \rangle (R \parallel \langle \nu y \rangle S), \langle \nu y \rangle \langle \nu z \rangle (R \parallel S)) \in Rel$

shows $P \parallel \langle \nu x \rangle Q \rightsquigarrow_{[Rel]} \langle \nu x \rangle (P \parallel Q)$
using $\langle eqvt Rel \rangle$
proof(*induct rule: simCasesCont*[**where** $C=(x, P, Q)$])
 case(*Bound a y xPQ*)
 from $\langle y \# (x, P, Q) \rangle$ **have** $y \neq x$ **and** $y \# P$ **and** $y \# Q$ **by** *simp+*
 hence $y \neq x$ **and** $y \# P \parallel Q$ **by**(*auto simp add: abs-fresh*)
 with $\langle \langle \nu x \rangle (P \parallel Q) \mapsto a \langle y \rangle \prec xPQ \rangle$ **show** *?case*
 proof(*induct rule: resCasesB*)
 case(*cOpen a PQ*)
 from $\langle P \parallel Q \mapsto a[x] \prec PQ \rangle$ **show** *?case*
 proof(*induct rule: parCasesF*[**where** $C=()$])
 case(*cPar1 P'*)
 from $\langle P \mapsto a[x] \prec P' \rangle \langle x \# P \rangle$ **have** $x \neq x$ **by**(*force dest: freshFreeDerivative*)
 thus *?case* **by** *simp*
 next
 case(*cPar2 Q'*)
 from $\langle Q \mapsto a[x] \prec Q' \rangle \langle y \# Q \rangle$ **have** $y \# Q'$ **by**(*force dest: freshFreeDerivative*)
 from $\langle Q \mapsto a[x] \prec Q' \rangle \langle a \neq x \rangle$ **have** $\langle \nu x \rangle Q \mapsto a \langle \nu x \rangle \prec Q'$ **by**(*rule*
Open)
 hence $P \parallel \langle \nu x \rangle Q \mapsto a \langle \nu x \rangle \prec P \parallel Q'$ **using** $\langle x \# P \rangle$ **by**(*rule Par2B*)
 with $\langle y \# P \rangle \langle y \# Q' \rangle \langle x \# P \rangle$ **have** $P \parallel \langle \nu x \rangle Q \mapsto a \langle \nu y \rangle \prec ([(y, x)] \cdot (P \parallel Q'))$
 by(*subst alphaBoundResidual*[**where** $x'=x$]) (*auto simp add: fresh-left calc-atm*)
 moreover with *Id* **have** *derivative* $([(y, x)] \cdot (P \parallel Q'))$
 $([(y, x)] \cdot (P \parallel Q'))$ (*BoundOutputS a*) *y Rel*
 by(*auto simp add: derivative-def*)
 ultimately show *?case* **by** *blast*
 next
 case(*cComm1 P' Q' b c y*)
 from $\langle a[x] = \tau \rangle$ **show** *?case* **by** *simp*
 next
 case(*cComm2 P' Q' b c y*)
 from $\langle a[x] = \tau \rangle$ **show** *?case* **by** *simp*
 next
 case(*cClose1 P' Q' b y z*)
 from $\langle a[x] = \tau \rangle$ **show** *?case* **by** *simp*
 next
 case(*cClose2 P' Q' b y z*)
 from $\langle a[x] = \tau \rangle$ **show** *?case* **by** *simp*
 qed
 next
 case(*cRes PQ*)
 from $\langle P \parallel Q \mapsto a \langle y \rangle \prec PQ \rangle \langle y \# P \rangle \langle y \# Q \rangle$
 show *?case*

```

proof(induct rule: parCasesB)
  case(cPar1 P')
    from  $\langle y \neq x \rangle \langle x \# P \rangle \langle P \mapsto a \langle y \rangle \prec P' \rangle$  have  $x \# P'$  by(force dest: fresh-
BoundDerivative)

    from  $\langle P \mapsto a \langle y \rangle \prec P' \rangle \langle y \# Q \rangle$  have  $P \parallel \langle \nu x \rangle Q \mapsto a \langle y \rangle \prec P' \parallel \langle \nu x \rangle Q$ 
      by(rule-tac Par1B) (auto simp add: abs-fresh)
    moreover have derivative  $(P' \parallel \langle \nu x \rangle Q) (\langle \nu x \rangle (P' \parallel Q))$  a y Rel
    proof(cases a, auto simp add: derivative-def)
      fix u::name
      obtain z::name where  $z \# Q$  and  $y \neq z$  and  $z \neq u$  and  $z \# P$  and  $z \# P'$ 
        by(generate-fresh name) auto
      thus  $(P'[y::=u] \parallel (\langle \nu x \rangle Q)[y::=u], (\langle \nu x \rangle (P' \parallel Q))[y::=u]) \in Rel$  using
 $\langle x \# P' \rangle$ 
        by(subst alphaRes[where c=z and a=x], auto)
          (subst alphaRes[where c=z and a=x], auto intro: Res simp add:
fresh-fact1)
      next
        from  $\langle x \# P' \rangle$  show  $(P' \parallel \langle \nu x \rangle Q, \langle \nu x \rangle (P' \parallel Q)) \in Rel$ 
          by(rule Res)
      qed

    ultimately show ?case by blast
  next
    case(cPar2 Q')
      from  $\langle Q \mapsto a \langle y \rangle \prec Q' \rangle$  have  $\langle \nu x \rangle Q \mapsto a \langle y \rangle \prec \langle \nu x \rangle Q'$  using  $\langle x \# a \rangle$ 
 $\langle y \neq x \rangle$ 
        by(rule-tac ResB) auto
      hence  $P \parallel \langle \nu x \rangle Q \mapsto a \langle y \rangle \prec P \parallel \langle \nu x \rangle Q'$  using  $\langle y \# P \rangle$  by(rule Par2B)

      moreover have derivative  $(P \parallel \langle \nu x \rangle Q') (\langle \nu x \rangle (P \parallel Q'))$  a y Rel
      proof(cases a, auto simp add: derivative-def)
        fix u::name
        obtain z::name where  $z \# Q$  and  $z \neq y$  and  $z \neq u$  and  $z \# P$  and  $z \# Q'$ 
          by(generate-fresh name) auto

        thus  $(P[y::=u] \parallel (\langle \nu x \rangle Q')[y::=u], (\langle \nu x \rangle (P \parallel Q'))[y::=u]) \in Rel$  using
 $\langle x \# P \rangle$ 
          by(subst alphaRes[where a=x and c=z], auto)
            (subst alphaRes[where a=x and c=z], auto intro: Res simp add:
fresh-fact1)
        next
          from  $\langle x \# P \rangle$  show  $(P \parallel \langle \nu x \rangle Q', \langle \nu x \rangle (P \parallel Q')) \in Rel$ 
            by(rule Res)
        qed

      ultimately show ?case by blast
    qed
  qed

```

```

next
case(Free  $\alpha$   $xPQ$ )
from  $\langle \nu x \rangle (P \parallel Q) \mapsto \alpha \prec xPQ$  show ?case
proof(induct rule: resCasesF)
  case(cRes  $PQ$ )
  from  $\langle P \parallel Q \mapsto \alpha \prec PQ \rangle$  show ?case
  proof(induct rule: parCasesF[where  $C=x$ ])
    case(cPar1  $P'$ )
    from  $\langle P \mapsto \alpha \prec P' \rangle$  have  $P \parallel \langle \nu x \rangle Q \mapsto \alpha \prec P' \parallel \langle \nu x \rangle Q$  by(rule Par1F)
    moreover from  $\langle P \mapsto \alpha \prec P' \rangle \langle x \# P \rangle$  have  $x \# P'$  by(rule freshFreeDerivative)
    hence  $(P' \parallel \langle \nu x \rangle Q, \langle \nu x \rangle (P' \parallel Q)) \in Rel$  by(rule Res)
    ultimately show ?case by blast
  next
  case(cPar2  $Q'$ )
  from  $\langle Q \mapsto \alpha \prec Q' \rangle \langle x \# \alpha \rangle$  have  $\langle \nu x \rangle Q \mapsto \alpha \prec \langle \nu x \rangle Q'$  by(rule ResF)
  hence  $P \parallel \langle \nu x \rangle Q \mapsto \alpha \prec P \parallel \langle \nu x \rangle Q'$  by(rule Par2F)
  moreover from  $\langle x \# P \rangle$  have  $(P \parallel \langle \nu x \rangle Q', \langle \nu x \rangle (P \parallel Q')) \in Rel$  by(rule Res)
  ultimately show ?case by blast
  next
  case(cComm1  $P' Q' a b y$ )
  from  $\langle x \# P \rangle \langle y \# x \rangle \langle P \mapsto a \langle y \rangle \prec P' \rangle$  have  $x \neq a$  and  $x \# P'$  by(force dest: freshBoundDerivative)+
  show ?case
  proof(cases  $b=x$ )
    case True
    from  $\langle Q \mapsto a[b] \prec Q' \rangle \langle x \neq a \rangle \langle b = x \rangle$  have  $\langle \nu x \rangle Q \mapsto a \langle \nu x \rangle \prec Q'$ 
  by(rule-tac Open) auto
    with  $\langle P \mapsto a \langle y \rangle \prec P' \rangle$  have  $P \parallel \langle \nu x \rangle Q \mapsto \tau \prec \langle \nu x \rangle (P'[y::=x] \parallel Q')$ 
  using  $\langle x \# P \rangle$  by(rule Close1)
    moreover from Id have  $(\langle \nu x \rangle (P'[y::=b] \parallel Q'), \langle \nu x \rangle (P'[y::=b] \parallel Q')) \in Rel$ 
  by blast
    ultimately show ?thesis using  $\langle b=x \rangle$  by blast
  next
  case False
  from  $\langle Q \mapsto a[b] \prec Q' \rangle \langle x \neq a \rangle \langle b \neq x \rangle$  have  $\langle \nu x \rangle Q \mapsto a[b] \prec \langle \nu x \rangle Q'$ 
  by(rule-tac ResF) auto
    with  $\langle P \mapsto a \langle y \rangle \prec P' \rangle$  have  $P \parallel \langle \nu x \rangle Q \mapsto \tau \prec (P'[y::=b] \parallel \langle \nu x \rangle Q')$ 
  by(rule Comm1)
    moreover from  $\langle x \# P' \rangle \langle b \neq x \rangle$  have  $(P'[y::=b] \parallel \langle \nu x \rangle Q', \langle \nu x \rangle (P'[y::=b] \parallel Q')) \in Rel$ 
  by(force intro: Res simp add: fresh-fact1)
    ultimately show ?thesis by blast
  qed
  next
  case(cComm2  $P' Q' a b y$ )
  from  $\langle P \mapsto a[b] \prec P' \rangle \langle x \# P \rangle$  have  $x \neq a$  and  $x \neq b$  and  $x \# P'$  by(force dest: freshFreeDerivative)+

```

```

from  $\langle Q \mapsto a \langle y \rangle \prec Q' \rangle \langle y \# x \rangle \langle x \neq a \rangle$  have  $\langle \nu x \rangle Q \mapsto a \langle y \rangle \prec \langle \nu x \rangle Q'$ 
by(rule-tac ResB) auto
  with  $\langle P \mapsto a \langle b \rangle \prec P' \rangle$  have  $P \parallel \langle \nu x \rangle Q \mapsto \tau \prec P' \parallel (\langle \nu x \rangle Q')[y ::= b]$ 
by(rule Comm2)
  moreover from  $\langle x \# P' \rangle$  have  $(P' \parallel \langle \nu x \rangle (Q'[y ::= b]), \langle \nu x \rangle (P' \parallel Q'[y ::= b]))$ 
 $\in \text{Rel}$  by(rule Res)
  ultimately show ?case using  $\langle y \# x \rangle \langle x \neq b \rangle$  by force
next
  case(cClose1 P' Q' a y z)
  from  $\langle P \mapsto a \langle y \rangle \prec P' \rangle \langle x \# P \rangle \langle y \# x \rangle$  have  $x \neq a$  and  $x \# P'$  by(force
dest: freshBoundDerivative)+
  from  $\langle Q \mapsto a \langle \nu z \rangle \prec Q' \rangle \langle z \# x \rangle \langle x \neq a \rangle$  have  $\langle \nu x \rangle Q \mapsto a \langle \nu z \rangle \prec$ 
 $\langle \nu x \rangle Q'$  by(rule-tac ResB) auto
  with  $\langle P \mapsto a \langle y \rangle \prec P' \rangle$  have  $P \parallel \langle \nu x \rangle Q \mapsto \tau \prec \langle \nu z \rangle (P'[y ::= z]) \parallel$ 
 $\langle \nu x \rangle Q'$  using  $\langle z \# P \rangle$  by(rule Close1)
  moreover from  $\langle x \# P' \rangle \langle z \# x \rangle$  have  $(\langle \nu z \rangle (P'[y ::= z]) \parallel \langle \nu x \rangle Q'), \langle \nu x \rangle (\langle \nu z \rangle (P'[y ::= z])$ 
 $\parallel Q')$   $\in \text{Rel}$ 
  by(rule-tac ScopeExt) (auto simp add: fresh-fact1)
  ultimately show ?case by blast
next
  case(cClose2 P' Q' a y z)
  from  $\langle P \mapsto a \langle \nu z \rangle \prec P' \rangle \langle x \# P \rangle \langle z \# x \rangle$  have  $x \neq a$  and  $x \# P'$  by(force
dest: freshBoundDerivative)+
  from  $\langle Q \mapsto a \langle y \rangle \prec Q' \rangle \langle y \# x \rangle \langle x \neq a \rangle$  have  $\langle \nu x \rangle Q \mapsto a \langle y \rangle \prec \langle \nu x \rangle Q'$ 
by(rule-tac ResB) auto
  with  $\langle P \mapsto a \langle \nu z \rangle \prec P' \rangle$  have  $P \parallel \langle \nu x \rangle Q \mapsto \tau \prec \langle \nu z \rangle (P' \parallel (\langle \nu x \rangle Q')[y ::= z])$ 
using  $\langle z \# Q \rangle$ 
  by(rule-tac Close2) (auto simp add: abs-fresh)
  moreover from  $\langle x \# P' \rangle$  have  $(\langle \nu z \rangle (P' \parallel \langle \nu x \rangle (Q'[y ::= z])), \langle \nu x \rangle \langle \nu z \rangle (P'$ 
 $\parallel Q'[y ::= z])) \in \text{Rel}$  by(rule ScopeExt)
  ultimately show ?case using  $\langle z \# x \rangle \langle y \# x \rangle$  by force
qed
qed
qed

```

lemma *resNilRight*:

fixes $x :: \text{name}$
and $\text{Rel} :: (\text{pi} \times \text{pi}) \text{ set}$

shows $0 \rightsquigarrow[\text{Rel}] \langle \nu x \rangle 0$

by(*fastforce simp add: simulation-def pi.inject alpha' elim: resCasesB' resCasesF*)

lemma *resComm*:

fixes $a :: \text{name}$
and $b :: \text{name}$
and $P :: \text{pi}$
and $\text{Rel} :: (\text{pi} \times \text{pi}) \text{ set}$

assumes *ResComm*: $\bigwedge c d Q. (\langle \nu c \rangle \langle \nu d \rangle Q, \langle \nu d \rangle \langle \nu c \rangle Q) \in \text{Rel}$

```

and   Id:       $Id \subseteq Rel$ 
and   EqvtRel:  $eqvt\ Rel$ 

shows  $\langle \nu a \rangle \langle \nu b \rangle P \rightsquigarrow [Rel] \langle \nu b \rangle \langle \nu a \rangle P$ 
proof(cases a=b)
  assume  $a=b$ 
  with Id show ?thesis by(force intro: Strong-Late-Sim.reflexive)
next
  assume aineqb:  $a \neq b$ 
  from EqvtRel show ?thesis
  proof(induct rule: simCasesCont[where  $C=(a, b, P)$ ])
    case(Bound c x baP)
      from  $\langle x \# (a, b, P) \rangle$  have  $x \neq a$  and  $x \neq b$  and  $x \# P$  by simp+
      from  $\langle x \# P \rangle$  have  $x \# \langle \nu a \rangle P$  by(simp add: abs-fresh)
      with  $\langle \nu b \rangle \langle \nu a \rangle P \mapsto c \langle x \rangle \prec baP \langle x \neq b \rangle$  show ?case
      proof(induct rule: resCasesB)
        case(cOpen c aP)
          from  $\langle \nu a \rangle P \mapsto c[b] \prec aP$ 
          show ?case
        proof(induct rule: resCasesF)
          case(cRes P')
            from  $\langle a \# c[b] \rangle$  have  $a \neq c$  and  $a \neq b$  by simp+
            from  $\langle x \# P \rangle \langle P \mapsto c[b] \prec P' \rangle$  have  $x \neq c$  and  $x \# P'$  by(force dest: freshFreeDerivative)+
            from  $\langle P \mapsto c[b] \prec P' \rangle$  have  $[(x, b)] \cdot P \mapsto [(x, b)] \cdot (c[b] \prec P')$  by(rule transitions.eqvt)
            with  $\langle x \neq c \rangle \langle c \neq b \rangle \langle x \neq b \rangle$  have  $[(x, b)] \cdot P \mapsto c[x] \prec [(x, b)] \cdot P'$ 
by(simp add: eqvts calc-atm)
            hence  $\langle \nu x \rangle ([x, b]) \cdot P \mapsto c \langle \nu x \rangle \prec [(x, b)] \cdot P'$  using  $\langle x \neq c \rangle$ 
by(rule-tac Open) auto
            with  $\langle x \# P \rangle$  have  $\langle \nu b \rangle P \mapsto c \langle \nu x \rangle \prec [(x, b)] \cdot P'$  by(simp add: alphaRes)
            hence  $\langle \nu a \rangle \langle \nu b \rangle P \mapsto c \langle \nu x \rangle \prec \langle \nu a \rangle ([x, b]) \cdot P'$  using  $\langle a \neq c \rangle \langle x \neq a \rangle$ 
by(rule-tac ResB) auto
            moreover from Id have derivative ( $\langle \nu a \rangle ([x, b]) \cdot P'$ ) ( $\langle \nu a \rangle ([x, b]) \cdot P'$ )
(BoundOutputS c) x Rel
            by(force simp add: derivative-def)
            ultimately show ?case using  $\langle a \neq b \rangle \langle x \neq a \rangle \langle a \neq c \rangle$  by(force simp add: eqvts calc-atm)
          qed
        next
          case(cRes aP)
            from  $\langle \nu a \rangle P \mapsto c \langle x \rangle \prec aP \langle x \neq a \rangle \langle x \# P \rangle \langle b \# c \rangle$  show ?case
            proof(induct rule: resCasesB)
              case(cOpen c P')
                from  $\langle P \mapsto c[a] \prec P' \rangle \langle x \# P \rangle$  have  $x \# P'$  by(force intro: freshFreeDerivative)
                from  $\langle b \# BoundOutputS c \rangle$  have  $b \neq c$  by simp

```

```

with  $\langle P \mapsto c[a] \prec P' \rangle \langle a \neq b \rangle$  have  $\langle \nu b \rangle P \mapsto c[a] \prec \langle \nu b \rangle P'$  by(rule-tac ResF) auto
with  $\langle c \neq a \rangle$  have  $\langle \nu a \rangle \langle \nu b \rangle P \mapsto c \langle \nu a \rangle \prec \langle \nu b \rangle P'$  by(rule-tac Open) auto
hence  $\langle \nu a \rangle \langle \nu b \rangle P \mapsto c \langle \nu x \rangle \prec \langle \nu b \rangle ((x, a) \cdot P')$  using  $\langle x \neq b \rangle \langle a \neq b \rangle \langle x \# P' \rangle$ 
apply(subst alphaBoundResidual[where  $x'=a$ ]) by(auto simp add: abs-fresh fresh-left calc-atm)
moreover have derivative ( $\langle \nu b \rangle ((x, a) \cdot P')$ ) ( $\langle \nu b \rangle ((x, a) \cdot P')$ )
(BoundOutputS c) x Rel using Id
by(force simp add: derivative-def)
ultimately show ?case by blast
next
case(cRes P')
from  $\langle P \mapsto c \langle x \rangle \prec P' \rangle \langle b \# c \rangle \langle x \neq b \rangle$  have  $\langle \nu b \rangle P \mapsto c \langle x \rangle \prec \langle \nu b \rangle P'$ 
by(rule-tac ResB) auto
hence  $\langle \nu a \rangle \langle \nu b \rangle P \mapsto c \langle x \rangle \prec \langle \nu a \rangle \langle \nu b \rangle P'$  using  $\langle a \# c \rangle \langle x \neq a \rangle$ 
by(rule-tac ResB) auto
moreover have derivative ( $\langle \nu a \rangle \langle \nu b \rangle P'$ ) ( $\langle \nu b \rangle \langle \nu a \rangle P'$ ) c x Rel
proof(cases c, auto simp add: derivative-def)
fix u::name
show ( $\langle \nu a \rangle \langle \nu b \rangle P'$ )[ $x::=u$ ], ( $\langle \nu b \rangle \langle \nu a \rangle P'$ )[ $x::=u$ ]  $\in Rel$ 
proof(cases u=a)
case True
from  $\langle u = a \rangle \langle a \neq b \rangle$  show ?thesis
by(subst injPermSubst[symmetric], auto simp add: abs-fresh)
(subst injPermSubst[symmetric], auto simp add: abs-fresh calc-atm)
intro: ResComm)
next
case False
show ?thesis
proof(cases u=b)
case True
from  $\langle u = b \rangle \langle u \neq a \rangle$  show ?thesis
by(subst injPermSubst[symmetric], auto simp add: abs-fresh)
(subst injPermSubst[symmetric], auto simp add: abs-fresh calc-atm)
intro: ResComm)
next
case False
from  $\langle u \neq a \rangle \langle u \neq b \rangle \langle x \neq a \rangle \langle x \neq b \rangle$  show ?thesis by(auto intro: ResComm)
qed
qed
next
show ( $\langle \nu a \rangle \langle \nu b \rangle P'$ ,  $\langle \nu b \rangle \langle \nu a \rangle P'$ )  $\in Rel$  by(rule ResComm)
qed
ultimately show ?case by blast
qed
qed

```

```

next
  case(Free  $\alpha$   $baP$ )
  from  $\langle \nu b \rangle \langle \nu a \rangle P \mapsto \alpha \prec baP$  show ?case
  proof(induct rule: resCasesF)
    case(cRes  $aP$ )
    from  $\langle \nu a \rangle P \mapsto \alpha \prec aP$  show ?case
    proof(induct rule: resCasesF)
      case(cRes  $P'$ )
      from  $\langle P \mapsto \alpha \prec P' \rangle \langle b \# \alpha \rangle$  have  $\langle \nu b \rangle P \mapsto \alpha \prec \langle \nu b \rangle P'$  by(rule ResF)
      hence  $\langle \nu a \rangle \langle \nu b \rangle P \mapsto \alpha \prec \langle \nu a \rangle \langle \nu b \rangle P'$  using  $\langle a \# \alpha \rangle$  by(rule ResF)
      moreover have  $(\langle \nu a \rangle \langle \nu b \rangle P', \langle \nu b \rangle \langle \nu a \rangle P') \in Rel$  by(rule ResComm)
      ultimately show ?case by blast
    qed
  qed
qed
qed

```

```

lemma bangLeftSC:
  fixes  $P :: pi$ 
  and  $Rel :: (pi \times pi)$  set

  assumes  $Id \subseteq Rel$ 

  shows  $!P \rightsquigarrow_{[Rel]} P \parallel !P$ 
  using assms
  by(force simp add: simulation-def dest: Bang derivativeReflexive)

```

```

lemma bangRightSC:
  fixes  $P :: pi$ 
  and  $Rel :: (pi \times pi)$  set

  assumes  $IdRel: Id \subseteq Rel$ 

  shows  $P \parallel !P \rightsquigarrow_{[Rel]} !P$ 
  using assms
  by(fastforce simp add: pi.inject simulation-def intro: derivativeReflexive elim: bang-
  Cases)

```

```

lemma resNilLeft:
  fixes  $x :: name$ 
  and  $y :: name$ 
  and  $P :: pi$ 
  and  $Rel :: (pi \times pi)$  set
  and  $b :: name$ 

  shows  $0 \rightsquigarrow_{[Rel]} \langle \nu x \rangle (x \langle y \rangle . P)$ 
  and  $0 \rightsquigarrow_{[Rel]} \langle \nu x \rangle (x \{b\} . P)$ 

```

by(*auto simp add: simulation-def*)

lemma *resInputLeft*:

fixes $x :: \text{name}$
and $a :: \text{name}$
and $y :: \text{name}$
and $P :: \text{pi}$
and $\text{Rel} :: (\text{pi} \times \text{pi}) \text{ set}$

assumes $x \text{ineq} a: x \neq a$
and $x \text{ineq} y: x \neq y$
and $\text{Eqvt}: \text{eqvt Rel}$
and $\text{Id}: \text{Id} \subseteq \text{Rel}$

shows $\langle \nu x \rangle a \langle y \rangle . P \rightsquigarrow [\text{Rel}] a \langle y \rangle . (\langle \nu x \rangle P)$

using *Eqvt*

proof(*induct rule: simCasesCont*[**where** $C = (x, y, a, P)$])

case(*Bound* $b \ z \ P'$)

from $\langle z \# (x, y, a, P) \rangle$ **have** $z \neq x$ **and** $z \neq y$ **and** $z \# P$ **and** $z \neq a$ **by** *simp+*

from $\langle z \# P \rangle$ **have** $z \# \langle \nu x \rangle P$ **by**(*simp add: abs-fresh*)

with $\langle a \langle y \rangle . (\langle \nu x \rangle P) \mapsto b \langle z \rangle \prec P' \rangle$ $\langle z \neq a \rangle$ $\langle z \neq y \rangle$ **show** *?case*

proof(*induct rule: inputCases*)

case *cInput*

have $a \langle y \rangle . P \mapsto a \langle y \rangle \prec P$ **by**(*rule Input*)

with $\langle x \neq y \rangle$ $\langle x \neq a \rangle$ **have** $\langle \nu x \rangle a \langle y \rangle . P \mapsto a \langle y \rangle \prec \langle \nu x \rangle P$ **by**(*rule-tac ResB*) *auto*

hence $\langle \nu x \rangle a \langle y \rangle . P \mapsto a \langle z \rangle \prec [(y, z)] \cdot \langle \nu x \rangle P$ **using** $\langle z \# P \rangle$

by(*subst alphaBoundResidual*[**where** $x' = y$]) (*auto simp add: abs-fresh fresh-left calc-atm*)

moreover from *Id* **have** *derivative* ($[(y, z)] \cdot \langle \nu x \rangle P$) ($[(y, z)] \cdot \langle \nu x \rangle P$)
(*InputS a*) $z \text{ Rel}$

by(*rule derivativeReflexive*)

ultimately show *?case* **by** *blast*

qed

next

case(*Free* $\alpha \ P'$)

from $\langle a \langle y \rangle . (\langle \nu x \rangle P) \mapsto \alpha \prec P' \rangle$ **have** *False* **by** *auto*

thus *?case* **by** *simp*

qed

lemma *resInputRight*:

fixes $a :: \text{name}$
and $y :: \text{name}$
and $x :: \text{name}$
and $P :: \text{pi}$
and $\text{Rel} :: (\text{pi} \times \text{pi}) \text{ set}$

assumes $x \text{ineq} a: x \neq a$

and $x \text{ineq} y: x \neq y$

```

and   Eqvt: eqvt Rel
and   Id:  $Id \subseteq Rel$ 

shows  $a \langle y \rangle . (\langle \nu x \rangle P) \rightsquigarrow [Rel] \langle \nu x \rangle a \langle y \rangle . P$ 
using Eqvt
proof(induct rule: simCasesCont[where  $C = (x, y, a, P)$ ])
  case(Bound  $b \ z \ xP$ )
    from  $\langle z \# (x, y, a, P) \rangle$  have  $z \neq x$  and  $z \neq y$  and  $z \# P$  and  $z \neq a$  by simp+
    from  $\langle z \neq a \rangle \langle z \# P \rangle$  have  $z \# a \langle y \rangle . P$  by(simp add: abs-fresh)
    with  $\langle \langle \nu x \rangle a \langle y \rangle . P \mapsto b \langle z \rangle \prec xP \rangle \langle z \neq x \rangle$  show ?case
    proof(induct rule: resCasesB)
      case(cOpen  $b \ P'$ )
        from  $\langle a \langle y \rangle . P \mapsto b[x] \prec P' \rangle$  have False by auto
        thus ?case by simp
      next
        case(cRes  $P'$ )
          from  $\langle a \langle y \rangle . P \mapsto b \langle z \rangle \prec P' \rangle \langle z \neq a \rangle \langle z \neq y \rangle \langle z \# P \rangle$  show ?case
          proof(induct rule: inputCases)
            case cInput
              have  $a \langle y \rangle . (\langle \nu x \rangle P) \mapsto a \langle y \rangle \prec (\langle \nu x \rangle P)$  by(rule Input)
              with  $\langle z \# P \rangle \langle x \neq y \rangle \langle z \neq x \rangle$  have  $a \langle y \rangle . (\langle \nu x \rangle P) \mapsto a \langle z \rangle \prec (\langle \nu x \rangle ([ (y, z) ] \cdot P))$ 
                by(subst alphaBoundResidual[where  $x' = y$ ]) (auto simp add: abs-fresh calc-atm fresh-left)
              moreover from Id have derivative  $(\langle \nu x \rangle ([ (y, z) ] \cdot P)) (\langle \nu x \rangle ([ (y, z) ] \cdot P))$  (InputS  $a$ )  $z \ Rel$ 
                by(rule derivativeReflexive)
              ultimately show ?case by blast
            qed
          qed
        next
          case(Free  $\alpha \ P'$ )
            from  $\langle \langle \nu x \rangle a \langle y \rangle . P \mapsto \alpha \prec P' \rangle$  have False by auto
            thus ?case by simp
          qed
    qed

lemma resOutputLeft:
  fixes  $x \ :: \ name$ 
  and    $a \ :: \ name$ 
  and    $b \ :: \ name$ 
  and    $P \ :: \ pi$ 
  and    $Rel \ :: \ (pi \times pi) \ set$ 

  assumes xineqa:  $x \neq a$ 
  and   xineqb:  $x \neq b$ 
  and   Id:  $Id \subseteq Rel$ 

  shows  $\langle \nu x \rangle a \{b\} . P \rightsquigarrow [Rel] a \{b\} . (\langle \nu x \rangle P)$ 
using assms

```

by(*fastforce simp add: simulation-def elim: outputCases intro: Output ResF*)

lemma *resOutputRight*:

fixes $x :: \text{name}$
and $a :: \text{name}$
and $b :: \text{name}$
and $P :: \text{pi}$
and $\text{Rel} :: (\text{pi} \times \text{pi}) \text{ set}$

assumes $x \text{ineq} a: x \neq a$
and $x \text{ineq} b: x \neq b$
and $\text{Id}: \text{Id} \subseteq \text{Rel}$
and $\text{Eqvt}: \text{eqvt Rel}$

shows $a\{b\}.(\nu x > P) \rightsquigarrow[\text{Rel}] <\nu x > a\{b\}.P$

using *assms*

by(*erule-tac simCasesCont[where C=x]*)

(*force simp add: abs-fresh elim: resCasesB resCasesF outputCases intro: ResF Output*)+

lemma *resTauLeft*:

fixes $x :: \text{name}$
and $P :: \text{pi}$
and $\text{Rel} :: (\text{pi} \times \text{pi}) \text{ set}$

assumes $\text{Id}: \text{Id} \subseteq \text{Rel}$

shows $<\nu x >(\tau.(P)) \rightsquigarrow[\text{Rel}] \tau.<\nu x >P$

using *assms*

by(*force simp add: simulation-def elim: tauCases resCasesF intro: Tau ResF*)

lemma *resTauRight*:

fixes $x :: \text{name}$
and $P :: \text{pi}$
and $\text{Rel} :: (\text{pi} \times \text{pi}) \text{ set}$

assumes $\text{Id}: \text{Id} \subseteq \text{Rel}$

shows $\tau.<\nu x >P \rightsquigarrow[\text{Rel}] <\nu x >(\tau.(P))$

using *assms*

by(*force simp add: simulation-def elim: tauCases resCasesF intro: Tau ResF*)

end

theory *Strong-Late-Bisim-SC*

imports *Strong-Late-Bisim-Pres Strong-Late-Sim-SC*

begin

lemma *nilBisim[dest]*:

```

fixes  $a :: name$ 
and  $b :: name$ 
and  $x :: name$ 
and  $P :: pi$ 

shows  $\tau.(P) \sim \mathbf{0} \implies False$ 
and  $a\langle x \rangle.P \sim \mathbf{0} \implies False$ 
and  $a\{b\}.P \sim \mathbf{0} \implies False$ 
and  $\mathbf{0} \sim \tau.(P) \implies False$ 
and  $\mathbf{0} \sim a\langle x \rangle.P \implies False$ 
and  $\mathbf{0} \sim a\{b\}.P \implies False$ 
by(auto dest: bisimE symmetric)

```

lemma *matchId*:

```

fixes  $a :: name$ 
and  $P :: pi$ 

```

shows $[a\curvearrowright a]P \sim P$

proof –

let $?X = \{([a\curvearrowright a]P, P), (P, [a\curvearrowright a]P)\}$

have $([a\curvearrowright a]P, P) \in ?X$ **by** *simp*

thus *?thesis*

by(*coinduct rule: bisimCoinduct*) (*auto intro: matchIdLeft matchIdRight reflexive*)

qed

lemma *matchNil*:

```

fixes  $a :: name$ 
and  $b :: name$ 

```

assumes $a \neq b$

shows $[a\curvearrowright b]P \sim \mathbf{0}$

proof –

let $?X = \{([a\curvearrowright b]P, \mathbf{0}), (\mathbf{0}, [a\curvearrowright b]P)\}$

have $([a\curvearrowright b]P, \mathbf{0}) \in ?X$ **by** *simp*

thus *?thesis using* $\langle a \neq b \rangle$

by(*coinduct rule: bisimCoinduct*) (*auto intro: matchNilLeft nilSimRight reflexive*)

qed

lemma *mismatchId*:

```

fixes  $a :: name$ 
and  $b :: name$ 
and  $P :: pi$ 

```

assumes $a \neq b$

```

shows  $[a \neq b]P \sim P$ 
proof -
  let  $?X = \{([a \neq b]P, P), (P, [a \neq b]P)\}$ 
  have  $([a \neq b]P, P) \in ?X$  by simp
  thus ?thesis using  $\langle a \neq b \rangle$ 
  by(coinduct rule: bisimCoinduct) (auto intro: mismatchIdLeft mismatchIdRight
reflexive)
qed

```

lemma *mismatchNil*:

```

fixes  $a :: name$ 
and  $P :: pi$ 

```

```

shows  $[a \neq a]P \sim \mathbf{0}$ 
proof -
  let  $?X = \{([a \neq a]P, \mathbf{0}), (\mathbf{0}, [a \neq a]P)\}$ 
  have  $([a \neq a]P, \mathbf{0}) \in ?X$  by simp
  thus ?thesis
  by(coinduct rule: bisimCoinduct) (auto intro: mismatchNilLeft nilSimRight
reflexive)
qed

```

lemma *nilRes*:

```

fixes  $x :: name$ 

```

```

shows  $\langle \nu x \rangle \mathbf{0} \sim \mathbf{0}$ 
proof -
  let  $?X = \{(\langle \nu x \rangle \mathbf{0}, \mathbf{0}), (\mathbf{0}, \langle \nu x \rangle \mathbf{0})\}$ 
  have  $(\langle \nu x \rangle \mathbf{0}, \mathbf{0}) \in ?X$  by simp
  thus ?thesis
  by(coinduct rule: bisimCoinduct) (auto intro: nilSimRight resNilRight)
qed

```

lemma *resComm*:

```

fixes  $x :: name$ 
and  $y :: name$ 
and  $P :: pi$ 

```

```

shows  $\langle \nu x \rangle \langle \nu y \rangle P \sim \langle \nu y \rangle \langle \nu x \rangle P$ 
proof -
  let  $?X = \{(\langle \nu x \rangle \langle \nu y \rangle P, \langle \nu y \rangle \langle \nu x \rangle P) \mid x \ y \ P. \ True\}$ 
  have  $(\langle \nu x \rangle \langle \nu y \rangle P, \langle \nu y \rangle \langle \nu x \rangle P) \in ?X$  by auto
  thus ?thesis
  proof(coinduct rule: bisimCoinduct)
    case(cSim xyP yxP)
    {
      fix  $x \ y \ P$ 

```

```

    have  $\bigwedge x y P. (\langle \nu x \rangle \langle \nu y \rangle P, \langle \nu y \rangle \langle \nu x \rangle P) \in ?X \cup \text{bisim}$  by auto
    moreover have  $\text{Id} \subseteq ?X \cup \text{bisim}$  by (auto intro: reflexive)
    moreover have eqvt  $?X$  by (force simp add: eqvt-def)
    hence eqvt  $(?X \cup \text{bisim})$  by auto
    ultimately have  $\langle \nu x \rangle \langle \nu y \rangle P \rightsquigarrow [(?X \cup \text{bisim})] \langle \nu y \rangle \langle \nu x \rangle P$  by (rule
resComm)
  }
  with  $\langle (xyP, yxP) \in ?X \rangle$  show ?case by auto
next
case (cSym  $xyP yxP$ )
thus ?case by auto
qed
qed

```

```

lemma sumSym:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 

  shows  $P \oplus Q \sim Q \oplus P$ 
proof -
  let  $?X = \{(P \oplus Q, Q \oplus P), (Q \oplus P, P \oplus Q)\}$ 
  have  $(P \oplus Q, Q \oplus P) \in ?X$  by simp
  thus ?thesis
  by (coinduct rule: bisimCoinduct) (auto intro: reflexive sumSym)
qed

```

```

lemma sumIdemp:
  fixes  $P :: pi$ 

  shows  $P \oplus P \sim P$ 
proof -
  let  $?X = \{(P \oplus P, P), (P, P \oplus P)\}$ 
  have  $(P \oplus P, P) \in ?X$  by simp
  thus ?thesis
  by (coinduct rule: bisimCoinduct) (auto intro: reflexive sumIdempLeft sumIdempRight)
qed

```

```

lemma sumAssoc:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $R :: pi$ 

  shows  $(P \oplus Q) \oplus R \sim P \oplus (Q \oplus R)$ 
proof -
  let  $?X = \{((P \oplus Q) \oplus R, P \oplus (Q \oplus R)), (P \oplus (Q \oplus R), (P \oplus Q) \oplus R)\}$ 
  have  $((P \oplus Q) \oplus R, P \oplus (Q \oplus R)) \in ?X$  by simp

```

thus *?thesis*
by(*coinduct rule: bisimCoinduct*) (*auto intro: reflexive sumAssocLeft sumAssocRight*)
qed

lemma *sumZero*:
fixes $P :: pi$

shows $P \oplus \mathbf{0} \sim P$

proof –

let $?X = \{(P \oplus \mathbf{0}, P), (P, P \oplus \mathbf{0})\}$

have $(P \oplus \mathbf{0}, P) \in ?X$ **by** *simp*

thus *?thesis*

by(*coinduct rule: bisimCoinduct*) (*auto intro: reflexive sumZeroLeft sumZeroRight*)
qed

lemma *parZero*:
fixes $P :: pi$

shows $P \parallel \mathbf{0} \sim P$

proof –

let $?X = \{(P \parallel \mathbf{0}, P) \mid P. True\} \cup \{(P, P \parallel \mathbf{0}) \mid P. True\}$

have $(P \parallel \mathbf{0}, P) \in ?X$ **by** *blast*

thus *?thesis*

by(*coinduct rule: bisimCoinduct, auto intro: parZeroRight parZeroLeft*)

qed

lemma *parSym*:
fixes $P :: pi$
and $Q :: pi$

shows $P \parallel Q \sim Q \parallel P$

proof –

let $?X = \{(resChain\ lst\ (P \parallel Q), resChain\ lst\ (Q \parallel P)) \mid lst\ P\ Q. True\}$

have $(P \parallel Q, Q \parallel P) \in ?X$ **by**(*blast intro: resChain.base[THEN sym]*)

thus *?thesis*

proof(*coinduct rule: bisimCoinduct*)

case(*cSim PQ QP*)

{

fix $lst\ P\ Q$

have $\bigwedge P\ Q. (P \parallel Q, Q \parallel P) \in ?X \cup bisim$ **by**(*blast intro: resChain.base[THEN sym]*)

moreover have $Res: \bigwedge x\ P\ Q. (P, Q) \in ?X \cup bisim \implies \langle \nu x \rangle P, \langle \nu x \rangle Q) \in ?X \cup bisim$

by(*auto intro: resPres resChain.step[THEN sym]*)

ultimately have $P \parallel Q \rightsquigarrow[(?X \cup bisim)] Q \parallel P$ **by**(*rule parSym*)

```

moreover have eqvt ?X by(force simp add: eqvt-def)
hence eqvt(?X ∪ bisim) by auto
ultimately have resChain lst (P ∥ Q) ∼[ (?X ∪ bisim)] resChain lst (Q ∥
P) using Res
by(rule resChainI)
}
with ⟨(PQ, QP) ∈ ?X⟩ show ?case by auto
next
case(cSym PQ QP)
thus ?case by auto
qed
qed

```

lemma scopeExtPar:

```

fixes P :: pi
and Q :: pi
and x :: name

```

```

assumes x # P

```

```

shows <νx>(P ∥ Q) ∼ P ∥ <νx>Q

```

proof –

```

let ?X = {(resChain lst (<νx>(P ∥ Q)), resChain lst (P ∥ <νx>Q)) | lst x P
Q. x # P} ∪
{(resChain lst (P ∥ <νx>Q), resChain lst (<νx>(P ∥ Q)) | lst x P Q.
x # P}

```

```

let ?Y = bisim O (?X ∪ bisim) O bisim

```

```

have Res: ∧P Q x. (P, Q) ∈ ?X ⇒ (<νx>P, <νx>Q) ∈ ?X by(blast intro:
resChain.step[THEN sym])

```

```

from ⟨x # P⟩ have (<νx>(P ∥ Q), P ∥ <νx>Q) ∈ ?X by(blast intro: resChain.base[THEN
sym])

```

```

moreover have EqvtX: eqvt ?X by(fastforce simp add: eqvt-def name-fresh-left
name-rev-per)

```

```

ultimately show ?thesis

```

```

proof(coinduct rule: bisimTransitiveCoinduct)

```

```

case(cSim P Q)

```

```

{

```

```

fix P Q lst x

```

```

assume (x::name) # (P::pi)

```

```

moreover have Id ⊆ ?Y by(blast intro: reflexive)

```

```

moreover from ⟨eqvt ?X⟩ bisimEqvt have eqvt ?Y by blast

```

```

moreover have ∧P Q x. x # P ⇒ (<νx>(P ∥ Q), P ∥ <νx>Q) ∈ ?Y
by(blast intro: resChain.base[THEN sym] reflexive)

```

```

moreover {

```

```

fix P Q x y

```

```

have <νx><νy>(P ∥ Q) ∼ <νy><νx>(P ∥ Q) by(rule resComm)

```

```

moreover assume x # P

```

```

hence ( $\langle \nu x \rangle (P \parallel Q), P \parallel \langle \nu x \rangle Q$ )  $\in ?X$  by(fastforce intro: resChain.base[THEN sym])
  hence ( $\langle \nu y \rangle \langle \nu x \rangle (P \parallel Q), \langle \nu y \rangle (P \parallel \langle \nu x \rangle Q)$ )  $\in ?X$  by(rule Res)
  ultimately have ( $\langle \nu x \rangle \langle \nu y \rangle (P \parallel Q), \langle \nu y \rangle (P \parallel \langle \nu x \rangle Q)$ )  $\in ?Y$  by(blast intro: reflexive)
}
ultimately have  $\langle \nu x \rangle (P \parallel Q) \rightsquigarrow[?Y] (P \parallel \langle \nu x \rangle Q)$  by(rule scopeExtParLeft)
moreover note  $\langle \text{eqvt } ?Y \rangle$ 
moreover from Res have  $\bigwedge P Q x. (P, Q) \in ?Y \implies (\langle \nu x \rangle P, \langle \nu x \rangle Q) \in ?Y$ 
  by(blast intro: resChain.step[THEN sym] dest: resPres)
  ultimately have resChain lst ( $\langle \nu x \rangle (P \parallel Q)$ )  $\rightsquigarrow[?Y]$  resChain lst ( $P \parallel \langle \nu x \rangle Q$ )
by(rule resChainI)
}
moreover {
  fix  $P Q lst x$ 
  assume ( $x::\text{name}$ )  $\# (P::\text{pi})$ 
  moreover have  $\text{Id} \subseteq ?Y$  by(blast intro: reflexive)
  moreover from  $\langle \text{eqvt } ?X \rangle$  bisimEqvt have  $\text{eqvt } ?Y$  by blast
  moreover have  $\bigwedge P Q x. x \# P \implies (P \parallel \langle \nu x \rangle Q, \langle \nu x \rangle (P \parallel Q)) \in ?Y$ 
    by(blast intro: resChain.base[THEN sym] reflexive)
  moreover {
    fix  $P Q x y$ 
    have  $\langle \nu y \rangle \langle \nu x \rangle (P \parallel Q) \sim \langle \nu x \rangle \langle \nu y \rangle (P \parallel Q)$  by(rule resComm)
    moreover assume  $x \# P$ 
  }
  hence ( $P \parallel \langle \nu x \rangle Q, \langle \nu x \rangle (P \parallel Q)$ )  $\in ?X$  by(fastforce intro: resChain.base[THEN sym])
  hence ( $\langle \nu y \rangle (P \parallel \langle \nu x \rangle Q), \langle \nu y \rangle \langle \nu x \rangle (P \parallel Q)$ )  $\in ?X$  by(rule Res)
  ultimately have ( $\langle \nu y \rangle (P \parallel \langle \nu x \rangle Q), \langle \nu x \rangle \langle \nu y \rangle (P \parallel Q)$ )  $\in ?Y$  by(blast intro: reflexive)
}
ultimately have ( $P \parallel \langle \nu x \rangle Q$ )  $\rightsquigarrow[?Y] \langle \nu x \rangle (P \parallel Q)$ 
  by(rule scopeExtParRight)
moreover note  $\langle \text{eqvt } ?Y \rangle$ 
moreover from Res have  $\bigwedge P Q x. (P, Q) \in ?Y \implies (\langle \nu x \rangle P, \langle \nu x \rangle Q) \in ?Y$ 
  by(blast intro: resChain.step[THEN sym] dest: resPres)
  ultimately have resChain lst ( $P \parallel \langle \nu x \rangle Q$ )  $\rightsquigarrow[?Y]$  resChain lst ( $\langle \nu x \rangle (P \parallel Q)$ )
by(rule resChainI)
}
ultimately show  $?case$  using  $\langle (P, Q) \in ?X \rangle$  by auto
next
case(cSym P Q)
thus  $?case$ 
  by auto (blast dest: symmetric transitive intro: resChain.base[THEN sym] reflexive)+

```

```

qed
qed

lemma scopeExtPar':
  fixes P :: pi
  and Q :: pi
  and x :: name

  assumes xFreshQ: x # Q

  shows <νx>(P || Q) ~ (<νx>P) || Q
proof -
  have <νx>(P || Q) ~ <νx>(Q || P)
  proof -
    have P || Q ~ Q || P by(rule parSym)
    thus ?thesis by(rule resPres)
  qed
  moreover from xFreshQ have <νx>(Q || P) ~ Q || (<νx>P) by(rule scope-
  ExtPar)
  moreover have Q || <νx>P ~ (<νx>P) || Q by(rule parSym)
  ultimately show ?thesis by(blast intro: transitive)
qed

lemma parAssoc:
  fixes P :: pi
  and Q :: pi
  and R :: pi

  shows (P || Q) || R ~ P || (Q || R)
proof -
  let ?X = {(resChain lst ((P || Q) || R), resChain lst (P || (Q || R))) | lst P Q R.
  True}
  let ?Y = bisim O (?X ∪ bisim) O bisim

  have ResX: ∧P Q x. (P, Q) ∈ ?X ⇒ (<νx>P, <νx>Q) ∈ ?X
    by(blast intro: resChain.step[symmetric])
  hence ResY: ∧P Q x. (P, Q) ∈ ?Y ⇒ (<νx>P, <νx>Q) ∈ ?Y
    by(blast intro: resChain.step[symmetric] dest: resPres)

  have ((P || Q) || R, P || (Q || R)) ∈ ?X by(blast intro: resChain.base[symmetric])
  moreover have eqvt ?X by(fastforce simp add: eqvt-def)
  ultimately show ?thesis
  proof(coinduct rule: bisimTransitiveCoinduct)
    case(cSim P Q)
    {
      fix P Q R lst
      have ∧P Q R. ((P || Q) || R, P || (Q || R)) ∈ ?Y by(blast intro: reflexive
      resChain.base[symmetric])
      moreover have ∧P Q x. (P, Q) ∈ ?Y ⇒ (<νx>P, <νx>Q) ∈ ?Y by(blast

```

```

intro: resChain.step[symmetric] resPres)
  moreover {
    fix P Q R x
    have (<νx>((P || Q) || R), <νx>(P || (Q || R))) ∈ ?X by(rule-tac ResX)
    (blast intro: resChain.base[symmetric])
    moreover assume x ‡ P
    hence <νx>(P || (Q || R)) ~ P || <νx>(Q || R) by(rule scopeExtPar)
    ultimately have (<νx>((P || Q) || R), P || <νx>(Q || R)) ∈ ?Y by(blast
intro: reflexive)
  }
  moreover {
    fix P Q R x
    have (<νx>((P || Q) || R), <νx>(P || (Q || R))) ∈ ?X by(rule-tac ResX)
    (blast intro: resChain.base[symmetric])
    moreover assume x ‡ R
    hence <νx>(P || Q) || R ~ <νx>((P || Q) || R) by(metis scopeExtPar'
symmetric)
    ultimately have (<νx>(P || Q) || R, <νx>(P || (Q || R))) ∈ ?Y by(blast
intro: reflexive)
  }
  ultimately have (P || Q) || R ~[?Y] P || (Q || R) by(rule parAssocLeft)
  moreover from <eqvt ?X> bisimEqvt have eqvt ?Y by blast
  ultimately have resChain.lst ((P || Q) || R) ~[?Y] resChain.lst (P || (Q ||
R)) using ResY
    by(rule resChainI)
  }
  with <(P, Q) ∈ ?X> show ?case by auto
next
case(cSym P Q)
{
  fix P Q R lst
  have P || (Q || R) ~ (R || Q) || P by(metis parPres parSym transitive)
  moreover have ((R || Q) || P, R || (Q || P)) ∈ ?X by(blast intro: resChain.base[symmetric])
  moreover have R || (Q || P) ~ (P || Q) || R by(metis parPres parSym
transitive)
  ultimately have (P || (Q || R), (P || Q) || R) ∈ ?Y by blast
  hence (resChain.lst (P || (Q || R)), resChain.lst ((P || Q) || R)) ∈ ?Y using
ResY
    by(induct lst) auto
  }
  with <(P, Q) ∈ ?X> show ?case by blast
qed
qed

```

lemma *scopeFresh*:

fixes $x :: \text{name}$

and $P :: \text{pi}$

assumes $x \# P$

shows $\langle \nu x \rangle P \sim P$
proof –
have $\langle \nu x \rangle P \sim \langle \nu x \rangle P \parallel \mathbf{0}$ **by**(rule parZero[THEN symmetric])

moreover have $\langle \nu x \rangle P \parallel \mathbf{0} \sim \mathbf{0} \parallel \langle \nu x \rangle P$ **by**(rule parSym)
moreover have $\mathbf{0} \parallel \langle \nu x \rangle P \sim \langle \nu x \rangle (\mathbf{0} \parallel P)$ **by**(rule scopeExtPar[THEN symmetric]) *auto*
moreover have $\langle \nu x \rangle (\mathbf{0} \parallel P) \sim \langle \nu x \rangle (P \parallel \mathbf{0})$ **by**(rule resPres[OF parSym])
moreover from $\langle x \# P \rangle$ **have** $\langle \nu x \rangle (P \parallel \mathbf{0}) \sim P \parallel \langle \nu x \rangle \mathbf{0}$ **by**(rule scopeExtPar)
moreover have $P \parallel \langle \nu x \rangle \mathbf{0} \sim \langle \nu x \rangle \mathbf{0} \parallel P$ **by**(rule parSym)
moreover have $\langle \nu x \rangle \mathbf{0} \parallel P \sim \mathbf{0} \parallel P$ **by**(rule parPres[OF nilRes])
moreover have $\mathbf{0} \parallel P \sim P \parallel \mathbf{0}$ **by**(rule parSym)
moreover have $P \parallel \mathbf{0} \sim P$ **by**(rule parZero)
ultimately show *?thesis* **by**(metis transitive)
qed

lemma *sumRes*:
fixes $x :: \text{name}$
and $P :: \text{pi}$
and $Q :: \text{pi}$

shows $\langle \nu x \rangle (P \oplus Q) \sim (\langle \nu x \rangle P) \oplus (\langle \nu x \rangle Q)$
proof –
let $?X = \{(\langle \nu x \rangle (P \oplus Q), \langle \nu x \rangle P \oplus \langle \nu x \rangle Q) \mid x P Q. \text{True}\} \cup$
 $\{(\langle \nu x \rangle P \oplus \langle \nu x \rangle Q, \langle \nu x \rangle (P \oplus Q)) \mid x P Q. \text{True}\}$
have $(\langle \nu x \rangle (P \oplus Q), \langle \nu x \rangle P \oplus \langle \nu x \rangle Q) \in ?X$ **by** *auto*
moreover have *eqvt* $?X$ **by**(fastforce simp add: eqvt-def)
ultimately show *?thesis*
by(coinduct rule: bisimCoinduct) (fastforce intro: sumResLeft sumResRight reflexive)+
qed

lemma *scopeExtSum*:
fixes $P :: \text{pi}$
and $Q :: \text{pi}$
and $x :: \text{name}$

assumes $x \# P$

shows $\langle \nu x \rangle (P \oplus Q) \sim P \oplus \langle \nu x \rangle Q$
proof –
have $\langle \nu x \rangle (P \oplus Q) \sim \langle \nu x \rangle P \oplus \langle \nu x \rangle Q$ **by**(rule sumRes)
moreover from $\langle x \# P \rangle$ **have** $\langle \nu x \rangle P \oplus \langle \nu x \rangle Q \sim P \oplus \langle \nu x \rangle Q$
by(rule sumPres[OF scopeFresh])
ultimately show *?thesis* **by**(rule transitive)
qed

```

lemma bangSC:
  fixes  $P :: pi$ 

  shows  $!P \sim P \parallel !P$ 
proof –
  let  $?X = \{(!P, P \parallel !P), (P \parallel !P, !P)\}$ 
  have  $(!P, P \parallel !P) \in ?X$  by simp
  thus ?thesis
  by(coinduct rule: bisimCoinduct) (auto intro: bangLeftSC bangRightSC reflexive)
qed

```

```

lemma resNil:
  fixes  $x :: name$ 
  and  $y :: name$ 
  and  $P :: pi$ 
  and  $b :: name$ 

  shows  $\langle \nu x \rangle x \langle y \rangle . P \sim \mathbf{0}$ 
  and  $\langle \nu x \rangle x \{b\} . P \sim \mathbf{0}$ 
proof –
  let  $?X = \{(\langle \nu x \rangle x \langle y \rangle . P, \mathbf{0}), (\mathbf{0}, \langle \nu x \rangle x \langle y \rangle . P)\}$ 
  have  $(\langle \nu x \rangle x \langle y \rangle . P, \mathbf{0}) \in ?X$  by simp
  thus  $\langle \nu x \rangle x \langle y \rangle . P \sim \mathbf{0}$ 
  by(coinduct rule: bisimCoinduct) (auto simp add: simulation-def)
next
  let  $?X = \{(\langle \nu x \rangle x \{b\} . P, \mathbf{0}), (\mathbf{0}, \langle \nu x \rangle x \{b\} . P)\}$ 
  have  $(\langle \nu x \rangle x \{b\} . P, \mathbf{0}) \in ?X$  by simp
  thus  $\langle \nu x \rangle x \{b\} . P \sim \mathbf{0}$ 
  by(coinduct rule: bisimCoinduct) (auto simp add: simulation-def)
qed

```

```

lemma resInput:
  fixes  $x :: name$ 
  and  $a :: name$ 
  and  $y :: name$ 
  and  $P :: pi$ 

  assumes  $x \neq a$ 
  and  $x \neq y$ 

  shows  $\langle \nu x \rangle a \langle y \rangle . P \sim a \langle y \rangle . (\langle \nu x \rangle P)$ 
proof –
  let  $?X = \{(\langle \nu x \rangle a \langle y \rangle . P, a \langle y \rangle . (\langle \nu x \rangle P)) \mid x \ a \ y \ P. \ x \neq a \wedge x \neq y\} \cup$ 
     $\{(a \langle y \rangle . (\langle \nu x \rangle P), \langle \nu x \rangle a \langle y \rangle . P) \mid x \ a \ y \ P. \ x \neq a \wedge x \neq y\}$ 
  from assms have  $(\langle \nu x \rangle a \langle y \rangle . P, a \langle y \rangle . (\langle \nu x \rangle P)) \in ?X$  by auto
  moreover have eqvt  $?X$  by(fastforce simp add: eqvt-def pt-bij[OF pt-name-inst,
OF at-name-inst])
  ultimately show ?thesis
  by(coinduct rule: bisimCoinduct) (fastforce intro: resInputLeft reflexive resIn-

```

putRight)+
qed

lemma *resOutput*:

fixes $x :: \text{name}$
and $a :: \text{name}$
and $b :: \text{name}$
and $P :: \text{pi}$

assumes $x \neq a$
and $x \neq b$

shows $\langle \nu x \rangle a\{b\}.P \sim a\{b\}.(\langle \nu x \rangle P)$

proof –

let $?X = \{(\langle \nu x \rangle a\{b\}.P, a\{b\}.(\langle \nu x \rangle P)) \mid x \ a \ b \ P. \ x \neq a \wedge x \neq b\} \cup$
 $\{a\{b\}.(\langle \nu x \rangle P), \langle \nu x \rangle a\{b\}.P \mid x \ a \ b \ P. \ x \neq a \wedge x \neq b\}$

from *assms* **have** $(\langle \nu x \rangle a\{b\}.P, a\{b\}.(\langle \nu x \rangle P)) \in ?X$ **by** *blast*

moreover **have** *eqvt* $?X$ **by** (*fastforce simp add: eqvt-def pt-bij[OF pt-name-inst, OF at-name-inst]*)

ultimately show *?thesis*

by (*coinduct rule: bisimCoinduct*) (*fastforce intro: resOutputLeft resOutputRight reflexive*)+

qed

lemma *resTau*:

fixes $x :: \text{name}$
and $P :: \text{pi}$

shows $\langle \nu x \rangle \tau.(P) \sim \tau.(\langle \nu x \rangle P)$

proof –

let $?X = \{(\langle \nu x \rangle \tau.(P), \tau.(\langle \nu x \rangle P)), (\tau.(\langle \nu x \rangle P), \langle \nu x \rangle \tau.(P))\}$

have $(\langle \nu x \rangle \tau.(P), \tau.(\langle \nu x \rangle P)) \in ?X$ **by** *auto*

thus *?thesis*

by (*coinduct rule: bisimCoinduct*) (*fastforce intro: resTauLeft resTauRight reflexive*)+

qed

inductive *structCong* :: $\text{pi} \Rightarrow \text{pi} \Rightarrow \text{bool}$ ($\langle _ \equiv_s _ \rangle [70, 70] 70$)

where

Refl: $P \equiv_s P$

| *Sym*: $P \equiv_s Q \Longrightarrow Q \equiv_s P$

| *Trans*: $\llbracket P \equiv_s Q; Q \equiv_s R \rrbracket \Longrightarrow P \equiv_s R$

| *SumComm*: $P \oplus Q \equiv_s Q \oplus P$

| *SumAssoc*: $(P \oplus Q) \oplus R \equiv_s P \oplus (Q \oplus R)$

| *SumId*: $P \oplus \mathbf{0} \equiv_s P$

| *ParComm*: $P \parallel Q \equiv_s Q \parallel P$

| *ParAssoc*: $(P \parallel Q) \parallel R \equiv_s P \parallel (Q \parallel R)$

| *ParId*: $P \parallel \mathbf{0} \equiv_s P$

| *MatchId*: $[a \frown a]P \equiv_s P$

| *ResNil*: $\langle \nu x \rangle \mathbf{0} \equiv_s \mathbf{0}$

| *ResComm*: $\langle \nu x \rangle \langle \nu y \rangle P \equiv_s \langle \nu y \rangle \langle \nu x \rangle P$

| *ResSum*: $\langle \nu x \rangle (P \oplus Q) \equiv_s \langle \nu x \rangle P \oplus \langle \nu x \rangle Q$

| *ScopeExtPar*: $x \# P \implies \langle \nu x \rangle (P \parallel Q) \equiv_s P \parallel \langle \nu x \rangle Q$

| *InputRes*: $\llbracket x \neq a; x \neq y \rrbracket \implies \langle \nu x \rangle a \langle y \rangle . P \equiv_s a \langle y \rangle . (\langle \nu x \rangle P)$

| *OutputRes*: $\llbracket x \neq a; x \neq b \rrbracket \implies \langle \nu x \rangle a \{b\} . P \equiv_s a \{b\} . (\langle \nu x \rangle P)$

| *TauRes*: $\langle \nu x \rangle \tau . (P) \equiv_s \tau . (\langle \nu x \rangle P)$

| *BangUnfold*: $!P \equiv_s P \parallel !P$

lemma *structCongBisim*:

fixes $P :: pi$

and $Q :: pi$

assumes $P \equiv_s Q$

shows $P \sim Q$

using *assms*

by(*induct rule: structCong.induct*)

(*auto intro: reflexive symmetric transitive sumSym sumAssoc sumZero parSym parAssoc parZero*

nilRes resComm resInput resOutput resTau sumRes scopeExtPar bangSC matchId mismatchId)

end

theory *Strong-Late-Bisim-Subst-SC*

imports *Strong-Late-Bisim-Subst-Pres Strong-Late-Bisim-SC*

begin

lemma *matchId*:

fixes $a :: name$

and $P :: pi$

shows $[a \frown a]P \sim^s P$

by(*auto simp add: substClosed-def intro: Strong-Late-Bisim-SC.matchId*)

lemma *mismatchNil*:

fixes $a :: name$

and $P :: pi$

shows $[a \neq a]P \sim^s \mathbf{0}$

by(*auto simp add: substClosed-def intro: Strong-Late-Bisim-SC.mismatchNil*)

lemma *scopeFresh*:

```

fixes  $P :: pi$ 
and  $x :: name$ 

assumes  $xFreshP: x \# P$ 

shows  $\langle \nu x \rangle P \sim^s P$ 
proof(auto simp add: substClosed-def)
  fix  $s :: (name \times name) list$ 

  have  $\exists c::name. c \# (P, s)$  by(blast intro: name-exists-fresh)
  then obtain  $c::name$  where  $cFreshP: c \# P$  and  $cFreshs: c \# s$  by(force simp add: fresh-prod)

  have  $\langle \nu x \rangle P = \langle \nu c \rangle P$ 
  proof –
    from  $cFreshP$  have  $\langle \nu x \rangle P = \langle \nu c \rangle ([ (x, c) ] \cdot P)$  by(simp add: alphaRes)
    with  $cFreshP$   $xFreshP$  show ?thesis by(simp add: name-fresh-fresh)
  qed

  with  $cFreshP$   $cFreshs$  show  $(\langle \nu x \rangle P)[\langle s \rangle] \sim P[\langle s \rangle]$ 
  by(force intro: Strong-Late-Bisim-SC.scopeFresh)
qed

lemma resComm:
  fixes  $P :: pi$ 
  and  $x :: name$ 
  and  $y :: name$ 

  shows  $\langle \nu x \rangle \langle \nu y \rangle P \sim^s \langle \nu y \rangle \langle \nu x \rangle P$ 
proof(cases x=y)
  assume  $x=y$ 
  have  $P \sim^s P$  by(rule Strong-Late-Bisim-Subst.reflexive)
  hence  $\langle \nu x \rangle P \sim^s \langle \nu x \rangle P$  by(rule resPres)
  hence  $\langle \nu x \rangle \langle \nu x \rangle P \sim^s \langle \nu x \rangle \langle \nu x \rangle P$  by(rule resPres)
  with  $x=y$  show ?thesis by simp
next
  assume  $x \neq y$ 
  show ?thesis
  proof(auto simp add: substClosed-def)
    fix  $s::(name \times name) list$ 

    have  $\exists c::name. c \# (P, s, y)$  by(blast intro: name-exists-fresh)
    then obtain  $c::name$  where  $cFreshP: c \# P$  and  $cFreshs: c \# s$  and  $cineqy: c \neq y$ 
    by(force simp add: fresh-prod)

    have  $\exists d::name. d \# (P, s, c, x, y)$  by (blast intro: name-exists-fresh)
    then obtain  $d::name$  where  $dFreshP: d \# P$  and  $dFreshs: d \# s$  and  $dineqc: d \neq c$ 

```

and *dineqx*: $d \neq x$ **and** *dineqy*: $d \neq y$
by(*force simp add: fresh-prod*)

have $\langle \nu x \rangle \langle \nu y \rangle P = \langle \nu c \rangle \langle \nu d \rangle ([x, c] \cdot [(y, d)] \cdot P)$
proof –
from *cineqy cFreshP* **have** *cFreshyP*: $c \# \langle \nu y \rangle P$ **by**(*simp add: name-fresh-abs*)
from *dFreshP* **have** $\langle \nu y \rangle P = \langle \nu d \rangle ([y, d] \cdot P)$ **by**(*rule alphaRes*)
moreover from *cFreshyP* **have** $\langle \nu x \rangle \langle \nu y \rangle P = \langle \nu c \rangle ([x, c] \cdot (\langle \nu y \rangle P))$
by(*rule alphaRes*)
ultimately show *?thesis* **using** *dineqc dineqx* **by**(*simp add: name-calc*)
qed
moreover have $\langle \nu y \rangle \langle \nu x \rangle P = \langle \nu d \rangle \langle \nu c \rangle ([x, c] \cdot [(y, d)] \cdot P)$
proof –
from *dineqx dFreshP* **have** *dFreshxP*: $d \# \langle \nu x \rangle P$ **by**(*simp add: name-fresh-abs*)
from *cFreshP* **have** $\langle \nu x \rangle P = \langle \nu c \rangle ([x, c] \cdot P)$ **by**(*rule alphaRes*)
moreover from *dFreshxP* **have** $\langle \nu y \rangle \langle \nu x \rangle P = \langle \nu d \rangle ([y, d] \cdot (\langle \nu x \rangle P))$
by(*rule alphaRes*)
ultimately have $\langle \nu y \rangle \langle \nu x \rangle P = \langle \nu d \rangle \langle \nu c \rangle ([y, d] \cdot [(x, c)] \cdot P)$ **using**
dineqc cineqy
by(*simp add: name-calc*)
thus *?thesis* **using** *dineqx dineqc cineqy xineqy*
by(*subst name-perm-compose, simp add: name-calc*)
qed

ultimately show $(\langle \nu x \rangle \langle \nu y \rangle P)[\langle s \rangle] \sim (\langle \nu y \rangle \langle \nu x \rangle P)[\langle s \rangle]$ **using** *cFreshs*
dFreshs
by(*force intro: Strong-Late-Bisim-SC.resComm*)
qed
qed

lemma *sumZero*:
fixes $P :: pi$

shows $P \oplus \mathbf{0} \sim^s P$
by(*force simp add: substClosed-def intro: Strong-Late-Bisim-SC.sumZero*)

lemma *sumSym*:
fixes $P :: pi$
and $Q :: pi$

shows $P \oplus Q \sim^s Q \oplus P$
by(*force simp add: substClosed-def intro: Strong-Late-Bisim-SC.sumSym*)

lemma *sumAssoc*:
fixes $P :: pi$
and $Q :: pi$
and $R :: pi$

shows $(P \oplus Q) \oplus R \sim^s P \oplus (Q \oplus R)$

by(force simp add: substClosed-def intro: Strong-Late-Bisim-SC.sumAssoc)

lemma sumRes:

fixes $P :: pi$

and $Q :: pi$

and $x :: name$

shows $\langle \nu x \rangle (P \oplus Q) \sim^s \langle \nu x \rangle P \oplus \langle \nu x \rangle Q$

proof(auto simp add: substClosed-def)

fix $s :: (name \times name) list$

have $\exists c :: name. c \# (P, Q, s)$ **by**(blast intro: name-exists-fresh)

then obtain $c :: name$ **where** $cFreshP: c \# P$ **and** $cFreshQ: c \# Q$ **and** $cFreshs:$
 $c \# s$

by(force simp add: fresh-prod)

have $\langle \nu x \rangle (P \oplus Q) = \langle \nu c \rangle ([(x, c)] \cdot P \oplus [(x, c)] \cdot Q)$

proof –

from $cFreshP$ $cFreshQ$ **have** $c \# P \oplus Q$ **by** simp

hence $\langle \nu x \rangle (P \oplus Q) = \langle \nu c \rangle ([(x, c)] \cdot (P \oplus Q))$ **by**(simp add: alphaRes)

thus $?thesis$ **by**(simp add: name-fresh-fresh)

qed

moreover from $cFreshP$ **have** $\langle \nu x \rangle P = \langle \nu c \rangle ([(x, c)] \cdot P)$ **by**(simp add:
 $alphaRes$)

moreover from $cFreshQ$ **have** $\langle \nu x \rangle Q = \langle \nu c \rangle ([(x, c)] \cdot Q)$ **by**(simp add:
 $alphaRes$)

ultimately show $(\langle \nu x \rangle (P \oplus Q))[\langle s \rangle] \sim (\langle \nu x \rangle P)[\langle s \rangle] \oplus (\langle \nu x \rangle Q)[\langle s \rangle]$
using $cFreshs$

by(force intro: Strong-Late-Bisim-SC.sumRes)

qed

lemma scopeExtSum:

fixes $P :: pi$

and $Q :: pi$

and $x :: name$

assumes $xFreshP: x \# P$

shows $\langle \nu x \rangle (P \oplus Q) \sim^s P \oplus \langle \nu x \rangle Q$

proof(auto simp add: substClosed-def)

fix $s :: (name \times name) list$

have $\exists c :: name. c \# (P, Q, s)$ **by**(blast intro: name-exists-fresh)

then obtain $c :: name$ **where** $cFreshP: c \# P$ **and** $cFreshQ: c \# Q$ **and** $cFreshs:$
 $c \# s$

by(force simp add: fresh-prod)

have $\langle \nu x \rangle (P \oplus Q) = \langle \nu c \rangle (P \oplus ((x, c)] \cdot Q))$
proof –
from $cFreshP$ $cFreshQ$ **have** $c \# P \oplus Q$ **by** *simp*
hence $\langle \nu x \rangle (P \oplus Q) = \langle \nu c \rangle ((x, c)] \cdot (P \oplus Q))$ **by** (*simp add: alphaRes*)
with $xFreshP$ $cFreshP$ **show** *?thesis* **by** (*simp add: name-fresh-fresh*)
qed

moreover from $cFreshQ$ **have** $\langle \nu x \rangle Q = \langle \nu c \rangle ((x, c)] \cdot Q)$ **by** (*simp add: alphaRes*)

ultimately show $(\langle \nu x \rangle (P \oplus Q))[\langle s \rangle] \sim P[\langle s \rangle] \oplus (\langle \nu x \rangle Q)[\langle s \rangle]$ **using**
 $cFreshs$ $cFreshP$
by (*force intro: Strong-Late-Bisim-SC.scopeExtSum*)
qed

lemma *parZero*:
fixes $P :: pi$

shows $P \parallel \mathbf{0} \sim^s P$
by (*force simp add: substClosed-def intro: Strong-Late-Bisim-SC.parZero*)

lemma *parSym*:
fixes $P :: pi$
and $Q :: pi$

shows $P \parallel Q \sim^s Q \parallel P$
by (*force simp add: substClosed-def intro: Strong-Late-Bisim-SC.parSym*)

lemma *parAssoc*:
fixes $P :: pi$
and $Q :: pi$
and $R :: pi$

shows $(P \parallel Q) \parallel R \sim^s P \parallel (Q \parallel R)$
by (*force simp add: substClosed-def intro: Strong-Late-Bisim-SC.parAssoc*)

lemma *scopeExtPar*:
fixes $P :: pi$
and $Q :: pi$
and $x :: name$

assumes $xFreshP: x \# P$

shows $\langle \nu x \rangle (P \parallel Q) \sim^s P \parallel \langle \nu x \rangle Q$
proof (*auto simp add: substClosed-def*)
fix $s :: (name \times name)$ *list*

have $\exists c :: name. c \# (P, Q, s)$ **by** (*blast intro: name-exists-fresh*)
then obtain $c :: name$ **where** $cFreshP: c \# P$ **and** $cFreshQ: c \# Q$ **and** $cFreshs:$

$c \# s$
by(*force simp add: fresh-prod*)

have $\langle \nu x \rangle (P \parallel Q) = \langle \nu c \rangle (P \parallel ((x, c) \cdot Q))$
proof –
from *cFreshP cFreshQ* **have** $c \# P \parallel Q$ **by** *simp*
hence $\langle \nu x \rangle (P \parallel Q) = \langle \nu c \rangle ((x, c) \cdot (P \parallel Q))$ **by**(*simp add: alphaRes*)
with *xFreshP cFreshP* **show** *?thesis* **by**(*simp add: name-fresh-fresh*)
qed

moreover from *cFreshQ* **have** $\langle \nu x \rangle Q = \langle \nu c \rangle ((x, c) \cdot Q)$ **by**(*simp add: alphaRes*)

ultimately show $(\langle \nu x \rangle (P \parallel Q))[\langle s \rangle] \sim P[\langle s \rangle] \parallel (\langle \nu x \rangle Q)[\langle s \rangle]$ **using**
cFreshs cFreshP
by(*force intro: Strong-Late-Bisim-SC.scopeExtPar*)
qed

lemma *scopeExtPar'*:
fixes $P :: pi$
and $Q :: pi$
and $x :: name$

assumes *xFreshP*: $x \# Q$

shows $\langle \nu x \rangle (P \parallel Q) \sim^s (\langle \nu x \rangle P) \parallel Q$
proof –
have $\langle \nu x \rangle (P \parallel Q) \sim^s \langle \nu x \rangle (Q \parallel P)$ **by**(*blast intro: parSym resPres*)
moreover from *xFreshP* **have** $\langle \nu x \rangle (Q \parallel P) \sim^s Q \parallel \langle \nu x \rangle P$ **by**(*rule scope-ExtPar*)
moreover have $Q \parallel \langle \nu x \rangle P \sim^s (\langle \nu x \rangle P) \parallel Q$ **by**(*rule parSym*)
ultimately show *?thesis* **by** (*blast intro: transitive*)
qed

lemma *bangSC*:
fixes $P :: pi$

shows $!P \sim^s P \parallel !P$
by(*auto simp add: substClosed-def intro: Strong-Late-Bisim-SC.bangSC*)

lemma *nilRes*:
fixes $x :: name$

shows $\langle \nu x \rangle \mathbf{0} \sim^s \mathbf{0}$
proof(*auto simp add: substClosed-def*)
fix $\sigma :: (name \times name) list$
obtain $y :: name$ **where** $y \# \sigma$
by(*generate-fresh name*) *auto*
have $\langle \nu y \rangle \mathbf{0} \sim \mathbf{0}$ **by** (*rule Strong-Late-Bisim-SC.nilRes*)

```

with ⟨y # σ⟩ have (<νy>0)[<σ>] ~ 0 by simp
thus (<νx>0)[<σ>] ~ 0
  by(subst alphaRes[where c=y]) auto
qed

```

lemma *resTau*:

```

fixes x :: name
and P :: pi

```

```

shows <νx>(τ.(P)) ~s τ.<νx>P
proof(auto simp add: substClosed-def)
  fix σ::(name × name) list
  obtain y::name where y # P and y # σ
    by(generate-fresh name, auto)
  have <νy>(τ.([(x, y)] · P)[<σ>]) ~ τ.<νy>([(x, y)] · P)[<σ>]
    by(rule resTau)
  with ⟨y # σ⟩ have (<νy>(τ.([(x, y)] · P)))[<σ>] ~ (τ.<νy>([(x, y)] · P)))[<σ>]
    by simp
  with ⟨y # P⟩ show (<νx>τ.(P)))[<σ>] ~ τ.<νx>P)[<σ>]
    apply(subst alphaRes[where c=y])
    apply simp
    apply(subst alphaRes[where c=y and a=x])
    by simp+
qed

```

lemma *resOutput*:

```

fixes x :: name
and a :: name
and b :: name
and P :: pi

```

```

assumes x ≠ a
and x ≠ b

```

```

shows <νx>(a{b}.(P)) ~s a{b}.<νx>P
proof(auto simp add: substClosed-def)
  fix σ::(name × name) list
  obtain y::name where y # P and y # σ and y ≠ a and y ≠ b
    by(generate-fresh name, auto)
  have <νy>((seq-subst-name a σ){seq-subst-name b σ}([(x, y)] · P)[<σ>]) ~
seq-subst-name a σ{seq-subst-name b σ}.<νy>([(x, y)] · P)[<σ>]
    using ⟨y ≠ a⟩ ⟨y ≠ b⟩ ⟨y # σ⟩ freshSeqSubstName
    by(rule-tac resOutput) auto
  with ⟨y # σ⟩ have (<νy>(a{b}([(x, y)] · P)))[<σ>] ~ (a{b}.<νy>([(x, y)] ·
P)))[<σ>]
    by simp
  with ⟨y # P⟩ ⟨y ≠ a⟩ ⟨y ≠ b⟩ ⟨x ≠ a⟩ ⟨x ≠ b⟩ show (<νx>a{b}.(P)))[<σ>] ~
seq-subst-name a σ{seq-subst-name b σ}.<νx>P)[<σ>]
    apply(subst alphaRes[where c=y])

```

```

  apply simp
  apply(subst alphaRes[where c=y and a=x])
  by simp+
qed

```

lemma *resInput*:

```

  fixes x :: name
  and a :: name
  and b :: name
  and P :: pi

```

```

  assumes x ≠ a
  and x ≠ y

```

```

  shows <νx>(a<y>.(P)) ~s a<y>.<νx>P
proof(auto simp add: substClosed-def)
  fix σ::(name × name) list
  obtain x'::name where x' # P and x' # σ and x' ≠ a and x' ≠ x and x' ≠ y
    by(generate-fresh name, auto)
  obtain y'::name where y' # P and y' # σ and y' ≠ a and y' ≠ x and y' ≠ y
  and x' ≠ y'
    by(generate-fresh name, auto)
  have <νx'>((seq-subst-name a σ)<y'>(((y, y') · [(x, x')] · P)[<σ>])) ~
seq-subst-name a σ<y'>.<νx'>(((y, y') · [(x, x')] · P)[<σ>]))
    using ⟨x' ≠ a⟩ ⟨x' ≠ y'⟩ ⟨x' # σ⟩ ⟨y' # σ⟩ freshSeqSubstName
    by(rule-tac resInput) auto
  with ⟨x' # σ⟩ ⟨y' # σ⟩ have (<νx'>(a<y'>(((y, y') · [(x, x')] · P)))[<σ>] ~
(a<y'>.<νx'>([(y, y')] · [(x, x')] · P)))[<σ>]
    by simp
  with ⟨x' # P⟩ ⟨y' ≠ x⟩ ⟨x' ≠ y⟩ ⟨y' # P⟩ ⟨x' ≠ y'⟩ ⟨x' ≠ a⟩ ⟨y' ≠ a⟩ ⟨x ≠ a⟩ ⟨x
≠ y⟩ show (<νx>a<y>.(P)) [<σ>] ~ a<y>.<νx>P [<σ>]
    apply(subst alphaInput[where c=y'])
    apply simp
    apply(subst alphaRes[where c=x'])
    apply(simp add: abs-fresh fresh-left calc-atm)
    apply(simp add: eqvts calc-atm)
    apply(subst alphaRes[where c=x' and a=x])
    apply simp
    apply(subst alphaInput[where c=y' and x=y])
    apply(simp add: abs-fresh fresh-left calc-atm)
    apply(simp add: eqvts calc-atm)
    apply(subst perm-compose)
    by(simp add: eqvts calc-atm)
qed

```

lemma *bisimSubstStructCong*:

```

  fixes P :: pi
  and Q :: pi

```

```

assumes  $P \equiv_s Q$ 
shows  $P \sim^s Q$ 

using assms
apply(induct rule: structCong.induct)
by(auto intro: reflexive symmetric transitive sumSym sumAssoc sumZero parSym
parAssoc parZero
nilRes resComm resInput resOutput resTau sumRes scopeExtPar bangSC
matchId mismatchId)

end

theory Weak-Late-Cong-Subst-SC
imports Weak-Late-Cong-Subst Strong-Late-Bisim-Subst-SC
begin

lemma resComm:
  fixes  $P :: pi$ 

  shows  $\langle \nu a \rangle \langle \nu b \rangle P \simeq^s \langle \nu b \rangle \langle \nu a \rangle P$ 
proof –
  have  $\langle \nu a \rangle \langle \nu b \rangle P \sim^s \langle \nu b \rangle \langle \nu a \rangle P$ 
    by(rule Strong-Late-Bisim-Subst-SC.resComm)
  thus ?thesis by(rule strongEqWeakCong)
qed

lemma matchId:
  fixes  $a :: name$ 
  and  $P :: pi$ 

  shows  $[a \frown a]P \simeq^s P$ 
proof –
  have  $[a \frown a]P \sim^s P$  by(rule Strong-Late-Bisim-Subst-SC.matchId)
  thus ?thesis by(rule strongEqWeakCong)
qed

lemma matchNil:
  fixes  $a :: name$ 
  and  $P :: pi$ 

```

```

  shows  $[a \neq a]P \simeq^s \mathbf{0}$ 
proof -
  have  $[a \neq a]P \sim^s \mathbf{0}$  by(rule Strong-Late-Bisim-Subst-SC.mismatchNil)
  thus ?thesis by(rule strongEqWeakCong)
qed

```

lemma *sumSym*:

```

  fixes  $P :: pi$ 
  and  $Q :: pi$ 

```

```

  shows  $P \oplus Q \simeq^s Q \oplus P$ 
proof -
  have  $P \oplus Q \sim^s Q \oplus P$  by(rule Strong-Late-Bisim-Subst-SC.sumSym)
  thus ?thesis by(rule strongEqWeakCong)
qed

```

lemma *sumAssoc*:

```

  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $R :: pi$ 

```

```

  shows  $(P \oplus Q) \oplus R \simeq^s P \oplus (Q \oplus R)$ 
proof -
  have  $(P \oplus Q) \oplus R \sim^s P \oplus (Q \oplus R)$  by(rule Strong-Late-Bisim-Subst-SC.sumAssoc)
  thus ?thesis by(rule strongEqWeakCong)
qed

```

lemma *sumZero*:

```

  fixes  $P :: pi$ 

```

```

  shows  $P \oplus \mathbf{0} \simeq^s P$ 
proof -
  have  $P \oplus \mathbf{0} \sim^s P$  by(rule Strong-Late-Bisim-Subst-SC.sumZero)
  thus ?thesis by(rule strongEqWeakCong)
qed

```

lemma *parZero*:

```

  fixes  $P :: pi$ 

```

```

  shows  $P \parallel \mathbf{0} \simeq^s P$ 
proof -
  have  $P \parallel \mathbf{0} \sim^s P$  by(rule Strong-Late-Bisim-Subst-SC.parZero)
  thus ?thesis by(rule strongEqWeakCong)
qed

```

lemma *parSym*:

fixes $P :: pi$

and $Q :: pi$

shows $P \parallel Q \simeq^s Q \parallel P$

proof –

have $P \parallel Q \sim^s Q \parallel P$ **by**(*rule Strong-Late-Bisim-Subst-SC.parSym*)

thus *?thesis* **by**(*rule strongEqWeakCong*)

qed

lemma *scopeExtPar*:

fixes $P :: pi$

and $Q :: pi$

and $x :: name$

assumes $x \# P$

shows $\langle \nu x \rangle (P \parallel Q) \simeq^s P \parallel \langle \nu x \rangle Q$

proof –

from *assms* **have** $\langle \nu x \rangle (P \parallel Q) \sim^s P \parallel \langle \nu x \rangle Q$ **by**(*rule Strong-Late-Bisim-Subst-SC.scopeExtPar*)

thus *?thesis* **by**(*rule strongEqWeakCong*)

qed

lemma *scopeExtPar'*:

fixes $P :: pi$

and $Q :: pi$

and $x :: name$

assumes $x \text{Fresh} Q: x \# Q$

shows $\langle \nu x \rangle (P \parallel Q) \simeq^s (\langle \nu x \rangle P) \parallel Q$

proof –

from *assms* **have** $\langle \nu x \rangle (P \parallel Q) \sim^s (\langle \nu x \rangle P) \parallel Q$ **by**(*rule Strong-Late-Bisim-Subst-SC.scopeExtPar'*)

thus *?thesis* **by**(*rule strongEqWeakCong*)

qed

lemma *parAssoc*:

fixes $P :: pi$

and $Q :: pi$

and $R :: pi$

shows $(P \parallel Q) \parallel R \simeq^s P \parallel (Q \parallel R)$

proof –

have $(P \parallel Q) \parallel R \sim^s P \parallel (Q \parallel R)$ **by**(*rule Strong-Late-Bisim-Subst-SC.parAssoc*)

thus *?thesis* **by**(*rule strongEqWeakCong*)

qed

lemma *scopeFresh*:

fixes $P :: pi$

```

and   a :: name

assumes aFreshP: a # P

shows <νa>P ≃s P
proof –
from assms have <νa>P ∼s P by(rule Strong-Late-Bisim-Subst-SC.scopeFresh)
thus ?thesis by(rule strongEqWeakCong)
qed

lemma scopeExtSum:
  fixes P :: pi
  and   Q :: pi
  and   x :: name

  assumes x # P

  shows <νx>(P ⊕ Q) ≃s P ⊕ <νx>Q
proof –
from assms have <νx>(P ⊕ Q) ∼s P ⊕ <νx>Q by(rule Strong-Late-Bisim-Subst-SC.scopeExtSum)
thus ?thesis by(rule strongEqWeakCong)
qed

lemma bangSC:
  fixes P

  shows !P ≃s P || !P
proof –
have !P ∼s P || !P by(rule Strong-Late-Bisim-Subst-SC.bangSC)
thus ?thesis by(rule strongEqWeakCong)
qed

end

theory Weak-Late-Step-Sim-Pres
  imports Weak-Late-Step-Sim
begin

lemma tauPres:
  fixes P   :: pi
  and   Q   :: pi
  and   Rel :: (pi × pi) set
  and   Rel' :: (pi × pi) set

  assumes PRelQ: (P, Q) ∈ Rel

  shows τ.(P) ↪<Rel> τ.(Q)
proof(induct rule: simCases)
  case(Bound Q' a y)

```

```

have  $\tau.(Q) \mapsto a \langle \nu y \rangle \prec Q'$  by fact
hence False by auto
thus ?case by simp
next
case(Input  $Q' a x$ )
have  $\tau.(Q) \mapsto a \langle x \rangle \prec Q'$  by fact
hence False by auto
thus ?case by simp
next
case(Free  $Q' \alpha$ )
have  $\tau.(Q) \mapsto \alpha \prec Q'$  by fact
thus ?case using PRelQ
proof(induct rule: tauCases, auto simp add: pi.inject residual.inject)
  have  $\tau.(P) \Longrightarrow_I \tau \prec P$  by(rule Weak-Late-Step-Semantics.Tau)
  moreover assume  $(P, Q') \in \text{Rel}$ 
  ultimately show  $\exists P'. \tau.(P) \Longrightarrow_I \tau \prec P' \wedge (P', Q') \in \text{Rel}$  by blast
qed
qed

```

lemma *inputPres*:

```

fixes  $P$  :: pi
and  $Q$  :: pi
and  $a$  :: name
and  $x$  :: name
and  $\text{Rel}$  ::  $(\text{pi} \times \text{pi})$  set

```

```

assumes PRelQ:  $\forall y. (P[x::=y], Q[x::=y]) \in \text{Rel}$ 
and Eqvt: eqvt Rel

```

shows $a \langle x \rangle . P \rightsquigarrow_{\langle \text{Rel} \rangle} a \langle x \rangle . Q$

proof –

show *?thesis* **using** *Eqvt*

proof(*induct rule: simCasesCont[of - (P, a, x, Q)]*)

case(*Bound* $Q' b y$)

have $a \langle x \rangle . Q \mapsto b \langle \nu y \rangle \prec Q'$ **by fact**

hence *False* **by auto**

thus *?case* **by simp**

next

case(*Input* $Q' b y$)

have $y \# (P, a, x, Q)$ **by fact**

hence *yFreshP*: $(y::\text{name}) \# P$ **and** *yineqx*: $y \neq x$ **and** $y \neq a$ **and** $y \# Q$

by(*simp add: fresh-prod*)**+**

have $a \langle x \rangle . Q \mapsto b \langle y \rangle \prec Q'$ **by fact**

thus *?case* **using** $\langle y \neq a \rangle \langle y \neq x \rangle \langle y \# Q \rangle$

proof(*induct rule: inputCases, auto simp add: subject.inject*)

have $\forall u. \exists P'. a \langle x \rangle . P \Longrightarrow_I u \text{ in } ((x, y) \cdot P) \rightarrow a \langle y \rangle \prec P' \wedge (P', ((x, y)$

$\cdot Q)[y::=u]) \in \text{Rel}$

proof(*rule allI*)

fix u

have $a\langle x \rangle.P \Rightarrow_{lu} \text{in } ([x, y] \cdot P) \rightarrow a\langle y \rangle \prec ([x, y] \cdot P)[y::=u]$ (**is**
 $?goal$)
proof –
from $yFreshP$ **have** $a\langle x \rangle.P = a\langle y \rangle.([x, y] \cdot P)$ **by** (*rule Agent.alphaInput*)
moreover have $a\langle y \rangle.([x, y] \cdot P) \Rightarrow_{lu} \text{in } ([x, y] \cdot P) \rightarrow a\langle y \rangle \prec ([x,$
 $y]) \cdot P)[y::=u]$
by (*rule Weak-Late-Step-Semantics.Input*)
ultimately show $?goal$ **by** (*simp add: name-swap*)
qed

moreover have $(([x, y] \cdot P)[y::=u], ([x, y] \cdot Q)[y::=u]) \in Rel$
proof –
from $PRelQ$ **have** $(P[x::=u], Q[x::=u]) \in Rel$ **by** *auto*
with $\langle y \# P \rangle \langle y \# Q \rangle$ **show** $?thesis$ **by** (*simp add: renaming*)
qed

ultimately show $\exists P'. a\langle x \rangle.P \Rightarrow_{lu} \text{in } ([x, y] \cdot P) \rightarrow a\langle y \rangle \prec P' \wedge (P',$
 $([x, y] \cdot Q)[y::=u]) \in Rel$
by *blast*
qed

thus $\exists P''. \forall u. \exists P'. a\langle x \rangle.P \Rightarrow_{lu} \text{in } P'' \rightarrow a\langle y \rangle \prec P' \wedge (P', ([x, y] \cdot$
 $Q)[y::=u]) \in Rel$ **by** *blast*
qed
next
case (*Free Q' α*)
have $a\langle x \rangle.Q \mapsto \alpha \prec Q'$ **by** *fact*
hence *False* **by** *auto*
thus $?case$ **by** *simp*
qed
qed

lemma *outputPres*:

fixes $P :: pi$
and $Q :: pi$
and $a :: name$
and $b :: name$
and $Rel :: (pi \times pi)$ *set*
and $Rel' :: (pi \times pi)$ *set*

assumes $PRelQ: (P, Q) \in Rel$

shows $a\{b\}.P \rightsquigarrow_{\langle Rel \rangle} a\{b\}.Q$

proof (*induct rule: simCases*)

case (*Bound Q' c x*)

have $a\{b\}.Q \mapsto c\langle \nu x \rangle \prec Q'$ **by** *fact*

hence *False* **by** *auto*

thus $?case$ **by** *simp*

next

```

    case(Input Q' c x)
    have a{b}.Q  $\mapsto$  c<x> < Q' by fact
    hence False by auto
    thus ?case by simp
next
    case(Free Q'  $\alpha$ )
    have a{b}.Q  $\mapsto$   $\alpha$  < Q' by fact
    thus ?case using PRelQ
    proof(induct rule: outputCases, auto simp add: pi.inject residual.inject)
      have a{b}.P  $\implies$  a[b] < P by(rule Weak-Late-Step-Semantics.Output)
      moreover assume (P, Q')  $\in$  Rel
      ultimately show  $\exists P'. a\{b\}.P \implies$  a[b] < P'  $\wedge$  (P', Q')  $\in$  Rel by blast
    qed
  qed

lemma matchPres:
  fixes P :: pi
  and Q :: pi
  and a :: name
  and b :: name
  and Rel :: (pi  $\times$  pi) set
  and Rel' :: (pi  $\times$  pi) set

  assumes PSimQ: P  $\rightsquigarrow$  <Rel> Q
  and RelRel': Rel  $\subseteq$  Rel'

  shows [a $\frown$ b]P  $\rightsquigarrow$  <Rel'> [a $\frown$ b]Q
proof(induct rule: simCases)
  case(Bound Q' c x)
  have x  $\#$  [a $\frown$ b]P by fact
  hence xFreshP: (x::name)  $\#$  P by simp
  have [a $\frown$ b]Q  $\mapsto$  c< $\nu$ x> < Q' by fact
  thus ?case
  proof(induct rule: matchCases)
    case cMatch
    have Q  $\mapsto$  c< $\nu$ x> < Q' by fact
    with PSimQ xFreshP obtain P' where PTrans: P  $\implies$  c< $\nu$ x> < P'
      and P'RelQ': (P', Q')  $\in$  Rel
    by(blast dest: simE)
    from PTrans have [a $\frown$ a]P  $\implies$  c< $\nu$ x> < P' by(rule Weak-Late-Step-Semantics.Match)
    moreover from P'RelQ' RelRel' have (P', Q')  $\in$  Rel' by blast
    ultimately show ?case by blast
  qed
next
  case(Input Q' c x)
  have x  $\#$  [a $\frown$ b]P by fact
  hence xFreshP: (x::name)  $\#$  P by simp
  have [a $\frown$ b]Q  $\mapsto$  c<x> < Q' by fact
  thus ?case

```

```

proof(induct rule: matchCases)
  case cMatch
    have  $Q \mapsto c\langle x \rangle \prec Q'$  by fact
    with PSimQ xFreshP obtain  $P''$  where  $L1: \forall u. \exists P'. P \Rightarrow_{\iota} u$  in  $P'' \rightarrow c\langle x \rangle$ 
 $\prec P' \wedge (P', Q'[x::=u]) \in Rel$ 
    by(blast dest: simE)
    have  $\forall u. \exists P'. [a \frown a]P \Rightarrow_{\iota} u$  in  $P'' \rightarrow c\langle x \rangle \prec P' \wedge (P', Q'[x::=u]) \in Rel'$ 
    proof(rule allI)
      fix  $u$ 
      from  $L1$  obtain  $P'$  where  $PTrans: P \Rightarrow_{\iota} u$  in  $P'' \rightarrow c\langle x \rangle \prec P'$  and  $P'RelQ':$ 
 $(P', Q'[x::=u]) \in Rel$ 
      by blast
      from  $PTrans$  have  $[a \frown a]P \Rightarrow_{\iota} u$  in  $P'' \rightarrow c\langle x \rangle \prec P'$  by(rule Weak-Late-Step-Semantics.Match)
      with  $P'RelQ' RelRel'$  show  $\exists P'. [a \frown a]P \Rightarrow_{\iota} u$  in  $P'' \rightarrow c\langle x \rangle \prec P' \wedge (P',$ 
 $Q'[x::=u]) \in Rel'$ 
      by blast
    qed
    thus ?case by blast
  qed
next
  case(Free Q'  $\alpha$ )
  have  $[a \frown b]Q \mapsto \alpha \prec Q'$  by fact
  thus ?case
  proof(induct rule: matchCases)
    case cMatch
      have  $Q \mapsto \alpha \prec Q'$  by fact
      with PSimQ obtain  $P'$  where  $PTrans: P \Rightarrow_{\iota} \alpha \prec P'$  and  $PRel: (P', Q') \in$ 
 $Rel$ 
      by(blast dest: simE)
      from  $PTrans$  have  $[a \frown a]P \Rightarrow_{\iota} \alpha \prec P'$  by(rule Weak-Late-Step-Semantics.Match)
      with  $PRel RelRel'$  show ?case by blast
    qed
  qed

lemma mismatchPres:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $a :: name$ 
  and  $b :: name$ 
  and  $Rel :: (pi \times pi)$  set
  and  $Rel' :: (pi \times pi)$  set

  assumes PSimQ:  $P \rightsquigarrow \langle Rel \rangle Q$ 
  and  $RelRel': Rel \subseteq Rel'$ 

  shows  $[a \neq b]P \rightsquigarrow \langle Rel' \rangle [a \neq b]Q$ 
proof(cases a=b)
  assume  $a=b$ 
  thus ?thesis

```

```

    by(auto simp add: weakStepSimDef)
next
assume aineqb: a≠b
show ?thesis
proof(induct rule: simCases)
  case(Bound Q' c x)
  have x # [a≠b]P by fact
  hence xFreshP: (x::name) # P by simp
  have [a≠b]Q ⟶ c<νx> < Q' by fact
  thus ?case
proof(induct rule: mismatchCases)
  case cMismatch
  have Q ⟶ c<νx> < Q' by fact
  with PSimQ xFreshP obtain P' where PTrans: P ⟶1 c<νx> < P'
    and P'RelQ': (P', Q') ∈ Rel

    by(blast dest: simE)
  from PTrans aineqb have [a≠b]P ⟶1 c<νx> < P' by(rule Weak-Late-Step-Semantics.Mismatch)
  moreover from P'RelQ' RelRel' have (P', Q') ∈ Rel' by blast
  ultimately show ?case by blast
qed
next
case(Input Q' c x)
have x # [a≠b]P by fact
hence xFreshP: (x::name) # P by simp
have [a≠b]Q ⟶ c<x> < Q' by fact
thus ?case
proof(induct rule: mismatchCases)
  case cMismatch
  have Q ⟶ c<x> < Q' by fact
  with PSimQ xFreshP obtain P'' where L1: ∀ u. ∃ P'. P ⟶1 u in P'' → c<x>
    < P' ∧ (P', Q'[x::=u]) ∈ Rel
    by(blast dest: simE)
  have ∀ u. ∃ P'. [a≠b]P ⟶1 u in P'' → c<x> < P' ∧ (P', Q'[x::=u]) ∈ Rel'
  proof(rule allI)
    fix u
    from L1 obtain P' where PTrans: P ⟶1 u in P'' → c<x> < P' and
      P'RelQ': (P', Q'[x::=u]) ∈ Rel
    by blast
    from PTrans aineqb have [a≠b]P ⟶1 u in P'' → c<x> < P' by(rule
      Weak-Late-Step-Semantics.Mismatch)
    with P'RelQ' RelRel' show ∃ P'. [a≠b]P ⟶1 u in P'' → c<x> < P' ∧ (P',
      Q'[x::=u]) ∈ Rel'
    by blast
  qed
  thus ?case by blast
qed
next
case(Free Q' α)
have [a≠b]Q ⟶ α < Q' by fact

```

```

thus ?case
proof(induct rule: mismatchCases)
  case cMismatch
  have  $Q \mapsto \alpha \prec Q'$  by fact
  with PSimQ obtain  $P'$  where  $P \Longrightarrow_1 \alpha \prec P'$  and  $P' \text{Rel} (P', Q')$ 
 $\in \text{Rel}$ 
  by(blast dest: simE)
  from  $P \text{Trans} \langle a \neq b \rangle$  have  $[a \neq b]P \Longrightarrow_1 \alpha \prec P'$  by(rule Weak-Late-Step-Semantics.Mismatch)
  with  $P' \text{Rel} \text{Rel}'$  show ?case by blast
qed
qed
qed

```

lemma *sumCompose*:

```

fixes  $P :: pi$ 
and  $Q :: pi$ 
and  $R :: pi$ 
and  $T :: pi$ 

assumes  $PSimQ: P \rightsquigarrow \langle \text{Rel} \rangle Q$ 
and  $R \text{Sim} T: R \rightsquigarrow \langle \text{Rel} \rangle T$ 
and  $\text{Rel}' \text{Rel}'': \text{Rel} \subseteq \text{Rel}'$ 

shows  $P \oplus R \rightsquigarrow \langle \text{Rel}' \rangle Q \oplus T$ 
proof(induct rule: simCases)
  case(Bound Q' a x)
  have  $x \# P \oplus R$  by fact
  hence  $x \text{Fresh} P: (x :: \text{name}) \# P$  and  $x \text{Fresh} R: x \# R$  by simp+
  have  $Q \oplus T \mapsto a \langle \nu x \rangle \prec Q'$  by fact
  thus ?case
  proof(induct rule: sumCases)
    case cSum1
    have  $Q \mapsto a \langle \nu x \rangle \prec Q'$  by fact
    with  $x \text{Fresh} P$   $PSimQ$  obtain  $P'$  where  $P \Longrightarrow_1 a \langle \nu x \rangle \prec P'$  and
 $P' \text{Rel} Q': (P', Q') \in \text{Rel}$ 
    by(blast dest: simE)
    from  $P \text{Trans} \langle a \neq b \rangle$  have  $P \oplus R \Longrightarrow_1 a \langle \nu x \rangle \prec P'$  by(rule Weak-Late-Step-Semantics.Sum1)
    moreover from  $P' \text{Rel} Q' \text{Rel}'$  have  $(P', Q') \in \text{Rel}'$  by blast
    ultimately show ?case by blast
  next
  case cSum2
  have  $T \mapsto a \langle \nu x \rangle \prec Q'$  by fact
  with  $x \text{Fresh} R$   $R \text{Sim} T$  obtain  $R'$  where  $R \Longrightarrow_1 a \langle \nu x \rangle \prec R'$  and
 $R' \text{Rel} Q': (R', Q') \in \text{Rel}$ 
  by(blast dest: simE)
  from  $R \text{Trans} \langle a \neq b \rangle$  have  $P \oplus R \Longrightarrow_1 a \langle \nu x \rangle \prec R'$  by(rule Weak-Late-Step-Semantics.Sum2)
  moreover from  $R' \text{Rel} Q' \text{Rel}'$  have  $(R', Q') \in \text{Rel}'$  by blast
  ultimately show ?thesis by blast
qed

```

```

next
  case(Input Q' a x)
  have x # P ⊕ R by fact
  hence xFreshP: (x::name) # P and xFreshR: x # R by simp+
  have Q ⊕ T ⟶ a⟨x⟩ < Q' by fact
  thus ?case
  proof(induct rule: sumCases)
    case cSum1
    have Q ⟶ a⟨x⟩ < Q' by fact
    with xFreshP PSimQ obtain P'' where L1: ∀ u. ∃ P'. P ⟶lu in P'' → a⟨x⟩
    < P' ∧ (P', Q'[x::=u]) ∈ Rel
    by(blast dest: simE)
    have ∀ u. ∃ P'. P ⊕ R ⟶lu in P'' → a⟨x⟩ < P' ∧ (P', Q'[x::=u]) ∈ Rel'
    proof(rule allI)
      fix u
      from L1 obtain P' where PTrans: P ⟶lu in P'' → a⟨x⟩ < P'
      and P'RelQ': (P', Q'[x::=u]) ∈ Rel by blast
      from PTrans have P ⊕ R ⟶lu in P'' → a⟨x⟩ < P' by(rule Weak-Late-Step-Semantics.Sum1)
      with P'RelQ' RelRel' show ∃ P'. P ⊕ R ⟶lu in P'' → a⟨x⟩ < P' ∧ (P',
    Q'[x::=u]) ∈ Rel' by blast
    qed
    thus ?case by blast
  next
  case cSum2
  have T ⟶ a⟨x⟩ < Q' by fact
  with xFreshR RSimT obtain R'' where L1: ∀ u. ∃ R'. R ⟶lu in R'' → a⟨x⟩
  < R' ∧ (R', Q'[x::=u]) ∈ Rel
  by(blast dest: simE)
  have ∀ u. ∃ P'. P ⊕ R ⟶lu in R'' → a⟨x⟩ < P' ∧ (P', Q'[x::=u]) ∈ Rel'
  proof(rule allI)
    fix u
    from L1 obtain R' where RTrans: R ⟶lu in R'' → a⟨x⟩ < R'
    and R'RelQ': (R', Q'[x::=u]) ∈ Rel by blast
    from RTrans have P ⊕ R ⟶lu in R'' → a⟨x⟩ < R' by(rule Weak-Late-Step-Semantics.Sum2)
    with R'RelQ' RelRel' show ∃ P'. P ⊕ R ⟶lu in R'' → a⟨x⟩ < P' ∧ (P',
  Q'[x::=u]) ∈ Rel' by blast
  qed
  thus ?case by blast
  qed
  next
  case(Free Q' α)
  have Q ⊕ T ⟶ α < Q' by fact
  thus ?case
  proof(induct rule: sumCases)
    case cSum1
    have Q ⟶ α < Q' by fact
    with PSimQ obtain P' where PTrans: P ⟶lα < P' and PRel: (P', Q') ∈
  Rel
    by(blast dest: simE)

```

```

from PTrans have  $P \oplus R \Longrightarrow_1 \alpha \prec P'$  by(rule Weak-Late-Step-Semantics.Sum1)
with RelRel' PRel show ?case by blast
next
  case cSum2
  have  $T \mapsto \alpha \prec Q'$  by fact
  with RSimT obtain  $R'$  where  $RTrans: R \Longrightarrow_1 \alpha \prec R'$  and  $RRel: (R', Q') \in$ 
Rel
  by(blast dest: simE)
  from RTrans have  $P \oplus R \Longrightarrow_1 \alpha \prec R'$  by(rule Weak-Late-Step-Semantics.Sum2)
  with RelRel' RRel show ?case by blast
qed
qed

```

lemma *sumPres*:

```

fixes  $P :: pi$ 
and  $Q :: pi$ 
and  $R :: pi$ 

```

```

assumes PSimQ:  $P \rightsquigarrow \langle Rel \rangle Q$ 
and  $Id: Id \subseteq Rel$ 
and  $RelRel': Rel \subseteq Rel'$ 

```

shows $P \oplus R \rightsquigarrow \langle Rel' \rangle Q \oplus R$

proof –

```

from Id have Refl:  $R \rightsquigarrow \langle Rel \rangle R$  by(rule reflexive)
from PSimQ Refl RelRel' show ?thesis by(rule sumCompose)

```

qed

lemma *parPres*:

```

fixes  $P :: pi$ 
and  $Q :: pi$ 
and  $R :: pi$ 
and  $Rel :: (pi \times pi)$  set
and  $Rel' :: (pi \times pi)$  set

```

```

assumes PSimQ:  $P \rightsquigarrow \langle Rel \rangle Q$ 
and PRelQ:  $(P, Q) \in Rel$ 
and Par:  $\bigwedge P Q R. (P, Q) \in Rel \Longrightarrow (P \parallel R, Q \parallel R) \in Rel'$ 
and Res:  $\bigwedge P Q a. (P, Q) \in Rel' \Longrightarrow (\langle \nu a \rangle P, \langle \nu a \rangle Q) \in Rel'$ 
and EqvtRel: eqvt  $Rel$ 
and EqvtRel': eqvt  $Rel'$ 

```

shows $P \parallel R \rightsquigarrow \langle Rel' \rangle Q \parallel R$

using *EqvtRel'*

proof(*induct rule: simCasesCont*[**where** $C=(P, Q, R)$])

case(*Bound* $Q' a x$)

have $x \# (P, Q, R)$ **by** *fact*

hence $xFreshP: x \# P$ **and** $xFreshR: x \# R$ **and** $x \# Q$ **by** *simp+*

from $\langle Q \parallel R \mapsto a \langle \nu x \rangle \prec Q' \rangle \langle x \# Q \rangle \langle x \# R \rangle$ **show** ?*case*

proof (*induct rule: parCasesB*)
case (*cPar1 Q'*)
have $QTrans: Q \mapsto a \langle \nu x \rangle \prec Q'$ **by fact**

from $xFreshP$ $PSimQ$ $QTrans$ **obtain** P' **where** $PTrans: P \Rightarrow_l a \langle \nu x \rangle \prec P'$
and $P'RelQ': (P', Q') \in Rel$

by (*blast dest: simE*)
from $PTrans$ $xFreshR$ **have** $P \parallel R \Rightarrow_l a \langle \nu x \rangle \prec (P' \parallel R)$ **by** (*rule Weak-Late-Step-Semantics.Par1B*)
moreover from $P'RelQ'$ **have** $(P' \parallel R, Q' \parallel R) \in Rel'$ **by** (*rule Par*)
ultimately show *?case* **by blast**
next
case (*cPar2 R'*)
have $RTrans: R \mapsto a \langle \nu x \rangle \prec R'$ **by fact**
hence $R \Rightarrow_l a \langle \nu x \rangle \prec R'$
by (*auto simp add: weakTransition-def dest: Weak-Late-Step-Semantics.singleActionChain*)
with $xFreshP$ $xFreshR$ **have** $ParTrans: P \parallel R \Rightarrow_l a \langle \nu x \rangle \prec P \parallel R'$
by (*blast intro: Weak-Late-Step-Semantics.Par2B*)
moreover from $P'RelQ'$ **have** $(P \parallel R', Q \parallel R') \in Rel'$ **by** (*rule Par*)
ultimately show *?case* **by blast**
qed
next
case (*Input Q' a x*)
have $x \# (P, Q, R)$ **by fact**
hence $xFreshP: x \# P$ **and** $xFreshR: x \# R$ **and** $x \# Q$ **by simp+**
from $\langle Q \parallel R \mapsto a \langle x \rangle \prec Q' \rangle \langle x \# Q \rangle \langle x \# R \rangle$
show *?case*
proof (*induct rule: parCasesB*)
case (*cPar1 Q'*)
have $QTrans: Q \mapsto a \langle x \rangle \prec Q'$ **by fact**
from $xFreshP$ $PSimQ$ $QTrans$ **obtain** P''
where $L1: \forall u. \exists P'. P \Rightarrow_l u$ *in* $P'' \rightarrow a \langle x \rangle \prec P' \wedge (P', Q'[x::=u]) \in Rel$
by (*blast dest: simE*)
have $\forall u. \exists P'. P \parallel R \Rightarrow_l u$ *in* $(P'' \parallel R) \rightarrow a \langle x \rangle \prec P' \wedge (P', Q'[x::=u] \parallel R[x::=u]) \in Rel'$
proof (*rule allI*)
fix u
from $L1$ **obtain** P' **where** $PTrans: P \Rightarrow_l u$ *in* $P'' \rightarrow a \langle x \rangle \prec P'$
and $P'RelQ': (P', Q'[x::=u]) \in Rel$ **by blast**
from $PTrans$ $xFreshR$ **have** $P \parallel R \Rightarrow_l u$ *in* $(P'' \parallel R) \rightarrow a \langle x \rangle \prec (P' \parallel R)$
by (*rule Weak-Late-Step-Semantics.Par1B*)
moreover from $P'RelQ'$ **have** $(P' \parallel R, Q'[x::=u] \parallel R) \in Rel'$
by (*rule Par*)
ultimately show $\exists P'. P \parallel R \Rightarrow_l u$ *in* $(P'' \parallel R) \rightarrow a \langle x \rangle \prec P' \wedge (P', Q'[x::=u] \parallel (R[x::=u])) \in Rel'$
using $xFreshR$
by (*force simp add: forget*)
qed
thus *?case* **by force**
next

```

case(cPar2  $R'$ )
have  $RTrans: R \mapsto a\langle x \rangle \prec R'$  by fact
have  $\forall u. \exists P'. P \parallel R \Longrightarrow_{lu} in (P \parallel R') \rightarrow a\langle x \rangle \prec P' \wedge (P', Q \parallel R'[x::=u])$ 
 $\in Rel'$ 
proof
  fix  $u$ 
  from  $RTrans$  have  $R \Longrightarrow_{lu} in R' \rightarrow a\langle x \rangle \prec R'[x::=u]$ 
    by(rule Weak-Late-Step-Semantics.singleActionChain)
  hence  $P \parallel R \Longrightarrow_{lu} in P \parallel R' \rightarrow a\langle x \rangle \prec P \parallel R'[x::=u]$  using  $\langle x \# P \rangle$ 
    by(rule Weak-Late-Step-Semantics.Par2B)
  moreover from  $PRelQ$  have  $(P \parallel R'[x::=u], Q \parallel R'[x::=u]) \in Rel'$  by(rule
Par)
  ultimately show  $\exists P'. P \parallel R \Longrightarrow_{lu} in (P \parallel R') \rightarrow a\langle x \rangle \prec P' \wedge$ 
     $(P', Q \parallel R'[x::=u]) \in Rel'$  by blast

  qed
  thus ?case using  $\langle x \# Q \rangle$  by(fastforce simp add: forget)
qed
next
case(Free  $QR' \alpha$ )
have  $Q \parallel R \mapsto \alpha \prec QR'$  by fact
thus ?case
proof(induct rule: parCasesF[of - - - - (P, R)])
  case(cPar1  $Q'$ )
  have  $Q \mapsto \alpha \prec Q'$  by fact
  with  $PSimQ$  obtain  $P'$  where  $PTrans: P \Longrightarrow_{l\alpha} \prec P'$  and  $PRel: (P', Q') \in$ 
 $Rel$ 
    by(blast dest: simE)
  from  $PTrans$  have  $Trans: P \parallel R \Longrightarrow_{l\alpha} \prec P' \parallel R$  by(rule Weak-Late-Step-Semantics.Par1F)
  moreover from  $PRel$  have  $(P' \parallel R, Q' \parallel R) \in Rel'$  by(blast intro: Par)
  ultimately show ?case by blast
next
  case(cPar2  $R'$ )
  have  $R \mapsto \alpha \prec R'$  by fact
  hence  $R \Longrightarrow_{l\alpha} \prec R'$ 
    by(rule Weak-Late-Step-Semantics.singleActionChain)
  hence  $P \parallel R \Longrightarrow_{l\alpha} \prec (P \parallel R')$  by(rule Weak-Late-Step-Semantics.Par2F)
  moreover from  $PRelQ$  have  $(P \parallel R', Q \parallel R') \in Rel'$  by(blast intro: Par)
  ultimately show ?case by blast
next
  case(cComm1  $Q' R' a b x$ )
  have  $QTrans: Q \mapsto a\langle x \rangle \prec Q'$  and  $RTrans: R \mapsto a[b] \prec R'$  by fact+
  have  $x \# (P, R)$  by fact
  hence  $xFreshP: x \# P$  by(simp add: fresh-prod)

  from  $PSimQ$   $QTrans$   $xFreshP$  obtain  $P' P''$  where  $PTrans: P \Longrightarrow_{lb} in$ 
 $P'' \rightarrow a\langle x \rangle \prec P'$ 
    and  $P'RelQ': (P', Q'[x::=b]) \in Rel$ 
    by(blast dest: simE)

```

from $RTrans$ **have** $R \Longrightarrow_I a[b] \prec R'$
by(rule *Weak-Late-Step-Semantics.singleActionChain*)

with $PTrans$ **have** $P \parallel R \Longrightarrow_I \tau \prec P' \parallel R'$ **by**(rule *Weak-Late-Step-Semantics.Comm1*)
moreover from $P'RelQ'$ **have** $(P' \parallel R', Q'[x::=b] \parallel R') \in Rel'$ **by**(rule *Par*)
ultimately show *?case by blast*

next
case(*cComm2 Q' R' a b x*)
have $QTrans: Q \mapsto a[b] \prec Q'$ **and** $RTrans: R \mapsto a\langle x \rangle \prec R'$ **by** *fact+*
have $x \# (P, R)$ **by** *fact*
hence $xFreshR: x \# R$ **by**(*simp add: fresh-prod*)

from $PSimQ QTrans$ **obtain** P' **where** $PTrans: P \Longrightarrow_I a[b] \prec P'$
and $PRel: (P', Q') \in Rel$
by(*blast dest: simE*)
from $RTrans$ **have** $R \Longrightarrow_I b$ **in** $R' \rightarrow a\langle x \rangle \prec R'[x::=b]$
by(rule *Weak-Late-Step-Semantics.singleActionChain*)
with $PTrans$ **have** $P \parallel R \Longrightarrow_I \tau \prec P' \parallel R'[x::=b]$ **by**(rule *Weak-Late-Step-Semantics.Comm2*)
moreover from $PRel$ **have** $(P' \parallel R'[x::=b], Q' \parallel R'[x::=b]) \in Rel'$ **by**(rule *Par*)
ultimately show *?case by blast*

next
case(*cClose1 Q' R' a x y*)
have $QTrans: Q \mapsto a\langle x \rangle \prec Q'$ **and** $RTrans: R \mapsto a\langle \nu y \rangle \prec R'$ **by** *fact+*
have $x \# (P, R)$ **and** $y \# (P, R)$ **by** *fact+*
hence $xFreshP: x \# P$ **and** $yFreshR: y \# R$ **and** $yFreshP: y \# P$ **by**(*simp add: fresh-prod*)+

from $PSimQ QTrans xFreshP$ **obtain** $P' P''$ **where** $PTrans: P \Longrightarrow_I y$ **in**
 $P'' \rightarrow a\langle x \rangle \prec P'$
and $P'RelQ': (P', Q'[x::=y]) \in Rel$
by(*blast dest: simE*)
from $RTrans$ **have** $R \Longrightarrow_I a\langle \nu y \rangle \prec R'$
by(*auto simp add: weakTransition-def dest: Weak-Late-Step-Semantics.singleActionChain*)
with $PTrans$ **have** $Trans: P \parallel R \Longrightarrow_I \tau \prec \langle \nu y \rangle (P' \parallel R')$ **using** $yFreshP$
 $yFreshR$
by(rule *Weak-Late-Step-Semantics.Close1*)
moreover from $P'RelQ'$ **have** $(\langle \nu y \rangle (P' \parallel R'), \langle \nu y \rangle (Q'[x::=y] \parallel R')) \in Rel'$
by(*blast intro: Par Res*)
ultimately show *?case by blast*

next
case(*cClose2 Q' R' a x y*)
have $QTrans: Q \mapsto a\langle \nu y \rangle \prec Q'$ **and** $RTrans: R \mapsto a\langle x \rangle \prec R'$ **by** *fact+*
have $x \# (P, R)$ **and** $y \# (P, R)$ **by** *fact+*
hence $xFreshR: x \# R$ **and** $yFreshP: y \# P$ **and** $yFreshR: y \# R$ **by**(*simp add: fresh-prod*)+

from $PSimQ QTrans yFreshP$ **obtain** P' **where** $PTrans: P \Longrightarrow_I a\langle \nu y \rangle \prec P'$
and $P'RelQ': (P', Q') \in Rel$

by(*blast dest: simE*)
from *RTrans* **have** $R \Longrightarrow_{\iota} y$ in $R' \rightarrow a \langle x \rangle \prec R'[x::=y]$
by(*rule Weak-Late-Step-Semantics.singleActionChain*)
with *PTrans* **have** $P \parallel R \Longrightarrow_{\iota} \tau \prec \langle \nu y \rangle (P' \parallel R'[x::=y])$ **using** *yFreshP*
yFreshR
by(*rule Weak-Late-Step-Semantics.Close2*)
moreover from $P' \text{Rel} Q'$ **have** $(\langle \nu y \rangle (P' \parallel R'[x::=y]), \langle \nu y \rangle (Q' \parallel R'[x::=y]))$
 $\in \text{Rel}'$
by(*blast intro: Par Res*)
ultimately show *?case* **by** *blast*
qed
qed

lemma *resPres*:

fixes $P \quad :: \text{pi}$
and $Q \quad :: \text{pi}$
and $\text{Rel} \quad :: (\text{pi} \times \text{pi}) \text{ set}$
and $x \quad :: \text{name}$
and $\text{Rel}' \quad :: (\text{pi} \times \text{pi}) \text{ set}$

assumes $\text{PSimQ}: P \rightsquigarrow \langle \text{Rel} \rangle Q$
and $\text{ResRel}: \bigwedge (P::\text{pi}) (Q::\text{pi}) (x::\text{name}). (P, Q) \in \text{Rel} \Longrightarrow (\langle \nu x \rangle P, \langle \nu x \rangle Q)$
 $\in \text{Rel}'$
and $\text{RelRel}': \text{Rel} \subseteq \text{Rel}'$
and $\text{EqvtRel}: \text{eqvt Rel}$
and $\text{EqvtRel}': \text{eqvt Rel}'$

shows $\langle \nu x \rangle P \rightsquigarrow \langle \text{Rel}' \rangle \langle \nu x \rangle Q$

proof –

from $\text{EqvtRel}'$ **show** *?thesis*

proof(*induct rule: simCasesCont[of - (P, Q, x)]*)

case(*Bound Q' a y*)

have $\text{Trans}: \langle \nu x \rangle Q \mapsto a \langle \nu y \rangle \prec Q'$ **by** *fact*

have $y \# (P, Q, x)$ **by** *fact*

hence $y \text{ineq}: y \neq x$ **and** $y \text{FreshP}: y \# P$ **and** $y \# Q$ **by**(*simp add: fresh-prod*)**+**

from $\text{Trans} \langle y \neq x \rangle \langle y \# Q \rangle$ **show** *?case*

proof(*induct rule: resCasesB*)

case(*cOpen a Q'*)

have $Q \text{Trans}: Q \mapsto a[x] \prec Q'$ **and** $a \text{ineq}: a \neq x$ **by** *fact***+**

from $\text{PSimQ} Q \text{Trans}$ **obtain** P' **where** $P \text{Trans}: P \Longrightarrow_{\iota} a[x] \prec P'$

and $P' \text{Rel} Q': (P', Q') \in \text{Rel}$

by(*blast dest: simE*)

have $\langle \nu x \rangle P \Longrightarrow_{\iota} a \langle \nu y \rangle \prec ([y, x] \cdot P')$

proof –

from $P \text{Trans} a \text{ineq}$ **have** $\langle \nu x \rangle P \Longrightarrow_{\iota} a \langle \nu x \rangle \prec P'$ **by**(*rule Weak-Late-Step-Semantics.Open*)

moreover from $P \text{Trans} y \text{FreshP}$ **have** $y \# P'$ **by**(*force intro: Weak-Late-Step-Semantics.freshTransition*)

ultimately show *?thesis* **by**(*simp add: alphaBoundResidual name-swap*)
qed
moreover from *EqvtRel P'RelQ' RelRel'* **have** $((y, x) \cdot P', [(y, x)] \cdot Q') \in$
Rel'
by(*blast intro: eqvtRelI*)
ultimately show *?case by blast*
next
case(*cRes Q'*)
have *QTrans: Q \mapsto a <v y> < Q'* **by fact**
from $\langle x \# \text{BoundOutputS } a \rangle$ **have** $x \neq a$ **by simp**

from *PSimQ yFreshP QTrans* **obtain** *P'* **where** *PTrans: P \Rightarrow a <v y> <*
P'
and *P'RelQ': (P', Q') \in Rel*
by(*blast dest: simE*)
from *PTrans* $\langle x \neq a \rangle$ *yineqx yFreshP* **have** *ResTrans: <v x>P \Rightarrow a <v y> <*
 $\langle v x \rangle P'$
by(*blast intro: Weak-Late-Step-Semantics.ResB*)
moreover from *P'RelQ'* **have** $((\langle v x \rangle P'), (\langle v x \rangle Q')) \in \text{Rel}'$
by(*rule ResRel*)
ultimately show *?case by blast*
qed
next
case(*Input Q' a y*)
have $y \# (P, Q, x)$ **by fact**
hence *yineqx: y \neq x* **and** *yFreshP: y $\#$ P* **and** $y \# Q$ **by**(*simp add: fresh-prod*)

have $\langle v x \rangle Q \mapsto a \langle y \rangle < Q'$ **by fact**
thus *?case using yineqx* $\langle y \# Q \rangle$
proof(*induct rule: resCasesB*)
case(*cOpen a Q'*)
thus *?case by simp*
next
case(*cRes Q'*)
have *QTrans: Q \mapsto a <v y> < Q'* **by fact**
from $\langle x \# \text{InputS } a \rangle$ **have** $x \neq a$ **by simp**

from *PSimQ QTrans yFreshP* **obtain** *P''*
where *L1: $\forall u. \exists P'. P \Rightarrow u$ in $P'' \rightarrow a \langle y \rangle < P' \wedge (P', Q'[y::=u]) \in \text{Rel}$*
by(*blast dest: simE*)
have $\forall u. \exists P'. \langle v x \rangle P \Rightarrow u$ in $\langle v x \rangle P'' \rightarrow a \langle y \rangle < P' \wedge (P', (\langle v x \rangle Q')[y::=u])$
 $\in \text{Rel}'$
proof(*rule allI*)
fix *u*
show $\exists P'. \langle v x \rangle P \Rightarrow u$ in $\langle v x \rangle P'' \rightarrow a \langle y \rangle < P' \wedge (P', (\langle v x \rangle Q')[y::=u])$
 $\in \text{Rel}'$
proof(*cases x=u*)
assume *xqu: x=u*

have $\exists c::\text{name}. c \# (P, P'', Q', x, y, a)$ **by** (*blast intro: name-exists-fresh*)
then obtain $c::\text{name}$ **where** $c\text{Fresh}P: c \# P$ **and** $c\text{Fresh}P'': c \# P''$ **and**
 $c\text{Fresh}Q': c \# Q'$
and $c\text{ineq}x: c \neq x$ **and** $c\text{ineq}y: c \neq y$ **and** $c\text{ineq}a: c \neq a$
by (*force simp add: fresh-prod*)

from $L1$ **obtain** P' **where** $P\text{Trans}: P \Longrightarrow_1 c \text{ in } P'' \rightarrow a \langle y \rangle \prec P'$
and $P'\text{Rel}Q': (P', Q'[y::=c]) \in \text{Rel}$
by *blast*
have $\langle \nu x \rangle P \Longrightarrow_1 u \text{ in } (\langle \nu x \rangle P'') \rightarrow a \langle y \rangle \prec \langle \nu c \rangle ([x, c] \cdot P')$
proof –
from $P\text{Trans}$ $y\text{ineq}x \langle x \neq a \rangle$ $c\text{ineq}x$ **have** $\langle \nu x \rangle P \Longrightarrow_1 c \text{ in } (\langle \nu x \rangle P'') \rightarrow a \langle y \rangle$
 $\prec \langle \nu x \rangle P'$
by (*blast intro: Weak-Late-Step-Semantics.ResB*)
hence $([x, c] \cdot \langle \nu x \rangle P) \Longrightarrow_1 ([x, c] \cdot c) \text{ in } ([x, c] \cdot \langle \nu x \rangle P'') \rightarrow ([x, c] \cdot a) \langle ([x, c] \cdot y) \rangle \prec [x, c] \cdot \langle \nu x \rangle P'$
by (*rule Weak-Late-Step-Semantics.eqvtI*)
moreover from $c\text{Fresh}P$ **have** $\langle \nu c \rangle ([x, c] \cdot P) = \langle \nu x \rangle P$ **by** (*simp add: alphaRes*)
moreover from $c\text{Fresh}P''$ **have** $\langle \nu c \rangle ([x, c] \cdot P'') = \langle \nu x \rangle P''$ **by** (*simp add: alphaRes*)
ultimately show *?thesis using* $\langle x \neq a \rangle$ $c\text{ineq}a$ $y\text{ineq}x$ $c\text{ineq}y$ $c\text{ineq}x$ $x\text{eq}u$
by (*simp add: name-calc*)
qed
moreover have $(\langle \nu c \rangle ([x, c] \cdot P'), (\langle \nu x \rangle Q')[y::=u]) \in \text{Rel}'$
proof –
from $P'\text{Rel}Q'$ **have** $(\langle \nu x \rangle P', \langle \nu x \rangle (Q'[y::=c])) \in \text{Rel}'$ **by** (*rule ResRel*)
with $\text{EqvtRel}'$ **have** $([x, c] \cdot \langle \nu x \rangle P', [x, c] \cdot \langle \nu x \rangle (Q'[y::=c])) \in \text{Rel}'$ **by** (*rule eqvtRelI*)
with $c\text{ineq}y$ $y\text{ineq}x$ $c\text{ineq}x$ **have** $(\langle \nu c \rangle ([x, c] \cdot P'), (\langle \nu c \rangle ([x, c] \cdot Q'))[y::=x]) \in \text{Rel}'$
by (*simp add: name-calc eqvt-subs*)
with $c\text{Fresh}Q'$ $x\text{eq}u$ **show** *?thesis* **by** (*simp add: alphaRes*)
qed
ultimately show *?thesis* **by** *blast*

next
assume $x\text{ineq}u: x \neq u$
from $L1$ **obtain** P' **where** $P\text{Trans}: P \Longrightarrow_1 u \text{ in } P'' \rightarrow a \langle y \rangle \prec P'$
and $P'\text{Rel}Q': (P', Q'[y::=u]) \in \text{Rel}$ **by** *blast*

from $P\text{Trans}$ $\langle x \neq a \rangle$ $y\text{ineq}x$ $x\text{ineq}u$ **have** $\langle \nu x \rangle P \Longrightarrow_1 u \text{ in } (\langle \nu x \rangle P'') \rightarrow a \langle y \rangle$
 $\prec \langle \nu x \rangle P'$
by (*blast intro: Weak-Late-Step-Semantics.ResB*)
moreover from $P'\text{Rel}Q'$ $x\text{ineq}u$ $y\text{ineq}x$ **have** $(\langle \nu x \rangle P', (\langle \nu x \rangle Q')[y::=u]) \in \text{Rel}'$
by (*force intro: ResRel*)
ultimately show *?thesis* **by** *blast*
qed
qed

```

    thus ?case by blast
  qed
next
case(Free Q' α)
have <νx>Q ⟶ α < Q' by fact
thus ?case
proof(induct rule: resCasesF)
  case(cRes Q')
  have Q ⟶ α < Q' by fact
  with PSimQ obtain P' where PTrans: P ⟶lα < P'
    and P'RelQ': (P', Q') ∈ Rel
    by(blast dest: simE)

  have <νx>P ⟶lα < <νx>P'
  proof -
    have xFreshAlpha: x ‡ α by fact
    with PTrans show ?thesis by(rule Weak-Late-Step-Semantics.ResF)
  qed
  moreover from P'RelQ' have (<νx>P', <νx>Q') ∈ Rel' by(rule ResRel)
  ultimately show ?case by blast
qed
qed
qed

lemma bangPres:
  fixes P   :: pi
  and Q     :: pi
  and Rel   :: (pi × pi) set

  assumes PSimQ: P ~><Rel'> Q
  and PRelQ: (P, Q) ∈ Rel
  and Sim: ⋀P Q. (P, Q) ∈ Rel ⟹ P ~><Rel'> Q
  and RelRel': ⋀P Q. (P, Q) ∈ Rel ⟹ (P, Q) ∈ Rel'
  and eqvRel': eqvt Rel'

  shows !P ~><bangRel Rel'> !Q
proof -
  from eqvRel' have EqvBangRel': eqvt(bangRel Rel') by(rule eqvtBangRel)
  from RelRel' have BRelRel': ⋀P Q. (P, Q) ∈ bangRel Rel ⟹ (P, Q) ∈ bangRel Rel'
  by(auto intro: bangRelSubset)

  have ⋀Rs P. [!Q ⟶ Rs; (P, !Q) ∈ bangRel Rel] ⟹ weakStepSimAct P Rs P
  (bangRel Rel')
  proof -
    fix Rs P
    assume !Q ⟶ Rs and (P, !Q) ∈ bangRel Rel
    thus weakStepSimAct P Rs P (bangRel Rel')
    proof(nominal-induct avoiding: P rule: bangInduct)

```

```

case(cPar1B aa x Q' P)
have QTrans:  $Q \mapsto aa \langle x \rangle \prec Q'$  and xFreshQ:  $x \# Q$  by fact+
have  $(P, Q \parallel !Q) \in \text{bangRel } Rel$  and  $x \# P$  by fact+
thus ?case
proof(induct rule: BRParCases)
  case(BRPar P R)
    have PRelQ:  $(P, Q) \in Rel$  and RBangRelQ:  $(R, !Q) \in \text{bangRel } Rel$  by
fact+
    have  $x \# P \parallel R$  by fact
    hence xFreshP:  $x \# P$  and xFreshR:  $x \# R$  by simp+
    from PRelQ have PSimQ:  $P \rightsquigarrow \langle Rel \rangle Q$  by(rule Sim)
    from EqvtBangRel' show ?case
    proof(induct rule: simActBoundCases)
      case(Input a)
        have  $aa = \text{InputS } a$  by fact
        with PSimQ QTrans xFreshP obtain  $P''$ 
          where  $L1: \forall u. \exists P'. P \Longrightarrow_{!u} \text{in } P'' \rightarrow a \langle x \rangle \prec P' \wedge (P', Q'[x::=u]) \in$ 
Rel'
          by(blast dest: simE)
          have  $\forall u. \exists P'. P \parallel R \Longrightarrow_{!u} \text{in } (P'' \parallel R) \rightarrow a \langle x \rangle \prec P' \wedge (P', (Q' \parallel$ 
!Q)[x::=u]) \in \text{bangRel } Rel'
          proof(rule allI)
            fix  $u$ 
            from  $L1$  obtain  $P'$  where  $PTrans: P \Longrightarrow_{!u} \text{in } P'' \rightarrow a \langle x \rangle \prec P'$ 
and  $P'RelQ': (P', Q'[x::=u]) \in Rel'$ 
            by blast
            from  $PTrans$  xFreshR have  $P \parallel R \Longrightarrow_{!u} \text{in } (P'' \parallel R) \rightarrow a \langle x \rangle \prec P' \parallel R$ 
by(rule Weak-Late-Step-Semantics.Par1B)
            moreover have  $(P' \parallel R, (Q' \parallel !Q)[x::=u]) \in \text{bangRel } Rel'$ 
            proof –
            from  $P'RelQ' RBangRelQ$  have  $(P' \parallel R, Q'[x::=u] \parallel !Q) \in \text{bangRel}$ 
Rel'
            by(blast intro: BRelRel' Rel.BRPar)
            with xFreshQ show ?thesis by(force simp add: forget)
            qed
            ultimately show  $\exists P'. P \parallel R \Longrightarrow_{!u} \text{in } (P'' \parallel R) \rightarrow a \langle x \rangle \prec P' \wedge$ 
 $(P', (Q' \parallel !Q)[x::=u]) \in \text{bangRel } Rel'$ 
            by blast
            qed
          thus ?case by blast
        next
        case(BoundOutput a)
        have  $aa = \text{BoundOutputS } a$  by fact
        with PSimQ QTrans xFreshP obtain  $P'$  where  $PTrans: P \Longrightarrow_{!a} \langle \nu x \rangle$ 
 $\prec P'$  and  $P'RelQ': (P', Q') \in Rel'$ 
        by(force dest: simE)
        from  $PTrans$  xFreshR have  $P \parallel R \Longrightarrow_{!a} \langle \nu x \rangle \prec P' \parallel R$ 
by(rule Weak-Late-Step-Semantics.Par1B)
        moreover from  $P'RelQ' RBangRelQ$  have  $(P' \parallel R, Q' \parallel !Q) \in \text{bangRel}$ 

```

```

Rel'
  by(blast intro: Rel.BRPar BRelRel')
  ultimately show ?case by blast
qed
qed
next
case(cPar1F  $\alpha$   $Q'$   $P$ )
have  $QTrans: Q \mapsto \alpha \prec Q'$  by fact
have  $(P, Q \parallel !Q) \in \text{bangRel } Rel$  by fact
thus ?case
proof(induct rule: BRParCases)
case(BRPar  $P$   $R$ )
  have  $PRelQ: (P, Q) \in Rel$  and  $RBangRelQ: (R, !Q) \in \text{bangRel } Rel$  by
fact+
  show ?case
  proof(induct rule: simActFreeCases)
  case Free
    from  $PRelQ$  have  $P \rightsquigarrow \langle Rel' \rangle Q$  by(rule Sim)
    with  $QTrans$  obtain  $P'$  where  $PTrans: P \Longrightarrow_l \alpha \prec P'$  and  $P'RelQ': (P',$ 
 $Q') \in Rel'$ 
    by(blast dest: simE)

    from  $PTrans$  have  $P \parallel R \Longrightarrow_l \alpha \prec P' \parallel R$  by(rule Weak-Late-Step-Semantics.Par1F)
    moreover from  $P'RelQ'$   $RBangRelQ$  have  $(P' \parallel R, Q' \parallel !Q) \in \text{bangRel}$ 
Rel'
  by(blast intro: BRelRel' Rel.BRPar)
  ultimately show ?case by blast
qed
qed
next
case(cPar2B  $aa$   $x$   $Q'$   $P$ )
have  $IH: \bigwedge P. (P, !Q) \in \text{bangRel } Rel \implies \text{weakStepSimAct } P (aa \langle x \rangle \prec Q')$ 
 $P (\text{bangRel } Rel')$  by fact
have  $xFreshQ: x \# Q$  by fact
have  $(P, Q \parallel !Q) \in \text{bangRel } Rel$  and  $x \# P$  by fact+
thus ?case
proof(induct rule: BRParCases)
case(BRPar  $P$   $R$ )
  have  $PRelQ: (P, Q) \in Rel$  and  $RBangRelQ: (R, !Q) \in \text{bangRel } Rel$  by
fact+
  have  $x \# P \parallel R$  by fact
  hence  $xFreshP: x \# P$  and  $xFreshR: x \# R$  by simp+
  from  $RBangRelQ$  have  $IH: \text{weakStepSimAct } R (aa \langle x \rangle \prec Q') R (\text{bangRel}$ 
 $Rel')$  by(rule IH)
  from  $EqvBangRel'$  show ?case
  proof(induct rule: simActBoundCases)
  case(Input  $a$ )
    have  $aa = \text{InputS } a$  by fact
    with  $xFreshR$   $IH$  obtain  $R''$  where  $L1: \forall u. \exists R'. R \Longrightarrow_l u$  in  $R'' \rightarrow a \langle x \rangle$ 

```

$\prec R' \wedge$
 $(R', Q'[x::=u]) \in \text{bangRel Rel}'$
by (*simp add: weakStepSimAct-def, blast*)
have $\forall u. \exists P'. P \parallel R \Longrightarrow_l u$ in $(P \parallel R'') \rightarrow a \langle x \rangle \prec P' \wedge (P', (Q \parallel Q')[x::=u]) \in \text{bangRel Rel}'$
proof (*rule allI*)
fix u
from $L1$ **obtain** R' **where** $RTrans: R \Longrightarrow_l u$ in $R'' \rightarrow a \langle x \rangle \prec R'$
and $R'BangRelT': (R', Q'[x::=u]) \in \text{bangRel Rel}'$
by *blast*

from $RTrans$ $xFreshP$ **have** $P \parallel R \Longrightarrow_l u$ in $(P \parallel R'') \rightarrow a \langle x \rangle \prec P \parallel R'$
by (*rule Weak-Late-Step-Semantics.Par2B*)
moreover **have** $(P \parallel R', (Q \parallel Q')[x::=u]) \in \text{bangRel Rel}'$
proof –
from $PRelQ$ $R'BangRelT'$ **have** $(P \parallel R', Q \parallel Q'[x::=u]) \in \text{bangRel Rel}'$
by (*blast intro: RelRel' Rel.BRPar*)
with $xFreshQ$ **show** *?thesis* **by** (*simp add: forget*)
qed
ultimately show $\exists P'. P \parallel R \Longrightarrow_l u$ in $(P \parallel R'') \rightarrow a \langle x \rangle \prec P' \wedge (P', (Q \parallel Q')[x::=u]) \in \text{bangRel Rel}'$
by *blast*
qed
thus *?case* **by** *blast*
next
case (*BoundOutput a*)
have $aa = \text{BoundOutputS } a$ **by** *fact*
with IH $xFreshR$ **obtain** R' **where** $RTrans: R \Longrightarrow_l a \langle \nu x \rangle \prec R'$
and $R'BangRelT': (R', Q') \in \text{bangRel Rel}'$
by (*simp add: weakStepSimAct-def, blast*)

from $RTrans$ $xFreshP$ **have** $P \parallel R \Longrightarrow_l a \langle \nu x \rangle \prec P \parallel R'$
by (*auto intro: Weak-Late-Step-Semantics.Par2B*)
moreover **from** $PRelQ$ $R'BangRelT'$ **have** $(P \parallel R', Q \parallel Q') \in \text{bangRel Rel}'$
by (*blast intro: RelRel' Rel.BRPar*)
ultimately show *?case* **by** *blast*
qed
qed
next
case (*cPar2F α Q'*)
have $IH: \bigwedge P. (P, !Q) \in \text{bangRel Rel} \Longrightarrow \text{weakStepSimAct } P (\alpha \prec Q') P$
(bangRel Rel') **by** *fact+*
have $(P, Q \parallel !Q) \in \text{bangRel Rel}$ **by** *fact+*
thus *?case*
proof (*induct rule: BRParCases*)
case (*BRPar P R*)
have $PRelQ: (P, Q) \in \text{Rel}$ **and** $RBangRelQ: (R, !Q) \in \text{bangRel Rel}$ **by**

```

fact+
  show ?case
  proof(induct rule: simActFreeCases)
    case Free
      from RBangRelQ have weakStepSimAct R ( $\alpha \prec Q'$ ) R (bangRel Rel')
by(rule IH)
      then obtain R' where RTrans:  $R \Longrightarrow_{\iota} \alpha \prec R'$  and R'BangRelQ':  $(R', Q') \in \text{bangRel Rel}'$ 
      by(simp add: weakStepSimAct-def, blast)

      from RTrans have  $P \parallel R \Longrightarrow_{\iota} \alpha \prec P \parallel R'$  by(rule Weak-Late-Step-Semantics.Par2F)
      moreover from PRelQ R'BangRelQ' have  $(P \parallel R', Q \parallel Q') \in \text{bangRel Rel}'$ 
      by(blast intro: RelRel' Rel.BRPar)
      ultimately show ?case by blast
    qed
  qed
next
case(cComm1 a x Q' b Q'' P)
have QTrans:  $Q \mapsto a \langle x \rangle \prec Q'$  by fact
have IH:  $\bigwedge P. (P, !Q) \in \text{bangRel Rel} \Longrightarrow \text{weakStepSimAct } P (a[b] \prec Q'') P$ 
(bangRel Rel') by fact+
have  $(P, Q \parallel !Q) \in \text{bangRel Rel}$  and  $x \# P$  by fact+
thus ?case
proof(induct rule: BRParCases)
  case(BRPar P R)
    have PRelQ:  $(P, Q) \in \text{Rel}$  and RBangRelQ:  $(R, !Q) \in \text{bangRel Rel}$  by
fact+
    have  $x \# P \parallel R$  by fact
    hence xFreshP:  $x \# P$  by simp
    show ?case
    proof(induct rule: simActFreeCases)
      case Free
        from PRelQ have  $P \rightsquigarrow \langle \text{Rel}' \rangle Q$  by(rule Sim)
        with QTrans xFreshP obtain  $P' P''$  where PTrans:  $P \Longrightarrow_{\iota} b$  in  $P'' \rightarrow a \langle x \rangle$ 
        < P'
          and P'RelQ':  $(P', Q'[x::=b]) \in \text{Rel}'$ 
          by(blast dest: simE)

        from RBangRelQ have weakStepSimAct R ( $a[b] \prec Q''$ ) R (bangRel Rel')
by(rule IH)
        then obtain R' where RTrans:  $R \Longrightarrow_{\iota} a[b] \prec R'$ 
          and R'RelT':  $(R', Q'') \in \text{bangRel Rel}'$ 
          by(simp add: weakStepSimAct-def, blast)
        from PTrans RTrans have  $P \parallel R \Longrightarrow_{\iota} \tau \prec (P' \parallel R')$ 
          by(rule Weak-Late-Step-Semantics.Comm1)
        moreover from P'RelQ' R'RelT' have  $(P' \parallel R', Q'[x::=b] \parallel Q'') \in$ 
bangRel Rel'
          by(blast intro: RelRel' Rel.BRPar)
    end
  end
end

```

```

    ultimately show ?case by blast
  qed
  qed
  next
  case(cComm2 a b Q' x Q'' P)
  have QTrans:  $Q \mapsto a[b] \prec Q'$  by fact
  have IH:  $\bigwedge P. (P, !Q) \in \text{bangRel Rel} \implies \text{weakStepSimAct } P (a\langle x \rangle \prec Q'')$ 
  P (bangRel Rel')
  by fact
  have  $(P, Q \parallel !Q) \in \text{bangRel Rel}$  and  $x \# P$  by fact+
  thus ?case
  proof(induct rule: BRParCases)
  case(BRPar P R)
  have PRelQ:  $(P, Q) \in \text{Rel}$  and RBangRelQ:  $(R, !Q) \in \text{bangRel Rel}$  by
  fact+
  have  $x \# P \parallel R$  by fact
  hence xFreshR:  $x \# R$  by simp
  show ?case
  proof(induct rule: simActFreeCases)
  case Free

  from PRelQ have  $P \rightsquigarrow \langle \text{Rel}' \rangle Q$  by(rule Sim)
  with QTrans obtain  $P'$  where PTrans:  $P \implies_{\iota} a[b] \prec P'$ 
  and P'RelQ':  $(P', Q') \in \text{Rel}'$ 
  by(blast dest: simE)

  from RBangRelQ have  $\text{weakStepSimAct } R (a\langle x \rangle \prec Q'')$  R (bangRel Rel')
  by(rule IH)
  with xFreshR obtain  $R' R''$  where RTrans:  $R \implies_{\iota} b \text{ in } R'' \rightarrow a\langle x \rangle \prec R'$ 
  and R'BangRelQ'':  $(R', Q''[x::=b]) \in \text{bangRel Rel}'$ 
  by(simp add: weakStepSimAct-def, blast)

  from PTrans RTrans have  $P \parallel R \implies_{\iota\tau} \prec (P' \parallel R')$ 
  by(rule Weak-Late-Step-Semantics.Comm2)
  moreover from P'RelQ' R'BangRelQ'' have  $(P' \parallel R', Q' \parallel Q''[x::=b])$ 
   $\in \text{bangRel Rel}'$ 
  by(rule Rel.BRPar)
  ultimately show ?case by blast
  qed
  qed
  next
  case(cClose1 a x Q' y Q'' P)
  have QTrans:  $Q \mapsto a\langle x \rangle \prec Q'$  by fact
  have IH:  $\bigwedge P. (P, !Q) \in \text{bangRel Rel} \implies \text{weakStepSimAct } P (a\langle \nu y \rangle \prec Q'')$ 
  P (bangRel Rel')
  by fact
  have  $(P, Q \parallel !Q) \in \text{bangRel Rel}$  and  $x \# P$  and  $y \# P$  by fact+
  thus ?case
  proof(induct rule: BRParCases)

```

case(*BRPar P R*)
have *PRelQ*: $(P, Q) \in \text{Rel}$ **and** *RBangRelQ*: $(R, !Q) \in \text{bangRel Rel}$ **by**
fact+
have $x \# P \parallel R$ **and** $y \# P \parallel R$ **by** *fact+*
hence *xFreshP*: $x \# P$ **and** *yFreshR*: $y \# R$ **and** *yFreshP*: $y \# P$ **by** *simp+*
show *?case*
proof(*induct rule: simActFreeCases*)
case *Free*
from *PRelQ* **have** $P \rightsquigarrow \langle \text{Rel}' \rangle Q$ **by**(*rule Sim*)
with *QTrans xFreshP* **obtain** $P' P''$ **where** *PTrans*: $P \Longrightarrow_{\tau} y$ *in* $P'' \rightarrow a \langle x \rangle$
 $\prec P'$
and *P'RelQ'*: $(P', Q'[x::=y]) \in \text{Rel}'$
by(*blast dest: simE*)

from *RBangRelQ* **have** *weakStepSimAct R* $(a \langle \nu y \rangle \prec Q'') R$ (*bangRel*
Rel') **by**(*rule IH*)
with *yFreshR* **obtain** R' **where** *RTrans*: $R \Longrightarrow_{\tau} a \langle \nu y \rangle \prec R'$
and *R'BangRelQ''*: $(R', Q'') \in \text{bangRel Rel}'$
by(*simp add: weakStepSimAct-def, blast*)
from *PTrans RTrans yFreshP yFreshR* **have** $P \parallel R \Longrightarrow_{\tau} \prec \langle \nu y \rangle (P' \parallel$
 $R')$
by(*rule Weak-Late-Step-Semantics.Close1*)
moreover from *P'RelQ' R'BangRelQ''* **have** $(\langle \nu y \rangle (P' \parallel R'), \langle \nu y \rangle (Q'[x::=y]$
 $\parallel Q'')) \in \text{bangRel Rel}'$
by(*force intro: Rel.BRPar Rel.BRRes*)
ultimately show *?case by blast*
qed
qed
next
case(*cClose2 a y Q' x Q''*)
have *QTrans*: $Q \mapsto a \langle \nu y \rangle \prec Q'$ **by** *fact*
have *IH*: $\bigwedge P. (P, !Q) \in \text{bangRel Rel} \Longrightarrow \text{weakStepSimAct } P (a \langle x \rangle \prec Q'')$
 P (*bangRel Rel'*)
by *fact*
have $(P, Q \parallel !Q) \in \text{bangRel Rel}$ **and** $x \# P$ **and** $y \# P$ **by** *fact+*
thus *?case*
proof(*induct rule: BRParCases*)
case(*BRPar P R*)
have *PRelQ*: $(P, Q) \in \text{Rel}$ **and** *RBangRelQ*: $(R, !Q) \in \text{bangRel Rel}$ **by**
fact+
have $x \# P \parallel R$ **and** $y \# P \parallel R$ **by** *fact+*
hence *xFreshR*: $x \# R$ **and** *yFreshR*: $y \# R$ **and** *yFreshP*: $y \# P$ **by** *simp+*
show *?case*
proof(*induct rule: simActFreeCases*)
case *Free*
from *PRelQ* **have** $P \rightsquigarrow \langle \text{Rel}' \rangle Q$ **by**(*rule Sim*)
with *QTrans yFreshP* **obtain** P' **where** *PTrans*: $P \Longrightarrow_{\tau} a \langle \nu y \rangle \prec P'$
and *P'RelQ'*: $(P', Q') \in \text{Rel}'$
by(*blast dest: simE*)

```

from  $RBangRelQ$  have  $weakStepSimAct\ R\ (a\langle x \rangle \prec Q')$   $R\ (bangRel\ Rel')$ 
  by(rule IH)
with  $xFreshR$  obtain  $R'\ R''$  where  $RTrans: R \Longrightarrow_{\iota} y\ in\ R'' \rightarrow a\langle x \rangle \prec R'$ 
  and  $R'BangRelT': (R', Q'[x::=y]) \in bangRel\ Rel'$ 
  by(simp add: weakStepSimAct-def, blast)

from  $PTrans\ RTrans\ yFreshP\ yFreshR$  have  $P \parallel R \Longrightarrow_{\iota} \tau \prec \langle \nu y \rangle (P' \parallel$ 
 $R')$ 
  by(rule Weak-Late-Step-Semantics.Close2)
  moreover from  $P'RelQ'\ R'BangRelT'$  have  $\langle \nu y \rangle (P' \parallel R'), \langle \nu y \rangle (Q'$ 
 $\parallel Q''[x::=y]) \in bangRel\ Rel'$ 
  by(force intro: Rel.BRPar Rel.BRRes)
  ultimately show ?case by blast
qed
qed
next
case( $cBang\ Rs$ )
  have  $IH: \bigwedge P. (P, Q \parallel !Q) \in bangRel\ Rel \Longrightarrow weakStepSimAct\ P\ Rs\ P$ 
( $bangRel\ Rel'$ )
  by fact
  have  $(P, !Q) \in bangRel\ Rel$  by fact
  thus ?case
proof(induct rule: BRBangCases)
  case( $BRBang\ P$ )
  have  $PRelQ: (P, Q) \in Rel$  by fact
  hence  $(!P, !Q) \in bangRel\ Rel$  by(rule Rel.BRBang)
  with  $PRelQ$  have  $(P \parallel !P, Q \parallel !Q) \in bangRel\ Rel$  by(rule Rel.BRPar)
  hence  $weakStepSimAct\ (P \parallel !P)\ Rs\ (P \parallel !P)$  ( $bangRel\ Rel'$ ) by(rule IH)
  thus ?case
proof(simp (no-asm) add: weakStepSimAct-def, auto)
  fix  $Q'\ a\ x$ 
  assume  $weakStepSimAct\ (P \parallel !P)\ (a\langle \nu x \rangle \prec Q')\ (P \parallel !P)$  ( $bangRel\ Rel'$ )
and  $x \# P$ 
  then obtain  $P'$  where  $PTrans: (P \parallel !P) \Longrightarrow_{\iota} a\langle \nu x \rangle \prec P'$ 
  and  $P'RelQ': (P', Q') \in (bangRel\ Rel')$ 
  by(simp add: weakStepSimAct-def, blast)
  from  $PTrans$  have  $!P \Longrightarrow_{\iota} a\langle \nu x \rangle \prec P'$ 
  by(rule Weak-Late-Step-Semantics.Bang)
  with  $P'RelQ'$  show  $\exists P'. !P \Longrightarrow_{\iota} a\langle \nu x \rangle \prec P' \wedge (P', Q') \in bangRel\ Rel'$ 
by blast
  next
  fix  $Q'\ a\ x$ 
  assume  $weakStepSimAct\ (P \parallel !P)\ (a\langle x \rangle \prec Q')\ (P \parallel !P)$  ( $bangRel\ Rel'$ )
and  $x \# P$ 
  then obtain  $P''$  where  $L1: \forall u. \exists P'. P \parallel !P \Longrightarrow_{\iota} u\ in\ P'' \rightarrow a\langle x \rangle \prec P'$ 
 $\wedge (P', Q'[x::=u]) \in (bangRel\ Rel')$ 
  by(simp add: weakStepSimAct-def, blast)
  have  $\forall u. \exists P'. !P \Longrightarrow_{\iota} u\ in\ P'' \rightarrow a\langle x \rangle \prec P' \wedge (P', Q'[x::=u]) \in (bangRel$ 

```

```

Rel')
  proof(rule allI)
    fix u
    from L1 obtain P' where PTrans: P || !P ==>_l u in P'' -> a <x> < P'
      and P'RelQ': (P', Q'[x::=u]) ∈ (bangRel Rel')
    by blast
    from PTrans have !P ==>_l u in P'' -> a <x> < P' by(rule Weak-Late-Step-Semantics.Bang)
      with P'RelQ' show ∃ P'. !P ==>_l u in P'' -> a <x> < P' ∧ (P', Q'[x::=u])
    ∈ (bangRel Rel') by blast
  qed
  thus ∃ P''. ∀ u. ∃ P'. !P ==>_l u in P'' -> a <x> < P' ∧ (P', Q'[x::=u]) ∈
    (bangRel Rel') by blast
  next
  fix Q' α
  assume weakStepSimAct (P || !P) (α < Q') (P || !P) (bangRel Rel')
  then obtain P' where PTrans: (P || !P) ==>_l α < P'
    and P'RelQ': (P', Q') ∈ (bangRel Rel')
  by(simp add: weakStepSimAct-def, blast)
  from PTrans have !P ==>_l α < P'
  by(rule Weak-Late-Step-Semantics.Bang)
  with P'RelQ' show ∃ P'. !P ==>_l α < P' ∧ (P', Q') ∈ (bangRel Rel') by
blast
  qed
  qed
  qed
  qed
  moreover from PRelQ have (!P, !Q) ∈ bangRel Rel by(rule Rel.BRBang)
  ultimately show ?thesis by(simp add: weakStepSim-def)
qed
end

theory Weak-Late-Bisim-SC
  imports Weak-Late-Bisim Strong-Late-Bisim-SC
begin

```

lemma *resComm*:

fixes $P :: pi$

shows $\langle \nu a \rangle \langle \nu b \rangle P \approx \langle \nu b \rangle \langle \nu a \rangle P$

proof –

have $\langle \nu a \rangle \langle \nu b \rangle P \sim \langle \nu b \rangle \langle \nu a \rangle P$ **by**(rule *Strong-Late-Bisim-SC.resComm*)

thus *?thesis* **by**(rule *strongBisimWeakBisim*)

qed

```

lemma matchId:
  fixes a :: name
  and P :: pi

  shows  $[a \frown a]P \approx P$ 
proof -
  have  $[a \frown a]P \sim P$  by(rule Strong-Late-Bisim-SC.matchId)
  thus ?thesis by(rule strongBisimWeakBisim)
qed

```

```

lemma mismatchId:
  fixes a :: name
  and b :: name
  and P :: pi

  assumes  $a \neq b$ 

  shows  $[a \neq b]P \approx P$ 
proof -
  from assms have  $[a \neq b]P \sim P$  by(rule Strong-Late-Bisim-SC.mismatchId)
  thus ?thesis by(rule strongBisimWeakBisim)
qed

```

```

lemma mismatchZero:
  fixes a :: name
  and P :: pi

  shows  $[a \neq a]P \approx \mathbf{0}$ 
proof -
  have  $[a \neq a]P \sim \mathbf{0}$  by(rule Strong-Late-Bisim-SC.mismatchNil)
  thus ?thesis by(rule strongBisimWeakBisim)
qed

```

```

lemma sumSym:
  fixes P :: pi
  and Q :: pi

  shows  $P \oplus Q \approx Q \oplus P$ 
proof -
  have  $P \oplus Q \sim Q \oplus P$  by(rule Strong-Late-Bisim-SC.sumSym)
  thus ?thesis by(rule strongBisimWeakBisim)
qed

```

lemma *sumAssoc*:
fixes $P :: pi$
and $Q :: pi$
and $R :: pi$

shows $(P \oplus Q) \oplus R \approx P \oplus (Q \oplus R)$
proof –
have $(P \oplus Q) \oplus R \sim P \oplus (Q \oplus R)$ **by**(*rule Strong-Late-Bisim-SC.sumAssoc*)
thus *?thesis* **by**(*rule strongBisimWeakBisim*)
qed

lemma *sumZero*:
fixes $P :: pi$

shows $P \oplus \mathbf{0} \approx P$
proof –
have $P \oplus \mathbf{0} \sim P$ **by**(*rule Strong-Late-Bisim-SC.sumZero*)
thus *?thesis* **by**(*rule strongBisimWeakBisim*)
qed

lemma *parZero*:
fixes $P :: pi$

shows $P \parallel \mathbf{0} \approx P$
proof –
have $P \parallel \mathbf{0} \sim P$ **by**(*rule Strong-Late-Bisim-SC.parZero*)
thus *?thesis* **by**(*rule strongBisimWeakBisim*)
qed

lemma *parSym*:
fixes $P :: pi$
and $Q :: pi$

shows $P \parallel Q \approx Q \parallel P$
proof –
have $P \parallel Q \sim Q \parallel P$ **by**(*rule Strong-Late-Bisim-SC.parSym*)
thus *?thesis* **by**(*rule strongBisimWeakBisim*)
qed

lemma *scopeExtPar*:
fixes $P :: pi$
and $Q :: pi$
and $x :: name$

assumes $x \# P$

shows $\langle \nu x \rangle (P \parallel Q) \approx P \parallel \langle \nu x \rangle Q$

```

proof –
  from assms have  $\langle \nu x \rangle (P \parallel Q) \sim P \parallel \langle \nu x \rangle Q$  by(rule Strong-Late-Bisim-SC.scopeExtPar)
  thus ?thesis by(rule strongBisimWeakBisim)
qed

lemma scopeExtPar':
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $x :: name$ 

  assumes  $xFreshQ: x \# Q$ 

  shows  $\langle \nu x \rangle (P \parallel Q) \approx (\langle \nu x \rangle P) \parallel Q$ 
proof –
  from assms have  $\langle \nu x \rangle (P \parallel Q) \sim (\langle \nu x \rangle P) \parallel Q$  by(rule Strong-Late-Bisim-SC.scopeExtPar')
  thus ?thesis by(rule strongBisimWeakBisim)
qed

lemma parAssoc:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $R :: pi$ 

  shows  $(P \parallel Q) \parallel R \approx P \parallel (Q \parallel R)$ 
proof –
  have  $(P \parallel Q) \parallel R \sim P \parallel (Q \parallel R)$  by(rule Strong-Late-Bisim-SC.parAssoc)
  thus ?thesis by(rule strongBisimWeakBisim)
qed

lemma freshRes:
  fixes  $P :: pi$ 
  and  $a :: name$ 

  assumes  $aFreshP: a \# P$ 

  shows  $\langle \nu a \rangle P \approx P$ 
proof –
  from assms have  $\langle \nu a \rangle P \sim P$  by(rule Strong-Late-Bisim-SC.scopeFresh)
  thus ?thesis by(rule strongBisimWeakBisim)
qed

lemma scopeExtSum:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $x :: name$ 

  assumes  $x \# P$ 

  shows  $\langle \nu x \rangle (P \oplus Q) \approx P \oplus \langle \nu x \rangle Q$ 

```

```

proof –
  from assms have  $\langle \nu x \rangle (P \oplus Q) \sim P \oplus \langle \nu x \rangle Q$  by(rule Strong-Late-Bisim-SC.scopeExtSum)
  thus ?thesis by(rule strongBisimWeakBisim)
qed

lemma bangSC:
  fixes  $P$ 

  shows  $!P \approx P \parallel !P$ 
proof –
  have  $!P \sim P \parallel !P$  by(rule Strong-Late-Bisim-SC.bangSC)
  thus ?thesis by(rule strongBisimWeakBisim)
qed

end

theory Weak-Late-Sim-Pres
  imports Weak-Late-Sim
begin

lemma tauPres:
  fixes  $P \quad :: \text{pi}$ 
  and  $Q \quad :: \text{pi}$ 
  and  $Rel \quad :: (\text{pi} \times \text{pi}) \text{ set}$ 
  and  $Rel' \quad :: (\text{pi} \times \text{pi}) \text{ set}$ 

  assumes  $PRelQ: (P, Q) \in Rel$ 

  shows  $\tau.(P) \rightsquigarrow^{\wedge} \langle Rel \rangle \tau.(Q)$ 
proof(induct rule: simCases)
  case(Bound Q' a x)
  have  $\tau.(Q) \mapsto a \langle \nu x \rangle \prec Q'$  by fact
  hence False by auto
  thus ?case by simp
next
  case(Input Q' a x)
  have  $\tau.(Q) \mapsto a \langle x \rangle \prec Q'$  by fact
  hence False by auto
  thus ?case by simp
next
  case(Free Q' alpha)
  have  $\tau.(Q) \mapsto (\alpha \prec Q')$  by fact
  thus ?case using  $PRelQ$ 
proof(induct rule: tauCases, auto simp add: pi.inject residual.inject)
  have  $\tau.(P) \Longrightarrow_l^{\wedge} \tau \prec P$  by(rule Tau)
  moreover assume  $(P, Q') \in Rel$ 
  ultimately show  $\exists P'. \tau.(P) \Longrightarrow_l^{\wedge} \tau \prec P' \wedge (P', Q') \in Rel$  by blast
qed
qed

```

```

lemma inputPres:
  fixes P    :: pi
  and Q      :: pi
  and a      :: name
  and x      :: name
  and Rel    :: (pi × pi) set

  assumes PRelQ:  $\forall y. (P[x::=y], Q[x::=y]) \in Rel$ 
  and      Eqvt: eqvt Rel

  shows  $a \langle x \rangle . P \rightsquigarrow^{\langle Rel \rangle} a \langle x \rangle . Q$ 
proof -
  show ?thesis using Eqvt
proof(induct rule: simCasesCont[of - (P, a, x, Q)])
  case(Bound Q' b y)
  have  $a \langle x \rangle . Q \mapsto b \langle \nu y \rangle \prec Q'$  by fact
  hence False by auto
  thus ?case by simp
next
  case(Input Q' b y)
  have  $y \# (P, a, x, Q)$  by fact
  hence  $yFreshP: (y::name) \# P$  and  $yineqx: y \neq x$  and  $y \neq a$  and  $y \# Q$ 
  by(simp add: fresh-prod)+
  have  $a \langle x \rangle . Q \mapsto b \langle y \rangle \prec Q'$  by fact
  thus ?case using  $\langle y \neq a \rangle \langle y \neq x \rangle \langle y \# Q \rangle$ 
  proof(induct rule: inputCases, auto simp add: subject.inject)
  have  $\forall u. \exists P'. a \langle x \rangle . P \Longrightarrow_{lu} in ((x, y) \cdot P) \rightarrow a \langle y \rangle \prec P' \wedge (P', ((x, y) \cdot Q)[y::=u]) \in Rel$ 
  proof(rule allI)
  fix u
  have  $a \langle x \rangle . P \Longrightarrow_{lu} in ((x, y) \cdot P) \rightarrow a \langle y \rangle \prec ((x, y) \cdot P)[y::=u]$  (is
  ?goal)
  proof -
  from  $yFreshP$  have  $a \langle x \rangle . P = a \langle y \rangle . ((x, y) \cdot P)$  by(rule Agent.alphaInput)
  moreover have  $a \langle y \rangle . ((x, y) \cdot P) \Longrightarrow_{lu} in ((x, y) \cdot P) \rightarrow a \langle y \rangle \prec ((x, y) \cdot P)[y::=u]$ 
  by(rule Weak-Late-Step-Semantics.Input)
  ultimately show ?goal by(simp add: name-swap)
  qed
  moreover have  $((x, y) \cdot P)[y::=u], ((x, y) \cdot Q)[y::=u] \in Rel$ 
  proof -
  from PRelQ have  $(P[x::=u], Q[x::=u]) \in Rel$  by auto
  with  $\langle y \# P \rangle \langle y \# Q \rangle$  show ?thesis by(simp add: renaming)
  qed
  ultimately show  $\exists P'. a \langle x \rangle . P \Longrightarrow_{lu} in ((x, y) \cdot P) \rightarrow a \langle y \rangle \prec P' \wedge (P', ((x, y) \cdot Q)[y::=u]) \in Rel$ 

```

```

    by blast
qed

    thus  $\exists P''. \forall u. \exists P'. a\langle x \rangle.P \Longrightarrow_{Iu} \text{in } P'' \rightarrow a\langle y \rangle \prec P' \wedge (P', [(x, y)] \cdot Q)[y::=u]) \in \text{Rel}$  by blast
    qed
  next
    case(Free Q'  $\alpha$ )
    have  $a\langle x \rangle.Q \mapsto \alpha \prec Q'$  by fact
    hence False by auto
    thus ?case by simp
  qed
qed

```

lemma *outputPres*:

```

fixes P   :: pi
and Q     :: pi
and a     :: name
and b     :: name
and Rel   :: (pi  $\times$  pi) set
and Rel'  :: (pi  $\times$  pi) set

```

assumes *PRelQ*: $(P, Q) \in \text{Rel}$

```

shows  $a\{b\}.P \rightsquigarrow^{\wedge} \langle \text{Rel} \rangle a\{b\}.Q$ 
proof(induct rule: simCases)
  case(Bound Q' c x)
  have  $a\{b\}.Q \mapsto c\langle \nu x \rangle \prec Q'$  by fact
  hence False by auto
  thus ?case by simp
next
  case(Input Q' c x)
  have  $a\{b\}.Q \mapsto c\langle x \rangle \prec Q'$  by fact
  hence False by auto
  thus ?case by simp
next
  case(Free Q'  $\alpha$ )
  have  $a\{b\}.Q \mapsto \alpha \prec Q'$  by fact
  thus  $\exists P'. a\{b\}.P \Longrightarrow_{I^{\wedge}} \alpha \prec P' \wedge (P', Q') \in \text{Rel}$  using PRelQ
  proof(induct rule: outputCases, auto simp add: pi.inject residual.inject)
    have  $a\{b\}.P \Longrightarrow_{I^{\wedge}} a[b] \prec P$  by(rule Output)
    moreover assume  $(P, Q') \in \text{Rel}$ 
    ultimately show  $\exists P'. a\{b\}.P \Longrightarrow_{I^{\wedge}} a[b] \prec P' \wedge (P', Q') \in \text{Rel}$  by blast
  qed
qed

```

lemma *matchPres*:

```

fixes P   :: pi
and Q     :: pi

```

```

and  $a :: \text{name}$ 
and  $b :: \text{name}$ 
and  $\text{Rel} :: (\text{pi} \times \text{pi}) \text{ set}$ 
and  $\text{Rel}' :: (\text{pi} \times \text{pi}) \text{ set}$ 

assumes  $\text{PSimQ}: P \rightsquigarrow^{\wedge} \langle \text{Rel} \rangle Q$ 
and  $\text{RelStay}: \bigwedge P Q a. (P, Q) \in \text{Rel} \implies ([a \frown a]P, Q) \in \text{Rel}$ 
and  $\text{RelRel}': \text{Rel} \subseteq \text{Rel}'$ 

shows  $[a \frown b]P \rightsquigarrow^{\wedge} \langle \text{Rel}' \rangle [a \frown b]Q$ 
proof(induct rule: simCases)
  case(Bound  $Q' c x$ )
    have  $x \# [a \frown b]P$  by fact
    hence  $x\text{Fresh}P: (x::\text{name}) \# P$  by simp
    have  $[a \frown b]Q \mapsto c \langle \nu x \rangle \prec Q'$  by fact
    thus ?case
  proof(induct rule: matchCases)
    case cMatch
      have  $Q \mapsto c \langle \nu x \rangle \prec Q'$  by fact
      with  $\text{PSimQ } x\text{Fresh}P$  obtain  $P'$  where  $P\text{Trans}: P \implies_i \hat{c} \langle \nu x \rangle \prec P'$ 
        and  $P'\text{Rel}Q': (P', Q') \in \text{Rel}$ 
      by(blast dest: simE)
      from  $P\text{Trans}$  have  $[a \frown a]P \implies_i \hat{c} \langle \nu x \rangle \prec P'$  by(rule Weak-Late-Semantics.Match)
      with  $P'\text{Rel}Q' \text{RelRel}'$  show ?case by blast
    qed
  next
    case(Input  $Q' c x$ )
      have  $x \# [a \frown b]P$  by fact
      hence  $x\text{Fresh}P: x \# P$  by simp
      have  $[a \frown b]Q \mapsto c \langle x \rangle \prec Q'$  by fact
      thus ?case
    proof(induct rule: matchCases)
      case cMatch
        have  $Q \mapsto c \langle x \rangle \prec Q'$  by fact
        with  $\text{PSimQ } x\text{Fresh}P$  obtain  $P''$  where  $L1: \forall u. \exists P'. P \implies_{!u} \text{in } P'' \rightarrow c \langle x \rangle$ 
           $\prec P' \wedge (P', Q'[x::=u]) \in \text{Rel}$ 
        by(force intro: simE)
        have  $\forall u. \exists P'. [a \frown a]P \implies_{!u} \text{in } P'' \rightarrow c \langle x \rangle \prec P' \wedge (P', Q'[x::=u]) \in \text{Rel}'$ 
        proof(rule allI)
          fix  $u$ 
          from  $L1$  obtain  $P'$  where  $P\text{Trans}: P \implies_{!u} \text{in } P'' \rightarrow c \langle x \rangle \prec P'$  and  $P'\text{Rel}Q':$ 
             $(P', Q'[x::=u]) \in \text{Rel}$ 
          by blast
          from  $P\text{Trans}$  have  $[a \frown a]P \implies_{!u} \text{in } P'' \rightarrow c \langle x \rangle \prec P'$  by(rule Weak-Late-Step-Semantics.Match)
          with  $P'\text{Rel}Q' \text{RelRel}'$  show  $\exists P'. [a \frown a]P \implies_{!u} \text{in } P'' \rightarrow c \langle x \rangle \prec P' \wedge (P',$ 
             $Q'[x::=u]) \in \text{Rel}'$ 
          by blast
        qed
      thus ?case by blast
    qed
  thus ?case by blast

```

```

qed
next
case(Free Q' α)
have [a↔b]Q ⟶ α < Q' by fact
thus ?case
proof(induct rule: matchCases)
  case cMatch
  have Q ⟶ α < Q' by fact
  with PSimQ obtain P' where PTrans: P ⟶l α < P' and PRel: (P', Q') ∈
Rel
  by(blast dest: simE)
  from PTrans show ?case
  proof(induct rule: transitionCases)
    case Step
    have P ⟶l α < P' by fact
    hence [a↔a]P ⟶l α < P' by(rule Weak-Late-Step-Semantics.Match)
    with PRel RelRel' show ?case by(force simp add: weakTransition-def)
  next
  case Stay
  have α < P' = τ < P by fact
  hence alphaEqTau: α = τ and PeqP': P = P' by(simp add: residual.inject)+
  have [a↔a]P ⟶l τ < [a↔a]P by(simp add: weakTransition-def)
  moreover from PeqP' PRel have ([a↔a]P, Q') ∈ Rel by(blast intro: RelStay)
  ultimately show ?case using RelRel' alphaEqTau by blast
  qed
qed
qed

lemma mismatchPres:
  fixes P :: pi
  and Q :: pi
  and a :: name
  and b :: name
  and Rel :: (pi × pi) set
  and Rel' :: (pi × pi) set

  assumes PSimQ: P ~>^<Rel> Q
  and RelStay: ⋀P Q a b. [(P, Q) ∈ Rel; a ≠ b] ⟹ ([a≠b]P, Q) ∈ Rel
  and RelRel': Rel ⊆ Rel'

  shows [a≠b]P ~>^<Rel'> [a≠b]Q
proof(cases a = b)
  assume a = b
  thus ?thesis by(auto simp add: weakSimulation-def)
next
  assume aineqb: a ≠ b
  show ?thesis
  proof(induct rule: simCases)
    case(Bound Q' c x)

```

```

have  $x \# [a \neq b]P$  by fact
hence  $xFreshP: (x::name) \# P$  by simp
have  $[a \neq b]Q \mapsto c \langle \nu x \rangle \prec Q'$  by fact
thus ?case
proof(induct rule: mismatchCases)
  case cMismatch
  have  $Q \mapsto c \langle \nu x \rangle \prec Q'$  by fact
  with  $PSimQ$   $xFreshP$  obtain  $P'$  where  $PTrans: P \Longrightarrow_i \hat{c} \langle \nu x \rangle \prec P'$ 
  and  $P'RelQ': (P', Q') \in Rel$ 
  by(blast dest: simE)
  from  $PTrans$   $aineqb$  have  $[a \neq b]P \Longrightarrow_i \hat{c} \langle \nu x \rangle \prec P'$  by(rule Weak-Late-Semantics.Mismatch)
  with  $P'RelQ'$   $RelRel'$  show ?case by blast
qed
next
case(Input  $Q' c x$ )
have  $x \# [a \neq b]P$  by fact
hence  $xFreshP: x \# P$  by simp
have  $[a \neq b]Q \mapsto c \langle x \rangle \prec Q'$  by fact
thus ?case
proof(induct rule: mismatchCases)
  case cMismatch
  have  $Q \mapsto c \langle x \rangle \prec Q'$  by fact
  with  $PSimQ$   $xFreshP$  obtain  $P''$  where  $L1: \forall u. \exists P'. P \Longrightarrow_{Iu} in P'' \rightarrow c \langle x \rangle$ 
 $\prec P' \wedge (P', Q'[x::=u]) \in Rel$ 
  by(force intro: simE)
  have  $\forall u. \exists P'. [a \neq b]P \Longrightarrow_{Iu} in P'' \rightarrow c \langle x \rangle \prec P' \wedge (P', Q'[x::=u]) \in Rel'$ 
  proof(rule allI)
    fix  $u$ 
    from  $L1$  obtain  $P'$  where  $PTrans: P \Longrightarrow_{Iu} in P'' \rightarrow c \langle x \rangle \prec P'$  and
 $P'RelQ': (P', Q'[x::=u]) \in Rel$ 
    by blast
    from  $PTrans$   $aineqb$  have  $[a \neq b]P \Longrightarrow_{Iu} in P'' \rightarrow c \langle x \rangle \prec P'$  by(rule
Weak-Late-Step-Semantics.Mismatch)
    with  $P'RelQ'$   $RelRel'$  show  $\exists P'. [a \neq b]P \Longrightarrow_{Iu} in P'' \rightarrow c \langle x \rangle \prec P' \wedge (P',
Q'[x::=u]) \in Rel'$ 
    by blast
  qed
  thus ?case by blast
qed
next
case(Free  $Q' \alpha$ )
have  $[a \neq b]Q \mapsto \alpha \prec Q'$  by fact
thus ?case
proof(induct rule: mismatchCases)
  case cMismatch
  have  $a \neq b$  by fact
  have  $Q \mapsto \alpha \prec Q'$  by fact
  with  $PSimQ$  obtain  $P'$  where  $PTrans: P \Longrightarrow_i \hat{\alpha} \prec P'$  and  $P'Rel: (P', Q')
\in Rel$ 

```

```

    by(blast dest: simE)
  from PTrans show ?case
  proof(induct rule: transitionCases)
    case Step
    have  $P \Rightarrow_1 \alpha \prec P'$  by fact
  hence  $[a \neq b]P \Rightarrow_1 \alpha \prec P'$  using  $\langle a \neq b \rangle$  by(rule Weak-Late-Step-Semantics.Mismatch)
    with PRel RelRel' show ?case by(force simp add: weakTransition-def)
  next
    case Stay
    have  $\alpha \prec P' = \tau \prec P$  by fact
  hence alphaEqTau:  $\alpha = \tau$  and PegP':  $P = P'$  by(simp add: residual.inject)+
    have  $[a \neq b]P \Rightarrow_1 \hat{\tau} \prec [a \neq b]P$  by(simp add: weakTransition-def)
    moreover from PegP' PRel aineqb have  $([a \neq b]P, Q') \in Rel$  by(blast intro:
  RelStay)
    ultimately show ?case using alphaEqTau RelRel' by blast
  qed
  qed
  qed
  qed

```

lemma parCompose:

```

  fixes P    :: pi
  and Q      :: pi
  and R      :: pi
  and T      :: pi
  and Rel    :: (pi × pi) set
  and Rel'   :: (pi × pi) set
  and Rel''  :: (pi × pi) set

  assumes PSimQ:   $P \rightsquigarrow^{\wedge} \langle Rel \rangle Q$ 
  and RSimT:      $R \rightsquigarrow^{\wedge} \langle Rel' \rangle T$ 
  and PRelQ:      $(P, Q) \in Rel$ 
  and RRel'T:     $(R, T) \in Rel'$ 
  and Par:        $\bigwedge P Q R T. \llbracket (P, Q) \in Rel; (R, T) \in Rel' \rrbracket \implies (P \parallel R, Q \parallel T) \in Rel''$ 
  and Res:        $\bigwedge P Q a. (P, Q) \in Rel'' \implies (\langle \nu a \rangle P, \langle \nu a \rangle Q) \in Rel''$ 
  and EqvtRel:   eqvt Rel
  and EqvtRel': eqvt Rel'
  and EqvtRel'': eqvt Rel''

```

```

  shows  $P \parallel R \rightsquigarrow^{\wedge} \langle Rel'' \rangle Q \parallel T$ 
  using  $\langle eqvt Rel'' \rangle$ 
  proof(induct rule: simCasesCont[where C=(P, Q, R, T)])
    case(Bound Q' a x)
  from  $\langle x \# (P, Q, R, T) \rangle$  have  $x \# P$  and  $x \# R$  and  $x \# Q$  and  $x \# T$  by simp+
  from  $\langle Q \parallel T \mapsto a \langle \nu x \rangle \prec Q' \rangle \langle x \# Q \rangle \langle x \# T \rangle$ 
  show ?case
  proof(induct rule: parCasesB)
    case(cPar1 Q')

```

from $PSimQ \langle Q \mapsto a \langle \nu x \rangle \prec Q' \rangle \langle x \# P \rangle$ **obtain** P' **where** $PTrans: P \Longrightarrow_i \hat{a} \langle \nu x \rangle \prec P'$

and $P'RelQ': (P', Q') \in Rel$

by(*blast dest: simE*)

from $PTrans \langle x \# R \rangle$ **have** $P \parallel R \Longrightarrow_i \hat{a} \langle \nu x \rangle \prec (P' \parallel R)$ **by**(*rule Weak-Late-Semantics.Par1B*)

moreover from $P'RelQ' RRel'T$ **have** $(P' \parallel R, Q' \parallel T) \in Rel''$ **by**(*rule Par*)

ultimately show *?case by blast*

next

case(*cPar2 T'*)

from $RSimT \langle T \mapsto a \langle \nu x \rangle \prec T' \rangle \langle x \# R \rangle$ **obtain** R' **where** $RTrans: R \Longrightarrow_i \hat{a} \langle \nu x \rangle \prec R'$

and $R'Rel'T': (R', T') \in Rel'$

by(*blast dest: simE*)

from $RTrans \langle x \# P \rangle \langle x \# R \rangle$ **have** $ParTrans: P \parallel R \Longrightarrow_i \hat{a} \langle \nu x \rangle \prec (P \parallel R')$

by(*blast intro: Weak-Late-Semantics.Par2B*)

moreover from $P'RelQ' R'Rel'T'$ **have** $(P \parallel R', Q' \parallel T') \in Rel''$ **by**(*rule Par*)

ultimately show *?case by blast*

qed

next

case(*Input Q' a x*)

from $\langle x \# (P, Q, R, T) \rangle$ **have** $x \# P$ **and** $x \# R$ **and** $x \# Q$ **and** $x \# T$ **by** *simp+*

from $\langle Q \parallel T \mapsto a \langle x \rangle \prec Q' \rangle \langle x \# Q \rangle \langle x \# T \rangle$

show *?case*

proof(*induct rule: parCasesB*)

case(*cPar1 Q'*)

from $PSimQ \langle Q \mapsto a \langle x \rangle \prec Q' \rangle \langle x \# P \rangle$ **obtain** P''

where $L1: \forall u. \exists P'. P \Longrightarrow_{iu} in P'' \rightarrow a \langle x \rangle \prec P' \wedge (P', Q'[x::=u]) \in Rel$

by(*blast dest: simE*)

have $\forall u. \exists P'. P \parallel R \Longrightarrow_{iu} in (P'' \parallel R) \rightarrow a \langle x \rangle \prec P' \wedge (P', Q'[x::=u] \parallel T[x::=u]) \in Rel''$

proof(*rule allI*)

fix u

from $L1$ **obtain** P' **where** $PTrans: P \Longrightarrow_{iu} in P'' \rightarrow a \langle x \rangle \prec P'$

and $P'RelQ': (P', Q'[x::=u]) \in Rel$ **by** *blast*

from $PTrans \langle x \# R \rangle$ **have** $P \parallel R \Longrightarrow_{iu} in (P'' \parallel R) \rightarrow a \langle x \rangle \prec (P' \parallel R)$

by(*rule Weak-Late-Step-Semantics.Par1B*)

moreover from $P'RelQ' RRel'T$ **have** $(P' \parallel R, Q'[x::=u] \parallel T) \in Rel''$

by(*rule Par*)

ultimately show $\exists P'. P \parallel R \Longrightarrow_{iu} in (P'' \parallel R) \rightarrow a \langle x \rangle \prec P' \wedge (P', Q'[x::=u] \parallel (T[x::=u])) \in Rel''$ **using** $\langle x \# T \rangle$

by(*force simp add: forget*)

qed

thus *?case by force*

next

case(*cPar2 T'*)

from $RSimT \langle T \mapsto a \langle x \rangle \prec T' \rangle \langle x \# R \rangle$ **obtain** R''

where $L1: \forall u. \exists R'. R \Longrightarrow_{iu} in R'' \rightarrow a \langle x \rangle \prec R' \wedge (R', T'[x::=u]) \in Rel'$

by(*blast dest: simE*)

have $\forall u. \exists P'. P \parallel R \Longrightarrow_{iu} in (P \parallel R'') \rightarrow a \langle x \rangle \prec P' \wedge (P', Q[x::=u] \parallel T[x::=u]) \in Rel''$

$T'[x::=u] \in Rel''$
proof(rule allI)
fix u
from $L1$ **obtain** R' **where** $RTrans: R \Longrightarrow_l u$ **in** $R'' \rightarrow a \langle x \rangle \prec R'$
and $R'Rel'T': (R', T'[x::=u]) \in Rel'$ **by** *blast*
from $RTrans \langle x \# P \rangle$ **have** $ParTrans: P \parallel R \Longrightarrow_l u$ **in** $(P \parallel R'') \rightarrow a \langle x \rangle \prec$
 $(P \parallel R')$
by(rule *Weak-Late-Step-Semantics.Par2B*)

moreover from $PRelQ R'Rel'T'$ **have** $(P \parallel R', Q \parallel T'[x::=u]) \in Rel''$
by(rule *Par*)

ultimately show $\exists P'. P \parallel R \Longrightarrow_l u$ **in** $(P \parallel R'') \rightarrow a \langle x \rangle \prec P' \wedge$
 $(P', Q[x::=u] \parallel T'[x::=u]) \in Rel''$ **using** $\langle x \# Q \rangle$
by(force *simp add: forget*)
qed
thus *?case by force*
qed
next
case(*Free QT' α*)
have $Q \parallel T \mapsto \alpha \prec QT'$ **by** *fact*
thus *?case*
proof(*induct rule: parCasesF[of - - - - (P, R)]*)
case(*cPar1 Q'*)
have $Q \mapsto \alpha \prec Q'$ **by** *fact*
with $PSimQ$ **obtain** P' **where** $PTrans: P \Longrightarrow_l \hat{\alpha} \prec P'$ **and** $PRel: (P', Q')$
 $\in Rel$
by(*blast dest: simE*)
from $PTrans$ **have** $Trans: P \parallel R \Longrightarrow_l \hat{\alpha} \prec P' \parallel R$ **by**(rule *Weak-Late-Semantics.Par1F*)
moreover from $PRel RRel'T$ **have** $(P' \parallel R, Q' \parallel T) \in Rel''$ **by**(*blast intro:*
Par)
ultimately show *?case by blast*
next
case(*cPar2 T'*)
have $T \mapsto \alpha \prec T'$ **by** *fact*
with $RSimT$ **obtain** R' **where** $RTrans: R \Longrightarrow_l \hat{\alpha} \prec R'$ **and** $RRel: (R', T')$
 $\in Rel'$
by(*blast dest: simE*)
from $RTrans$ **have** $Trans: P \parallel R \Longrightarrow_l \hat{\alpha} \prec P \parallel R'$ **by**(rule *Weak-Late-Semantics.Par2F*)
moreover from $PRelQ RRel$ **have** $(P \parallel R', Q \parallel T') \in Rel''$ **by**(*blast intro:*
Par)
ultimately show *?case by blast*
next
case(*cComm1 Q' T' a b x*)
have $QTrans: Q \mapsto a \langle x \rangle \prec Q'$ **and** $TTrans: T \mapsto a[b] \prec T'$ **by** *fact+*
have $x \# (P, R)$ **by** *fact*
hence $xFreshP: x \# P$ **by**(*simp add: fresh-prod*)

from $PSimQ QTrans xFreshP$ **obtain** $P' P''$ **where** $PTrans: P \Longrightarrow_l b$ **in**

$P'' \rightarrow a \langle x \rangle \prec P'$
and $P'RelQ': (P', Q'[x::=b]) \in Rel$
by(blast dest: simE)

from $RSimT TTrans$ **obtain** R' **where** $RTrans: R \Rightarrow_l \hat{a}[b] \prec R'$
and $RRel: (R', T') \in Rel'$
by(blast dest: simE)

from $PTrans RTrans$ **have** $P \parallel R \Rightarrow_l \hat{\tau} \prec P' \parallel R'$ **by**(rule Weak-Late-Semantics.Comm1)
moreover from $P'RelQ' RRel$ **have** $(P' \parallel R', Q'[x::=b] \parallel T') \in Rel''$ **by**(rule Par)

ultimately show ?case **by** blast
next
case(cComm2 $Q' T' a b x$)
have $QTrans: Q \mapsto a[b] \prec Q'$ **and** $TTrans: T \mapsto a \langle x \rangle \prec T'$ **by** fact+
have $x \# (P, R)$ **by** fact
hence $xFreshR: x \# R$ **by**(simp add: fresh-prod)

from $PSimQ QTrans$ **obtain** P' **where** $PTrans: P \Rightarrow_l \hat{a}[b] \prec P'$
and $PRel: (P', Q') \in Rel$
by(blast dest: simE)

from $RSimT TTrans xFreshR$ **obtain** $R' R''$ **where** $RTrans: R \Rightarrow_l b$ in
 $R'' \rightarrow a \langle x \rangle \prec R'$
and $R'Rel'T': (R', T'[x::=b]) \in Rel'$
by(blast dest: simE)

from $PTrans RTrans$ **have** $P \parallel R \Rightarrow_l \hat{\tau} \prec P' \parallel R'$ **by**(rule Weak-Late-Semantics.Comm2)
moreover from $PRel R'Rel'T'$ **have** $(P' \parallel R', Q' \parallel T'[x::=b]) \in Rel''$ **by**(rule Par)

ultimately show ?case **by** blast
next
case(cClose1 $Q' T' a x y$)
have $QTrans: Q \mapsto a \langle x \rangle \prec Q'$ **and** $TTrans: T \mapsto a \langle \nu y \rangle \prec T'$ **by** fact+
have $x \# (P, R)$ **and** $y \# (P, R)$ **by** fact+
hence $xFreshP: x \# P$ **and** $yFreshR: y \# R$ **and** $yFreshP: y \# P$ **by**(simp add: fresh-prod)+

from $PSimQ QTrans xFreshP$ **obtain** $P' P''$ **where** $PTrans: P \Rightarrow_l y$ in
 $P'' \rightarrow a \langle x \rangle \prec P'$
and $P'RelQ': (P', Q'[x::=y]) \in Rel$
by(blast dest: simE)

from $RSimT TTrans yFreshR$ **obtain** R' **where** $RTrans: R \Rightarrow_l \hat{a} \langle \nu y \rangle \prec R'$
and $R'Rel'T': (R', T') \in Rel'$
by(blast dest: simE)

from $PTrans RTrans yFreshP yFreshR$ **have** $Trans: P \parallel R \Rightarrow_l \hat{\tau} \prec \langle \nu y \rangle (P'$

$\parallel R')$
by(*rule Weak-Late-Semantics.Close1*)
moreover from $P' \text{Rel} Q' R' \text{Rel}' T'$ **have** $(\langle \nu y \rangle (P' \parallel R'), \langle \nu y \rangle (Q'[x::=y] \parallel T')) \in \text{Rel}''$
by(*blast intro: Par Res*)
ultimately show *?case by blast*
next
case(*cClose2 Q' T' a x y*)
have $Q \text{Trans}: Q \mapsto a \langle \nu y \rangle \prec Q'$ **and** $T \text{Trans}: T \mapsto a \langle x \rangle \prec T'$ **by** *fact+*
have $x \# (P, R)$ **and** $y \# (P, R)$ **by** *fact+*
hence $x \text{Fresh} R: x \# R$ **and** $y \text{Fresh} P: y \# P$ **and** $y \text{Fresh} R: y \# R$ **by**(*simp add: fresh-prod*)

from $PSimQ Q \text{Trans} y \text{Fresh} P$ **obtain** P' **where** $P \text{Trans}: P \Longrightarrow_l \hat{a} \langle \nu y \rangle \prec P'$
and $P' \text{Rel} Q': (P', Q') \in \text{Rel}$
by(*blast dest: simE*)

from $R \text{Sim} T T \text{Trans} x \text{Fresh} R$ **obtain** $R' R''$ **where** $R \text{Trans}: R \Longrightarrow_l y$ *in*
 $R'' \rightarrow a \langle x \rangle \prec R'$
and $R' \text{Rel}' T': (R', T'[x::=y]) \in \text{Rel}'$
by(*blast dest: simE*)

from $P \text{Trans} R \text{Trans} y \text{Fresh} P y \text{Fresh} R$ **have** $\text{Trans}: P \parallel R \Longrightarrow_l \hat{\tau} \prec \langle \nu y \rangle (P' \parallel R')$
 $\parallel R')$
by(*rule Weak-Late-Semantics.Close2*)
moreover from $P' \text{Rel} Q' R' \text{Rel}' T'$ **have** $(\langle \nu y \rangle (P' \parallel R'), \langle \nu y \rangle (Q' \parallel T'[x::=y])) \in \text{Rel}''$
by(*blast intro: Par Res*)
ultimately show *?case by blast*
qed
qed

lemma *parPres:*

fixes $P :: pi$
and $Q :: pi$
and $R :: pi$
and $a :: name$
and $b :: name$
and $\text{Rel} :: (pi \times pi) \text{ set}$
and $\text{Rel}' :: (pi \times pi) \text{ set}$

assumes $PSimQ: P \rightsquigarrow \hat{\langle \text{Rel} \rangle} Q$
and $P \text{Rel} Q: (P, Q) \in \text{Rel}$
and $\text{Par}: \bigwedge P Q R. (P, Q) \in \text{Rel} \Longrightarrow (P \parallel R, Q \parallel R) \in \text{Rel}'$
and $\text{Res}: \bigwedge P Q a. (P, Q) \in \text{Rel}' \Longrightarrow (\langle \nu a \rangle P, \langle \nu a \rangle Q) \in \text{Rel}'$
and $\text{EqvtRel}: \text{eqvt } \text{Rel}$
and $\text{EqvtRel}': \text{eqvt } \text{Rel}'$

shows $P \parallel R \rightsquigarrow \hat{\langle \text{Rel}' \rangle} Q \parallel R$

proof –
note $PSimQ$
moreover have $RSimR: R \rightsquigarrow^{\widehat{\langle Id \rangle}} R$ **by** (*auto intro: reflexive*)
moreover note $PRelQ$ **moreover have** $(R, R) \in Id$ **by** *auto*
moreover from Par **have** $\bigwedge P Q R T. \llbracket (P, Q) \in Rel; (R, T) \in Id \rrbracket \implies (P \parallel R, Q \parallel T) \in Rel'$
by *auto*
moreover note $Res \langle eqvt Rel \rangle$
moreover have $eqvt Id$ **by** (*auto simp add: eqvt-def*)
ultimately show *?thesis* **using** $EqvtRel'$ **by** (*rule parCompose*)
qed

lemma $resPres$:

fixes $P :: pi$
and $Q :: pi$
and $Rel :: (pi \times pi)$ *set*
and $x :: name$
and $Rel' :: (pi \times pi)$ *set*

assumes $PSimQ: P \rightsquigarrow^{\widehat{\langle Rel \rangle}} Q$
and $ResRel: \bigwedge (P::pi) (Q::pi) (x::name). (P, Q) \in Rel \implies (\langle \nu x \rangle P, \langle \nu x \rangle Q) \in Rel'$
and $RelRel': Rel \subseteq Rel'$
and $EqvtRel: eqvt Rel$
and $EqvtRel': eqvt Rel'$

shows $\langle \nu x \rangle P \rightsquigarrow^{\widehat{\langle Rel' \rangle}} \langle \nu x \rangle Q$

proof –

from $EqvtRel'$ **show** *?thesis*
proof (*induct rule: simCasesCont[of - (P, Q, x)]*)
case ($Bound Q' a y$)
have $Trans: \langle \nu x \rangle Q \mapsto a \langle \nu y \rangle \prec Q'$ **by** *fact*
have $y \# (P, Q, x)$ **by** *fact*
hence $y \# x$ **and** $y \# P$ **and** $y \# Q$ **by** (*simp add: fresh-prod*)
from $Trans \langle y \# x \rangle \langle y \# Q \rangle$ **show** *?case*
proof (*induct rule: resCasesB*)
case ($cOpen a Q'$)
have $QTrans: Q \mapsto a[x] \prec Q'$ **and** $a \# x$ **by** *fact*

from $PSimQ QTrans$ **obtain** P' **where** $PTrans: P \Longrightarrow_i \widehat{a[x]} \prec P'$
and $P'RelQ': (P', Q') \in Rel$
by (*blast dest: simE*)

have $\langle \nu x \rangle P \Longrightarrow_i \widehat{a \langle \nu y \rangle} \prec ((y, x) \cdot P')$

proof –

from $PTrans a \# x$ **have** $\langle \nu x \rangle P \Longrightarrow_i \widehat{a \langle \nu x \rangle} \prec P'$ **by** (*rule Weak-Late-Semantics.Open*)
moreover from $PTrans y \# P$ **have** $y \# P'$ **by** (*force intro: freshTransition*)
ultimately show *?thesis* **by** (*simp add: alphaBoundResidual name-swap*)
qed

moreover from $\text{EqvtRel } P' \text{Rel} Q' \text{Rel} \text{Rel}'$ **have** $([(y, x)] \cdot P', [(y, x)] \cdot Q') \in \text{Rel}'$
by (*blast intro: eqvtRelI*)
ultimately show *?case by blast*
next
case (*cRes Q'*)
have $Q \text{Trans}: Q \mapsto a \langle \nu y \rangle \prec Q'$ **by fact**
from $\langle x \# \text{BoundOutputS } a \rangle$ **have** $x \neq a$ **by simp**

from $\text{PSimQ } y \text{FreshP } Q \text{Trans}$ **obtain** P' **where** $P \text{Trans}: P \Longrightarrow_l \hat{a} \langle \nu y \rangle \prec$
 P' **and** $P' \text{Rel} Q': (P', Q') \in \text{Rel}$
by (*blast dest: simE*)
from $P \text{Trans} \langle x \neq a \rangle y \text{ineqx } y \text{FreshP}$ **have** $\text{ResTrans}: \langle \nu x \rangle P \Longrightarrow_l \hat{a} \langle \nu y \rangle$
 $\prec (\langle \nu x \rangle P')$
by (*blast intro: Weak-Late-Semantics.ResB*)
moreover from $P' \text{Rel} Q'$ **have** $(\langle \nu x \rangle P', \langle \nu x \rangle Q') \in \text{Rel}'$
by (*rule ResRel*)
ultimately show *?case by blast*
qed
next
case (*Input Q' a y*)
have $y \# (P, Q, x)$ **by fact**
hence $y \text{ineqx}: y \neq x$ **and** $y \text{FreshP}: y \# P$ **and** $y \# Q$ **by** (*simp add: fresh-prod*)

have $\langle \nu x \rangle Q \mapsto a \langle y \rangle \prec Q'$ **by fact**
thus *?case using yineqx* $\langle y \# Q \rangle$
proof (*induct rule: resCasesB*)
case (*cOpen a Q'*)
thus *?case by simp*
next
case (*cRes Q'*)
have $Q \text{Trans}: Q \mapsto a \langle y \rangle \prec Q'$ **by fact**
from $\langle x \# \text{InputS } a \rangle$ **have** $x \neq a$ **by simp**

from $\text{PSimQ } Q \text{Trans } y \text{FreshP}$ **obtain** P''
where $L1: \forall u. \exists P'. P \Longrightarrow_l u \text{ in } P'' \rightarrow a \langle y \rangle \prec P' \wedge (P', Q'[y::=u]) \in \text{Rel}$
by (*blast dest: simE*)
have $\forall u. \exists P'. \langle \nu x \rangle P \Longrightarrow_l u \text{ in } (\langle \nu x \rangle P'') \rightarrow a \langle y \rangle \prec P' \wedge (P', (\langle \nu x \rangle Q')[y::=u])$
 $\in \text{Rel}'$
proof (*rule allI*)
fix u
show $\exists P'. \langle \nu x \rangle P \Longrightarrow_l u \text{ in } \langle \nu x \rangle P'' \rightarrow a \langle y \rangle \prec P' \wedge (P', (\langle \nu x \rangle Q')[y::=u])$
 $\in \text{Rel}'$
proof (*cases x=u*)
assume $x \text{equ}: x=u$

have $\exists c::\text{name}. c \# (P, P'', Q', x, y, a)$ **by** (*blast intro: name-exists-fresh*)
then obtain $c::\text{name}$ **where** $c \text{FreshP}: c \# P$ **and** $c \text{FreshP}'': c \# P''$ **and**

$cFreshQ': c \# Q'$
and $cineqx: c \neq x$ **and** $cineqy: c \neq y$ **and** $cineqa: c \neq a$
by(*force simp add: fresh-prod*)

from $L1$ **obtain** P' **where** $PTrans: P \Longrightarrow_1 c$ **in** $P'' \rightarrow a < y > \prec P'$
and $P'RelQ': (P', Q'[y::=c]) \in Rel$

by *blast*
have $\langle \nu x \rangle P \Longrightarrow_1 u$ **in** $(\langle \nu x \rangle P'') \rightarrow a < y > \prec \langle \nu c \rangle ([(x, c)] \cdot P')$
proof –
from $PTrans$ $yineqx \langle x \neq a \rangle$ $cineqx$ **have** $\langle \nu x \rangle P \Longrightarrow_1 c$ **in** $(\langle \nu x \rangle P'') \rightarrow a < y >$
 $\prec \langle \nu x \rangle P'$
by(*blast intro: Weak-Late-Step-Semantics.ResB*)
hence $([(x, c)] \cdot \langle \nu x \rangle P) \Longrightarrow_1 ([(x, c)] \cdot c)$ **in** $([(x, c)] \cdot \langle \nu x \rangle P'') \rightarrow ([(x, c)] \cdot a) < ([(x, c)] \cdot y) > \prec [(x, c)] \cdot \langle \nu x \rangle P'$
by(*rule Weak-Late-Step-Semantics.eqvtI*)
moreover from $cFreshP$ **have** $\langle \nu c \rangle ([(x, c)] \cdot P) = \langle \nu x \rangle P$ **by**(*simp add: alphaRes*)
moreover from $cFreshP''$ **have** $\langle \nu c \rangle ([(x, c)] \cdot P'') = \langle \nu x \rangle P''$ **by**(*simp add: alphaRes*)
ultimately show *?thesis using* $\langle x \neq a \rangle$ $cineqa$ $yineqx$ $cineqy$ $cineqx$ $xequ$
by(*simp add: name-calc*)
qed
moreover have $(\langle \nu c \rangle ([(x, c)] \cdot P'), (\langle \nu x \rangle Q')[y::=u]) \in Rel'$
proof –
from $P'RelQ'$ **have** $(\langle \nu x \rangle P', \langle \nu x \rangle (Q'[y::=c])) \in Rel'$ **by**(*rule ResRel*)
with $EqvtRel'$ **have** $([(x, c)] \cdot \langle \nu x \rangle P', [(x, c)] \cdot \langle \nu x \rangle (Q'[y::=c])) \in Rel'$ **by**(*rule eqvtRelI*)
with $cineqy$ $yineqx$ $cineqx$ **have** $(\langle \nu c \rangle ([(x, c)] \cdot P'), (\langle \nu c \rangle ([(x, c)] \cdot Q'))[y::=x]) \in Rel'$
by(*simp add: name-calc eqvt-subs*)
with $cFreshQ'$ $xequ$ **show** *?thesis* **by**(*simp add: alphaRes*)
qed
ultimately show *?thesis* **by** *blast*

next
assume $xinequ: x \neq u$
from $L1$ **obtain** P' **where** $PTrans: P \Longrightarrow_1 u$ **in** $P'' \rightarrow a < y > \prec P'$
and $P'RelQ': (P', Q'[y::=u]) \in Rel$ **by** *blast*

from $PTrans \langle x \neq a \rangle$ $yineqx$ $xinequ$ **have** $\langle \nu x \rangle P \Longrightarrow_1 u$ **in** $(\langle \nu x \rangle P'') \rightarrow a < y >$
 $\prec \langle \nu x \rangle P'$
by(*blast intro: Weak-Late-Step-Semantics.ResB*)
moreover from $P'RelQ'$ $xinequ$ $yineqx$ **have** $(\langle \nu x \rangle P', (\langle \nu x \rangle Q')[y::=u]) \in Rel'$
by(*force intro: ResRel*)
ultimately show *?thesis* **by** *blast*
qed
qed
thus *?case* **by** *blast*
qed

```

next
  case(Free  $Q' \alpha$ )
  have  $\langle \nu x \rangle Q \mapsto \alpha \prec Q'$  by fact
  thus ?case
  proof(induct rule: resCasesF)
    case(cRes  $Q'$ )
    have  $Q \mapsto \alpha \prec Q'$  by fact
    with PSimQ obtain  $P'$  where  $PTrans: P \Rightarrow_i \hat{\alpha} \prec P'$ 
      and  $P'RelQ': (P', Q') \in Rel$ 
    by(blast dest: simE)

    have  $\langle \nu x \rangle P \Rightarrow_i \hat{\alpha} \prec \langle \nu x \rangle P'$ 
    proof –
      have  $xFreshAlpha: x \# \alpha$  by fact
      with  $PTrans$  show ?thesis by(rule ResF)
    qed
    moreover from  $P'RelQ'$  have  $(\langle \nu x \rangle P', \langle \nu x \rangle Q') \in Rel'$  by(rule ResRel)
    ultimately show ?case by blast
  qed
qed
qed

lemma resChainI:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $Rel :: (pi \times pi)$  set
  and  $lst :: name$  list

  assumes eqvRel: eqv  $Rel$ 
  and  $Res: \bigwedge P Q a. (P, Q) \in Rel \Rightarrow (\langle \nu a \rangle P, \langle \nu a \rangle Q) \in Rel$ 
  and  $PRelQ: P \rightsquigarrow \hat{\langle Rel \rangle} Q$ 

  shows  $(resChain\ lst)\ P \rightsquigarrow \hat{\langle Rel \rangle} (resChain\ lst)\ Q$ 
  proof –
    show ?thesis
    proof(induct  $lst$ )
      from  $PRelQ$  show  $resChain\ []\ P \rightsquigarrow \hat{\langle Rel \rangle} resChain\ []\ Q$  by simp
    next
      fix  $a\ lst$ 
      assume IH:  $(resChain\ lst\ P) \rightsquigarrow \hat{\langle Rel \rangle} (resChain\ lst\ Q)$ 
      moreover from  $Res$  have  $\bigwedge P Q a. (P, Q) \in Rel \Rightarrow (\langle \nu a \rangle P, \langle \nu a \rangle Q) \in Rel$ 
    Rel
      by simp
      moreover have  $Rel \subseteq Rel$  by simp
      ultimately have  $\langle \nu a \rangle (resChain\ lst\ P) \rightsquigarrow \hat{\langle Rel \rangle} \langle \nu a \rangle (resChain\ lst\ Q)$ 
    using eqvRel
      by(rule-tac resPres)
      thus  $resChain\ (a \# lst)\ P \rightsquigarrow \hat{\langle Rel \rangle} resChain\ (a \# lst)\ Q$ 
      by simp
  qed

```

qed
qed

lemma *bangPres*:

fixes $P :: pi$
and $Q :: pi$
and $Rel :: (pi \times pi)$ set

assumes *PSimQ*: $P \rightsquigarrow^{\wedge} \langle Rel \rangle Q$
and *PRelQ*: $(P, Q) \in Rel$
and *Sim*: $\bigwedge P Q. (P, Q) \in Rel \implies P \rightsquigarrow^{\wedge} \langle Rel \rangle Q$

and *ParComp*: $\bigwedge P Q R T. \llbracket (P, Q) \in Rel; (R, T) \in Rel \rrbracket \implies (P \parallel R, Q \parallel T) \in Rel'$

and *Res*: $\bigwedge P Q x. (P, Q) \in Rel' \implies (\langle \nu x \rangle P, \langle \nu x \rangle Q) \in Rel'$

and *RelStay*: $\bigwedge P Q. (P \parallel !P, Q) \in Rel' \implies (!P, Q) \in Rel'$
and *BangRelRel'*: $(bangRel\ Rel) \subseteq Rel'$
and *eqvtRel'*: $eqvt\ Rel'$

shows $!P \rightsquigarrow^{\wedge} \langle Rel' \rangle !Q$

proof –

have $\bigwedge Rs\ P. \llbracket !Q \mapsto Rs; (P, !Q) \in bangRel\ Rel \rrbracket \implies weakSimAct\ P\ Rs\ P\ Rel'$

proof –

fix $Rs\ P$

assume $!Q \mapsto Rs$ and $(P, !Q) \in bangRel\ Rel$

thus $weakSimAct\ P\ Rs\ P\ Rel'$

proof(*nominal-induct avoiding: P rule: bangInduct*)

case(*cPar1B aa x Q'*)

have *QTrans*: $Q \mapsto aa \langle x \rangle \prec Q'$ and *xFreshQ*: $x \# Q$ by *fact+*

have $(P, Q \parallel !Q) \in bangRel\ Rel$ and $x \# P$ by *fact+*

thus ?*case*

proof(*induct rule: BRParCases*)

case(*BRPar P R*)

have *PRelQ*: $(P, Q) \in Rel$ and *RBangRelT*: $(R, !Q) \in bangRel\ Rel$ by *fact+*

have $x \# P \parallel R$ by *fact*

hence *xFreshP*: $x \# P$ and *xFreshR*: $x \# R$ by *simp+*

from *PRelQ* have *PSimQ*: $P \rightsquigarrow^{\wedge} \langle Rel \rangle Q$ by (*rule Sim*)

from *eqvtRel'* show ?*case*

proof(*induct rule: simActBoundCases*)

case(*Input a*)

have $aa = InputS\ a$ by *fact*

with *PSimQ QTrans xFreshP* obtain P''

where *L1*: $\forall u. \exists P'. P \implies_{\iota u} in\ P'' \rightarrow a \langle x \rangle \prec P' \wedge (P', Q'[x::=u]) \in$

Rel

by (*blast dest: simE*)

have $\forall u. \exists P'. P \parallel R \implies_{\iota u} in\ (P'' \parallel R) \rightarrow a \langle x \rangle \prec P' \wedge (P', (Q' \parallel !Q)[x::=u]) \in Rel'$

```

proof(rule allI)
  fix u
  from L1 obtain P' where PTrans: P  $\Longrightarrow_l$  u in P''  $\rightarrow_a \langle x \rangle \prec$  P'
    and P'RelQ': (P', Q'[x::=u])  $\in$  Rel
    by blast

from PTrans xFreshR have P  $\parallel$  R  $\Longrightarrow_l$  u in (P''  $\parallel$  R)  $\rightarrow_a \langle x \rangle \prec$  P'  $\parallel$  R
  by(rule Weak-Late-Step-Semantics.Par1B)
moreover have (P'  $\parallel$  R, (Q'  $\parallel$  !Q)[x::=u])  $\in$  Rel'
proof -
  from P'RelQ' RBangRelT have (P'  $\parallel$  R, Q'[x::=u]  $\parallel$  !Q)  $\in$  bangRel
  by(rule Rel.BRPar)
  with xFreshQ BangRelRel' show ?thesis by(auto simp add: forget)
qed
ultimately show  $\exists$  P'. P  $\parallel$  R  $\Longrightarrow_l$  u in (P''  $\parallel$  R)  $\rightarrow_a \langle x \rangle \prec$  P'  $\wedge$ 
  (P', (Q'  $\parallel$  !Q)[x::=u])  $\in$  Rel' by blast
qed
thus ?case by blast
next
case(BoundOutput a)
have aa = BoundOutputS a by fact
with PSimQ QTrans xFreshP obtain P' where PTrans: P  $\Longrightarrow_l$  a  $\langle \nu x \rangle$ 
  and P'RelQ': (P', Q')  $\in$  Rel
  by(blast dest: simE)
from PTrans xFreshR have P  $\parallel$  R  $\Longrightarrow_l$  a  $\langle \nu x \rangle \prec$  P'  $\parallel$  R
  by(rule Weak-Late-Semantics.Par1B)
moreover from P'RelQ' RBangRelT BangRelRel' have (P'  $\parallel$  R, Q'  $\parallel$ 
!Q)  $\in$  Rel'
  by(blast intro: Rel.BRPar)
ultimately show ?case by blast
qed
qed
next
case(cPar1F  $\alpha$  Q' P)
have QTrans: Q  $\mapsto$   $\alpha \prec$  Q' by fact
have (P, Q  $\parallel$  !Q)  $\in$  bangRel Rel by fact
thus ?case
proof(induct rule: BRParCases)
case(BRPar P R)
have PRelQ: (P, Q)  $\in$  Rel and RBangRelQ: (R, !Q)  $\in$  bangRel Rel by
fact+
show ?case
proof(induct rule: simActFreeCases)
case Der
from PRelQ have P  $\rightsquigarrow$   $\hat{\langle$  Rel  $\rangle}$  Q by(rule Sim)
with QTrans obtain P' where PTrans: P  $\Longrightarrow_l$   $\alpha \prec$  P' and P'RelQ':
(P', Q')  $\in$  Rel

```

```

    by(blast dest: simE)

from PTrans have  $P \parallel R \Longrightarrow_i \hat{\alpha} \prec P' \parallel R$  by(rule Weak-Late-Semantics.Par1F)
    moreover from  $P' \text{Rel} Q' \text{RBangRel} Q$  have  $(P' \parallel R, Q' \parallel !Q) \in \text{bangRel}$ 
Rel
    by(rule Rel.BRPar)
    ultimately show ?case using BangRelRel' by blast
    qed
  qed
  next
  case(cPar2B aa x Q' P)
  have IH:  $\bigwedge P. (P, !Q) \in \text{bangRel Rel} \Longrightarrow \text{weakSimAct } P (aa \langle x \rangle \prec Q') P$ 
Rel' by fact
  have xFreshQ:  $x \# Q$  by fact
  have  $(P, Q \parallel !Q) \in \text{bangRel Rel}$  and  $x \# P$  by fact+
  thus ?case
  proof(induct rule: BRParCases)
  case(BRPar P R)
  have PRelQ:  $(P, Q) \in \text{Rel}$  and RBangRelQ:  $(R, !Q) \in \text{bangRel Rel}$  by
fact+
  have  $x \# P \parallel R$  by fact
  hence xFreshP:  $x \# P$  and xFreshR:  $x \# R$  by simp+
  from eqvtRel' show ?case
  proof(induct rule: simActBoundCases)
  case(Input a)
  have  $aa = \text{InputS } a$  by fact
  with RBangRelQ IH have  $\text{weakSimAct } R (a \langle x \rangle \prec Q') R \text{Rel}'$  by blast
  with xFreshR obtain  $R''$  where  $L1: \forall u. \exists R'. R \Longrightarrow_i u \text{ in } R'' \rightarrow a \langle x \rangle \prec$ 
 $R' \wedge (R', Q'[x::=u]) \in \text{Rel}'$ 
  by(force simp add: weakSimAct-def)
  have  $\forall u. \exists P'. P \parallel R \Longrightarrow_i u \text{ in } (P \parallel R'') \rightarrow a \langle x \rangle \prec P' \wedge (P', (Q \parallel$ 
 $Q')[x::=u]) \in \text{Rel}'$ 
  proof(rule allI)
  fix  $u$ 
  from  $L1$  obtain  $R'$  where  $R \text{Trans}: R \Longrightarrow_i u \text{ in } R'' \rightarrow a \langle x \rangle \prec R'$ 
  and  $R' \text{Rel}' Q': (R', Q'[x::=u]) \in \text{Rel}'$ 
  by blast

from  $R \text{Trans } x \text{FreshP}$  have  $P \parallel R \Longrightarrow_i u \text{ in } (P \parallel R'') \rightarrow a \langle x \rangle \prec P \parallel R'$ 
  by(rule Weak-Late-Step-Semantics.Par2B)
  moreover have  $(P \parallel R', (Q \parallel Q')[x::=u]) \in \text{Rel}'$ 
  proof –
  from PRelQ R'Rel'Q' have  $(P \parallel R', Q \parallel Q'[x::=u]) \in \text{Rel}'$ 
  by(rule ParComp)
  with xFreshQ show ?thesis by(simp add: forget)
  qed
  ultimately show  $\exists P'. P \parallel R \Longrightarrow_i u \text{ in } (P \parallel R'') \rightarrow a \langle x \rangle \prec P' \wedge (P',$ 
 $(Q \parallel Q')[x::=u]) \in \text{Rel}'$ 
  by blast

```

```

    qed
  thus ?case by blast
next
  case(BoundOutput a)
  have aa = BoundOutputS a by fact
  with IH RBangRelQ have weakSimAct R (a<νx> < Q') R Rel' by blast
    with xFreshR obtain R' where RTrans: R  $\Longrightarrow_i \hat{a}<\nu x> < R'$  and
R'BangRelQ': (R', Q') ∈ Rel'
    by(simp add: weakSimAct-def, blast)

  from RTrans xFreshP have P || R  $\Longrightarrow_i \hat{a}<\nu x> < P || R'$ 
    by(auto intro: Weak-Late-Semantics.Par2B)
  moreover from PRelQ R'BangRelQ' have (P || R', Q || Q') ∈ Rel'
    by(rule ParComp)
  ultimately show ?case by blast
qed
qed
next
  case(cPar2F α Q' P)
  have IH:  $\bigwedge P. (P, !Q) \in \text{bangRel Rel} \Longrightarrow \text{weakSimAct } P (\alpha < Q') P \text{ Rel'}$  by
fact
  have (P, Q || !Q) ∈ bangRel Rel by fact
  thus ?case
  proof(induct rule: BRParCases)
    case(BRPar P R)
    have PRelQ: (P, Q) ∈ Rel and RBangRelQ: (R, !Q) ∈ bangRel Rel by
fact+
  show ?case
  proof(induct rule: simActFreeCases)
    case Der
    from RBangRelQ have weakSimAct R (α < Q') R Rel' by(rule IH)
    then obtain R' where RTrans: R  $\Longrightarrow_i \hat{\alpha} < R'$  and R'RelQ': (R', Q') ∈
Rel'
    by(simp add: weakSimAct-def, blast)

  from RTrans have P || R  $\Longrightarrow_i \hat{\alpha} < P || R'$  by(rule Weak-Late-Semantics.Par2F)
  moreover from PRelQ R'RelQ' have (P || R', Q || Q') ∈ Rel' by(rule
ParComp)
  ultimately show ?case by blast
qed
qed
next
  case(cComm1 a x Q' b Q'' P)
  have QTrans: Q  $\mapsto a<x> < Q'$  by fact
  have IH:  $\bigwedge P. (P, !Q) \in \text{bangRel Rel} \Longrightarrow \text{weakSimAct } P (a[b] < Q'') P \text{ Rel'}$ 
by fact
  have (P, Q || !Q) ∈ bangRel Rel and x # P by fact+
  thus ?case
  proof(induct rule: BRParCases)

```

case(*BRPar P R*)
have *PRelQ*: $(P, Q) \in \text{Rel}$ **and** *RBangRelQ*: $(R, !Q) \in \text{bangRel Rel}$ **by**
fact+
have $x \# P \parallel R$ **by** *fact*
hence *xFreshP*: $x \# P$ **by** *simp*
show *?case*
proof(*induct rule: simActFreeCases*)
case *Der*
from *PRelQ* **have** $P \rightsquigarrow \hat{\langle \text{Rel} \rangle} Q$ **by**(*rule Sim*)
with *QTrans xFreshP* **obtain** $P' P''$ **where** *PTrans*: $P \Longrightarrow_l b$ *in* $P'' \rightarrow a \langle x \rangle$
 $\prec P'$
and *P'RelQ'*: $(P', Q'[x::=b]) \in \text{Rel}$
by(*blast dest: simE*)

from *RBangRelQ* **have** *weakSimAct R* $(a[b] \prec Q'') R \text{ Rel}'$ **by**(*rule IH*)
then obtain R' **where** *RTrans*: $R \Longrightarrow_i a[b] \prec R'$
and *R'RelQ''*: $(R', Q'') \in \text{Rel}'$
by(*simp add: weakSimAct-def, blast*)

from *PTrans RTrans* **have** $P \parallel R \Longrightarrow_l \tau \prec (P' \parallel R')$
by(*rule Weak-Late-Semantics.Comm1*)
moreover from *P'RelQ' R'RelQ''* **have** $(P' \parallel R', Q'[x::=b] \parallel Q'') \in \text{Rel}'$
by(*rule ParComp*)
ultimately show *?case* **by** *blast*
qed
qed
next
case(*cComm2 a b Q' x Q'' P*)
have *QTrans*: $Q \mapsto a[b] \prec Q'$ **by** *fact*
have *IH*: $\bigwedge P. (P, !Q) \in \text{bangRel Rel} \implies \text{weakSimAct } P (a \langle x \rangle \prec Q'') P$
Rel' **by** *fact*
have $(P, Q \parallel !Q) \in \text{bangRel Rel}$ **and** $x \# P$ **by** *fact+*
thus *?case*
proof(*induct rule: BRParCases*)
case(*BRPar P R*)
have *PRelQ*: $(P, Q) \in \text{Rel}$ **and** *RBangRelQ*: $(R, !Q) \in \text{bangRel Rel}$ **by**
fact+
have $x \# P \parallel R$ **by** *fact*
hence *xFreshR*: $x \# R$ **by** *simp*
show *?case*
proof(*induct rule: simActFreeCases*)
case *Der*
from *PRelQ* **have** $P \rightsquigarrow \hat{\langle \text{Rel} \rangle} Q$ **by**(*rule Sim*)
with *QTrans* **obtain** P' **where** *PTrans*: $P \Longrightarrow_i a[b] \prec P'$ **and** *P'RelQ'*:
 $(P', Q') \in \text{Rel}$
by(*blast dest: simE*)

from *RBangRelQ* **have** *weakSimAct R* $(a \langle x \rangle \prec Q'') R \text{ Rel}'$ **by**(*rule IH*)
with *xFreshR* **obtain** $R' R''$ **where** *RTrans*: $R \Longrightarrow_l b$ *in* $R'' \rightarrow a \langle x \rangle \prec R'$

and $R' \text{BangRel} Q'' : (R', Q''[x::=b]) \in \text{Rel}'$

by(*simp add: weakSimAct-def, blast*)

from $P \text{Trans } R \text{Trans}$ **have** $P \parallel R \implies_l \hat{\tau} \prec (P' \parallel R')$

by(*rule Weak-Late-Semantics.Comm2*)

moreover from $P' \text{Rel} Q' R' \text{BangRel} Q''$ **have** $(P' \parallel R', Q' \parallel Q''[x::=b])$

$\in \text{Rel}'$

by(*rule ParComp*)

ultimately show *?case* **by** *blast*

qed

qed

next

case(*cClose1 a x Q' y Q'' P*)

have $Q \text{Trans} : Q \mapsto a \langle x \rangle \prec Q'$ **by** *fact*

have $IH : \bigwedge P. (P, !Q) \in \text{bangRel } \text{Rel} \implies \text{weakSimAct } P (a \langle \nu y \rangle \prec Q') P$

Rel' **by** *fact*

have $(P, Q \parallel !Q) \in \text{bangRel } \text{Rel}$ **and** $x \# P$ **and** $y \# P$ **by** *fact+*

thus *?case*

proof(*induct rule: BRParCases*)

case(*BRPar P R*)

have $P \text{Rel} Q : (P, Q) \in \text{Rel}$ **and** $R \text{BangRel} Q : (R, !Q) \in \text{bangRel } \text{Rel}$ **by**

fact+

have $x \# P \parallel R$ **by** *fact*

hence $x \text{Fresh} P : x \# P$ **by** *simp*

have $y \# P \parallel R$ **by** *fact*

hence $y \text{Fresh} R : y \# R$ **and** $y \text{Fresh} P : y \# P$ **by** *simp+*

show *?case*

proof(*induct rule: simActFreeCases*)

case *Der*

from $P \text{Rel} Q$ **have** $P \rightsquigarrow \hat{\langle \text{Rel} \rangle} Q$ **by**(*rule Sim*)

with $Q \text{Trans } x \text{Fresh} P$ **obtain** $P' P''$ **where** $P \text{Trans} : P \implies_l y$ **in** $P'' \rightarrow a \langle x \rangle$

$\prec P'$

and $P' \text{Rel} Q' : (P', Q'[x::=y]) \in \text{Rel}$

by(*blast dest: simE*)

from $R \text{BangRel} Q$ **have** $\text{weakSimAct } R (a \langle \nu y \rangle \prec Q'') R \text{Rel}'$ **by**(*rule IH*)

with $y \text{Fresh} R$ **obtain** R' **where** $R \text{Trans} : R \implies_l a \langle \nu y \rangle \prec R'$

and $R' \text{Rel} Q'' : (R', Q'') \in \text{Rel}'$

by(*simp add: weakSimAct-def, blast*)

from $P \text{Trans } R \text{Trans } y \text{Fresh} P y \text{Fresh} R$ **have** $P \parallel R \implies_l \hat{\tau} \prec \langle \nu y \rangle (P' \parallel R')$

R'

by(*rule Weak-Late-Semantics.Close1*)

moreover from $P' \text{Rel} Q' R' \text{Rel} Q''$ **have** $(\langle \nu y \rangle (P' \parallel R'), \langle \nu y \rangle (Q'[x::=y] \parallel Q'')) \in \text{Rel}'$

by(*force intro: ParComp Res*)

ultimately show *?case* **by** *blast*

qed

qed

```

next
  case(cClose2 a y Q' x Q'' P)
  have QTrans:  $Q \mapsto a \langle \nu y \rangle \prec Q'$  by fact
  have IH:  $\bigwedge P. (P, !Q) \in \text{bangRel Rel} \implies \text{weakSimAct } P (a \langle x \rangle \prec Q'') P$ 
  Rel' by fact
  have  $(P, Q \parallel !Q) \in \text{bangRel Rel}$  and  $x \# P$  and  $y \# P$  by fact+
  thus ?case
  proof(induct rule: BRParCases)
    case(BRPar P R)
    have PRelQ:  $(P, Q) \in \text{Rel}$  and RBangRelQ:  $(R, !Q) \in \text{bangRel Rel}$  by
  fact+
  have  $x \# P \parallel R$  by fact
  hence xFreshR:  $x \# R$  by simp
  have  $y \# P \parallel R$  by fact
  hence yFreshP:  $y \# P$  and yFreshR:  $y \# R$  by simp+
  show ?case
  proof(induct rule: simActFreeCases)
    case Der
    from PRelQ have  $P \rightsquigarrow \hat{\langle \text{Rel} \rangle} Q$  by(rule Sim)
    with QTrans yFreshP obtain P' where PTrans:  $P \implies_l \hat{a} \langle \nu y \rangle \prec P'$ 
      and P'RelQ':  $(P', Q') \in \text{Rel}$ 
      by(blast dest: simE)

    from RBangRelQ have weakSimAct R  $(a \langle x \rangle \prec Q'')$  R Rel' by(rule IH)
    with xFreshR obtain R' R'' where RTrans:  $R \implies_l y$  in  $R'' \rightarrow a \langle x \rangle \prec R'$ 
      and R'RelQ'':  $(R', Q''[x::=y]) \in \text{Rel}'$ 
      by(simp add: weakSimAct-def, blast)

    from PTrans RTrans yFreshP yFreshR have  $P \parallel R \implies_l \hat{\tau} \prec \langle \nu y \rangle (P' \parallel$ 
  R')
      by(rule Weak-Late-Semantics.Close2)
      moreover from P'RelQ' R'RelQ'' have  $(\langle \nu y \rangle (P' \parallel R'), \langle \nu y \rangle (Q' \parallel$ 
  Q''[x::=y]))  $\in \text{Rel}'$ 
      by(force intro: ParComp Res)
      ultimately show ?case by blast
    qed
  qed
next
  case(cBang Rs)
  have IH:  $\bigwedge P. (P, Q \parallel !Q) \in \text{bangRel Rel} \implies \text{weakSimAct } P \text{ Rs } P \text{ Rel}'$  by
  fact
  have  $(P, !Q) \in \text{bangRel Rel}$  by fact
  thus ?case
  proof(induct rule: BRBangCases)
    case(BRBang P)
    have PRelQ:  $(P, Q) \in \text{Rel}$  by fact
    hence  $(!P, !Q) \in \text{bangRel Rel}$  by(rule Rel.BRBang)
    with PRelQ have  $(P \parallel !P, Q \parallel !Q) \in \text{bangRel Rel}$  by(rule Rel.BRPar)
    hence weakSimAct  $(P \parallel !P) \text{ Rs } (P \parallel !P) \text{ Rel}'$  by(rule IH)
  
```

thus *?case*
proof(*simp (no-asm) add: weakSimAct-def, auto*)
fix $Q' a x$
assume $weakSimAct (P \parallel !P) (a \langle \nu x \rangle \prec Q') (P \parallel !P) Rel'$ **and** $x \# P$
then obtain P' **where** $PTrans: (P \parallel !P) \Longrightarrow_l \hat{a} \langle \nu x \rangle \prec P'$
and $P'RelQ': (P', Q') \in Rel'$
by(*simp add: weakSimAct-def, blast*)
from $PTrans$ **have** $!P \Longrightarrow_l \hat{a} \langle \nu x \rangle \prec P'$
by(*force intro: Weak-Late-Step-Semantics.Bang simp add: weakTransition-def*)
with $P'RelQ'$ **show** $\exists P'. !P \Longrightarrow_l \hat{a} \langle \nu x \rangle \prec P' \wedge (P', Q') \in Rel'$ **by** *blast*
next
fix $Q' a x$
assume $weakSimAct (P \parallel !P) (a \langle x \rangle \prec Q') (P \parallel !P) Rel'$ **and** $x \# P$
then obtain P'' **where** $L1: \forall u. \exists P'. P \parallel !P \Longrightarrow_l u$ *in* $P'' \rightarrow a \langle x \rangle \prec P'$
 $\wedge (P', Q'[x::=u]) \in Rel'$
by(*simp add: weakSimAct-def, blast*)
have $\forall u. \exists P'. !P \Longrightarrow_l u$ *in* $P'' \rightarrow a \langle x \rangle \prec P' \wedge (P', Q'[x::=u]) \in Rel'$
proof(*rule allI*)
fix u
from $L1$ **obtain** P' **where** $PTrans: P \parallel !P \Longrightarrow_l u$ *in* $P'' \rightarrow a \langle x \rangle \prec P'$
and $P'RelQ': (P', Q'[x::=u]) \in Rel'$
by *blast*
from $PTrans$ **have** $!P \Longrightarrow_l u$ *in* $P'' \rightarrow a \langle x \rangle \prec P'$ **by**(*rule Weak-Late-Step-Semantics.Bang*)
with $P'RelQ'$ **show** $\exists P'. !P \Longrightarrow_l u$ *in* $P'' \rightarrow a \langle x \rangle \prec P' \wedge (P', Q'[x::=u]) \in Rel'$ **by** *blast*
qed
thus $\exists P''. \forall u. \exists P'. !P \Longrightarrow_l u$ *in* $P'' \rightarrow a \langle x \rangle \prec P' \wedge (P', Q'[x::=u]) \in Rel'$ **by** *blast*
next
fix $Q' \alpha$
assume $weakSimAct (P \parallel !P) (\alpha \prec Q') (P \parallel !P) Rel'$
then obtain P' **where** $PTrans: (P \parallel !P) \Longrightarrow_l \hat{\alpha} \prec P'$
and $P'RelQ': (P', Q') \in Rel'$
by(*simp add: weakSimAct-def, blast*)
from $PTrans$ **show** $\exists P'. !P \Longrightarrow_l \hat{\alpha} \prec P' \wedge (P', Q') \in Rel'$
proof(*induct rule: transitionCases*)
case *Step*
have $P \parallel !P \Longrightarrow_l \alpha \prec P'$ **by** *fact*
hence $!P \Longrightarrow_l \alpha \prec P'$ **by**(*rule Weak-Late-Step-Semantics.Bang*)
with $P'RelQ'$ **show** *?case* **by**(*force simp add: weakTransition-def*)
next
case *Stay*
have $\alpha \prec P' = \tau \prec P \parallel !P$ **by** *fact*
hence $\alpha eq \tau: \alpha = \tau$ **and** $P' eq P: P' = P \parallel !P$ **by**(*simp add: residual.inject*)
have $!P \Longrightarrow_l \tau \prec !P$ **by**(*simp add: weakTransition-def*)
moreover from $P' eq P$ $P'RelQ'$ **have** $(!P, Q') \in Rel'$ **by**(*blast intro: RelStay*)
ultimately show *?case* **using** $\alpha eq \tau$ **by** *blast*

```

      qed
    qed
  qed
  qed
  qed
  moreover from PRelQ have  $(!P, !Q) \in \text{bangRel Rel}$  by(rule Rel.BRBang)
  ultimately show ?thesis by(simp add: simDef)
qed

end

theory Weak-Late-Bisim-Pres
  imports Weak-Late-Bisim-SC Weak-Late-Sim-Pres Strong-Late-Bisim-SC
begin

lemma tauPres:
  fixes P :: pi
  and Q :: pi

  assumes  $P \approx Q$ 

  shows  $\tau.(P) \approx \tau.(Q)$ 
proof -
  let ?X =  $\{(\tau.(P), \tau.(Q)) \mid P Q. P \approx Q\}$ 
  from assms have  $(\tau.(P), \tau.(Q)) \in ?X$  by auto
  thus ?thesis
    by(coinduct rule: weakBisimCoinduct)
    (auto simp add: pi.inject intro: Weak-Late-Sim-Pres.tauPres symmetric)
qed

lemma inputPres:
  fixes P :: pi
  and Q :: pi
  and a :: name
  and x :: name

  assumes PSimQ:  $\forall y. P[x::=y] \approx Q[x::=y]$ 

  shows  $a\langle x \rangle.P \approx a\langle x \rangle.Q$ 
proof -
  let ?X =  $\{(a\langle x \rangle.P, a\langle x \rangle.Q) \mid a\ x\ P\ Q. \forall y. P[x::=y] \approx Q[x::=y]\}$ 
  {
    fix axP axQ p
    assume  $(axP, axQ) \in ?X$ 
    then obtain a x P Q where A:  $\forall y. P[x::=y] \approx Q[x::=y]$  and B: axP =
 $a\langle x \rangle.P$  and C: axQ =  $a\langle x \rangle.Q$ 
    by auto
    have  $\bigwedge y. ((p::\text{name } prm) \cdot P)[(p \cdot x)::=y] \approx (p \cdot Q)[(p \cdot x)::=y]$ 
    proof -

```

```

    fix y
    from A have P[x::=(rev p · y)] ≈ Q[x::=(rev p · y)]
      by blast
    hence (p · (P[x::=(rev p · y)])) ≈ p · (Q[x::=(rev p · y)])
      by(rule eqvtI)
    thus (p · P)[(p · x)::=y] ≈ (p · Q)[(p · x)::=y]
      by(simp add: eqvts pt-pi-rev[OF pt-name-inst, OF at-name-inst])
  qed
  hence ((p::name prm) · axP, p · axQ) ∈ ?X using B C
    by auto
}
hence eqvt ?X by(simp add: eqvt-def)

from PSimQ have (a<x>.P, a<x>.Q) ∈ ?X by auto
thus ?thesis
proof(coinduct rule: weakBisimCoinduct)
  case(cSim P Q)
  thus ?case using ‹eqvt ?X›
    by(force intro: inputPres)
next
  case(cSym P Q)
  thus ?case
    by(blast dest: symmetric)
qed
qed

lemma outputPres:
  fixes P :: pi
  and Q :: pi
  and a :: name
  and b :: name

  assumes P ≈ Q

  shows a{b}.(P) ≈ a{b}.(Q)
proof -
  let ?X = {(a{b}.(P), a{b}.(Q)) | a b P Q. P ≈ Q}
  from assms have (a{b}.(P), a{b}.(Q)) ∈ ?X by auto
  thus ?thesis
    by(coinduct rule: weakBisimCoinduct)
    (auto simp add: pi.inject intro: Weak-Late-Sim-Pres.outputPres symmetric)
qed

lemma resPres:
  fixes P :: pi
  and Q :: pi
  and x :: name

  assumes PBiSimQ: P ≈ Q

```

shows $\langle \nu x \rangle P \approx \langle \nu x \rangle Q$
proof –
let $?X = \{x. \exists P Q. P \approx Q \wedge (\exists a. x = (\langle \nu a \rangle P, \langle \nu a \rangle Q))\}$
from $PBiSimQ$ **have** $(\langle \nu x \rangle P, \langle \nu x \rangle Q) \in ?X$ **by** *blast*
moreover have $\bigwedge P Q a. P \rightsquigarrow^{\langle weakBisim \rangle} Q \implies \langle \nu a \rangle P \rightsquigarrow^{\langle ?X \cup weakBisim \rangle} \langle \nu a \rangle Q$
proof –
fix $P Q a$
assume $PSimQ: P \rightsquigarrow^{\langle weakBisim \rangle} Q$
moreover have $\bigwedge P Q a. P \approx Q \implies (\langle \nu a \rangle P, \langle \nu a \rangle Q) \in ?X \cup weakBisim$
by *blast*
moreover have $weakBisim \subseteq ?X \cup weakBisim$ **by** *blast*
moreover have *eqvt weakBisim* **by** (*rule eqvt*)
moreover have *eqvt (?X \cup weakBisim)*
by (*auto simp add: eqvt-def dest: eqvtI*)
ultimately show $\langle \nu a \rangle P \rightsquigarrow^{\langle ?X \cup weakBisim \rangle} \langle \nu a \rangle Q$
by (*rule Weak-Late-Sim-Pres.resPres*)
qed

ultimately show *?thesis* **using** $PBiSimQ$
by (*coinduct rule: weakBisimCoinductAux, blast dest: unfoldE*)
qed

lemma *matchPres*:

fixes $P :: pi$
and $Q :: pi$
and $a :: name$
and $b :: name$

assumes $P \approx Q$

shows $[a \frown b]P \approx [a \frown b]Q$

proof –

let $?X = \{([a \frown b]P, [a \frown b]Q) \mid a b P Q. P \approx Q\}$
from *assms* **have** $([a \frown b]P, [a \frown b]Q) \in ?X$ **by** *auto*
thus *?thesis*

proof (*coinduct rule: weakBisimCoinduct*)

case (*cSim P Q*)

{

fix $P Q a b$

assume $P \approx Q$

hence $P \rightsquigarrow^{\langle weakBisim \rangle} Q$ **by** (*rule unfoldE*)

moreover {

fix $P Q a$

assume $P \approx Q$

moreover have $[a \frown a]P \approx P$ **by** (*rule matchId*)

ultimately have $[a \frown a]P \approx Q$ **by** (*blast intro: transitive*)

}

```

    moreover have  $weakBisim \subseteq ?X \cup weakBisim$  by blast
    ultimately have  $[a \neq b]P \rightsquigarrow^{\wedge} \langle ?X \cup weakBisim \rangle [a \neq b]Q$ 
      by(rule mismatchPres)
  }
  with  $\langle (P, Q) \in ?X \rangle$  show ?case by auto
next
  case(cSym P Q)
  thus ?case by(auto simp add: pi.inject dest: symmetric)
qed
qed

lemma mismatchPres:
  fixes P :: pi
  and Q :: pi
  and a :: name
  and b :: name

  assumes  $P \approx Q$ 

  shows  $[a \neq b]P \approx [a \neq b]Q$ 
proof -
  let ?X =  $\{([a \neq b]P, [a \neq b]Q) \mid a \neq b \ P \ Q. \ P \approx Q\}$ 
  from assms have  $([a \neq b]P, [a \neq b]Q) \in ?X$  by auto
  thus ?thesis
proof(coinduct rule: weakBisimCoinduct)
  case(cSim P Q)
  {
    fix P Q a b
    assume  $P \approx Q$ 
    hence  $P \rightsquigarrow^{\wedge} \langle weakBisim \rangle Q$  by(rule unfoldE)
    moreover {
      fix P Q a b
      assume  $P \approx Q$  and  $(a::name) \neq b$ 
      note  $\langle P \approx Q \rangle$ 
      moreover from  $\langle a \neq b \rangle$  have  $[a \neq b]P \approx P$  by(rule mismatchId)
      ultimately have  $[a \neq b]P \approx Q$  by(blast intro: transitive)
    }
    moreover have  $weakBisim \subseteq ?X \cup weakBisim$  by blast
    ultimately have  $[a \neq b]P \rightsquigarrow^{\wedge} \langle ?X \cup weakBisim \rangle [a \neq b]Q$ 
      by(rule mismatchPres)
  }
  with  $\langle (P, Q) \in ?X \rangle$  show ?case by auto
next
  case(cSym P Q)
  thus ?case by(auto simp add: pi.inject dest: symmetric)
qed
qed

lemma parPres:

```

```

fixes  $P :: pi$ 
and  $Q :: pi$ 
and  $R :: pi$ 

assumes  $P \approx Q$ 

shows  $P \parallel R \approx Q \parallel R$ 
proof –
  let  $?ParSet = \{(resChain\ lst\ (P \parallel R),\ resChain\ lst\ (Q \parallel R)) \mid lst\ P\ Q\ R.\ P \approx Q\}$ 
  have  $BC: \bigwedge P\ Q.\ P \parallel Q = resChain\ []\ (P \parallel Q)$  by auto
  from assms have  $(P \parallel R, Q \parallel R) \in ?ParSet$  by (blast intro: BC)
  thus ?thesis
  proof (coinduct rule: weakBisimCoinduct)
    case (cSim PR QR)
    {
      fix  $P\ Q\ R\ lst$ 
      assume  $P \approx Q$ 

      from eqtI have eqt  $(?ParSet \cup weakBisim)$ 
      by (auto simp add: eqt-def, blast)
      moreover have  $\bigwedge P\ Q\ a.\ (P, Q) \in ?ParSet \cup weakBisim \implies \langle \nu a \rangle P,$ 
 $\langle \nu a \rangle Q \in ?ParSet \cup weakBisim$ 
      by (blast intro: resChain.step[THEN sym] resPres)
      moreover {
        from  $\langle P \approx Q \rangle$  have  $P \rightsquigarrow^{\langle weakBisim \rangle} Q$  by (rule unfoldE)
        moreover note  $\langle P \approx Q \rangle$ 
        moreover {
          fix  $P\ Q\ R$ 
          assume  $P \approx Q$ 
          moreover have  $P \parallel R = resChain\ []\ (P \parallel R)$  by simp
          moreover have  $Q \parallel R = resChain\ []\ (Q \parallel R)$  by simp
          ultimately have  $(P \parallel R, Q \parallel R) \in ?ParSet \cup weakBisim$  by blast
        }
      }
      moreover {
        fix  $P\ Q\ a$ 
        assume  $A: (P, Q) \in ?ParSet \cup weakBisim$ 
        hence  $\langle \nu a \rangle P, \langle \nu a \rangle Q \in ?ParSet \cup weakBisim$  (is ?goal)
        apply (auto intro: resPres)
        by (rule-tac x=a#lst in exI) auto
      }
      ultimately have  $(P \parallel R) \rightsquigarrow^{\langle (?ParSet \cup weakBisim) \rangle} (Q \parallel R)$  using
eqt  $\langle eqt (?ParSet \cup weakBisim) \rangle$ 
      by (rule Weak-Late-Sim-Pres.parPres)
    }
  }
  ultimately have  $resChain\ lst\ (P \parallel R) \rightsquigarrow^{\langle (?ParSet \cup weakBisim) \rangle} resChain\ lst\ (Q \parallel R)$ 
  by (rule resChainI)

```

```

}
with ⟨(PR, QR) ∈ ?ParSet⟩ show ?case by blast
next
case(cSym PR QR)
thus ?case by(auto dest: symmetric)
qed
qed

lemma bangPres:
  fixes P :: pi
  and Q :: pi

  assumes PBisimQ: P ≈ Q

  shows !P ≈ !Q
proof -
  let ?X = (bangRel weakBisim)
  let ?Y = Strong-Late-Bisim.bisim O (bangRel weakBisim) O Strong-Late-Bisim.bisim

  from eqvt Strong-Late-Bisim.bisimEqvt have eqvtY: eqvt ?Y by(blast intro: eqvt-
BangRel)
  have XsubY: ?X ⊆ ?Y by(auto intro: Strong-Late-Bisim.reflexive)

  have RelStay: ∧P Q. (P || !P, Q) ∈ ?Y ⟹ (!P, Q) ∈ ?Y
proof(auto)
  fix P Q R T
  assume PBisimQ: P || !P ~ Q
  and QBRR: (Q, R) ∈ bangRel weakBisim
  and RBisimT: R ~ T
  have !P ~ Q
proof -
  have !P ~ P || !P by(rule Strong-Late-Bisim-SC.bangSC)
  thus ?thesis using PBisimQ by(rule Strong-Late-Bisim.transitive)
qed
with QBRR RBisimT show (!P, T) ∈ ?Y by blast
qed

have ParCompose: ∧P Q R T. [!P ≈ Q; (R, T) ∈ ?Y] ⟹ (P || R, Q || T) ∈
?Y
proof -
  fix P Q R T
  assume PBisimQ: P ≈ Q
  and RYT: (R, T) ∈ ?Y
  thus (P || R, Q || T) ∈ ?Y
proof(auto)
  fix T' R'
  assume T'BisimT: T' ~ T and RBisimR': R ~ R'
  and R'BRT': (R', T') ∈ bangRel weakBisim
  have P || R ~ P || R'

```

```

proof –
from RBisimR' have  $R \parallel P \sim R' \parallel P$  by (rule Strong-Late-Bisim-Pres.parPres)
moreover have  $P \parallel R \sim R \parallel P$  and  $R' \parallel P \sim P \parallel R'$  by (rule Strong-Late-Bisim-SC.parSym)+
  ultimately show ?thesis by (blast intro: Strong-Late-Bisim.transitive)
qed
moreover from PBisimQ R'BRT' have  $(P \parallel R', Q \parallel T') \in \text{bangRel weakBisim}$ 
by (rule BRPar)
  moreover have  $Q \parallel T' \sim Q \parallel T$ 
  proof –
  from T'BisimT have  $T' \parallel Q \sim T \parallel Q$  by (rule Strong-Late-Bisim-Pres.parPres)
    moreover have  $Q \parallel T' \sim T' \parallel Q$  and  $T \parallel Q \sim Q \parallel T$  by (rule
  Strong-Late-Bisim-SC.parSym)+
    ultimately show ?thesis by (blast intro: Strong-Late-Bisim.transitive)
  qed
  ultimately show ?thesis by blast
qed
qed

have ResCong:  $\bigwedge P Q x. (P, Q) \in ?Y \implies (\langle \nu x \rangle P, \langle \nu x \rangle Q) \in ?Y$ 
by (auto intro: BRRes Strong-Late-Bisim-Pres.resPres transitive)

from PBisimQ have  $(!P, !Q) \in ?X$  by (rule BRBang)
moreover from eqvt have eqvt (bangRel weakBisim) by (rule eqvtBangRel)
ultimately show ?thesis
proof (coinduct rule: weakBisimTransitiveCoinduct)
  case (cSim P Q)
  from  $\langle (P, Q) \in ?X \rangle$ 
  show  $P \rightsquigarrow \langle ?Y \rangle Q$ 
  proof (induct)
  case (BRBang P Q)
  have  $P \approx Q$  by fact
  moreover hence  $P \rightsquigarrow \langle \text{weakBisim} \rangle Q$  by (blast dest: unfoldE)
  moreover have  $\bigwedge P Q. P \approx Q \implies P \rightsquigarrow \langle \text{weakBisim} \rangle Q$  by (blast dest:
  unfoldE)
  moreover from Strong-Late-Bisim.bisimEqvt eqvt have eqvt ?Y by (blast
  intro: eqvtBangRel)

  ultimately show  $!P \rightsquigarrow \langle ?Y \rangle !Q$  using ParCompose ResCong RelStay
  XsubY
  by (rule-tac Weak-Late-Sim-Pres.bangPres, simp-all)
next
  case (BRPar P Q R T)
  have PBiSimQ:  $P \approx Q$  by fact
  have RBangRelT:  $(R, T) \in ?X$  by fact
  have RSimT:  $R \rightsquigarrow \langle ?Y \rangle T$  by fact
  moreover from PBiSimQ have  $P \rightsquigarrow \langle \text{weakBisim} \rangle Q$  by (blast dest: un-
  foldE)
  moreover from RBangRelT have  $(R, T) \in ?Y$  by (blast intro: Strong-Late-Bisim.reflexive)
  ultimately show  $P \parallel R \rightsquigarrow \langle ?Y \rangle Q \parallel T$  using ParCompose ResCong eqvt

```

```

eqvtY ⟨P ≈ Q⟩
  by(rule-tac Weak-Late-Sim-Pres.parCompose)
next
case(BRRes P Q x)
have P ≈^⟨?Y⟩ Q by fact
thus ⟨νx⟩P ≈^⟨?Y⟩ ⟨νx⟩Q using ResCong eqvtY XsubY
  by(rule-tac Weak-Late-Sim-Pres.resPres, simp-all)
qed
next
case(cSym P Q)
thus ?case by(metis symmetric bangRelSymmetric)
qed
qed

end

theory Weak-Late-Cong-Pres
  imports Weak-Late-Cong Weak-Late-Step-Sim-Pres Weak-Late-Bisim-Pres
begin

lemma tauPres:
  fixes P :: pi
  and Q :: pi

  assumes P ≈ Q

  shows τ.(P) ≈ τ.(Q)
using assms
by(blast intro: unfoldI Weak-Late-Step-Sim-Pres.tauPres dest: congruenceWeakBisim
symetric)

lemma outputPres:
  fixes P :: pi
  and Q :: pi

  assumes P ≈ Q

  shows a{b}.P ≈ a{b}.Q
using assms
by(blast intro: unfoldI Weak-Late-Step-Sim-Pres.outputPres dest: congruenceWeak-
Bisim symetric)

lemma inputPres:
  fixes P :: pi
  and Q :: pi
  and a :: name
  and x :: name

  assumes PSimQ: ∀ y. P[x::=y] ≈ Q[x::=y]

```

```

  shows  $a\langle x \rangle.P \simeq a\langle x \rangle.Q$ 
using assms
apply(rule-tac unfoldI)
apply(rule-tac Weak-Late-Step-Sim-Pres.inputPres, auto intro: congruenceWeak-Bisim)
by(rule-tac Weak-Late-Step-Sim-Pres.inputPres, auto intro: congruenceWeakBisimWeak-Late-Bisim.symmetric)

```

lemma *matchPres*:

```

  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $a :: name$ 
  and  $b :: name$ 

```

assumes $P \simeq Q$

```

  shows  $[a\curvearrowright b]P \simeq [a\curvearrowright b]Q$ 
using assms
by(blast intro: unfoldI Weak-Late-Step-Sim-Pres.matchPres dest: unfoldE symmetric)

```

lemma *mismatchPres*:

```

  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $a :: name$ 
  and  $b :: name$ 

```

assumes $P \simeq Q$

```

  shows  $[a\neq b]P \simeq [a\neq b]Q$ 
using assms
by(blast intro: unfoldI Weak-Late-Step-Sim-Pres.mismatchPres dest: unfoldE symmetric)

```

lemma *sumPres*:

```

  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $R :: pi$ 

```

assumes $P \simeq Q$

```

  shows  $P \oplus R \simeq Q \oplus R$ 
using assms
by(blast intro: Weak-Late-Bisim.reflexive unfoldI Weak-Late-Step-Sim-Pres.sumPres dest: unfoldE symmetric)

```

lemma *parPres*:

```

  fixes  $P :: pi$ 

```

```

and  $Q :: pi$ 
and  $R :: pi$ 

assumes  $P \simeq Q$ 

shows  $P \parallel R \simeq Q \parallel R$ 
proof –
  have  $\bigwedge P Q R. \llbracket P \rightsquigarrow \langle weakBisim \rangle Q; P \approx Q \rrbracket \implies P \parallel R \rightsquigarrow \langle weakBisim \rangle Q \parallel R$ 
  proof –
    fix  $P Q R$ 
    assume  $P \rightsquigarrow \langle weakBisim \rangle Q$  and  $P \approx Q$ 
    thus  $P \parallel R \rightsquigarrow \langle weakBisim \rangle Q \parallel R$ 
    using Weak-Late-Bisim-Pres.parPres Weak-Late-Bisim-Pres.resPres Weak-Late-Bisim.reflexive Weak-Late-Bisim.eqvt
    by(blast intro: Weak-Late-Step-Sim-Pres.parPres)
  qed
  with assms show ?thesis
  by(blast intro: unfoldI dest: congruenceWeakBisim unfoldE symmetric)
qed

lemma resPres:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $x :: name$ 

  assumes  $P eqQ: P \simeq Q$ 

  shows  $\langle \nu x \rangle P \simeq \langle \nu x \rangle Q$ 
  proof –
    have  $\bigwedge P Q x. P \rightsquigarrow \langle weakBisim \rangle Q \implies \langle \nu x \rangle P \rightsquigarrow \langle weakBisim \rangle \langle \nu x \rangle Q$ 
    proof –
      fix  $P Q x$ 
      assume  $P \rightsquigarrow \langle weakBisim \rangle Q$ 
      with Weak-Late-Bisim.eqvt Weak-Late-Bisim-Pres.resPres show  $\langle \nu x \rangle P \rightsquigarrow \langle weakBisim \rangle \langle \nu x \rangle Q$ 
      by(blast intro: Weak-Late-Step-Sim-Pres.resPres)
    qed
    with assms show ?thesis
    by(blast intro: unfoldI dest: congruenceWeakBisim unfoldE symmetric)
  qed

lemma congruenceBang:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 

  assumes  $P \simeq Q$ 

  shows  $!P \simeq !Q$ 

```

```

proof –
  have  $\bigwedge P Q. \llbracket P \rightsquigarrow \langle \text{weakBisim} \rangle Q; P \simeq Q \rrbracket \implies !P \rightsquigarrow \langle \text{weakBisim} \rangle !Q$ 
  proof –
    fix  $P Q$ 
    assume  $P \rightsquigarrow \langle \text{weakBisim} \rangle Q$  and  $P \simeq Q$ 
    hence  $!P \rightsquigarrow \langle \text{bangRel weakBisim} \rangle !Q$  using unfoldE(1) congruenceWeakBisim
Weak-Late-Bisim.eqt
    by(rule Weak-Late-Step-Sim-Pres.bangPres)
    moreover have bangRel weakBisim  $\subseteq$  weakBisim
    proof auto
      fix  $a b$ 
      assume  $(a, b) \in \text{bangRel weakBisim}$ 
      thus  $a \approx b$ 
      apply(induct rule: bangRel.induct)
      apply (metis Weak-Late-Bisim-Pres.bangPres)
      apply (metis Weak-Late-Bisim.reflexive Weak-Late-Bisim.symmetric
Weak-Late-Bisim.transitive Weak-Late-Bisim-Pres.parPres Weak-Late-Bisim-SC.parSym)
      by (metis Weak-Late-Bisim-Pres.resPres)
    qed
    ultimately show  $!P \rightsquigarrow \langle \text{weakBisim} \rangle !Q$ 
    by(rule Weak-Late-Step-Sim.monotonic)
  qed

  with assms show ?thesis
  by(blast intro: unfoldI dest: unfoldE symmetric congruenceWeakBisim)
qed

end

theory Early-Semantics
  imports Agent
begin

  declare name-fresh[simp del]

  nominal-datatype freeRes = InputR name name ( $\langle \nu \rangle$  [110, 110]
  110)
    | OutputR name name ( $\langle [-] \rangle$  [110, 110] 110)
    | TauR ( $\langle \tau \rangle$  110)

  nominal-datatype residual = BoundOutputR name «name» pi ( $\langle \nu \rangle$   $\langle - \rangle$   $\langle - \rangle$ 
  [110, 110, 110] 110)
    | FreeR freeRes pi

  lemma alphaBoundOutput:
    fixes  $a :: \text{name}$ 
    and  $x :: \text{name}$ 
    and  $P :: \text{pi}$ 
    and  $x' :: \text{name}$ 

```

assumes $A1: x' \# P$

shows $a\langle\nu x\rangle \prec P = a\langle\nu x'\rangle \prec ((x, x') \cdot P)$

proof(cases $x=x'$)

assume $x=x'$

thus ?thesis by simp

next

assume $x \neq x'$

with $A1$ **show** ?thesis

by(simp add: residual.inject alpha name-fresh-left name-calc)

qed

declare name-fresh[simp]

abbreviation Transitions-Freejudge ($\prec \prec \rightarrow [80, 80] 80$) **where** $\alpha \prec P' \equiv (FreeR \alpha P')$

inductive TransitionsEarly :: $pi \Rightarrow residual \Rightarrow bool$ ($\prec \rightarrow \rightarrow [80, 80] 80$)

where

Tau: $\tau.(P) \mapsto \tau \prec P$

| Input: $\llbracket x \neq a; x \neq u \rrbracket \Longrightarrow a\langle x \rangle.P \mapsto a\langle u \rangle \prec (P[x::=u])$

| Output: $a\{b\}.P \mapsto a[b] \prec P$

| Match: $\llbracket P \mapsto V \rrbracket \Longrightarrow [b \sim b]P \mapsto V$

| Mismatch: $\llbracket P \mapsto V; a \neq b \rrbracket \Longrightarrow [a \neq b]P \mapsto V$

| Open: $\llbracket P \mapsto a[b] \prec P'; a \neq b \rrbracket \Longrightarrow \langle \nu b \rangle P \mapsto a\langle \nu b \rangle \prec P'$

| Sum1: $\llbracket P \mapsto V \rrbracket \Longrightarrow (P \oplus Q) \mapsto V$

| Sum2: $\llbracket Q \mapsto V \rrbracket \Longrightarrow (P \oplus Q) \mapsto V$

| Par1B: $\llbracket P \mapsto a\langle \nu x \rangle \prec P'; x \# P; x \# Q; x \neq a \rrbracket \Longrightarrow P \parallel Q \mapsto a\langle \nu x \rangle \prec (P' \parallel Q)$

| Par1F: $\llbracket P \mapsto \alpha \prec P' \rrbracket \Longrightarrow P \parallel Q \mapsto \alpha \prec (P' \parallel Q)$

| Par2B: $\llbracket Q \mapsto a\langle \nu x \rangle \prec Q'; x \# P; x \# Q; x \neq a \rrbracket \Longrightarrow P \parallel Q \mapsto a\langle \nu x \rangle \prec (P \parallel Q')$

| Par2F: $\llbracket Q \mapsto \alpha \prec Q' \rrbracket \Longrightarrow P \parallel Q \mapsto \alpha \prec (P \parallel Q')$

| Comm1: $\llbracket P \mapsto a\langle b \rangle \prec P'; Q \mapsto a[b] \prec Q' \rrbracket \Longrightarrow P \parallel Q \mapsto \tau \prec P' \parallel Q'$

| Comm2: $\llbracket P \mapsto a[b] \prec P'; Q \mapsto a\langle b \rangle \prec Q' \rrbracket \Longrightarrow P \parallel Q \mapsto \tau \prec P' \parallel Q'$

| Close1: $\llbracket P \mapsto a\langle x \rangle \prec P'; Q \mapsto a\langle \nu x \rangle \prec Q'; x \# P; x \# Q; x \neq a \rrbracket \Longrightarrow P \parallel Q \mapsto \tau \prec \langle \nu x \rangle (P' \parallel Q')$

| Close2: $\llbracket P \mapsto a\langle \nu x \rangle \prec P'; Q \mapsto a\langle x \rangle \prec Q'; x \# P; x \# Q; x \neq a \rrbracket \Longrightarrow P \parallel Q \mapsto \tau \prec \langle \nu x \rangle (P' \parallel Q')$

| ResB: $\llbracket P \mapsto a\langle \nu x \rangle \prec P'; y \neq a; y \neq x; x \# P; x \neq a \rrbracket \Longrightarrow \langle \nu y \rangle P \mapsto a\langle \nu x \rangle \prec (\langle \nu y \rangle P')$

| *ResF*: $\llbracket P \mapsto \alpha \prec P'; y \# \alpha \rrbracket \Longrightarrow \langle \nu y \rangle P \mapsto \alpha \prec \langle \nu y \rangle P'$

| *Bang*: $\llbracket P \parallel !P \mapsto V \rrbracket \Longrightarrow !P \mapsto V$

equivariance *TransitionsEarly*
nominal-inductive *TransitionsEarly*
by(*auto simp add: abs-fresh fresh-fact2*)

lemmas [*simp*] = *freeRes.inject*

lemma *freshOutputAction*:

fixes $P :: pi$
and $a :: name$
and $b :: name$
and $P' :: pi$
and $c :: name$

assumes $P \mapsto a[b] \prec P'$
and $c \# P$

shows $c \neq a$ **and** $c \neq b$ **and** $c \# P'$

proof –

from *assms* **have** $c \neq a \wedge c \neq b \wedge c \# P'$

by(*nominal-induct x2==a[b] < P' arbitrary: P' rule: TransitionsEarly.strong-induct*)
(*fastforce simp add: residual.inject abs-fresh freeRes.inject*)+

thus $c \neq a$ **and** $c \neq b$ **and** $c \# P'$
by *blast+*

qed

lemma *freshInputAction*:

fixes $P :: pi$
and $a :: name$
and $b :: name$
and $P' :: pi$
and $c :: name$

assumes $P \mapsto a \langle b \rangle \prec P'$
and $c \# P$

shows $c \neq a$

using *assms*

by(*nominal-induct x2==a < b > < P' arbitrary: P' rule: TransitionsEarly.strong-induct*)
(*auto simp add: residual.inject abs-fresh*)

lemma *freshBoundOutputAction*:

fixes $P :: pi$
and $a :: name$
and $x :: name$
and $P' :: pi$

```

and   c :: name

assumes P  $\mapsto$  a< $\nu$ x>  $\prec$  P'
and   c  $\#$  P

shows c  $\neq$  a
using assms
by(nominal-induct x2==a< $\nu$ x>  $\prec$  P' avoiding: x arbitrary: P' rule: TransitionsEarly.strong-induct) (auto simp add: residual.inject abs-fresh fresh-left calc-atm dest: freshOutputAction)

lemmas freshAction = freshOutputAction freshInputAction freshBoundOutputAction

lemma freshInputTransition:
  fixes P :: pi
  and   a :: name
  and   u :: name
  and   P' :: pi
  and   c :: name

  assumes P  $\mapsto$  a<u>  $\prec$  P'
  and   c  $\#$  P
  and   c  $\neq$  u

  shows c  $\#$  P'
using assms
by(nominal-induct x2==a<u>  $\prec$  P' arbitrary: P' rule: TransitionsEarly.strong-induct) (fastforce simp add: residual.inject name-fresh-abs fresh-fact1 fresh-fact2)+

lemma freshBoundOutputTransition:
  fixes P :: pi
  and   a :: name
  and   x :: name
  and   P' :: pi
  and   c :: name

  assumes P  $\mapsto$  a< $\nu$ x>  $\prec$  P'
  and   c  $\#$  P
  and   c  $\neq$  x

  shows c  $\#$  P'
using assms
apply(nominal-induct x2==a< $\nu$ x>  $\prec$  P' avoiding: x arbitrary: P' rule: TransitionsEarly.strong-induct)
apply(fastforce simp add: residual.inject name-fresh-abs alpha' fresh-left calc-atm dest: freshOutputAction | simp | auto simp add: abs-fresh residual.inject alpha' calc-atm)
apply(fastforce simp add: residual.inject name-fresh-abs alpha' fresh-left calc-atm)

```

```

dest: freshOutputAction | simp | auto simp add: abs-fresh residual.inject alpha'
calc-atm)
apply(fastforce simp add: residual.inject name-fresh-abs alpha' fresh-left calc-atm
dest: freshOutputAction | simp | auto simp add: abs-fresh residual.inject alpha'
calc-atm)
apply(force simp add: residual.inject name-fresh-abs alpha' fresh-left calc-atm dest:
freshOutputAction | simp | auto simp add: abs-fresh residual.inject alpha' calc-atm)
apply(force simp add: residual.inject name-fresh-abs alpha' fresh-left calc-atm dest:
freshOutputAction | simp | auto simp add: abs-fresh residual.inject alpha' calc-atm)
apply(force simp add: residual.inject name-fresh-abs alpha' fresh-left calc-atm dest:
freshOutputAction | simp | auto simp add: abs-fresh residual.inject alpha' calc-atm)
apply(force simp add: residual.inject name-fresh-abs alpha' fresh-left calc-atm dest:
freshOutputAction | simp | auto simp add: abs-fresh residual.inject alpha' calc-atm)
apply(force simp add: residual.inject name-fresh-abs alpha' fresh-left calc-atm dest:
freshOutputAction | simp | auto simp add: abs-fresh residual.inject alpha' calc-atm)
apply(force simp add: residual.inject name-fresh-abs alpha' fresh-left calc-atm dest:
freshOutputAction | simp | auto simp add: abs-fresh residual.inject alpha' calc-atm)
apply(force simp add: residual.inject name-fresh-abs alpha' fresh-left calc-atm dest:
freshOutputAction | simp | auto simp add: abs-fresh residual.inject alpha' calc-atm)
apply(force simp add: residual.inject name-fresh-abs alpha' fresh-left calc-atm dest:
freshOutputAction | simp | auto simp add: abs-fresh residual.inject alpha' calc-atm)
apply(force simp add: residual.inject name-fresh-abs alpha' fresh-left calc-atm dest:
freshOutputAction | simp | auto simp add: abs-fresh residual.inject alpha' calc-atm)
apply(fastforce simp add: residual.inject name-fresh-abs alpha' fresh-left calc-atm
dest: freshOutputAction)
apply(fastforce simp add: residual.inject name-fresh-abs alpha' fresh-left calc-atm
dest: freshOutputAction)
apply(fastforce simp add: residual.inject name-fresh-abs alpha' fresh-left calc-atm
dest: freshOutputAction)
apply(fastforce simp add: residual.inject name-fresh-abs alpha' fresh-left calc-atm
dest: freshOutputAction)
apply(auto simp add: residual.inject name-fresh-abs alpha' fresh-left calc-atm dest:
freshOutputAction)
done

```

lemma *freshTauTransition*:

```

fixes  $P :: pi$ 
and  $P' :: pi$ 
and  $c :: name$ 

assumes  $P \mapsto \tau \prec P'$ 
and  $c \# P$ 

shows  $c \# P'$ 
using assms
apply(nominal-induct  $x2 == \tau \prec P'$  arbitrary:  $P'$  rule: TransitionsEarly.strong-induct)
by(fastforce simp add: residual.inject abs-fresh dest: freshOutputAction freshInput-
Transition freshBoundOutputTransition)+

```

```

lemma freshFreeTransition:
  fixes  $P :: pi$ 
  and  $\alpha :: freeRes$ 
  and  $P' :: pi$ 
  and  $c :: name$ 

  assumes  $P \mapsto \alpha \prec P'$ 
  and  $c \# P$ 
  and  $c \# \alpha$ 

  shows  $c \# P'$ 
using assms
by(nominal-induct  $\alpha$  rule: freeRes.strong-inducts)
  (auto dest: freshInputTransition freshOutputAction freshTauTransition)

lemmas freshTransition = freshInputTransition freshOutputAction freshFreeTransition
freshBoundOutputTransition freshTauTransition

lemma substTrans[simp]:  $b \# P \implies ((P::pi)[a::=b])[b::=c] = P[a::=c]$ 
apply(simp add: injPermSubst[THEN sym])
apply(simp add: renaming)
by(simp add: pt-swap[OF pt-name-inst, OF at-name-inst])

lemma Input:
  fixes  $a :: name$ 
  and  $x :: name$ 
  and  $u :: name$ 
  and  $P :: pi$ 

  shows  $a \langle x \rangle . P \mapsto a \langle u \rangle \prec P[x::=u]$ 
proof –
  obtain  $y::name$  where  $y \neq a$  and  $y \neq u$  and  $y \# P$ 
  by(generate-fresh name) (auto simp add: fresh-prod)
  from  $\langle y \neq a \rangle \langle y \neq u \rangle$  have  $a \langle y \rangle . ([x, y]) \cdot P \mapsto a \langle u \rangle \prec ([x, y]) \cdot P[y::=u]$ 
  by(rule Input)
  with  $\langle y \# P \rangle$  show ?thesis by(simp add: alphaInput renaming name-swap)
qed

lemma Par1B:
  fixes  $P :: pi$ 
  and  $a :: name$ 
  and  $x :: name$ 
  and  $P' :: pi$ 
  and  $Q :: pi$ 

  assumes  $P \mapsto a \langle \nu x \rangle \prec P'$ 
  and  $x \# Q$ 

```

shows $P \parallel Q \mapsto a\langle\nu x\rangle \prec (P' \parallel Q)$
proof –
obtain $y::name$ **where** $y \# P$ **and** $y \# Q$ **and** $y \neq a$ **and** $y \# P'$
by(*generate-fresh name*) (*auto simp add: fresh-prod*)
from $\langle P \mapsto a\langle\nu x\rangle \prec P' \rangle \langle y \# P' \rangle$ **have** $P \mapsto a\langle\nu y\rangle \prec ((x, y) \cdot P')$
by(*simp add: alphaBoundOutput*)
hence $P \parallel Q \mapsto a\langle\nu y\rangle \prec ((x, y) \cdot P' \parallel Q)$ **using** $\langle y \# P \rangle \langle y \# Q \rangle \langle y \neq a \rangle$
by(*rule Par1B*)
with $\langle x \# Q \rangle \langle y \# Q \rangle \langle y \# P' \rangle$ **show** *?thesis*
by(*subst alphaBoundOutput*) (*auto simp add: name-fresh-fresh*)
qed

lemma *Par2B*:

fixes $Q :: pi$
and $a :: name$
and $x :: name$
and $Q' :: pi$
and $P :: pi$

assumes $Q \mapsto a\langle\nu x\rangle \prec Q'$
and $x \# P$

shows $P \parallel Q \mapsto a\langle\nu x\rangle \prec (P \parallel Q')$

proof –

obtain $y::name$ **where** $y \# P$ **and** $y \# Q$ **and** $y \neq a$ **and** $y \# Q'$
by(*generate-fresh name*) (*auto simp add: fresh-prod*)
from $\langle Q \mapsto a\langle\nu x\rangle \prec Q' \rangle \langle y \# Q' \rangle$ **have** $Q \mapsto a\langle\nu y\rangle \prec ((x, y) \cdot Q')$
by(*simp add: alphaBoundOutput*)
hence $P \parallel Q \mapsto a\langle\nu y\rangle \prec (P \parallel ((x, y) \cdot Q'))$ **using** $\langle y \# P \rangle \langle y \# Q \rangle \langle y \neq a \rangle$
by(*rule Par2B*)
with $\langle x \# P \rangle \langle y \# P \rangle \langle y \# Q' \rangle$ **show** *?thesis*
by(*subst alphaBoundOutput[of y]*) (*auto simp add: name-fresh-fresh*)
qed

lemma *inputInduct*[*consumes 1, case-names cInput cMatch cMismatch cSum1 cSum2 cPar1 cPar2 cRes cBang*]:

fixes $P :: pi$
and $a :: name$
and $u :: name$
and $P' :: pi$
and $F :: 'a::fs-name \Rightarrow pi \Rightarrow name \Rightarrow name \Rightarrow pi \Rightarrow bool$
and $C :: 'a::fs-name$

assumes *Trans*: $P \mapsto a\langle u \rangle \prec P'$

and $\bigwedge a x P u C. \llbracket x \# C; x \neq u; x \neq a \rrbracket \Longrightarrow F C (a\langle x \rangle.P) a u (P[x::=u])$

and $\bigwedge P a u P' b C. \llbracket P \mapsto a\langle u \rangle \prec P'; \bigwedge C. F C P a u P' \rrbracket \Longrightarrow F C (\llbracket b \frown b \rrbracket P)$
 $a u P'$

and $\bigwedge P a u P' b c C. \llbracket P \mapsto a\langle u \rangle \prec P'; \bigwedge C. F C P a u P'; b \neq c \rrbracket \Longrightarrow F C (\llbracket b \neq c \rrbracket P) a u P'$

and $\bigwedge P a u P' Q C. \llbracket P \mapsto a \langle u \rangle \prec P'; \bigwedge C. F C P a u P \rrbracket \Longrightarrow F C (P \oplus Q) a u P'$
and $\bigwedge Q a u Q' P C. \llbracket Q \mapsto a \langle u \rangle \prec Q'; \bigwedge C. F C Q a u Q \rrbracket \Longrightarrow F C (P \oplus Q) a u Q'$
and $\bigwedge P a u P' Q C. \llbracket P \mapsto a \langle u \rangle \prec P'; \bigwedge C. F C P a u P \rrbracket \Longrightarrow F C (P \parallel Q) a u (P' \parallel Q)$
and $\bigwedge Q a u Q' P C. \llbracket Q \mapsto a \langle u \rangle \prec Q'; \bigwedge C. F C Q a u Q \rrbracket \Longrightarrow F C (P \parallel Q) a u (P \parallel Q')$
and $\bigwedge P a u P' x C. \llbracket P \mapsto a \langle u \rangle \prec P'; x \neq a; x \neq u; x \# C; \bigwedge C. F C P a u P \rrbracket \Longrightarrow F C (\langle \nu x \rangle P) a u (\langle \nu x \rangle P')$
and $\bigwedge P a u P' C. \llbracket P \parallel !P \mapsto a \langle u \rangle \prec P'; \bigwedge C. F C (P \parallel !P) a u P \rrbracket \Longrightarrow F C (!P) a u P'$

shows $F C P a u P'$

using *assms*

by(*nominal-induct* $x2 == a \langle u \rangle \prec P'$ *avoiding: C arbitrary: P' rule: Transition-
sEarly.strong-induct*)

(*auto simp add: residual.inject*)

lemma *inputAlpha*:

assumes $P \mapsto a \langle u \rangle \prec P'$

and $u \# P$

and $r \# P'$

shows $P \mapsto a \langle r \rangle \prec ((u, r) \cdot P')$

using *assms*

proof(*nominal-induct* *avoiding: r rule: inputInduct*)

case(*cInput* $a x P u r$)

from $\langle x \neq u \rangle \langle u \# a \langle x \rangle . P \rangle$ **have** $u \neq a$ **and** $u \# P$ **by**(*simp add: abs-fresh*)**+**

have $a \langle x \rangle . P \mapsto a \langle r \rangle \prec P[x ::= r]$

by(*rule Input*)

thus $?case$ **using** $\langle r \# P[x ::= u] \rangle \langle u \# P \rangle$

by(*simp add: injPermSubst substTrans*)

next

case(*cMatch* $P a u P' b r$)

thus $?case$ **by**(*force intro: Match*)

next

case(*cMismatch* $P a u P' b c r$)

thus $?case$ **by**(*force intro: Mismatch*)

next

case(*cSum1* $P a u P' Q r$)

thus $?case$ **by**(*force intro: Sum1*)

next

case(*cSum2* $Q a u Q' P r$)

thus $?case$ **by**(*force intro: Sum2*)

next

case(*cPar1* $P a u P' Q r$)

thus $?case$ **by**(*force intro: Par1F simp add: eqvts name-fresh-fresh*)

next

```

  case(cPar2 Q a u Q' P r)
  thus ?case by(force intro: Par2F simp add: eqvts name-fresh-fresh)
next
  case(cRes P a u P' x r)
  thus ?case by(force intro: ResF simp add: eqvts calc-atm abs-fresh)
next
  case(cBang P a u P' R)
  thus ?case by(force intro: Bang)
qed

```

lemma *Close1*:

```

  fixes P :: pi
  and a :: name
  and x :: name
  and P' :: pi
  and Q :: pi
  and Q' :: pi

  assumes P ⟶a<x> < P'
  and Q ⟶a<νx> < Q'
  and x # P

  shows P || Q ⟶τ < <νx>(P' || Q')
proof -
  obtain y::name where y # P and y # Q and y ≠ a and y # Q' and y # P'
  by(generate-fresh name) (auto simp add: fresh-prod)
  from <P ⟶a<x> < P'> <x # P> <y # P'> have P ⟶a<y> < [(x, y)] · P'
  by(rule inputAlpha)
  moreover from <Q ⟶a<νx> < Q'> <y # Q'> have Q ⟶a<νy> < [(x, y)] ·
  Q'
  by(simp add: alphaBoundOutput)

  ultimately have P || Q ⟶τ < <νy>([(x, y)] · P') || [(x, y)] · Q' using <y
  # P> <y # Q> <y ≠ a>
  by(rule Close1)
  with <y # P'> <y # Q'> show ?thesis by(subst alphaRes) (auto simp add: name-fresh-fresh)
qed

```

lemma *Close2*:

```

  fixes P :: pi
  and a :: name
  and x :: name
  and P' :: pi
  and Q :: pi
  and Q' :: pi

  assumes P ⟶a<νx> < P'
  and Q ⟶a<x> < Q'
  and x # Q

```

```

shows  $P \parallel Q \mapsto \tau \prec \langle \nu x \rangle (P' \parallel Q')$ 
proof -
  obtain  $y :: \text{name}$  where  $y \# P$  and  $y \# Q$  and  $y \neq a$  and  $y \# Q'$  and  $y \# P'$ 
    by (generate-fresh name) (auto simp add: fresh-prod)
  from  $\langle P \mapsto a \langle \nu x \rangle \prec P' \rangle \langle y \# P' \rangle$  have  $P \mapsto a \langle \nu y \rangle \prec ([x, y] \cdot P')$ 
    by (simp add: alphaBoundOutput)
  moreover from  $\langle Q \mapsto a \langle x \rangle \prec Q' \rangle \langle x \# Q \rangle \langle y \# Q' \rangle$  have  $Q \mapsto a \langle y \rangle \prec ([x, y] \cdot Q')$ 
    by (rule inputAlpha)

  ultimately have  $P \parallel Q \mapsto \tau \prec \langle \nu y \rangle ([x, y] \cdot P' \parallel [x, y] \cdot Q')$  using  $\langle y \# P \rangle \langle y \# Q \rangle \langle y \neq a \rangle$ 
    by (rule Close2)
  with  $\langle y \# P' \rangle \langle y \# Q' \rangle$  show ?thesis by (subst alphaRes) (auto simp add: name-fresh-fresh)
qed

```

lemma ResB:

```

fixes  $P :: pi$ 
and  $a :: name$ 
and  $x :: name$ 
and  $P' :: pi$ 
and  $y :: name$ 

```

```

assumes  $P \mapsto a \langle \nu x \rangle \prec P'$ 
and  $y \neq a$ 
and  $y \neq x$ 

```

```

shows  $\langle \nu y \rangle P \mapsto a \langle \nu x \rangle \prec (\langle \nu y \rangle P')$ 

```

proof -

```

  obtain  $z :: \text{name}$  where  $z \# P$  and  $z \# P'$  and  $z \neq a$  and  $z \neq y$ 
    by (generate-fresh name) (auto simp add: fresh-prod)
  from  $\langle P \mapsto a \langle \nu x \rangle \prec P' \rangle \langle z \# P' \rangle$  have  $P \mapsto a \langle \nu z \rangle \prec ([x, z] \cdot P')$ 
    by (simp add: alphaBoundOutput)
  hence  $\langle \nu y \rangle P \mapsto a \langle \nu z \rangle \prec (\langle \nu y \rangle ([x, z] \cdot P'))$  using  $\langle y \neq a \rangle \langle z \neq y \rangle \langle z \# P \rangle \langle z \neq a \rangle$ 
    by (rule-tac ResB) auto
  thus ?thesis using  $\langle z \neq y \rangle \langle y \neq x \rangle \langle z \# P' \rangle$ 
    by (subst alphaBoundOutput[where  $x'=z$ ]) (auto simp add: eqvts calc-atm abs-fresh)
qed

```

lemma outputInduct[consumes 1, case-names Output Match Mismatch Sum1 Sum2 Par1 Par2 Res Bang]:

```

fixes  $P :: pi$ 
and  $a :: name$ 
and  $b :: name$ 
and  $P' :: pi$ 
and  $F :: 'a::fs-name \Rightarrow pi \Rightarrow name \Rightarrow name \Rightarrow pi \Rightarrow bool$ 
and  $C :: 'a::fs-name$ 

```

assumes *Trans*: $P \mapsto a[b] \prec P'$
and $\bigwedge a b P C. F C (a\{b\}.P) a b P$
and $\bigwedge P a b P' c C. \llbracket P \mapsto \text{OutputR } a b \prec P'; \bigwedge C. F C P a b P \rrbracket \implies F C$
 $([c \frown c]P) a b P'$
and $\bigwedge P a b P' c d C. \llbracket P \mapsto \text{OutputR } a b \prec P'; \bigwedge C. F C P a b P'; c \neq d \rrbracket$
 $\implies F C ([c \neq d]P) a b P'$
and $\bigwedge P a b P' Q C. \llbracket P \mapsto \text{OutputR } a b \prec P'; \bigwedge C. F C P a b P \rrbracket \implies F C$
 $(P \oplus Q) a b P'$
and $\bigwedge Q a b Q' P C. \llbracket Q \mapsto \text{OutputR } a b \prec Q'; \bigwedge C. F C Q a b Q \rrbracket \implies F$
 $C (P \oplus Q) a b Q'$
and $\bigwedge P a b P' Q C. \llbracket P \mapsto \text{OutputR } a b \prec P'; \bigwedge C. F C P a b P \rrbracket \implies F C$
 $(P \parallel Q) a b (P' \parallel Q)$
and $\bigwedge Q a b Q' P C. \llbracket Q \mapsto \text{OutputR } a b \prec Q'; \bigwedge C. F C Q a b Q \rrbracket \implies F$
 $C (P \parallel Q) a b (P \parallel Q')$
and $\bigwedge P a b P' x C. \llbracket P \mapsto \text{OutputR } a b \prec P'; x \neq a; x \neq b; x \# C; \bigwedge C. F$
 $C P a b P \rrbracket \implies$
 $F C (\langle \nu x \rangle P) a b (\langle \nu x \rangle P')$
and $\bigwedge P a b P' C. \llbracket P \parallel !P \mapsto \text{OutputR } a b \prec P'; \bigwedge C. F C (P \parallel !P) a b P \rrbracket$
 $\implies F C (!P) a b P'$

shows $F C P a b P'$
using *assms*
by(*nominal-induct* $x2 == a[b] \prec P'$ *avoiding*: C *arbitrary*: P' *rule*: *TransitionsEarly.strong-induct*)
(auto simp add: residual.inject)

lemma *boundOutputInduct*[*consumes 2, case-names Match Mismatch Open Sum1*
Sum2 Par1 Par2 Res Bang]:

fixes $P :: pi$
and $a :: name$
and $x :: name$
and $P' :: pi$
and $F :: ('a::fs-name) \Rightarrow pi \Rightarrow name \Rightarrow name \Rightarrow pi \Rightarrow bool$
and $C :: 'a::fs-name$

assumes $a: P \mapsto a \langle \nu x \rangle \prec P'$
and $xFreshP: x \# P$
and $cMatch: \bigwedge P a x P' b C. \llbracket P \mapsto a \langle \nu x \rangle \prec P'; \bigwedge C. F C P a x P \rrbracket \implies$
 $F C ([b \frown b]P) a x P'$
and $cMismatch: \bigwedge P a x P' b c C. \llbracket P \mapsto a \langle \nu x \rangle \prec P'; \bigwedge C. F C P a x P';$
 $b \neq c \rrbracket \implies F C ([b \neq c]P) a x P'$
and $cOpen: \bigwedge P a x P' C. \llbracket P \mapsto (\text{OutputR } a x) \prec P'; a \neq x \rrbracket \implies F C$
 $(\langle \nu x \rangle P) a x P'$
and $cSum1: \bigwedge P Q a x P' C. \llbracket P \mapsto a \langle \nu x \rangle \prec P'; \bigwedge C. F C P a x P \rrbracket \implies$
 $F C (P \oplus Q) a x P'$
and $cSum2: \bigwedge P Q a x Q' C. \llbracket Q \mapsto a \langle \nu x \rangle \prec Q'; \bigwedge C. F C Q a x Q \rrbracket$
 $\implies F C (P \oplus Q) a x Q'$
and $cPar1B: \bigwedge P P' Q a x C. \llbracket P \mapsto a \langle \nu x \rangle \prec P'; x \# Q; \bigwedge C. F C P a x$
 $P \rrbracket \implies$

$$\text{and } cPar2B: \bigwedge P Q Q' a x C. \llbracket Q \mapsto a \langle \nu x \rangle \prec Q'; x \# P; \bigwedge C. F C Q a x Q \rrbracket \implies$$

$$F C (P \parallel Q) a x (P' \parallel Q)$$

$$\text{and } cResB: \bigwedge P P' a x y C. \llbracket P \mapsto a \langle \nu x \rangle \prec P'; y \neq a; y \neq x; y \# C; \bigwedge C. F C P a x P' \rrbracket \implies F C (\langle \nu y \rangle P) a x (\langle \nu y \rangle P')$$

$$\text{and } cBang: \bigwedge P a x P' C. \llbracket P \parallel !P \mapsto a \langle \nu x \rangle \prec P'; \bigwedge C. F C (P \parallel !P) a x P' \rrbracket \implies$$

$$F C (!P) a x P'$$

shows $F C P a x P'$
using *assms*
proof –
have *Goal*: $\bigwedge P Rs a x P' C. \llbracket P \mapsto Rs; Rs = a \langle \nu x \rangle \prec P'; x \# P \rrbracket \implies F C P a x P'$
proof –
fix $P Rs a x P' C$
assume $P \mapsto Rs$ **and** $Rs = a \langle \nu x \rangle \prec P'$ **and** $x \# P$
thus $F C P a x P'$
proof (*nominal-induct avoiding: C a x P' rule: TransitionsEarly.strong-induct*)
case (*Tau P*)
thus ?case **by** (*simp add: residual.inject*)
next
case (*Input P a x*)
thus ?case **by** (*simp add: residual.inject*)
next
case (*Output P a b*)
thus ?case **by** (*simp add: residual.inject*)
next
case (*Match P Rs b C a x P'*)
thus ?case
by (*force intro: cMatch simp add: residual.inject*)
next
case (*Mismatch P Rs b c C a x P'*)
thus ?case
by (*force intro: cMismatch simp add: residual.inject*)
next
case (*Sum1 P Q Rs C*)
thus ?case **by** (*force intro: cSum1*)
next
case (*Sum2 P Q Rs C*)
thus ?case **by** (*force intro: cSum2*)
next
case (*Open P a b P' C a' x P''*)
have $b \# x$ **by** *fact* **hence** *bineqx*: $b \neq x$ **by** *simp*
moreover **have** $a \langle \nu b \rangle \prec P' = a' \langle \nu x \rangle \prec P''$ **by** *fact*
ultimately **have** *aeqa'*: $a = a'$ **and** *P'eqP''*: $P'' = [(b, x)] \cdot P'$
by (*simp add: residual.inject name-abs-eq*) +
have $x \# \langle \nu b \rangle P$ **by** *fact*
with *bineqx* **have** *xFreshP*: $x \# P$ **by** (*simp add: name-fresh-abs*)

have $a \text{ineq} b: a \neq b$ **by fact**

have $P \text{Trans}: P \mapsto a[b] \prec P'$ **by fact**
with $x \text{Fresh} P$ **have** $x \text{ineq} a: x \neq a$ **by** (*force dest: freshAction*)
from $P \text{Trans}$ **have** $([(b, x)] \cdot P) \mapsto [(b, x)] \cdot (a[b] \prec P')$ **by** (*rule Transition-
sEarly.eqv*)

with $P' \text{eq} P''$ $x \text{ineq} a$ $a \text{ineq} b$ **have** $\text{Trans}: ([(b, x)] \cdot P) \mapsto a[x] \prec P''$
by (*auto simp add: name-calc*)
hence $F C (\nu x) ([(b, x)] \cdot P)$ $a x P''$ **using** $x \text{ineq} a$ **by** (*blast intro: cOpen*)
with $x \text{Fresh} P$ $a \text{eq} a'$ **show** $? \text{case}$ **by** (*simp add: alphaRes*)

next

case ($\text{Par1} B P a x P' Q C a' x' P''$)
have $x \# x'$ **by fact** **hence** $x \text{ineq} x': x \neq x'$ **by simp**
moreover **have** $\text{Eq}: a \langle \nu x \rangle \prec (P' \parallel Q) = a' \langle \nu x' \rangle \prec P''$ **by fact**
hence $a \text{eq} a': a = a'$ **by** (*simp add: residual.inject*)
have $x \text{Fresh} Q: x \# Q$ **by fact**
have $x' \# P \parallel Q$ **by fact**
hence $x' \text{Fresh} P: x' \# P$ **and** $x' \text{Fresh} Q: x' \# Q$ **by simp+**
have $P'' \text{eq}: P'' = ([(x, x')] \cdot P') \parallel Q$
proof –
from Eq $x \text{ineq} x'$ **have** $(P' \parallel Q) = ([(x, x')] \cdot P''$
by (*simp add: residual.inject name-abs-eq*)
hence $([(x, x')] \cdot (P' \parallel Q)) = P''$ **by simp**
with $x' \text{Fresh} Q$ $x \text{Fresh} Q$ **show** $? \text{thesis}$ **by** (*simp add: name-fresh-fresh*)

qed

have $x \# P''$ **by fact**
with $P'' \text{eq}$ **have** $x' \text{Fresh} P': x' \# P'$ **by** (*simp add: name-fresh-left name-calc*)

have $P \mapsto a \langle \nu x \rangle \prec P'$ **by fact**
with $x' \text{Fresh} P'$ $a \text{eq} a'$ **have** $P \mapsto a' \langle \nu x' \rangle \prec ([(x, x')] \cdot P')$
by (*simp add: alphaBoundOutput*)
moreover **have** $\bigwedge C. F C P a x' ([(x, x')] \cdot P')$
proof –
fix C
have $\bigwedge C a' x' P''. \llbracket a \langle \nu x \rangle \prec P' = a' \langle \nu x' \rangle \prec P''; x' \# P \rrbracket \implies F C P a'$
 $x' P''$ **by fact**
moreover **with** $a \text{eq} a'$ $x \text{ineq} x'$ $x' \text{Fresh} P'$ **have** $a \langle \nu x \rangle \prec P' = a' \langle \nu x' \rangle \prec$
 $([(x, x')] \cdot P')$
by (*simp add: residual.inject name-abs-eq name-fresh-left name-calc*)
ultimately **show** $F C P a x' ([(x, x')] \cdot P')$ **using** $x' \text{Fresh} P'$ $a \text{eq} a'$ **by blast**

qed

ultimately **have** $F C (P \parallel Q) a' x' ([(x, x')] \cdot P') \parallel Q$ **using** $x' \text{Fresh} Q$
 $a \text{eq} a'$
by (*blast intro: cPar1B*)
with $P'' \text{eq}$ **show** $? \text{case}$ **by simp**

next

case ($\text{Par1} F P P' Q \alpha$)
thus $? \text{case}$ **by** (*simp add: residual.inject*)

```

next
  case(Par2B Q a x Q' P C a' x' Q'')
  have x # x' by fact hence xineqx': x ≠ x' by simp
  moreover have Eq: a<νx> < (P || Q') = a'<νx'> < Q'' by fact
  hence aeqa': a = a' by(simp add: residual.inject)
  have xFreshP: x # P by fact
  have x' # P || Q by fact
  hence x'FreshP: x' # P and x'FreshQ: x' # Q by simp+
  have Q''eq: Q'' = P || ((x, x') · Q')
  proof -
    from Eq xineqx' have (P || Q') = [(x, x')] · Q''
    by(simp add: residual.inject name-abs-eq)
    hence ((x, x') · (P || Q')) = Q'' by simp
    with x'FreshP xFreshP show ?thesis by(simp add: name-fresh-fresh)
  qed

  have x # Q'' by fact
  with Q''eq have x'FreshQ': x' # Q' by(simp add: name-fresh-left name-calc)

  have Q ↦ a<νx> < Q' by fact
  with x'FreshQ' aeqa' have Q ↦ a'<νx'> < ((x, x') · Q')
  by(simp add: alphaBoundOutput)
  moreover have ∧C. F C Q a x' ((x, x') · Q')
  proof -
    fix C
    have ∧C a' x' Q''. [[a<νx> < Q' = a'<νx'> < Q''; x' # Q]] ⇒ F C Q a'
    x' Q'' by fact
    moreover with aeqa' xineqx' x'FreshQ' have a<νx> < Q' = a'<νx'> <
    ((x, x') · Q')
    by(simp add: residual.inject name-abs-eq name-fresh-left name-calc)
    ultimately show F C Q a x' ((x, x') · Q') using x'FreshQ aeqa' by blast
  qed
  ultimately have F C (P || Q) a' x' (P || ((x, x') · Q')) using x'FreshP
  aeqa'
  by(blast intro: cPar2B)
  with Q''eq show ?case by simp
next
  case(Par2F P P' Q α)
  thus ?case by(simp add: residual.inject)
next
  case(Comm1 P P' Q Q' a b x)
  thus ?case by(simp add: residual.inject)
next
  case(Comm2 P P' Q Q' a b x)
  thus ?case by(simp add: residual.inject)
next
  case(Close1 P P' Q Q' a x y)
  thus ?case by(simp add: residual.inject)
next

```

```

    case(Close2 P P' Q Q' a x y)
    thus ?case by(simp add: residual.inject)
  next
    case(ResB P a x P' y C a' x' P'')
    have x # x' by fact hence xineqx': x ≠ x' by simp
    moreover have Eq: a<νx> < (<νy>P') = a'<νx'> < P'' by fact
    hence aeqa': a = a' by(simp add: residual.inject)
    have y # x' by fact hence yineqx': y ≠ x' by simp
    moreover have x' # <νy>P by fact
    ultimately have x'FreshP: x' # P by(simp add: name-fresh-abs)
    have yineqx: y ≠ x and yineqa: y ≠ a and yFreshC: y # C by fact+

    have P''eq: P'' = <νy>([(x, x')] · P')
    proof -
      from Eq yineqx' have <νy>P' = [(x, x')] · P''
      by(simp add: residual.inject name-abs-eq)
      hence ([(x, x')] · (<νy>P')) = P'' by simp
      with yineqx' yineqx show ?thesis by(simp add: name-fresh-fresh)
    qed

    have x # P'' by fact
    with P''eq yineqx have x'FreshP': x' # P' by(simp add: name-fresh-left
name-calc name-fresh-abs)

    have P ⟶ a<νx> < P' by fact
    with x'FreshP' aeqa' have P ⟶ a'<νx'> < ([(x, x')] · P')
    by(simp add: alphaBoundOutput)
    moreover have ∧C. F C P a x' ([(x, x')] · P')
    proof -
      fix C
      have ∧C a' x' P''. ⟦a<νx> < P' = a'<νx'> < P''; x' # P'⟧ ⟹ F C P a'
x' P'' by fact
      moreover with aeqa' yineqx' x'FreshP' have a<νx> < P' = a'<νx'> <
([(x, x')] · P')
      by(simp add: residual.inject name-abs-eq name-fresh-left name-calc)
      ultimately show F C P a x' ([(x, x')] · P') using x'FreshP' aeqa' by blast
    qed
    ultimately have F C (<νy>P) a' x' (<νy>([(x, x')] · P')) using yineqx'
yineqa yFreshC aeqa'
    by(force intro: cResB)
    with P''eq show ?case by simp
  next
    case(ResF P P' α y)
    thus ?case by(simp add: residual.inject)
  next
    case(Bang P Rs)
    thus ?case by(force intro: cBang)
  qed
qed

```

with a x $FreshP$ show $?thesis$ by simp
qed

lemma $\tau Induct$ [*consumes 1, case-names Tau Match Mismatch Sum1 Sum2 Par1 Par2 Comm1 Comm2 Close1 Close2 Res Bang*]:

fixes $P :: pi$
and $P' :: pi$
and $F :: 'a::fs-name \Rightarrow pi \Rightarrow pi \Rightarrow bool$
and $C :: 'a::fs-name$

assumes $Trans: P \mapsto_{\tau} \prec P'$
and $\bigwedge P C. F C (\tau.(P)) P$
and $\bigwedge P P' a C. \llbracket P \mapsto_{\tau} \prec P'; \bigwedge C. F C P P' \rrbracket \Longrightarrow F C ([a \frown a]P) P'$
and $\bigwedge P P' a b C. \llbracket P \mapsto_{\tau} \prec P'; \bigwedge C. F C P P'; a \neq b \rrbracket \Longrightarrow F C ([a \neq b]P) P'$

and $\bigwedge P P' Q C. \llbracket P \mapsto_{\tau} \prec P'; \bigwedge C. F C P P' \rrbracket \Longrightarrow F C (P \oplus Q) P'$
and $\bigwedge Q Q' P C. \llbracket Q \mapsto_{\tau} \prec Q'; \bigwedge C. F C Q Q' \rrbracket \Longrightarrow F C (P \oplus Q) Q'$
and $\bigwedge P P' Q C. \llbracket P \mapsto_{\tau} \prec P'; \bigwedge C. F C P P' \rrbracket \Longrightarrow F C (P \parallel Q) (P' \parallel Q)$
and $\bigwedge Q Q' P C. \llbracket Q \mapsto_{\tau} \prec Q'; \bigwedge C. F C Q Q' \rrbracket \Longrightarrow F C (P \parallel Q) (P \parallel Q')$
and $\bigwedge P a b P' Q Q' C. \llbracket P \mapsto_{a < b >} \prec P'; Q \mapsto_{OutputR} a b \prec Q' \rrbracket \Longrightarrow F C (P \parallel Q) (P' \parallel Q')$
and $\bigwedge P a b P' Q Q' C. \llbracket P \mapsto_{OutputR} a b \prec P'; Q \mapsto_{a < b >} \prec Q' \rrbracket \Longrightarrow F C (P \parallel Q) (P' \parallel Q')$
and $\bigwedge P a x P' Q Q' C. \llbracket P \mapsto_{a < x >} \prec P'; Q \mapsto_{a < \nu x >} \prec Q'; x \# P; x \# Q; x \neq a; x \# C \rrbracket \Longrightarrow F C (P \parallel Q) (<\nu x>(P' \parallel Q'))$
and $\bigwedge P a x P' Q Q' C. \llbracket P \mapsto_{a < \nu x >} \prec P'; Q \mapsto_{a < x >} \prec Q'; x \# P; x \# Q; x \neq a; x \# C \rrbracket \Longrightarrow F C (P \parallel Q) (<\nu x>(P' \parallel Q'))$
and $\bigwedge P P' x C. \llbracket P \mapsto_{\tau} \prec P'; x \# C; \bigwedge C. F C P P' \rrbracket \Longrightarrow F C (<\nu x>P) (<\nu x>P')$
and $\bigwedge P P' C. \llbracket P \parallel !P \mapsto_{\tau} \prec P'; \bigwedge C. F C (P \parallel !P) P' \rrbracket \Longrightarrow F C (!P) P'$

shows $F C P P'$
using $\langle P \mapsto_{\tau} \prec P' \rangle$
by(*nominal-induct x2== $\tau \prec P'$ avoiding: C arbitrary: P' rule: TransitionsEarly.strong-induct*)
(*auto simp add: residual.inject intro: assms*)+

inductive $bangPred :: pi \Rightarrow pi \Rightarrow bool$

where

$aux1: bangPred P (!P)$
 $| aux2: bangPred P (P \parallel !P)$

inductive-cases $\tau Cases$ [*simplified pi.distinct residual.distinct*]: $\tau.(P) \mapsto Rs$
inductive-cases $inputCases$ [*simplified pi.inject residual.inject*]: $a < b >.P \mapsto Rs$
inductive-cases $outputCases$ [*simplified pi.inject residual.inject*]: $a\{b\}.P \mapsto Rs$
inductive-cases $matchCases$ [*simplified pi.inject residual.inject*]: $[a \frown b]P \mapsto Rs$
inductive-cases $mismatchCases$ [*simplified pi.inject residual.inject*]: $[a \neq b]P \mapsto Rs$
inductive-cases $sumCases$ [*simplified pi.inject residual.inject*]: $P \oplus Q \mapsto Rs$


```

note Goal = this
obtain  $y :: \text{name}$  where  $y \# P$  and  $y \# \alpha$  and  $y \# P'$  and  $y \neq a$ 
  by(generate-fresh name) (auto simp add: fresh-prod)
from Input  $\langle y \# P \rangle$  have  $a \langle y \rangle.([(x, y)] \cdot P) \mapsto \alpha \prec P'$  by(simp add: alphaInput)
moreover note  $\langle y \# \alpha \rangle \langle y \# P' \rangle \langle y \neq a \rangle$ 
moreover from A  $\langle y \# P \rangle$  have  $\bigwedge u. \text{Prop } (a \langle u \rangle) (([(x, y)] \cdot P)[y ::= u])$ 
  by(simp add: renaming name-swap)
ultimately show ?thesis by(rule Goal)
qed

```

```

lemma outputCases:
  fixes  $P :: \text{pi}$ 
  and  $\alpha :: \text{freeRes}$ 
  and  $P' :: \text{pi}$ 

  assumes  $a\{b\}.P \mapsto \alpha \prec P'$ 
  and  $\text{Prop } (\text{OutputR } a \ b) \ P$ 

  shows  $\text{Prop } \alpha \ P'$ 
using assms
by(cases rule: outputCases') (auto simp add: pi.inject residual.inject)

```

```

lemma zeroTrans[dest]:
  fixes  $R_s :: \text{residual}$ 

  assumes  $\mathbf{0} \mapsto e \ R_s$ 

  shows False
using assms
by - (ind-cases  $\mathbf{0} \mapsto e \ R_s$ )

```

```

lemma mismatchTrans[dest]:
  fixes  $a :: \text{name}$ 
  and  $P :: \text{pi}$ 
  and  $R_s :: \text{residual}$ 

  assumes  $[a \neq a]P \mapsto R_s$ 

  shows False
using assms
by(erule-tac mismatchCases') auto

```

```

lemma matchCases[consumes 1, case-names Match]:
  fixes  $a :: \text{name}$ 
  and  $b :: \text{name}$ 
  and  $P :: \text{pi}$ 
  and  $R_s :: \text{residual}$ 
  and  $F :: \text{name} \Rightarrow \text{name} \Rightarrow \text{bool}$ 

```

```

assumes Trans:  $[a \sim b]P \mapsto Rs$ 
and cMatch:  $P \mapsto Rs \implies F a a$ 

shows  $F a b$ 
using assms
by(erule-tac matchCases', auto)

lemma mismatchCases[consumes 1, case-names Mismatch]:
  fixes  $a :: name$ 
  and  $b :: name$ 
  and  $P :: pi$ 
  and  $Rs :: residual$ 
  and  $F :: name \Rightarrow name \Rightarrow bool$ 

  assumes Trans:  $[a \neq b]P \mapsto Rs$ 
  and cMatch:  $\llbracket P \mapsto Rs; a \neq b \rrbracket \implies F a b$ 

  shows  $F a b$ 
  using assms
  by(erule-tac mismatchCases') auto

lemma sumCases[consumes 1, case-names Sum1 Sum2]:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $Rs :: residual$ 

  assumes Trans:  $P \oplus Q \mapsto Rs$ 
  and cSum1:  $P \mapsto Rs \implies F$ 
  and cSum2:  $Q \mapsto Rs \implies F$ 

  shows  $F$ 
  using assms
  by(erule-tac sumCases') auto

lemma parCasesB[consumes 1, case-names cPar1 cPar2]:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $a :: name$ 
  and  $x :: name$ 
  and  $PQ' :: pi$ 

  assumes Trans:  $P \parallel Q \mapsto a \langle \nu x \rangle \prec PQ'$ 
  and icPar1B:  $\bigwedge P'. \llbracket P \mapsto a \langle \nu x \rangle \prec P'; x \# Q \rrbracket \implies F (P' \parallel Q)$ 
  and icPar2B:  $\bigwedge Q'. \llbracket Q \mapsto a \langle \nu x \rangle \prec Q'; x \# P \rrbracket \implies F (P \parallel Q')$ 

  shows  $F PQ'$ 
proof –
  from Trans show ?thesis

```

proof (*induct rule: parCasesB', auto simp add: pi.inject residual.inject*)
fix $P' y$
assume $PTrans: P \mapsto a\langle \nu y \rangle \prec P'$
assume $yFreshQ: y \# (Q::pi)$
assume $absEq: [x].PQ' = [y].(P' \parallel Q)$

have $\exists c::name. c \# (P', x, y, Q)$ **by** (*blast intro: name-exists-fresh*)
then obtain c **where** $cFreshP': c \# P'$ **and** $cineqx: x \neq c$ **and** $cineqy: c \neq y$
and $cFreshQ: c \# Q$
by (*force simp add: fresh-prod name-fresh*)

from $cFreshP' PTrans$ **have** $Trans: P \mapsto a\langle \nu c \rangle \prec ([y, c] \cdot P')$ **by** (*simp add: alphaBoundOutput*)

from $cFreshP' cFreshQ$ **have** $c \# P' \parallel Q$ **by** *simp*
hence $[y].(P' \parallel Q) = [c].([y, c] \cdot (P' \parallel Q))$
by (*auto simp add: alpha fresh-left calc-atm*)
with $yFreshQ cFreshQ$ **have** $[y].(P' \parallel Q) = [c].([y, c] \cdot P' \parallel Q)$
by (*simp add: name-fresh-fresh*)

with $cineqx absEq$ **have** $L1: PQ' = [(x, c)] \cdot ([y, c] \cdot P' \parallel Q)$ **and** $L2: x \#$
 $([y, c] \cdot P' \parallel Q)$
by (*simp add: name-abs-eq*)+

from $L2$ **have** $xFreshQ: x \# Q$ **and** $xFreshP': x \# [(y, c)] \cdot P'$ **by** *simp+*
with $cFreshQ L1$ **have** $L3: PQ' = [(x, c)] \cdot [(y, c)] \cdot P' \parallel Q$ **by** (*simp add: name-fresh-fresh*)

from $Trans xFreshP'$ **have** $P \mapsto a\langle \nu x \rangle \prec [(x, c)] \cdot [(y, c)] \cdot P'$ **by** (*simp add: alphaBoundOutput name-swap*)

thus *?thesis using xFreshQ L3*
by (*blast intro: icPar1B*)

next
fix $Q' y$
assume $QTrans: Q \mapsto a\langle \nu y \rangle \prec Q'$
assume $yFreshP: y \# (P::pi)$
assume $absEq: [x].PQ' = [y].(P \parallel Q')$

have $\exists c::name. c \# (Q', x, y, P)$ **by** (*blast intro: name-exists-fresh*)
then obtain c **where** $cFreshQ': c \# Q'$ **and** $cineqx: x \neq c$ **and** $cineqy: c \neq y$
and $cFreshP: c \# P$
by (*force simp add: fresh-prod name-fresh*)

from $cFreshQ' QTrans$ **have** $Trans: Q \mapsto a\langle \nu c \rangle \prec ([y, c] \cdot Q')$ **by** (*simp add: alphaBoundOutput*)

from $cFreshQ' cFreshP$ **have** $c \# P \parallel Q'$ **by** *simp*
hence $[y].(P \parallel Q') = [c].([y, c] \cdot (P \parallel Q'))$

```

    by(auto simp add: alpha fresh-left calc-atm)
  with yFreshP cFreshP have [y].(P || Q') = [c].(P || ((y, c) · Q'))
  by(simp add: name-fresh-fresh)

  with cIneqx absEq have L1: PQ' = [(x, c)] · (P || ((y, c) · Q')) and L2: x #
P || ((y, c) · Q')
  by(simp add: name-abs-eq)+

  from L2 have xFreshP: x # P and xFreshQ': x # [(y, c)] · Q' by simp+
  with cFreshP L1 have L3: PQ' = P || [(x, c)] · [(y, c)] · Q' by(simp add:
name-fresh-fresh)

  from Trans xFreshQ' have Q ↦ a<νx> < [(x, c)] · [(y, c)] · Q' by(simp
add: alphaBoundOutput name-swap)

  thus ?thesis using xFreshP L3
  by(blast intro: icPar2B)
qed
qed

```

lemma *parCasesOutput*[*consumes 1, case-names Par1 Par2*]:

```

  fixes P :: pi
  and Q :: pi
  and a :: name
  and b :: name
  and P' :: pi

```

```

  assumes P || Q ↦ a[b] < PQ'
  and   ∧ P'. [[P ↦ a[b] < P']] ⇒ F (P' || Q)
  and   ∧ Q'. [[Q ↦ a[b] < Q']] ⇒ F (P || Q')

```

```

  shows F PQ'
  using assms
  by(erule-tac parCasesF', auto simp add: pi.inject residual.inject)

```

lemma *parCasesInput*[*consumes 1, case-names Par1 Par2*]:

```

  fixes P :: pi
  and Q :: pi
  and a :: name
  and b :: name
  and P' :: pi

```

```

  assumes Trans: P || Q ↦ a<b> < PQ'
  and   icPar1F: ∧ P'. [[P ↦ a<b> < P']] ⇒ F (P' || Q)
  and   icPar2F: ∧ Q'. [[Q ↦ a<b> < Q']] ⇒ F (P || Q')

```

```

  shows F PQ'
  using assms
  by(erule-tac parCasesF') (auto simp add: pi.inject residual.inject)

```

```

lemma parCasesF[consumes 1, case-names cPar1 cPar2 cComm1 cComm2 cClose1
cClose2]:
  fixes P :: pi
  and Q :: pi
  and α :: freeRes
  and P' :: pi
  and C :: 'a::fs-name

  assumes Trans:  $P \parallel Q \mapsto \alpha \prec PQ'$ 
  and icPar1F:  $\bigwedge P'. \llbracket P \mapsto \alpha \prec P \rrbracket \implies F \alpha (P' \parallel Q)$ 
  and icPar2F:  $\bigwedge Q'. \llbracket Q \mapsto \alpha \prec Q \rrbracket \implies F \alpha (P \parallel Q')$ 
  and icComm1:  $\bigwedge P' Q' a b. \llbracket P \mapsto a \langle b \rangle \prec P'; Q \mapsto a[b] \prec Q \rrbracket \implies F (\tau)$ 
  (P' \parallel Q')
  and icComm2:  $\bigwedge P' Q' a b. \llbracket P \mapsto a[b] \prec P'; Q \mapsto a \langle b \rangle \prec Q \rrbracket \implies F (\tau)$ 
  (P' \parallel Q')
  and icClose1:  $\bigwedge P' Q' a x. \llbracket P \mapsto a \langle x \rangle \prec P'; Q \mapsto a \langle \nu x \rangle \prec Q'; x \# P;$ 
   $x \# C \rrbracket \implies F (\tau) (\langle \nu x \rangle (P' \parallel Q'))$ 
  and icClose2:  $\bigwedge P' Q' a x. \llbracket P \mapsto a \langle \nu x \rangle \prec P'; Q \mapsto a \langle x \rangle \prec Q'; x \# Q;$ 
   $x \# C \rrbracket \implies F (\tau) (\langle \nu x \rangle (P' \parallel Q'))$ 

  shows  $F \alpha PQ'$ 
proof –
  from Trans show ?thesis
proof(rule parCasesF', auto)
  fix Pa Pa' Qa α'
  assume Trans':  $Pa \mapsto \alpha' \prec Pa'$ 
  assume Eq:  $P \parallel Q = Pa \parallel Qa$ 
  assume Eq':  $\alpha \prec PQ' = \alpha' \prec Pa' \parallel Qa$ 

  from Eq have  $P = Pa$  and  $Q = Qa$ 
  by(simp add: pi.inject)+

  moreover with Eq' have  $\alpha = \alpha'$  and  $PQ' = Pa' \parallel Q$ 
  by(simp add: residual.inject)+

  ultimately show ?thesis using icPar1F Trans'
  by simp
next
  fix Pa Qa Qa' α'
  assume Trans':  $Qa \mapsto \alpha' \prec Qa'$ 
  assume Eq:  $P \parallel Q = Pa \parallel Qa$ 
  assume Eq':  $\alpha \prec PQ' = \alpha' \prec Pa \parallel Qa'$ 

  from Eq have  $P = Pa$  and  $Q = Qa$ 
  by(simp add: pi.inject)+

  moreover with Eq' have  $\alpha = \alpha'$  and  $PQ' = P \parallel Qa'$ 
  by(simp add: residual.inject)+

```

```

ultimately show ?thesis using icPar2F Trans'
  by simp
next
fix Pa Pa' Qa Qa' a b
assume TransP: Pa  $\mapsto$  a<b>  $\prec$  Pa'
assume TransQ: Qa  $\mapsto$  a[b]  $\prec$  Qa'
assume Eq: P  $\parallel$  Q = Pa  $\parallel$  Qa
assume Eq':  $\alpha \prec$  PQ' =  $\tau \prec$  Pa'  $\parallel$  Qa'

from TransP TransQ Eq Eq' icComm1 show ?thesis
  by(simp add: pi.inject residual.inject)
next
fix Pa Pa' Qa Qa' a b x
assume TransP: Pa  $\mapsto$  (a::name)[b]  $\prec$  Pa'
assume TransQ: Qa  $\mapsto$  a<b>  $\prec$  Qa'
assume Eq: P  $\parallel$  Q = Pa  $\parallel$  Qa
assume Eq':  $\alpha \prec$  PQ' =  $\tau \prec$  Pa'  $\parallel$  Qa'

from TransP TransQ Eq Eq' icComm2 show ?thesis
  by(simp add: pi.inject residual.inject)
next
fix Pa Pa' Qa Qa' a x
assume TransP: Pa  $\mapsto$  a<x>  $\prec$  Pa'
assume TransQ: Qa  $\mapsto$  a< $\nu$ x>  $\prec$  Qa'
assume xFreshPa: x  $\#$  Pa
assume Eq: P  $\parallel$  Q = Pa  $\parallel$  Qa
assume Eq':  $\alpha \prec$  PQ' =  $\tau \prec$  < $\nu$ x>(Pa'  $\parallel$  Qa')

have  $\exists$  (c::name). c  $\#$  (Pa, Pa', x, Qa', a, C)
  by(blast intro: name-exists-fresh)
then obtain c::name where cFreshPa: c  $\#$  Pa and cFreshPa': c  $\#$  Pa' and
cineqy: c  $\neq$  x and cFreshQa': c  $\#$  Qa' and cFreshC: c  $\#$  C and cineqa: c  $\neq$  a
  by(force simp add: fresh-prod name-fresh)

from cFreshQa' have L1: a< $\nu$ x>  $\prec$  Qa' = a< $\nu$ c>  $\prec$  (([x, c])  $\cdot$  Qa')
  by(simp add: alphaBoundOutput)
with cFreshQa' cFreshPa' have c  $\#$  (Pa'  $\parallel$  Qa')
  by simp
then have L4: < $\nu$ x>(Pa'  $\parallel$  Qa') = < $\nu$ c>(((x, c])  $\cdot$  Pa')  $\parallel$  ((x, c])  $\cdot$  Qa')
  by(simp add: alphaRes)

have TransP: Pa  $\mapsto$  a<c>  $\prec$  [(x, c)]  $\cdot$  Pa'
proof -
  from xFreshPa TransP have xineqa: x  $\neq$  a by(force dest: freshAction)
  from TransP have ((x, c])  $\cdot$  Pa  $\mapsto$  [(x, c)]  $\cdot$  (a<x>  $\prec$  Pa')
    by(rule TransitionsEarly.eqt)
  with xineqa xFreshPa cFreshPa cineqa show ?thesis
    by(simp add: name-fresh-fresh name-calc)

```

```

qed

with TransQ L1 L4 icClose1 Eq Eq' cFreshPa cFreshC show ?thesis
  by(simp add: residual.inject, simp add: pi.inject)
next
fix Pa Pa' Qa Qa' a x
assume TransP: Pa  $\mapsto$  a< $\nu$ x> < Pa'
assume TransQ: Qa  $\mapsto$  a<x> < Qa'
assume xFreshQa: x # Qa
assume Eq: P || Q = Pa || Qa
assume Eq':  $\alpha$  < PQ' =  $\tau$  < < $\nu$ x>(Pa' || Qa')

have  $\exists$  (c::name). c # (Qa, Pa', x, Qa', a, C)
  by(blast intro: name-exists-fresh)
then obtain c::name where cFreshQa: c # Qa and cFreshPa': c # Pa' and
cineq: c  $\neq$  x and cFreshQa': c # Qa' and cFreshC: c # C and cineqa: c  $\neq$  a
  by(force simp add: fresh-prod name-fresh)

from cFreshPa' have L1: a< $\nu$ x> < Pa' = a< $\nu$ c> < [(x, c)]  $\cdot$  Pa'^
  by(simp add: alphaBoundOutput)
with cFreshQa' cFreshPa' have c # (Pa' || Qa')
  by simp
then have L4: < $\nu$ x>(Pa' || Qa') = < $\nu$ c>([(x, c)]  $\cdot$  Pa'^ || [(x, c)]  $\cdot$  Qa'^)
  by(simp add: alphaRes)

have TransQ: Qa  $\mapsto$  a<c> < [(x, c)]  $\cdot$  Qa'
proof -
  from xFreshQa TransQ have xineqa: x  $\neq$  a by(force dest: freshAction)
  from TransQ have [(x, c)]  $\cdot$  Qa  $\mapsto$  [(x, c)]  $\cdot$  (a<x> < Qa')
    by(rule TransitionsEarly.eqt)
  with xineqa xFreshQa cFreshQa cineqa show ?thesis
    by(simp add: name-fresh-fresh name-calc)
qed

with TransP L1 L4 icClose2 Eq Eq' cFreshQa cFreshC show ?thesis
  by(simp add: residual.inject, simp add: pi.inject)
qed
qed

lemma resCasesF[consumes 2, case-names Res]:
  fixes x :: name
  and P :: pi
  and  $\alpha$  :: freeRes
  and P' :: pi

  assumes Trans: < $\nu$ x>P  $\mapsto$   $\alpha$  < RP'
  and xFreshAlpha: x #  $\alpha$ 
  and rcResF:  $\bigwedge$ P'. P  $\mapsto$   $\alpha$  < P'  $\implies$  F (< $\nu$ x>P')

```

shows $F RP'$
proof –
from *Trans* **show** *?thesis*
proof(*induct rule: resCasesF', auto*)
fix $Pa Pa' \beta y$
assume $PTrans: Pa \mapsto \beta \prec Pa'$
assume $yFreshBeta: (y::name) \# \beta$
assume $TermEq: \langle \nu x \rangle P = \langle \nu y \rangle Pa$
assume $ResEq: \alpha \prec RP' = \beta \prec \langle \nu y \rangle Pa'$

hence $alphaeqbeta: \alpha = \beta$ **and** $L2: RP' = \langle \nu y \rangle Pa'$ **by**(*simp add: residual.inject*)

have $\exists (c::name). c \# (Pa, \alpha, Pa', x, y)$ **by**(*blast intro: name-exists-fresh*)
then obtain $c::name$ **where** $cFreshPa: c \# Pa$ **and** $cFreshAlpha: c \# \alpha$ **and**
 $cFreshPa': c \# Pa'$ **and** $cineqx: x \neq c$ **and** $cineqy: c \neq y$
by(*force simp add: fresh-prod name-fresh*)

from $cFreshPa$ **have** $\langle \nu y \rangle Pa = \langle \nu c \rangle ([y, c] \cdot Pa)$ **by**(*rule alphaRes*)
with $TermEq cineqx$ **have** $Peq: P = [(x, c)] \cdot [(y, c)] \cdot Pa$ **and** $xeq: x \# [(y, c)]$
 $\cdot Pa$
by(*simp add: pi.inject name-abs-eq*)

from $PTrans$ **have** $[(y, c)] \cdot Pa \mapsto [(y, c)] \cdot (\beta \prec Pa')$ **by**(*rule TransitionsEarly.eqt*)
with $yFreshBeta cFreshAlpha alphaeqbeta$ **have** $PTrans': [(y, c)] \cdot Pa \mapsto \alpha$
 $\prec [(y, c)] \cdot Pa'$
by(*simp add: name-fresh-fresh*)

from $PTrans'$ **have** $[(x, c)] \cdot [(y, c)] \cdot Pa \mapsto [(x, c)] \cdot (\alpha \prec [(y, c)] \cdot Pa')$
by(*rule TransitionsEarly.eqt*)
with $xFreshAlpha cFreshAlpha Peq$ **have** $PTrans'': P \mapsto \alpha \prec [(x, c)] \cdot [(y, c)]$
 $\cdot Pa'$
by(*simp add: name-fresh-fresh*)

from $PTrans'$ $xeq xFreshAlpha$ **have** $xeq': x \# [(y, c)] \cdot Pa'$
by(*nominal-induct α rule: freeRes.strong-induct*)
(*auto simp add: fresh-left calc-atm eqts dest: freshTransition*)

from $cFreshPa'$ **have** $\langle \nu y \rangle Pa' = \langle \nu c \rangle ([y, c] \cdot Pa')$ **by**(*rule alphaRes*)
moreover from xeq' **have** $\langle \nu c \rangle ([y, c] \cdot Pa') = \langle \nu x \rangle ([c, x] \cdot [(y, c)] \cdot Pa')$
 Pa'
by(*rule alphaRes*)
ultimately have $RP' = \langle \nu x \rangle ([c, x] \cdot [(y, c)] \cdot Pa')$ **using** $ResEq$
by(*simp add: residual.inject name-swap*)

with $PTrans'' xFreshAlpha$ **show** *?thesis*
by(*blast intro: rcResF*)
qed

qed

lemma *resCasesB*[*consumes 2, case-names Open Res*]:

fixes $x :: \text{name}$
and $P :: \text{pi}$
and $a :: \text{name}$
and $y :: \text{name}$
and $RP' :: \text{pi}$

assumes *Trans*: $\langle \nu y \rangle P \mapsto a \langle \nu x \rangle \prec RP'$
and *xineqy*: $x \neq y$
and *rcOpen*: $\bigwedge P'. \llbracket P \mapsto (\text{OutputR } a \ y) \prec P'; a \neq y \rrbracket \implies F \llbracket (x, y) \rrbracket \cdot P'$
and *rcResB*: $\bigwedge P'. \llbracket P \mapsto a \langle \nu x \rangle \prec P'; y \neq a \rrbracket \implies F \langle \nu y \rangle P'$

shows $F RP'$

proof –

from *Trans* **show** *?thesis*

proof(*induct rule: resCasesB', auto*)

fix $Pa \ Pa' \ aa \ b$

assume *PTrans*: $Pa \mapsto (aa :: \text{name})[b] \prec Pa'$

assume *aaineqb*: $aa \neq b$

assume *TermEq*: $\langle \nu y \rangle P = \langle \nu b \rangle Pa$

assume *ResEq*: $a \langle \nu x \rangle \prec RP' = aa \langle \nu b \rangle \prec Pa'$

have $\exists (c :: \text{name}). c \# (x, a, aa, y, Pa, Pa', b)$ **by**(*blast intro: name-exists-fresh*)
then obtain c **where** *cineqx*: $c \neq x$ **and** *cFresha*: $c \# a$ **and** *cineqy*: $c \neq y$ **and**
cineqaa: $c \neq aa$ **and** *cFreshPa*: $c \# Pa$ **and** *cFreshPa'*: $c \# Pa'$ **and** *cineqb*: $c \neq b$
by(*force simp add: fresh-prod name-fresh*)

from *cFreshPa* **have** $\langle \nu b \rangle Pa = \langle \nu c \rangle ((b, c) \cdot Pa)$ **by**(*rule alphaRes*)

with *cineqy TermEq* **have** *PEq*: $P = ((y, c) \cdot [(b, c)] \cdot Pa)$ **and** *yFreshPa*: $y \# [(b, c)] \cdot Pa$

by(*simp add: pi.inject name-abs-eq*)+

from *PTrans* **have** $[(b, c)] \cdot Pa \mapsto [(b, c)] \cdot (aa[b] \prec Pa')$ **by**(*rule TransitionsEarly.eqt*)

with *aaineqb cineqaa* **have** *L1*: $[(b, c)] \cdot Pa \mapsto aa[c] \prec [(b, c)] \cdot Pa'$ **by**(*simp add: name-calc*)

with *yFreshPa* **have** *yineqaa*: $y \neq aa$ **by**(*force dest: freshAction*)

from *L1 yFreshPa cineqy* **have** *yFreshPa'*: $y \# [(b, c)] \cdot Pa'$ **by**(*force intro: freshTransition*)

from *L1* **have** $[(y, c)] \cdot [(b, c)] \cdot Pa \mapsto [(y, c)] \cdot (aa[c] \prec [(b, c)] \cdot Pa')$

by(*rule TransitionsEarly.eqt*)

with *cineqaa yineqaa cineqy PEq* **have** *PTrans*: $P \mapsto aa[y] \prec [(y, c)] \cdot [(b, c)] \cdot Pa'$

by(*simp add: name-calc*)

moreover from *cFreshPa'* **have** $aa \langle \nu b \rangle \prec Pa' = aa \langle \nu c \rangle \prec ((b, c) \cdot Pa')$
by(*rule alphaBoundOutput*)

with $ResEq$ $cineq_x$ **have** $ResEq'$: $RP' = [(x, c)] \cdot [(b, c)] \cdot Pa'$ **and** $x \# [(b, c)] \cdot Pa'$
by (*simp add: residual.inject name-abs-eq*) +
with $xineq_y$ $cineq_y$ $cineq_x$ $yFreshPa'$ **have** $RP' = [(x, y)] \cdot [(y, c)] \cdot [(b, c)] \cdot Pa'$
by (*subst pt-perm-compose[OF pt-name-inst, OF at-name-inst], simp add: name-calc name-fresh-fresh*)
moreover from $ResEq$ **have** $a=aa$ **by** (*simp add: residual.inject*)
ultimately show *?thesis* **using** $yineq_{aa}$ $rcOpen$
by blast
next
fix Pa Pa' aa xa ya
assume $PTrans$: $Pa \mapsto aa \langle \nu xa \rangle \prec Pa'$
assume $yaFreshaa$: $(ya::name) \neq aa$
assume $yaIneq_xa$: $ya \neq xa$
assume $EqTrans$: $\langle \nu y \rangle P = \langle \nu ya \rangle Pa$
assume $EqRes$: $a \langle \nu x \rangle \prec RP' = aa \langle \nu xa \rangle \prec (\langle \nu ya \rangle Pa')$

hence aeq_{aa} : $a = aa$ **by** (*simp add: residual.inject*)
with $yaFreshaa$ **have** $yaFresha$: $ya \# a$ **by** *simp*

have $\exists (c::name). c \# (Pa', y, xa, ya, x, Pa, aa)$ **by** (*blast intro: name-exists-fresh*)
then obtain c **where** $cFreshPa'$: $c \# Pa'$ **and** $cineq_y$: $c \neq y$ **and** $cineq_xa$: $c \neq xa$ **and** $cineq_ya$: $c \neq ya$ **and** $cineq_x$: $c \neq x$ **and** $cFreshP$: $c \# Pa$ **and** $cFreshaa$: $c \# aa$
by (*force simp add: fresh-prod name-fresh*)

have $\exists (d::name). d \# (Pa, a, x, Pa', c, xa, ya, y)$ **by** (*blast intro: name-exists-fresh*)
then obtain d **where** $dFreshPa$: $d \# Pa$ **and** $dFresha$: $d \# a$ **and** $dineq_x$: $d \neq x$ **and** $dFreshPa'$: $d \# Pa'$ **and** $dineq_c$: $d \neq c$ **and** $dineq_xa$: $d \neq xa$ **and** $dineq_ya$: $d \neq ya$ **and** $dineq_y$: $d \neq y$
by (*force simp add: fresh-prod name-fresh*)

from $dFreshPa$ **have** $\langle \nu ya \rangle Pa = \langle \nu d \rangle ([ya, d] \cdot Pa)$ **by** (*rule alphaRes*)
with $EqTrans$ $dineq_y$ **have** PEq : $P = [(y, d)] \cdot [(ya, d)] \cdot Pa$
and $yFreshPa$: $y \# [(ya, d)] \cdot Pa$
by (*simp add: pi.inject name-abs-eq*) +

from $dFreshPa'$ **have** $L1$: $\langle \nu ya \rangle Pa' = \langle \nu d \rangle ([ya, d] \cdot Pa')$ **by** (*rule alphaRes*)
from $cFreshPa'$ $dineq_c$ $cineq_ya$ **have** $c \# \langle \nu d \rangle ([ya, d] \cdot Pa')$
by (*simp add: name-fresh-abs name-calc name-fresh-left*)
hence $aa \langle \nu xa \rangle \prec (\langle \nu d \rangle ([ya, d] \cdot Pa')) = aa \langle \nu c \rangle \prec ([xa, c] \cdot \langle \nu d \rangle ([ya, d] \cdot Pa'))$ (*is ?LHS = -*)
by (*rule alphaBoundOutput*)
with $dineq_xa$ $dineq_c$ **have** *?LHS* = $aa \langle \nu c \rangle \prec (\langle \nu d \rangle ([xa, c] \cdot [(ya, d)] \cdot Pa'))$
by (*simp add: name-calc*)
with $L1$ $EqRes$ $cineq_x$ $dineq_c$ $dineq_x$ **have**
 $RP'Eq$: $RP' = \langle \nu d \rangle ([x, c] \cdot [(xa, c)] \cdot [(ya, d)] \cdot Pa')$

and $xFreshPa'$: $x \# [(xa, c)] \cdot [(ya, d)] \cdot Pa'$
by(*simp add: residual.inject name-abs-eq name-fresh-abs name-calc*)

from $PTrans$ *aeqaa* **have** $[(ya, d)] \cdot Pa \mapsto [(ya, d)] \cdot (a < \nu xa > \prec Pa')$
by(*blast intro: TransitionsEarly.eqvt*)
with $yaineqxa$ $yaFresha$ $dineqxa$ $dFresha$ **have** $L1$:
 $[(ya, d)] \cdot Pa \mapsto a < \nu xa > \prec [(ya, d)] \cdot Pa'$ **by**(*simp add: name-calc name-fresh-fresh*)
with $yFreshPa$ **have** $yineqa$: $y \neq a$ **by**(*force dest: freshAction*)
from $dineqc$ $cinetya$ $cFreshPa'$ **have** $c \# [(ya, d)] \cdot Pa'$
by(*simp add: name-fresh-left name-calc*)
hence $a < \nu xa > \prec [(ya, d)] \cdot Pa' = a < \nu c > \prec [(xa, c)] \cdot [(ya, d)] \cdot Pa'$ (**is**
 $?LHS = -$)
by(*rule alphaBoundOutput*)
with $xFreshPa'$ **have** $L2$: $?LHS = a < \nu x > \prec [(c, x)] \cdot [(xa, c)] \cdot [(ya, d)] \cdot Pa'$
by(*simp add: alphaBoundOutput*)
with $L1$ PEq **have** $P \mapsto [(y, d)] \cdot (a < \nu x > \prec [(c, x)] \cdot [(xa, c)] \cdot [(ya, d)] \cdot Pa')$
by(*force intro: TransitionsEarly.eqvt simp del: residual.perm*)
with $yineqa$ $dFresha$ $xineqy$ $dineqx$ **have** $Trans$: $P \mapsto a < \nu x > \prec [(y, d)] \cdot [(c, x)] \cdot [(xa, c)] \cdot [(ya, d)] \cdot Pa'$
by(*simp add: name-calc name-fresh-fresh*)

from $L1$ $L2$ $yFreshPa$ $xineqy$ **have** $y \# [(c, x)] \cdot [(xa, c)] \cdot [(ya, d)] \cdot Pa'$
by(*force intro: freshTransition*)
with $RP'Eq$ **have** $RP' = < \nu y >([(y, d)] \cdot [(c, x)] \cdot [(xa, c)] \cdot [(ya, d)] \cdot Pa')$
by(*simp add: alphaRes name-swap*)

with $Trans$ $yineqa$ **show** $?thesis$
by(*blast intro: rcResB*)
qed
qed

lemma *bangInduct[consumes 1, case-names Par1B Par1F Par2B Par2F Comm1 Comm2 Close1 Close2 Bang]*:

fixes $F :: 'a::fs-name \Rightarrow pi \Rightarrow residual \Rightarrow bool$
and $P :: pi$
and $Rs :: residual$
and $C :: 'a::fs-name$

assumes $Trans$: $!P \mapsto Rs$
and $cPar1B$: $\bigwedge a x P' C. \llbracket P \mapsto a < \nu x > \prec P'; x \# P; x \# C \rrbracket \Longrightarrow F C (P \parallel !P) (a < \nu x > \prec (P' \parallel !P))$
and $cPar1F$: $\bigwedge (\alpha::freeRes) (P'::pi) C. \llbracket P \mapsto \alpha \prec P' \rrbracket \Longrightarrow F C (P \parallel !P) (\alpha \prec P' \parallel !P)$
and $cPar2B$: $\bigwedge a x P' C. \llbracket !P \mapsto a < \nu x > \prec P'; x \# P; x \# C; \bigwedge C. F C (!P) (a < \nu x > \prec P') \rrbracket \Longrightarrow F C (P \parallel !P) (a < \nu x > \prec (P \parallel P'))$
and $cPar2F$: $\bigwedge \alpha P' C. \llbracket !P \mapsto \alpha \prec P'; \bigwedge C. F C (!P) (\alpha \prec P') \rrbracket \Longrightarrow F C$

$(P \parallel !P) (\alpha \prec P \parallel P')$
and $cComm1: \bigwedge a P' b P'' C. \llbracket P \mapsto a \langle b \rangle \prec P'; !P \mapsto (OutputR a b) \prec P''; \bigwedge C. F C (!P) ((OutputR a b) \prec P'') \rrbracket \implies$
 $F C (P \parallel !P) (\tau \prec P' \parallel P'')$
and $cComm2: \bigwedge a b P' P'' C. \llbracket P \mapsto (OutputR a b) \prec P'; !P \mapsto a \langle b \rangle \prec P''; \bigwedge C. F C (!P) (a \langle b \rangle \prec P'') \rrbracket \implies$
 $F C (P \parallel !P) (\tau \prec P' \parallel P'')$
and $cClose1: \bigwedge a x P' P'' C. \llbracket P \mapsto a \langle x \rangle \prec P'; !P \mapsto a \langle \nu x \rangle \prec P''; x \# P; x \# C; \bigwedge C. F C (!P) (a \langle \nu x \rangle \prec P'') \rrbracket \implies$
 $F C (P \parallel !P) (\tau \prec \langle \nu x \rangle (P' \parallel P''))$
and $cClose2: \bigwedge a x P' P'' C. \llbracket P \mapsto a \langle \nu x \rangle \prec P'; !P \mapsto a \langle x \rangle \prec P''; x \# P; x \# C; \bigwedge C. F C (!P) (a \langle x \rangle \prec P'') \rrbracket \implies$
 $F C (P \parallel !P) (\tau \prec \langle \nu x \rangle (P' \parallel P''))$
and $cBang: \bigwedge Rs C. \llbracket P \parallel !P \mapsto Rs; \bigwedge C. F C (P \parallel !P) Rs \rrbracket \implies F C (!P) Rs$

shows $F C (!P) Rs$

proof –

have $\bigwedge X Y C. \llbracket X \mapsto Y; bangPred P X \rrbracket \implies F C X Y$

proof –

fix $X Y C$

assume $X \mapsto Y$ **and** $bangPred P X$

thus $F C X Y$

proof(*nominal-induct avoiding: C rule: TransitionsEarly.strong-induct*)

case($Tau Pa$)

thus *?case*

apply –

by(*ind-cases bangPred P* ($\tau.(Pa)$))

next

case($Input x a u Pa C$)

thus *?case*

by – (*ind-cases bangPred P* ($a \langle x \rangle . Pa$))

next

case($Output a b Pa C$)

thus *?case*

by – (*ind-cases bangPred P* ($a \{ b \} . Pa$))

next

case($Match Pa Rs b C$)

thus *?case*

by – (*ind-cases bangPred P* ($[b \curvearrowright] Pa$))

next

case($Mismatch Pa Rs a b C$)

thus *?case*

by – (*ind-cases bangPred P* ($[a \neq b] Pa$))

next

case($Open Pa a b Pa'$)

thus *?case*

by – (*ind-cases bangPred P* ($\langle \nu b \rangle Pa$))

next

```

    case(Sum1 Pa Rs Q)
  thus ?case
    by - (ind-cases bangPred P (Pa  $\oplus$  Q))
next
  case(Sum2 Q Rs Pa)
  thus ?case
    by - (ind-cases bangPred P (Pa  $\oplus$  Q))
next
  case(Par1B Pa a x P' Q C)
  thus ?case
    by - (ind-cases bangPred P (Pa  $\parallel$  Q), auto simp add: pi.inject cPar1B)
next
  case(Par1F Pa  $\alpha$  P' Q C)
  thus ?case
    by - (ind-cases bangPred P (Pa  $\parallel$  Q), auto simp add: pi.inject cPar1F)
next
  case(Par2B Q a x Q' Pa)
  thus ?case
    by - (ind-cases bangPred P (Pa  $\parallel$  Q), auto simp add: pi.inject aux1 cPar2B)
next
  case(Par2F Q  $\alpha$  Q' Pa)
  thus ?case
    by - (ind-cases bangPred P (Pa  $\parallel$  Q), auto simp add: pi.inject intro: cPar2F
aux1)
next
  case(Comm1 Pa a b Pa' Q Q' C)
  thus ?case
    by - (ind-cases bangPred P (Pa  $\parallel$  Q), auto simp add: pi.inject intro: cComm1
aux1)
next
  case(Comm2 Pa a b Pa' Q P'' C)
  thus ?case
    by - (ind-cases bangPred P (Pa  $\parallel$  Q), auto simp add: pi.inject intro: cComm2
aux1)
next
  case(Close1 Pa a x Pa' Q Q'' C)
  thus ?case
    by - (ind-cases bangPred P (Pa  $\parallel$  Q), auto simp add: pi.inject aux1 cClose1)
next
  case(Close2 Pa a x Pa' Q Q' C)
  thus ?case
    by - (ind-cases bangPred P (Pa  $\parallel$  Q), auto simp add: pi.inject aux1 cClose2)
next
  case(ResB Pa a x Pa' y)
  thus ?case
    by - (ind-cases bangPred P (< $\nu$ y>Pa))
next
  case(ResF Pa  $\alpha$  Pa' y)
  thus ?case

```

```

    by - (ind-cases bangPred P (<νy>Pa))
  next
  case(Bang Pa Rs)
  thus ?case
    by - (ind-cases bangPred P (!Pa), auto simp add: pi.inject intro: aux2
cBang)
  qed
  qed
  with Trans show ?thesis by(force intro: bangPred.aux1)
qed

end

```

```

theory Strong-Early-Sim
  imports Early-Semantics Rel
begin

```

```

definition strongSimEarly :: pi ⇒ (pi × pi) set ⇒ pi ⇒ bool (⟨- ~[-] -⟩ [80, 80,
80] 80) where
  P ~[-] [Rel] Q ≡ (∀ a y Q'. Q ⟶ a⟨νy⟩ < Q' ⟶ y # P ⟶ (∃ P'. P ⟶ a⟨νy⟩
< P' ∧ (P', Q') ∈ Rel)) ∧
  (∀ α Q'. Q ⟶ α < Q' ⟶ (∃ P'. P ⟶ α < P' ∧ (P', Q') ∈ Rel))

```

lemma *monotonic*:

```

fixes A :: (pi × pi) set
and B :: (pi × pi) set
and P :: pi
and P' :: pi

```

```

assumes P ~[-] [A] P'
and A ⊆ B

```

```

shows P ~[-] [B] P'

```

using *assms*

by(*fastforce simp add: strongSimEarly-def*)

lemma *freshUnit[simp]*:

```

fixes y :: name

```

```

shows y # ()

```

by(*auto simp add: fresh-def supp-unit*)

lemma *simCasesCont[consumes 1, case-names Bound Free]*:

```

fixes P :: pi
and Q :: pi
and Rel :: (pi × pi) set
and C :: 'a::fs-name

```

```

assumes Eqvt: eqvt Rel

```

and *Bound*: $\bigwedge a y Q'. \llbracket Q \mapsto a \langle \nu y \rangle \prec Q'; y \# P; y \# Q; y \# C \rrbracket \implies \exists P'. P \mapsto a \langle \nu y \rangle \prec P' \wedge (P', Q') \in \text{Rel}$

and *Free*: $\bigwedge \alpha Q'. Q \mapsto \alpha \prec Q' \implies \exists P'. P \mapsto \alpha \prec P' \wedge (P', Q') \in \text{Rel}$

shows $P \rightsquigarrow[\text{Rel}] Q$

proof –

from *Free* **show** *?thesis*

proof(*auto simp add: strongSimEarly-def*)

fix $Q' a y$

assume $y\text{Fresh}P: (y::\text{name}) \# P$

assume $\text{Trans}: Q \mapsto a \langle \nu y \rangle \prec Q'$

have $\exists c::\text{name}. c \# (P, Q', y, Q, C)$ **by**(*blast intro: name-exists-fresh*)

then obtain $c::\text{name}$ **where** $c\text{Fresh}P: c \# P$ **and** $c\text{Fresh}Q': c \# Q'$ **and** $c\text{Fresh}C: c \# C$

and $\text{cineqy}: c \neq y$ **and** $c \# Q$

by(*force simp add: fresh-prod name-fresh*)

from $\text{Trans } c\text{Fresh}Q'$ **have** $Q \mapsto a \langle \nu c \rangle \prec ((y, c) \cdot Q')$ **by**(*simp add: alphaBoundOutput*)

hence $\exists P'. P \mapsto a \langle \nu c \rangle \prec P' \wedge (P', [(y, c)] \cdot Q') \in \text{Rel}$ **using** $\langle c \# P \rangle \langle c \# Q \rangle \langle c \# C \rangle$

by(*rule Bound*)

then obtain P' **where** $P\text{Trans}: P \mapsto a \langle \nu c \rangle \prec P'$ **and** $P'\text{Rel}Q': (P', [(y, c)] \cdot Q') \in \text{Rel}$

by *blast*

from $P\text{Trans } y\text{Fresh}P$ cineqy **have** $y\text{Fresh}P': y \# P'$ **by**(*force intro: freshTransition*)

with $P\text{Trans}$ **have** $P \mapsto a \langle \nu y \rangle \prec ((y, c) \cdot P')$ **by**(*simp add: alphaBoundOutput name-swap*)

moreover have $((y, c) \cdot P', Q') \in \text{Rel}$ (**is** *?goal*)

proof –

from $\text{Eqvt } P'\text{Rel}Q'$ **have** $((y, c) \cdot P', [(y, c)] \cdot [(y, c)] \cdot Q') \in \text{Rel}$

by(*rule eqvtRelI*)

with cineqy **show** *?goal* **by**(*simp add: name-calc*)

qed

ultimately show $\exists P'. P \mapsto a \langle \nu y \rangle \prec P' \wedge (P', Q') \in \text{Rel}$ **by** *blast*

qed

qed

lemma *simCases*[*consumes 0, case-names Bound Free*]:

fixes $P :: pi$

and $Q :: pi$

and $\text{Rel} :: (pi \times pi)$ *set*

and $C :: 'a::\text{fs-name}$

assumes *Bound*: $\bigwedge a y Q'. \llbracket Q \mapsto a \langle \nu y \rangle \prec Q'; y \# P \rrbracket \implies \exists P'. P \mapsto a \langle \nu y \rangle \prec P' \wedge (P', Q') \in \text{Rel}$

and *Free*: $\bigwedge \alpha Q'. Q \mapsto \alpha \prec Q' \implies \exists P'. P \mapsto \alpha \prec P' \wedge (P', Q') \in \text{Rel}$

```

  shows  $P \rightsquigarrow[Rel] Q$ 
using assms
by(auto simp add: strongSimEarly-def)

lemma elim:
  fixes  $P :: pi$ 
  and  $Rel :: (pi \times pi) set$ 
  and  $Q :: pi$ 
  and  $a :: name$ 
  and  $x :: name$ 
  and  $Q' :: pi$ 

  assumes  $P \rightsquigarrow[Rel] Q$ 

  shows  $Q \mapsto a \langle \nu x \rangle \prec Q' \implies x \# P \implies \exists P'. P \mapsto a \langle \nu x \rangle \prec P' \wedge (P', Q') \in Rel$ 
  and  $Q \mapsto \alpha \prec Q' \implies \exists P'. P \mapsto \alpha \prec P' \wedge (P', Q') \in Rel$ 
using assms by(simp add: strongSimEarly-def)+

lemma eqvtI:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $Rel :: (pi \times pi) set$ 
  and  $perm :: name prm$ 

  assumes Sim:  $P \rightsquigarrow[Rel] Q$ 
  and RelRel':  $Rel \subseteq Rel'$ 
  and EqvtRel': eqvt  $Rel'$ 

  shows  $(perm \cdot P) \rightsquigarrow[Rel'] (perm \cdot Q)$ 
proof(induct rule: simCases)
  case (Bound  $a y Q'$ )
  have Trans:  $(perm \cdot Q) \mapsto a \langle \nu y \rangle \prec Q'$  by fact
  have yFreshP:  $y \# perm \cdot P$  by fact

  from Trans have  $(rev perm \cdot (perm \cdot Q)) \mapsto rev perm \cdot (a \langle \nu y \rangle \prec Q')$ 
  by(rule TransitionsEarly.eqvt)
  hence  $Q \mapsto (rev perm \cdot a) \langle \nu (rev perm \cdot y) \rangle \prec (rev perm \cdot Q')$ 
  by(simp add: name-rev-per)
  moreover from yFreshP have  $(rev perm \cdot y) \# P$  by(simp add: name-fresh-left)
  ultimately have  $\exists P'. P \mapsto (rev perm \cdot a) \langle \nu (rev perm \cdot y) \rangle \prec P' \wedge (P', rev perm \cdot Q') \in Rel$  using Sim
  by(force intro: elim)
  then obtain  $P'$  where PTrans:  $P \mapsto (rev perm \cdot a) \langle \nu (rev perm \cdot y) \rangle \prec P'$ 
  and P'RelQ':  $(P', rev perm \cdot Q') \in Rel$ 
  by blast

  from PTrans have  $(perm \cdot P) \mapsto perm \cdot ((rev perm \cdot a) \langle \nu (rev perm \cdot y) \rangle \prec$ 

```

P' **by**(rule *TransitionsEarly.eqt*)
hence $L1: (perm \cdot P) \mapsto a \langle \nu y \rangle \prec (perm \cdot P')$ **by**(simp add: *name-per-rev*)
from $P' Rel Q' Rel Rel'$ **have** $(P', rev\ perm \cdot Q') \in Rel'$ **by** blast
with $EqvRel'$ **have** $(perm \cdot P', perm \cdot (rev\ perm \cdot Q')) \in Rel'$
by(rule *eqvRelI*)
hence $(perm \cdot P', Q') \in Rel'$ **by**(simp add: *name-per-rev*)
with $L1$ **show** $?case$ **by** blast
next
case(*Free* $\alpha Q'$)
have $Trans: (perm \cdot Q) \mapsto \alpha \prec Q'$ **by** fact

from $Trans$ **have** $(rev\ perm \cdot (perm \cdot Q)) \mapsto rev\ perm \cdot (\alpha \prec Q')$
by(rule *TransitionsEarly.eqt*)
hence $Q \mapsto (rev\ perm \cdot \alpha) \prec (rev\ perm \cdot Q')$
by(simp add: *name-rev-per*)
with Sim **have** $\exists P'. P \mapsto (rev\ perm \cdot \alpha) \prec P' \wedge (P', (rev\ perm \cdot Q')) \in Rel$
by(force intro: *elim*)
then obtain P' **where** $PTrans: P \mapsto (rev\ perm \cdot \alpha) \prec P'$ **and** $PRel: (P', (rev\ perm \cdot Q')) \in Rel$ **by** blast

from $PTrans$ **have** $(perm \cdot P) \mapsto perm \cdot ((rev\ perm \cdot \alpha) \prec P')$ **by**(rule *TransitionsEarly.eqt*)
hence $L1: (perm \cdot P) \mapsto \alpha \prec (perm \cdot P')$ **by**(simp add: *name-per-rev*)
from $PRel EqvRel' Rel Rel'$ **have** $((perm \cdot P'), (perm \cdot (rev\ perm \cdot Q')) \in Rel'$
by(force intro: *eqvRelI*)
hence $((perm \cdot P'), Q') \in Rel'$ **by**(simp add: *name-per-rev*)
with $L1$ **show** $?case$ **by** blast
qed

lemma *reflexive*:
fixes $P :: pi$
and $Rel :: (pi \times pi)$ *set*

assumes $Id \subseteq Rel$

shows $P \rightsquigarrow[Rel] P$
using *assms*
by(auto simp add: *strongSimEarly-def*)

lemmas *fresh-prod*[*simp*]

lemma *transitive*:
fixes $P :: pi$
and $Q :: pi$
and $R :: pi$
and $Rel :: (pi \times pi)$ *set*

```

and Rel' :: (pi × pi) set
and Rel'' :: (pi × pi) set

assumes PSimQ: P ~>[Rel] Q
and QSimR: Q ~>[Rel'] R
and Eqvt': eqvt Rel''
and Trans: Rel O Rel' ⊆ Rel''

shows P ~>[Rel''] R
proof –
  from Eqvt' show ?thesis
  proof(induct rule: simCasesCont[where C=Q])
    case(Bound a y R')
      have RTrans: R ⟶ a<νy> < R' by fact

      from QSimR RTrans <y # Q> have ∃ Q'. Q ⟶ a<νy> < Q' ∧ (Q', R') ∈ Rel'
        by(rule elim)
      then obtain Q' where QTrans: Q ⟶ a<νy> < Q' and Q'RelR': (Q', R')
        ∈ Rel' by blast
      from PSimQ QTrans <y # P> have ∃ P'. P ⟶ a<νy> < P' ∧ (P', Q') ∈ Rel
        by(rule elim)
      then obtain P' where PTrans: P ⟶ a<νy> < P' and P'RelQ': (P', Q') ∈
        Rel by blast

      moreover from P'RelQ' Q'RelR' Trans have (P', R') ∈ Rel'' by blast

      ultimately show ?case by blast
    next
      case(Free α R')
        have RTrans: R ⟶ α < R' by fact
        with QSimR have ∃ Q'. Q ⟶ α < Q' ∧ (Q', R') ∈ Rel' by(rule elim)
        then obtain Q' where QTrans: Q ⟶ α < Q' and Q'RelR': (Q', R') ∈ Rel'
        by blast
        from PSimQ QTrans have ∃ P'. P ⟶ α < P' ∧ (P', Q') ∈ Rel by(rule elim)
        then obtain P' where PTrans: P ⟶ α < P' and P'RelQ': (P', Q') ∈ Rel
        by blast
        from P'RelQ' Q'RelR' Trans have (P', R') ∈ Rel'' by blast
        with PTrans show ∃ P'. P ⟶ α < P' ∧ (P', R') ∈ Rel'' by blast
      qed
    qed

  end

theory Strong-Early-Bisim
  imports Strong-Early-Sim
  begin

  lemma monoAux: A ⊆ B ⟹ P ~>[A] Q ⟹ P ~>[B] Q
  by(auto intro: Strong-Early-Sim.monotonic)

```

coinductive-set *bisim* :: (*pi* × *pi*) set

where

step: $\llbracket P \rightsquigarrow[bisim] Q; (Q, P) \in bisim \rrbracket \implies (P, Q) \in bisim$

monos *monoAux*

abbreviation *strongBisimJudge* (**infixr** $\langle \rightsquigarrow \rangle$ 65) **where** $P \sim Q \equiv (P, Q) \in bisim$

lemma *bisimCoinductAux*[*case-names bisim, case-conclusion StrongBisim step, consumes 1*]:

assumes *p*: $(P, Q) \in X$

and step: $\bigwedge P Q. (P, Q) \in X \implies P \rightsquigarrow[(X \cup bisim)] Q \wedge (Q, P) \in bisim \cup X$

shows $P \sim Q$

proof –

have *aux*: $X \cup bisim = \{(P, Q). (P, Q) \in X \vee P \sim Q\}$ **by** *blast*

from *p* **show** *?thesis*

by(*coinduct, force dest: step simp add: aux*)

qed

lemma *bisimCoinduct*[*consumes 1, case-names cSim cSym*]:

fixes *P* :: *pi*

and *Q* :: *pi*

assumes $(P, Q) \in X$

and $\bigwedge R S. (R, S) \in X \implies R \rightsquigarrow[(X \cup bisim)] S$

and $\bigwedge R S. (R, S) \in X \implies (S, R) \in X$

shows $P \sim Q$

using *assms*

by(*coinduct rule: bisimCoinductAux*) *auto*

lemma *weak-coinduct*[*case-names bisim, case-conclusion StrongBisim step, consumes 1*]:

assumes *p*: $(P, Q) \in X$

and step: $\bigwedge P Q. (P, Q) \in X \implies P \rightsquigarrow[X] Q \wedge (Q, P) \in X$

shows $P \sim Q$

using *p*

proof(*coinduct rule: bisimCoinductAux*)

case (*bisim P*)

from *step[OF this]* **show** *?case* **using** *Strong-Early-Sim.monotonic* **by** *blast*

qed

lemma *bisimWeakCoinduct*[*consumes 1, case-names cSim cSym*]:

fixes *P* :: *pi*

and *Q* :: *pi*

```

assumes  $(P, Q) \in X$ 
and  $\bigwedge P Q. (P, Q) \in X \implies P \rightsquigarrow[X] Q$ 
and  $\bigwedge P Q. (P, Q) \in X \implies (Q, P) \in X$ 

shows  $P \sim Q$ 
using assms
by(coinduct rule: weak-coinduct) auto

lemma monotonic:  $\text{mono}(\lambda p x1 x2.$ 
   $\exists P Q. x1 = P \wedge$ 
   $x2 = Q \wedge P \rightsquigarrow[\{(xa, x). p \ xa \ x\}] Q \wedge Q \rightsquigarrow[\{(xa, x). p \ xa \ x\}] P)$ 
apply(rule monoI)
by(auto intro: Strong-Early-Sim.monotonic)

lemma bisimE:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 

  assumes  $P \sim Q$ 

  shows  $P \rightsquigarrow[bisim] Q$ 
  and  $Q \sim P$ 
using assms
by(auto intro: bisim.cases)

lemma bisimClosed[eqvt]:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $p :: \text{name } prm$ 

  assumes  $P \sim Q$ 

  shows  $(p \cdot P) \sim (p \cdot Q)$ 
proof –
  let  $?X = \{(p \cdot P, p \cdot Q) \mid (p::\text{name } prm) P Q. P \sim Q\}$ 
  from assms have  $(p \cdot P, p \cdot Q) \in ?X$  by auto
  thus ?thesis
proof(coinduct rule: bisimWeakCoinduct)
  case(cSim P Q)
  moreover {
    fix  $P Q$ 
    fix  $p::\text{name } prm$ 
    assume  $P \rightsquigarrow[bisim] Q$ 

    moreover have  $bisim \subseteq ?X$ 
      by(auto, rule-tac x=[] in exI) auto
    moreover have eqvt ?X
      by(auto simp add: eqvt-def pt2[OF pt-name-inst, THEN sym]) blast
  }

```

```

    ultimately have  $(p \cdot P) \rightsquigarrow[?X] (p \cdot Q)$ 
      by(rule Strong-Early-Sim.eqtI)
  }
  ultimately show ?case by(blast dest: bisimE)
next
  case(cSym P Q)
  thus ?case by(blast dest: bisimE)
qed
qed

lemma eqt[simp]:
  shows eqt bisim
by(auto simp add: eqt-def eqvts)

lemma reflexive:
  fixes P :: pi

  shows P ~ P
proof -
  have  $(P, P) \in Id$  by simp
  then show ?thesis
    by(coinduct rule: bisimWeakCoinduct) (auto intro: Strong-Early-Sim.reflexive)
qed

lemma transitive:
  fixes P :: pi
  and Q :: pi
  and R :: pi

  assumes PBiSimQ:  $P \sim Q$ 
  and QBiSimR:  $Q \sim R$ 

  shows P ~ R
proof -
  let ?X = bisim O bisim
  from assms have  $(P, R) \in ?X$  by blast
  thus ?thesis
  proof(coinduct rule: bisimWeakCoinduct)
    case(cSim P Q)
    moreover {
      fix P Q R
      assume  $P \sim Q$  and  $Q \sim R$ 
      hence  $P \rightsquigarrow[bisim] Q$  and  $Q \rightsquigarrow[bisim] R$ 
        by(metis bisimE)+
      moreover from eqt have eqt ?X by(auto simp add: eqtTrans)
      moreover have bisim O bisim  $\subseteq$  ?X by auto
    }

    ultimately have  $P \rightsquigarrow[?X] R$ 
      by(rule Strong-Early-Sim.transitive)
  qed

```

```

    }
    ultimately show ?case by auto
  next
    case(cSym P Q)
    thus ?case by(auto dest: bisimE)
  qed
qed

end

theory Strong-Early-Bisim-Subst
  imports Strong-Early-Bisim
begin

abbreviation StrongCongEarlyJudge (infixr <~s> 65) where P ~s Q ≡ (P, Q)
∈ (substClosed bisim)

lemma congBisim:
  fixes P :: pi
  and Q :: pi

  assumes P ~s Q

  shows P ~ Q
using assms substClosedSubset by blast

lemma eqvt:
  shows eqvt (substClosed bisim)
by(rule eqvtSubstClosed[OF Strong-Early-Bisim.eqvt])

lemma eqvtI:
  fixes P :: pi
  and Q :: pi
  and perm :: name prm

  assumes P ~s Q

  shows (perm · P) ~s (perm · Q)
using assms
by(rule eqvtRelI[OF eqvt])

lemma reflexive:
  fixes P :: pi

  shows P ~s P
by(force simp add: substClosed-def intro: Strong-Early-Bisim.reflexive)

lemma symmetric:
  fixes P :: pi

```

```

and Q :: pi

assumes P ~s Q

shows Q ~s P
using assms
by(force simp add: substClosed-def intro: Strong-Early-Bisim.bisimE)

lemma transitive:
  fixes P :: pi
  and Q :: pi
  and R :: pi

  assumes P ~s Q
  and Q ~s R

  shows P ~s R
using assms
by(force simp add: substClosed-def intro: Strong-Early-Bisim.transitive)

lemma partUnfold:
  fixes P :: pi
  and Q :: pi
  and s :: (name × name) list

  assumes P ~s Q

  shows P[<s>] ~s Q[<s>]
using assms
proof(auto simp add: substClosed-def)
  fix s'
  assume  $\forall s. P[<s>] \sim Q[<s>]$ 
  hence P[<(s@s')>] ~ Q[<(s@s')>] by blast
  moreover have P[<(s@s')>] = (P[<s>])[<s'>]
    by(induct s', auto)
  moreover have Q[<(s@s')>] = (Q[<s>])[<s'>]
    by(induct s', auto)

  ultimately show (P[<s>])[<s'>] ~ (Q[<s>])[<s'>]
    by simp
qed

end

theory Strong-Early-Sim-Pres
  imports Strong-Early-Sim
begin

lemma tauPres:

```

```

fixes P  :: pi
and   Q  :: pi
and   Rel :: (pi × pi) set

assumes PRelQ: (P, Q) ∈ Rel

shows τ.(P) ~>[Rel] τ.(Q)
proof(induct rule: simCases)
  case(Bound a y Q')
    have τ.(Q) ⟶ a<νy> < Q' by fact
    hence False by(induct rule: tauCases', auto)
    thus ?case by simp
  next
    case(Free α Q')
    have τ.(Q) ⟶ α < Q' by fact
    thus ∃ P'. τ.(P) ⟶ α < P' ∧ (P', Q') ∈ Rel
    proof(induct rule: tauCases', auto simp add: pi.inject residual.inject)
      have τ.(P) ⟶ τ < P by(rule TransitionsEarly.Tau)
      with PRelQ show ∃ P'. τ.(P) ⟶ τ < P' ∧ (P', Q) ∈ Rel by blast
    qed
  qed

```

```

lemma inputPres:
  fixes P  :: pi
  and   x  :: name
  and   Q  :: pi
  and   a  :: name
  and   Rel :: (pi × pi) set

  assumes PRelQ: ∀ y. (P[x::=y], Q[x::=y]) ∈ Rel
  and     Eqvt: eqvt Rel

  shows a<x>.P ~>[Rel] a<x>.Q
using Eqvt
proof(induct rule: simCasesCont[where C=(x, a, P, Q)])
  case(Bound b y Q')
    from ⟨y # (x, a, P, Q)⟩ have y ≠ x y ≠ a y # P y # Q by simp+
    from ⟨a<x>.Q ⟶ b<νy> < Q'⟩ ⟨y ≠ a⟩ ⟨y ≠ x⟩ ⟨y # Q⟩ show ?case
      by(erule-tac inputCases') auto
  next
    case(Free α Q')
    from ⟨a<x>.Q ⟶ α < Q'⟩
    show ?case
    proof(induct rule: inputCases)
      case(cInput u)
      have a<x>.P ⟶ a<u> < P[x::=u] by(rule Input)
      moreover from PRelQ have (P[x::=u], Q[x::=u]) ∈ Rel by auto
      ultimately show ?case by blast

```

qed
qed

lemma *outputPres*:

fixes $P :: pi$
and $Q :: pi$
and $a :: name$
and $b :: name$
and $Rel :: (pi \times pi) set$
and $Rel' :: (pi \times pi) set$

assumes $PRelQ: (P, Q) \in Rel$

shows $a\{b\}.P \rightsquigarrow[Rel] a\{b\}.Q$

proof(*induct rule: simCases*)

case(*Bound c y Q'*)

have $a\{b\}.Q \mapsto c\langle \nu y \rangle \prec Q'$ by *fact*

hence *False* by(*induct rule: outputCases', auto*)

thus $\exists P'. a\{b\}.P \mapsto c\langle \nu y \rangle \prec P' \wedge (P', Q') \in Rel$ by *simp*

next

case(*Free α Q'*)

have $a\{b\}.Q \mapsto \alpha \prec Q'$ by *fact*

thus $\exists P'. a\{b\}.P \mapsto \alpha \prec P' \wedge (P', Q') \in Rel$

proof(*induct rule: outputCases', auto simp add: pi.inject residual.inject*)

have $a\{b\}.P \mapsto a[b] \prec P$ by(*rule TransitionsEarly.Output*)

with $PRelQ$ show $\exists P'. a\{b\}.P \mapsto a[b] \prec P' \wedge (P', Q) \in Rel$ by *blast*

qed

qed

lemma *matchPres*:

fixes $P :: pi$
and $Q :: pi$
and $a :: name$
and $b :: name$
and $Rel :: (pi \times pi) set$
and $Rel' :: (pi \times pi) set$

assumes $PSimQ: P \rightsquigarrow[Rel] Q$

and $RelRel': Rel \subseteq Rel'$

shows $[a\curvearrowright b]P \rightsquigarrow[Rel'] [a\curvearrowright b]Q$

proof(*induct rule: simCases*)

case(*Bound c y Q'*)

have $(y::name) \# [a\curvearrowright b]P$ by *fact*

hence $yFreshP: y \# P$ by *simp*

have $[a\curvearrowright b]Q \mapsto c\langle \nu y \rangle \prec Q'$ by *fact*

thus *?case*

proof(*induct rule: matchCases*)

case *Match*

have $Q \mapsto c\langle \nu y \rangle \prec Q'$ by *fact*

with $PSimQ$ $yFreshP$ **obtain** P' **where** $PTrans: P \mapsto c \langle \nu y \rangle \prec P'$ **and**
 $P'RelQ': (P', Q') \in Rel$
by(*blast dest: elim*)

from $PTrans$ **have** $[a \frown a]P \mapsto c \langle \nu y \rangle \prec P'$ **by**(*rule Early-Semantics.Match*)
moreover from $P'RelQ' RelRel'$ **have** $(P', Q') \in Rel'$ **by** *blast*
ultimately show $?case$ **by** *blast*

qed
next

case(*Free α Q'*)
assume $[a \frown b]Q \mapsto \alpha \prec Q'$
thus $?case$
proof(*induct rule: matchCases*)

case *Match*
have $Q \mapsto \alpha \prec Q'$ **by** *fact*
with $PSimQ$ **obtain** P' **where** $PTrans: P \mapsto \alpha \prec P'$ **and** $P'RelQ': (P', Q') \in Rel$
by(*blast dest: elim*)

from $PTrans$ **have** $[a \frown a]P \mapsto \alpha \prec P'$ **by**(*rule TransitionsEarly.Match*)
moreover from $P'RelQ' RelRel'$ **have** $(P', Q') \in Rel'$ **by** *blast*
ultimately show $?case$ **by** *blast*

qed
qed

lemma *mismatchPres*:
fixes P **::** pi
and Q **::** pi
and a **::** $name$
and b **::** $name$
and Rel **::** $(pi \times pi)$ *set*
and Rel' **::** $(pi \times pi)$ *set*

assumes $PSimQ: P \rightsquigarrow[Rel] Q$
and $RelRel': Rel \subseteq Rel'$

shows $[a \neq b]P \rightsquigarrow[Rel'] [a \neq b]Q$
proof(*cases $a = b$*)
assume $a = b$
thus $?thesis$
by(*auto simp add: strongSimEarly-def*)

next
assume $a \neq b$
show $?thesis$
proof(*induct rule: simCases*)
case(*Bound c x Q'*)
have $x \# [a \neq b]P$ **by** *fact*
hence $xFreshP: x \# P$ **by** *simp*
have $[a \neq b]Q \mapsto c \langle \nu x \rangle \prec Q'$ **by** *fact*

```

thus ?case
proof(induct rule: mismatchCases)
  case Mismatch
  have  $Q \mapsto c \langle \nu x \rangle \prec Q'$  by fact
  with  $PSimQ$   $xFreshP$  obtain  $P'$  where  $PTrans: P \mapsto c \langle \nu x \rangle \prec P'$ 
    and  $P'RelQ': (P', Q') \in Rel$ 
    by(blast dest: elim)

from  $PTrans$  aineqb have  $[a \neq b]P \mapsto c \langle \nu x \rangle \prec P'$  by(rule Early-Semantics.Mismatch)
moreover from  $P'RelQ' RelRel'$  have  $(P', Q') \in Rel'$  by blast
ultimately show ?case by blast
qed
next
case(Free  $\alpha$   $Q'$ )
have  $[a \neq b]Q \mapsto \alpha \prec Q'$  by fact
thus ?case
proof(induct rule: mismatchCases)
  case Mismatch
  have  $Q \mapsto \alpha \prec Q'$  by fact
  with  $PSimQ$  obtain  $P'$  where  $PTrans: P \mapsto \alpha \prec P'$ 
    and  $PRel: (P', Q') \in Rel$ 
    by(blast dest: elim)
from  $PTrans$   $\langle a \neq b \rangle$  have  $[a \neq b]P \mapsto \alpha \prec P'$  by(rule TransitionsEarly.Mismatch)
with  $RelRel' PRel$  show ?case by blast
qed
qed
qed

lemma sumPres:
  fixes  $P$     ::  $\pi i$ 
  and  $Q$      ::  $\pi i$ 
  and  $R$      ::  $\pi i$ 
  and  $Rel$    ::  $(\pi i \times \pi i)$  set
  and  $Rel'$  ::  $(\pi i \times \pi i)$  set

  assumes  $P \rightsquigarrow[Rel] Q$ 
  and  $C1: Id \subseteq Rel'$ 
  and  $Rel \subseteq Rel'$ 

  shows  $P \oplus R \rightsquigarrow[Rel'] Q \oplus R$ 
proof(induct rule: simCases)
  case(Bound  $a$   $y$   $Q'$ )
  have  $y \# P \oplus R$  by fact
  hence  $(y::name) \# P$  and  $y \# R$  by simp+
  from  $\langle Q \oplus R \mapsto a \langle \nu y \rangle \prec Q' \rangle$  show ?case
proof(induct rule: sumCases)
  case Sum1
  from  $\langle P \rightsquigarrow[Rel] Q \rangle$   $\langle Q \mapsto a \langle \nu y \rangle \prec Q' \rangle$   $\langle y \# P \rangle$  obtain  $P'$  where  $PTrans:$ 
 $P \mapsto a \langle \nu y \rangle \prec P'$  and  $P'RelQ': (P', Q') \in Rel$ 

```

```

    by(blast dest: elim)

    from PTrans have  $P \oplus R \mapsto a \langle \nu y \rangle \prec P'$  by(rule Early-Semantics.Sum1)
    moreover from  $P' \text{Rel} Q' \langle \text{Rel} \subseteq \text{Rel}' \rangle$  have  $(P', Q') \in \text{Rel}'$  by blast
    ultimately show ?case by blast
  next
    case Sum2
    from  $\langle R \mapsto a \langle \nu y \rangle \prec Q' \rangle$  have  $P \oplus R \mapsto a \langle \nu y \rangle \prec Q'$  by(rule Early-Semantics.Sum2)
    moreover from C1 have  $(Q', Q') \in \text{Rel}'$  by auto
    ultimately show ?case by blast
  qed
next
case(Free  $\alpha$   $Q'$ )
from  $\langle Q \oplus R \mapsto \alpha \prec Q' \rangle$  show  $\exists P'. P \oplus R \mapsto \alpha \prec P' \wedge (P', Q') \in \text{Rel}'$ 
proof(induct rule: sumCases)
  case Sum1
  have  $Q \mapsto \alpha \prec Q'$  by fact
  with  $\langle P \rightsquigarrow[\text{Rel}] Q \rangle$  obtain  $P'$  where PTrans:  $P \mapsto \alpha \prec P'$  and  $P' \text{Rel} Q'$ :
  ( $P', Q') \in \text{Rel}$ 
  by(blast dest: elim)

  from PTrans have  $P \oplus R \mapsto \alpha \prec P'$  by(rule TransitionsEarly.Sum1)
  moreover from  $P' \text{Rel} Q' \langle \text{Rel} \subseteq \text{Rel}' \rangle$  have  $(P', Q') \in \text{Rel}'$  by blast
  ultimately show ?case by blast
next
case Sum2
from  $\langle R \mapsto \alpha \prec Q' \rangle$  have  $P \oplus R \mapsto \alpha \prec Q'$  by(rule TransitionsEarly.Sum2)
moreover from C1 have  $(Q', Q') \in \text{Rel}'$  by blast
ultimately show ?case by blast
qed
qed

lemma parCompose:
  fixes  $P$     :: pi
  and  $Q$      :: pi
  and  $R$      :: pi
  and  $T$      :: pi
  and  $\text{Rel}$   ::  $(\text{pi} \times \text{pi})$  set
  and  $\text{Rel}'$  ::  $(\text{pi} \times \text{pi})$  set
  and  $\text{Rel}''$  ::  $(\text{pi} \times \text{pi})$  set

  assumes PSimQ:  $P \rightsquigarrow[\text{Rel}] Q$ 
  and RSimT:  $R \rightsquigarrow[\text{Rel}'] S$ 
  and PRelQ:  $(P, Q) \in \text{Rel}$ 
  and RRel'T:  $(R, S) \in \text{Rel}'$ 
  and Par:  $\bigwedge P' Q' R' S'. \llbracket (P', Q') \in \text{Rel}; (R', S') \in \text{Rel}' \rrbracket \implies (P' \parallel R', Q' \parallel S') \in \text{Rel}''$ 
  and Res:  $\bigwedge S T x. (S, T) \in \text{Rel}'' \implies (\langle \nu x \rangle S, \langle \nu x \rangle T) \in \text{Rel}''$ 

```

shows $P \parallel R \rightsquigarrow[Rel''] Q \parallel S$
proof(*induct rule: simCases*)
case(*Bound a x Q'*)
have $x \# P \parallel R$ **by fact**
hence $xFreshP: x \# P$ **and** $xFreshR: x \# R$ **by simp+**
have $Q \parallel S \mapsto a \langle \nu x \rangle \prec Q'$ **by fact**
thus *?case*
proof(*induct rule: parCasesB*)
case(*cPar1 Q'*)
have $Q \mapsto a \langle \nu x \rangle \prec Q'$ **by fact**
with $PSimQ$ $xFreshP$ **obtain** P' **where** $PTrans:P \mapsto a \langle \nu x \rangle \prec P'$ **and**
 $P'RelQ': (P', Q') \in Rel$
by(*blast dest: elim*)

from $PTrans$ $xFreshR$ **have** $P \parallel R \mapsto a \langle \nu x \rangle \prec (P' \parallel R)$ **by**(*rule Early-Semantics.Par1B*)
moreover from $P'RelQ'$ $RRel'T$ **have** $(P' \parallel R, Q' \parallel S) \in Rel''$ **by**(*rule Par*)
ultimately show *?case* **by blast**
next
case(*cPar2 S'*)
have $S \mapsto a \langle \nu x \rangle \prec S'$ **by fact**
with $RSimT$ $xFreshR$ **obtain** R' **where** $RTrans:R \mapsto a \langle \nu x \rangle \prec R'$ **and**
 $R'Rel'T': (R', S') \in Rel'$
by(*blast dest: elim*)

from $RTrans$ $xFreshP$ **have** $ParTrans: P \parallel R \mapsto a \langle \nu x \rangle \prec (P \parallel R')$ **by**(*rule Early-Semantics.Par2B*)
moreover from $PRelQ$ $R'Rel'T'$ **have** $(P \parallel R', Q \parallel S') \in Rel''$ **by**(*rule Par*)
ultimately show *?case* **by blast**
qed
next
case(*Free α QT'*)
have $Q \parallel S \mapsto \alpha \prec QT'$ **by fact**
thus *?case*
proof(*induct rule: parCasesF[of - - - - (P, R)]*)
case(*cPar1 Q'*)
have $Q \mapsto \alpha \prec Q'$ **by fact**
with $PSimQ$ **obtain** P' **where** $PTrans: P \mapsto \alpha \prec P'$ **and** $PRel: (P', Q') \in$
 Rel
by(*blast dest: elim*)

from $PTrans$ **have** $P \parallel R \mapsto \alpha \prec P' \parallel R$ **by**(*rule Early-Semantics.Par1F*)
moreover from $PRel$ $RRel'T$ **have** $(P' \parallel R, Q' \parallel S) \in Rel''$ **by**(*rule Par*)
ultimately show *?case* **by blast**
next
case(*cPar2 S'*)
have $S \mapsto \alpha \prec S'$ **by fact**
with $RSimT$ **obtain** R' **where** $RTrans: R \mapsto \alpha \prec R'$ **and** $RRel: (R', S') \in$
 Rel'
by(*blast dest: elim*)

from $RTrans$ **have** $P \parallel R \mapsto \alpha \prec P \parallel R'$ **by**(rule *Early-Semantics.Par2F*)
moreover from $PRelQ$ $RRel$ **have** $(P \parallel R', Q \parallel S') \in Rel''$ **by**(rule *Par*)
ultimately show *?case* **by** *blast*
next
case(*cComm1* $Q' S' a b$)
have $Q \mapsto a\langle b \rangle \prec Q'$ **by** *fact*
with $PSimQ$ **obtain** P' **where** $PTrans: P \mapsto a\langle b \rangle \prec P'$ **and** $P'RelQ': (P', Q') \in Rel$
by(*blast dest: elim*)

have $S \mapsto a[b] \prec S'$ **by** *fact*
with $RSimT$ **obtain** R' **where** $RTrans: R \mapsto a[b] \prec R'$ **and** $RRel: (R', S') \in Rel'$
by(*blast dest: elim*)

from $PTrans$ $RTrans$ **have** $P \parallel R \mapsto \tau \prec P' \parallel R'$ **by**(rule *Early-Semantics.Comm1*)
moreover from $P'RelQ'$ $RRel$ **have** $(P' \parallel R', Q' \parallel S') \in Rel''$ **by**(rule *Par*)
ultimately show *?case* **by** *blast*
next
case(*cComm2* $Q' S' a b$)
have $Q \mapsto (OutputR\ a\ b) \prec Q'$ **by** *fact*
with $PSimQ$ **obtain** P' **where** $PTrans: P \mapsto a[b] \prec P'$ **and** $PRel: (P', Q') \in Rel$
by(*blast dest: elim*)

have $S \mapsto a\langle b \rangle \prec S'$ **by** *fact*
with $RSimT$ **obtain** R' **where** $RTrans: R \mapsto a\langle b \rangle \prec R'$ **and** $R'Rel'T': (R', S') \in Rel'$
by(*blast dest: elim*)

from $PTrans$ $RTrans$ **have** $P \parallel R \mapsto \tau \prec P' \parallel R'$ **by**(rule *Early-Semantics.Comm2*)
moreover from $PRel$ $R'Rel'T'$ **have** $(P' \parallel R', Q' \parallel S') \in Rel''$ **by**(rule *Par*)
ultimately show *?case* **by** *blast*
next
case(*cClose1* $Q' S' a x$)
have $x \# (P, R)$ **by** *fact*
hence $xFreshP: x \# P$ **and** $xFreshR: x \# R$ **by** *simp+*

have $Q \mapsto a\langle x \rangle \prec Q'$ **by** *fact*
with $PSimQ$ **obtain** P' **where** $PTrans: P \mapsto a\langle x \rangle \prec P'$ **and** $P'RelQ': (P', Q') \in Rel$
by(*blast dest: elim*)

have $S \mapsto a\langle \nu x \rangle \prec S'$ **by** *fact*
with $RSimT$ $xFreshR$ **obtain** R' **where** $RTrans: R \mapsto a\langle \nu x \rangle \prec R'$ **and** $R'Rel'T': (R', S') \in Rel'$
by(*blast dest: elim*)

from $PTrans\ RTrans\ xFreshP$ **have** $P \parallel R \mapsto \tau \prec \langle \nu x \rangle (P' \parallel R')$
by(*rule Early-Semantics.Close1*)
moreover from $P'RelQ'\ R'Rel'T'$ **have** $(\langle \nu x \rangle (P' \parallel R'), \langle \nu x \rangle (Q' \parallel S')) \in Rel''$
by(*blast intro: Par Res*)
ultimately show *?case* **by** *blast*
next
case(*cClose2 Q' S' a x*)
have $x \# (P, R)$ **by** *fact*
hence $xFreshP: x \# P$ **and** $xFreshR: x \# R$ **by** *simp+*

have $Q \mapsto a \langle \nu x \rangle \prec Q'$ **by** *fact*
with $PSimQ\ xFreshP$ **obtain** P' **where** $PTrans: P \mapsto a \langle \nu x \rangle \prec P'$ **and**
 $P'RelQ': (P', Q') \in Rel$
by(*blast dest: elim*)

have $S \mapsto a \langle x \rangle \prec S'$ **by** *fact*
with $RSimT$ **obtain** R' **where** $RTrans: R \mapsto a \langle x \rangle \prec R'$ **and** $R'Rel'T': (R', S') \in Rel'$
by(*blast dest: elim*)

from $PTrans\ RTrans\ xFreshR$ **have** $P \parallel R \mapsto \tau \prec \langle \nu x \rangle (P' \parallel R')$
by(*rule Early-Semantics.Close2*)
moreover from $P'RelQ'\ R'Rel'T'$ **have** $(\langle \nu x \rangle (P' \parallel R'), \langle \nu x \rangle (Q' \parallel S')) \in Rel''$
by(*blast intro: Par Res*)
ultimately show *?case* **by** *blast*
qed
qed

lemma *parPres*:
fixes $P :: pi$
and $Q :: pi$
and $R :: pi$
and $a :: name$
and $b :: name$
and $Rel :: (pi \times pi)$ *set*
and $Rel' :: (pi \times pi)$ *set*

assumes $PSimQ: P \rightsquigarrow[Rel] Q$
and $PRelQ: (P, Q) \in Rel$
and $Par: \bigwedge S\ T\ U. (S, T) \in Rel \implies (S \parallel U, T \parallel U) \in Rel'$
and $Res: \bigwedge S\ T\ x. (S, T) \in Rel' \implies (\langle \nu x \rangle S, \langle \nu x \rangle T) \in Rel'$

shows $P \parallel R \rightsquigarrow[Rel'] Q \parallel R$
proof –
note $PSimQ$
moreover have $RSimR: R \rightsquigarrow[Id] R$ **by**(*auto intro: reflexive*)
moreover note $PRelQ$ **moreover have** $(R, R) \in Id$ **by** *auto*

moreover from *Par* **have** $\bigwedge P Q R T. \llbracket (P, Q) \in \text{Rel}; (R, T) \in \text{Id} \rrbracket \implies (P \parallel R, Q \parallel T) \in \text{Rel}'$
by *auto*
ultimately show *?thesis* **using** *Res* **by**(*rule parCompose*)
qed

lemma *resPres*:

fixes $P :: pi$
and $Q :: pi$
and $\text{Rel} :: (pi \times pi) \text{ set}$
and $x :: name$
and $\text{Rel}' :: (pi \times pi) \text{ set}$

assumes $\text{PSimQ}: P \rightsquigarrow[\text{Rel}] Q$
and $\text{ResSet}: \bigwedge (R::pi) (S::pi) (y::name). (R, S) \in \text{Rel} \implies (\langle \nu y \rangle R, \langle \nu y \rangle S) \in \text{Rel}'$
and $\text{RelRel}': \text{Rel} \subseteq \text{Rel}'$
and $\text{EqvtRel}: \text{eqvt Rel}$
and $\text{EqvtRel}': \text{eqvt Rel}'$

shows $\langle \nu x \rangle P \rightsquigarrow[\text{Rel}'] \langle \nu x \rangle Q$

proof –

from $\text{EqvtRel}'$ **show** *?thesis*

proof(*induct rule: simCasesCont*[**where** $C = (P, x)$])

case(*Bound* $a y Q'$)

have $\text{Trans}: \langle \nu x \rangle Q \mapsto a \langle \nu y \rangle \prec Q'$ **by** *fact*

have $y \# (P, x)$ **by** *fact*

hence $y \text{ineq}x: y \neq x$ **and** $y \text{Fresh}P: y \# (P::pi)$ **by** *simp+*

from Trans $y \text{ineq}x$ **show** *?case*

proof(*induct rule: resCasesB*)

case(*Open* Q')

have $Q \text{Trans}: Q \mapsto (a::name)[x] \prec Q'$ **by** *fact*

with PSimQ **obtain** P' **where** $P \text{Trans}: P \mapsto a[x] \prec P'$ **and** $P' \text{Rel}Q': (P',$

$Q') \in \text{Rel}$

by(*blast dest: elim*)

have $\langle \nu x \rangle P \mapsto a \langle \nu y \rangle \prec ((y, x)) \cdot P'$

proof –

have $a \text{ineq}x: a \neq x$ **by** *fact*

with $P \text{Trans}$ **have** $\langle \nu x \rangle P \mapsto a \langle \nu x \rangle \prec P'$ **by**(*rule TransitionsEarly.Open*)

moreover have $a \langle \nu x \rangle \prec P' = a \langle \nu y \rangle \prec ((y, x)) \cdot P'$

proof –

from $P \text{Trans}$ $y \text{Fresh}P$ **have** $y \text{Fresh}P': y \# P'$ **by**(*force intro: freshTransition*)

thus *?thesis* **by**(*simp add: alphaBoundOutput name-swap*)

qed

ultimately show *?thesis* **by** *simp*

qed

moreover from EqvtRel $P' \text{Rel}Q'$ RelRel' **have** $((y, x)) \cdot P', ((y, x)) \cdot Q' \in \text{Rel}'$

```

    by(blast intro: eqvtRelI)
  ultimately show ?case by blast
next
case(Res Q')
have QTrans:  $Q \mapsto a \langle \nu y \rangle \prec Q'$  by fact

  with PSimQ yFreshP obtain P' where PTrans:  $P \mapsto a \langle \nu y \rangle \prec P'$  and
P'RelQ':  $(P', Q') \in \text{Rel}$ 
  by(blast dest: elim)

  have xineqa:  $x \neq a$  by fact
  with PTrans yineqx have ResTrans:  $\langle \nu x \rangle P \mapsto a \langle \nu y \rangle \prec (\langle \nu x \rangle P')$ 
  by(blast intro: ResB)
  moreover from P'RelQ' have  $((\langle \nu x \rangle P'), (\langle \nu x \rangle Q')) \in \text{Rel}'$ 
  by(rule ResSet)

  ultimately show  $\exists P'. \langle \nu x \rangle P \mapsto a \langle \nu y \rangle \prec P' \wedge (P', (\langle \nu x \rangle Q')) \in \text{Rel}'$ 
by blast
qed
next
case(Free  $\alpha$  Q')
have Trans:  $\langle \nu x \rangle Q \mapsto \alpha \prec Q'$  by fact
have  $\exists c::\text{name}. c \# (P, Q, Q', \alpha)$  by(blast intro: name-exists-fresh)
  then obtain  $c::\text{name}$  where cFreshQ:  $c \# Q$  and cFreshAlpha:  $c \# \alpha$  and
cFreshQ':  $c \# Q'$  and cFreshP:  $c \# P$ 
  by(force simp add: fresh-prod)
  from cFreshP have  $\langle \nu x \rangle P = \langle \nu c \rangle ([x, c] \cdot P)$  by(simp add: alphaRes)
  moreover have  $\exists P'. \langle \nu c \rangle ([x, c] \cdot P) \mapsto \alpha \prec P' \wedge (P', Q') \in \text{Rel}'$ 
  proof -
    from Trans cFreshQ have  $\langle \nu c \rangle ([x, c] \cdot Q) \mapsto \alpha \prec Q'$  by(simp add:
alphaRes)
    moreover from EqvtRel PSimQ have  $([x, c] \cdot P) \rightsquigarrow[\text{Rel}] ([x, c] \cdot Q)$ 
    by(blast intro: eqvtI)
    ultimately show ?thesis using cFreshAlpha
    apply -
    apply(erule resCasesF)
    apply auto
    by(blast intro: ResF ResSet dest: elim)
  qed
qed

  ultimately show  $\exists P'. \langle \nu x \rangle P \mapsto \alpha \prec P' \wedge (P', Q') \in \text{Rel}'$  by auto
qed
qed

lemma resChainI:
  fixes P :: pi
  and Q :: pi
  and Rel :: (pi  $\times$  pi) set
  and lst :: name list

```

```

assumes eqvtRel: eqvt Rel
and Res:  $\bigwedge R S x. (R, S) \in Rel \implies (\langle \nu x \rangle R, \langle \nu x \rangle S) \in Rel$ 
and PRelQ:  $P \rightsquigarrow[Rel] Q$ 

shows (resChain lst)  $P \rightsquigarrow[Rel]$  (resChain lst)  $Q$ 
proof –
  show ?thesis
  proof(induct lst)
    from PRelQ show resChain []  $P \rightsquigarrow[Rel]$  resChain []  $Q$  by simp
  next
    fix a lst
    assume IH: (resChain lst  $P$ )  $\rightsquigarrow[Rel]$  (resChain lst  $Q$ )
    moreover from Res have  $\bigwedge P Q a. (P, Q) \in Rel \implies (\langle \nu a \rangle P, \langle \nu a \rangle Q) \in Rel$ 
    by simp
    moreover have  $Rel \subseteq Rel$  by simp
    ultimately have  $\langle \nu a \rangle$ (resChain lst  $P$ )  $\rightsquigarrow[Rel]$   $\langle \nu a \rangle$ (resChain lst  $Q$ ) using
    eqvtRel
    by(rule-tac resPres)
    thus resChain (a # lst)  $P \rightsquigarrow[Rel]$  resChain (a # lst)  $Q$ 
    by simp
  qed
qed

lemma bangPres:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $Rel :: (pi \times pi)$  set

  assumes PRelQ:  $(P, Q) \in Rel$ 
  and Sim:  $\bigwedge R S. (R, S) \in Rel \implies R \rightsquigarrow[Rel] S$ 
  and eqvtRel: eqvt Rel

  shows  $!P \rightsquigarrow[bangRel Rel] !Q$ 
  proof –
    let ?Sim =  $\lambda P Rs. (\forall a x Q'. Rs = a \langle \nu x \rangle \prec Q' \longrightarrow x \# P \longrightarrow (\exists P'. P \mapsto a \langle \nu x \rangle \prec P' \wedge (P', Q') \in bangRel Rel)) \wedge$ 
     $(\forall \alpha Q'. Rs = \alpha \prec Q' \longrightarrow (\exists P'. P \mapsto \alpha \prec P' \wedge (P', Q') \in bangRel Rel))$ 
    from eqvtRel have EqvtBangRel: eqvt(bangRel Rel) by(rule eqvtBangRel)

    {
      fix Pa Rs
      assume  $!Q \mapsto Rs$  and  $(Pa, !Q) \in bangRel Rel$ 
      hence ?Sim Pa Rs using PRelQ
      proof(nominal-induct avoiding: Pa P rule: bangInduct)
        case(Par1B a x  $Q' Pa P$ )
        have QTrans:  $Q \mapsto a \langle \nu x \rangle \prec Q'$  by fact
    }
  
```

have $(Pa, Q \parallel !Q) \in \text{bangRel Rel}$ **and** $x \# Pa$ **by** *fact+*
thus $?Sim Pa (a<\nu x> \prec (Q' \parallel !Q))$
proof(*induct rule: BRParCases*)
 case(*BRPar P R*)
 have $PRelQ: (P, Q) \in Rel$ **by** *fact*
 have $PBRQ: (R, !Q) \in \text{bangRel Rel}$ **by** *fact*
 have $x \# P \parallel R$ **by** *fact*
 hence $xFreshP: x \# P$ **and** $xFreshR: x \# R$ **by** *simp+*
 show *?case*
 proof(*auto simp add: residual.inject alpha^*)
 from $PRelQ$ **have** $P \rightsquigarrow[Rel] Q$ **by**(*rule Sim*)

with $QTrans xFreshP$ **obtain** P' **where** $PTrans: P \mapsto a<\nu x> \prec P'$ **and**
 $P'RelQ': (P', Q') \in Rel$
by(*blast dest: elim*)

from $PTrans xFreshR$ **have** $P \parallel R \mapsto a<\nu x> \prec (P' \parallel R)$
by(*force intro: Early-Semantics.Par1B*)
moreover from $P'RelQ' PBRQ$ **have** $(P' \parallel R, Q' \parallel !Q) \in \text{bangRel Rel}$
by(*rule Rel.BRPar*)
ultimately show $\exists P'. P \parallel R \mapsto a<\nu x> \prec P' \wedge (P', Q' \parallel !Q) \in \text{bangRel Rel}$ **by** *blast*

next
fix y
assume $(y::name) \# Q'$ **and** $y \# P$ **and** $y \# R$ **and** $y \# Q$
from $QTrans \langle y \# Q' \rangle$ **have** $Q \mapsto a<\nu y> \prec ((x, y) \cdot Q')$
by(*simp add: alphaBoundOutput*)
moreover from $PRelQ$ **have** $P \rightsquigarrow[Rel] Q$ **by**(*rule Sim*)
ultimately obtain P' **where** $PTrans: P \mapsto a<\nu y> \prec P'$ **and** $P'RelQ':$
 $(P', [(x, y)] \cdot Q') \in Rel$
using $\langle y \# P \rangle$
by(*blast dest: elim*)
from $PTrans \langle y \# R \rangle$ **have** $P \parallel R \mapsto a<\nu y> \prec (P' \parallel R)$ **by**(*force intro: Early-Semantics.Par1B*)
moreover from $P'RelQ' PBRQ$ **have** $(P' \parallel R, [(x, y)] \cdot Q' \parallel !Q) \in \text{bangRel Rel}$ **by**(*rule Rel.BRPar*)
with $\langle x \# Q \rangle \langle y \# Q \rangle$ **have** $(P' \parallel R, [(y, x)] \cdot Q' \parallel ![(y, x)] \cdot Q) \in \text{bangRel Rel}$
by(*simp add: name-fresh-fresh name-swap*)
ultimately show $\exists P'. P \parallel R \mapsto a<\nu y> \prec P' \wedge (P', [(y, x)] \cdot Q' \parallel ![(y, x)] \cdot Q) \in \text{bangRel Rel}$ **by** *blast*

qed
qed
next
case(*Par1F $\alpha Q' Pa P$*)
have $QTrans: Q \mapsto \alpha \prec Q'$ **by** *fact*
have $(Pa, Q \parallel !Q) \in \text{bangRel Rel}$ **by** *fact*
thus *?case*

```

proof(induct rule: BRParCases)
  case(BRPar P R)
  have PRelQ: (P, Q) ∈ Rel and BR: (R, !Q) ∈ bangRel Rel by fact+
  show ?case
  proof(auto simp add: residual.inject)
    from PRelQ have  $P \rightsquigarrow_{[Rel]} Q$  by(rule Sim)
    with QTrans obtain  $P'$  where  $PTrans: P \mapsto \alpha \prec P'$  and  $RRel: (P', Q') \in Rel$ 
    by(blast dest: elim)

    from PTrans have  $P \parallel R \mapsto \alpha \prec P' \parallel R$  by(rule TransitionsEarly.Par1F)
    moreover from RRel BR have  $(P' \parallel R, Q' \parallel !Q) \in bangRel Rel$  by(rule Rel.BRPar)
    ultimately show  $\exists P'. P \parallel R \mapsto \alpha \prec P' \wedge (P', Q' \parallel !Q) \in bangRel Rel$ 
by blast
  qed
  qed
next
  case(Par2B a x Q' Pa P)
  hence IH:  $\bigwedge Pa. (Pa, !Q) \in bangRel Rel \implies ?Sim Pa (a \langle \nu x \rangle \prec Q')$  by
simp
  have  $(Pa, Q \parallel !Q) \in bangRel Rel$  and  $x \# Pa$  by fact+
  thus  $?Sim Pa (a \langle \nu x \rangle \prec (Q \parallel Q'))$ 
  proof(induct rule: BRParCases)
    case(BRPar P R)
    have PRelQ: (P, Q) ∈ Rel and RBRQ: (R, !Q) ∈ bangRel Rel by fact+
    have  $x \# P \parallel R$  by fact
    hence  $xFreshP: x \# P$  and  $xFreshR: x \# R$  by simp+

    from EqvtBangRel show  $?Sim (P \parallel R) (a \langle \nu x \rangle \prec (Q \parallel Q'))$ 
    proof(auto simp add: residual.inject alpha')
      from RBRQ have  $?Sim R (a \langle \nu x \rangle \prec Q')$  by(rule IH)
      with  $xFreshR$  obtain  $R'$  where  $RTrans: R \mapsto a \langle \nu x \rangle \prec R'$  and  $R'BRQ': (R', Q') \in (bangRel Rel)$ 
      by(metis elim)
      from  $RTrans$   $xFreshP$  have  $P \parallel R \mapsto a \langle \nu x \rangle \prec (P \parallel R')$  by(auto intro: Early-Semantics.Par2B)
      moreover from PRelQ R'BRQ' have  $(P \parallel R', Q \parallel Q') \in (bangRel Rel)$ 
by(rule Rel.BRPar)
      ultimately show  $\exists P'. P \parallel R \mapsto a \langle \nu x \rangle \prec P' \wedge (P', Q \parallel Q') \in bangRel Rel$ 
by blast
    next
    fix  $y$ 
    assume  $(y::name) \# Q$  and  $y \# Q'$  and  $y \# P$  and  $y \# R$ 
    from RBRQ have  $?Sim R (a \langle \nu x \rangle \prec Q')$  by(rule IH)
    with  $\langle y \# Q' \rangle$  have  $?Sim R (a \langle \nu y \rangle \prec ([x, y] \cdot Q'))$  by(simp add: alphaBoundOutput)
    with  $\langle y \# R \rangle$  obtain  $R'$  where  $RTrans: R \mapsto a \langle \nu y \rangle \prec R'$  and  $R'BRQ': (R', ([x, y] \cdot Q')) \in (bangRel Rel)$ 

```

by(*metis elim*)
from $RTrans \langle y \# P \rangle$ **have** $P \parallel R \mapsto a \langle \nu y \rangle \prec (P \parallel R')$ **by**(*auto intro: Early-Semantics.Par2B*)
moreover from $PRelQ R'BRQ'$ **have** $(P \parallel R', Q \parallel [(x, y)] \cdot Q') \in$
(*bangRel Rel*) **by**(*rule Rel.BRPar*)
with $\langle y \# Q \rangle \langle x \# Q \rangle$ **have** $(P \parallel R', [(y, x)] \cdot Q \parallel [(y, x)] \cdot Q') \in$
(*bangRel Rel*)
by(*simp add: name-swap name-fresh-fresh*)
ultimately show $\exists P'. P \parallel R \mapsto a \langle \nu y \rangle \prec P' \wedge (P', [(y, x)] \cdot Q \parallel$
 $[(y, x)] \cdot Q') \in$ *bangRel Rel* **by** *blast*
qed
qed
next
case(*Par2F $\alpha Q' Pa P$*)
hence $IH: \bigwedge Pa. (Pa, !Q) \in$ *bangRel Rel* $\implies ?Sim Pa (\alpha \prec Q')$ **by** *simp*
have $(Pa, Q \parallel !Q) \in$ *bangRel Rel* **by** *fact*
thus *?case*
proof(*induct rule: BRParCases*)
case(*BRPar $P R$*)
have $PRelQ: (P, Q) \in Rel$ **and** $RBRQ: (R, !Q) \in$ *bangRel Rel* **by** *fact+*
show *?case*
proof(*auto simp add: residual.inject*)
from $RBRQ IH$ **have** $\exists R'. R \mapsto \alpha \prec R' \wedge (R', Q') \in$ *bangRel Rel*
by(*metis elim*)
then obtain R' **where** $RTrans: R \mapsto \alpha \prec R'$ **and** $R'RelQ': (R', Q') \in$
bangRel Rel
by *blast*

from $RTrans$ **have** $P \parallel R \mapsto \alpha \prec P \parallel R'$ **by**(*rule TransitionsEarly.Par2F*)
moreover from $PRelQ R'RelQ'$ **have** $(P \parallel R', Q \parallel Q') \in$ *bangRel Rel*
by(*rule Rel.BRPar*)
ultimately show $\exists P'. P \parallel R \mapsto \alpha \prec P' \wedge (P', Q \parallel Q') \in$ *bangRel Rel*
by *blast*
qed
qed
next
case(*Comm1 $a Q' b Q'' Pa P$*)
hence $IH: \bigwedge Pa. (Pa, !Q) \in$ *bangRel Rel* $\implies ?Sim Pa (a[b] \prec Q'')$ **by** *simp*
have $QTrans: Q \mapsto a \langle b \rangle \prec Q'$ **by** *fact*
have $(Pa, Q \parallel !Q) \in$ *bangRel Rel* **by** *fact*
thus *?case*
proof(*induct rule: BRParCases*)
case(*BRPar $P R$*)
have $PRelQ: (P, Q) \in Rel$ **and** $RBRQ: (R, !Q) \in$ *bangRel Rel* **by** *fact+*
show *?case*
proof(*auto simp add: residual.inject*)
from $PRelQ$ **have** $P \rightsquigarrow_{[Rel]} Q$ **by**(*rule Sim*)
with $QTrans$ **obtain** P' **where** $PTrans: P \mapsto a \langle b \rangle \prec P'$ **and** $P'RelQ':$
 $(P', Q') \in Rel$

by(*blast dest: elim*)

from *IH RBRQ* **have** $RTrans: \exists R'. R \mapsto a[b] \prec R' \wedge (R', Q'') \in \text{bangRel Rel}$

by(*metis elim*)

then obtain R' **where** $RTrans: R \mapsto a[b] \prec R'$ **and** $R'RelQ'': (R', Q'') \in \text{bangRel Rel}$

by *blast*

from *PTrans RTrans* **have** $P \parallel R \mapsto \tau \prec P' \parallel R'$ **by**(*rule Transition-
sEarly.Comm1*)

moreover from $P'RelQ' R'RelQ''$ **have** $(P' \parallel R', Q' \parallel Q'') \in \text{bangRel Rel}$ **by**(*rule Rel.BRPar*)

ultimately show $\exists P'. P \parallel R \mapsto \tau \prec P' \wedge (P', Q' \parallel Q'') \in \text{bangRel Rel}$

by *blast*

qed

qed

next

case(*Comm2 a b Q' Q''*)

hence *IH*: $\bigwedge Pa. (Pa, !Q) \in \text{bangRel Rel} \implies ?Sim Pa (a < b > \prec Q')$ **by** *simp*

have $QTrans: Q \mapsto a[b] \prec Q'$ **by** *fact*

have $(Pa, Q \parallel !Q) \in \text{bangRel Rel}$ **by** *fact*

thus *?case*

proof(*induct rule: BRParCases*)

case(*BRPar P R*)

have $PRelQ: (P, Q) \in \text{Rel}$ **and** $RBRQ: (R, !Q) \in \text{bangRel Rel}$ **by** *fact+*

show *?case*

proof(*auto simp add: residual.inject*)

from $PRelQ$ **have** $P \rightsquigarrow_{[Rel]} Q$ **by**(*rule Sim*)

with $QTrans$ **obtain** P' **where** $PTrans: P \mapsto a[b] \prec P'$ **and** $P'RelQ': (P', Q') \in \text{Rel}$

by(*blast dest: elim*)

from *IH RBRQ* **have** $RTrans: \exists R'. R \mapsto a < b > \prec R' \wedge (R', Q'') \in \text{bangRel Rel}$

by(*metis elim*)

then obtain R' **where** $RTrans: R \mapsto a < b > \prec R'$ **and** $R'RelQ'': (R', Q'') \in \text{bangRel Rel}$

by *blast*

from *PTrans RTrans* **have** $P \parallel R \mapsto \tau \prec P' \parallel R'$ **by**(*rule Transition-
sEarly.Comm2*)

moreover from $P'RelQ' R'RelQ''$ **have** $(P' \parallel R', Q' \parallel Q'') \in \text{bangRel Rel}$ **by**(*rule Rel.BRPar*)

ultimately show $\exists P'. P \parallel R \mapsto \tau \prec P' \wedge (P', Q' \parallel Q'') \in \text{bangRel Rel}$

by *blast*

qed

qed

next

case(*Close1* $a\ x\ Q'\ Q''\ Pa\ P$)
hence $IH: \bigwedge Pa. (Pa, !Q) \in \text{bangRel Rel} \longrightarrow ?Sim\ Pa\ (a\langle \nu x \rangle \prec Q'')$ **by**
simp
have $QTrans: Q \mapsto a\langle x \rangle \prec Q'$ **by fact**
have $xFreshQ: x \# Q$ **by fact**
have $(Pa, Q \parallel !Q) \in \text{bangRel Rel}$ **by fact**
moreover have $xFreshPa: x \# Pa$ **by fact**
ultimately show *?case*
proof(*induct rule: BRParCases*)
case(*BRPar P R*)
have $PRelQ: (P, Q) \in Rel$ **and** $RBRQ: (R, !Q) \in \text{bangRel Rel}$ **by fact+**
have $x \# P \parallel R$ **by fact**
hence $xFreshP: x \# P$ **and** $xFreshR: x \# R$ **by simp+**
show *?case*
proof(*auto simp add: residual.inject*)
from $PRelQ$ **have** $P \rightsquigarrow_{[Rel]} Q$ **by**(*rule Sim*)
with $QTrans\ xFreshP$ **obtain** P' **where** $PTrans: P \mapsto a\langle x \rangle \prec P'$ **and**
 $P'RelQ': (P', Q') \in Rel$
by(*blast dest: elim*)

from $RBRQ\ xFreshR\ IH$ **have** $\exists R'. R \mapsto a\langle \nu x \rangle \prec R' \wedge (R', Q') \in$
 bangRel Rel
by(*metis elim*)
then obtain R' **where** $RTrans: R \mapsto a\langle \nu x \rangle \prec R'$ **and** $R'RelQ'': (R',$
 $Q'') \in \text{bangRel Rel}$
by *blast*

from $PTrans\ RTrans\ xFreshP$ **have** $P \parallel R \mapsto \tau \prec \langle \nu x \rangle (P' \parallel R')$
by(*rule Early-Semantics.Close1*)
moreover from $P'RelQ'\ R'RelQ''$ **have** $(\langle \nu x \rangle (P' \parallel R'), \langle \nu x \rangle (Q' \parallel$
 $Q'')) \in \text{bangRel Rel}$
by(*force intro: Rel.BRPar BRRes*)
ultimately show $\exists P'. P \parallel R \mapsto \tau \prec P' \wedge (P', \langle \nu x \rangle (Q' \parallel Q'')) \in$
 bangRel Rel **by** *blast*
qed
qed
next
case(*Close2* $a\ x\ Q'\ Q''\ Pa\ P$)
hence $IH: \bigwedge Pa. (Pa, !Q) \in \text{bangRel Rel} \implies ?Sim\ Pa\ (a\langle x \rangle \prec Q'')$ **by** *simp*
have $QTrans: Q \mapsto a\langle \nu x \rangle \prec Q'$ **by fact**
have $xFreshQ: x \# Q$ **by fact**
have $(Pa, Q \parallel !Q) \in \text{bangRel Rel}$ **and** $x \# Pa$ **by fact+**
thus *?case*
proof(*induct rule: BRParCases*)
case(*BRPar P R*)
have $PRelQ: (P, Q) \in Rel$ **and** $RBRQ: (R, !Q) \in \text{bangRel Rel}$ **by fact+**
have $x \# P \parallel R$ **by fact**
hence $xFreshP: x \# P$ **and** $xFreshR: x \# R$ **by simp+**
show *?case*

```

proof(auto simp add: residual.inject)
  from  $P \text{Rel} Q$  have  $P \rightsquigarrow_{[Rel]} Q$  by(rule Sim)
  with  $Q \text{Trans } x \text{Fresh} P$  obtain  $P'$  where  $P \text{Trans}: P \mapsto a \langle \nu x \rangle \prec P'$  and
 $P' \text{Rel} Q': (P', Q') \in \text{Rel}$ 
    by(blast dest: elim)

  from  $R \text{BR} Q$   $IH$  have  $\exists R'. R \mapsto a \langle x \rangle \prec R' \wedge (R', Q') \in \text{bangRel Rel}$ 
    by auto
  then obtain  $R'$  where  $R \text{Trans}: R \mapsto a \langle x \rangle \prec R'$  and  $R' \text{Rel} Q': (R',$ 
 $Q') \in \text{bangRel Rel}$ 
    by blast

  from  $P \text{Trans } R \text{Trans } x \text{Fresh} R$  have  $P \parallel R \mapsto \tau \prec \langle \nu x \rangle (P' \parallel R')$ 
    by(rule Early-Semantics.Close2)
  moreover from  $P' \text{Rel} Q' R' \text{Rel} Q''$  have  $(\langle \nu x \rangle (P' \parallel R'), \langle \nu x \rangle (Q' \parallel$ 
 $Q'')) \in \text{bangRel Rel}$ 
    by(force intro: Rel.BRPar BRRes)
  ultimately show  $\exists P'. P \parallel R \mapsto \tau \prec P' \wedge (P', \langle \nu x \rangle (Q' \parallel Q'')) \in$ 
 $\text{bangRel Rel}$  by blast
  qed
qed
next
  case( $\text{Bang } Rs \text{ Pa } P$ )
  hence  $IH: \bigwedge Pa. (Pa, Q \parallel !Q) \in \text{bangRel Rel} \implies ?\text{Sim } Pa \text{ Rs}$  by simp
  have  $(Pa, !Q) \in \text{bangRel Rel}$  by fact
  thus ?case
  proof(induct rule: BRBangCases)
    case( $\text{BRBang } P$ )
    have  $P \text{Rel} Q: (P, Q) \in \text{Rel}$  by fact
    hence  $(!P, !Q) \in \text{bangRel Rel}$  by(rule Rel.BRBang)
    with  $P \text{Rel} Q$  have  $(P \parallel !P, Q \parallel !Q) \in \text{bangRel Rel}$  by(rule BRPar)
    with  $IH$  have  $?\text{Sim } (P \parallel !P) \text{ Rs}$  by simp
    thus ?case by(force intro: TransitionsEarly.Bang)
  qed
qed
}

  moreover from  $P \text{Rel} Q$  have  $(!P, !Q) \in \text{bangRel Rel}$  by(rule BRBang)
  ultimately show ?thesis by(auto simp add: strongSimEarly-def)
qed

end

theory Strong-Early-Bisim-Pres
  imports Strong-Early-Bisim Strong-Early-Sim-Pres
begin

```

```

lemma tauPres:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 

  assumes  $P \sim Q$ 

  shows  $\tau.(P) \sim \tau.(Q)$ 
proof –
  let  $?X = \{(\tau.(P), \tau.(Q)) \mid P \ Q. P \sim Q\}$ 
  from  $\langle P \sim Q \rangle$  have  $(\tau.(P), \tau.(Q)) \in ?X$  by auto
  thus ?thesis
  by(coinduct rule: bisimCoinduct) (auto intro: tauPres dest: bisimE)
qed

lemma inputPres:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $a :: name$ 
  and  $x :: name$ 

  assumes  $PSimQ: \forall y. P[x::=y] \sim Q[x::=y]$ 

  shows  $a\langle x \rangle.P \sim a\langle x \rangle.Q$ 
proof –
  let  $?X = \{(a\langle x \rangle.P, a\langle x \rangle.Q) \mid a \ x \ P \ Q. \forall y. P[x::=y] \sim Q[x::=y]\}$ 
  {
    fix  $axP \ axQ \ p$ 
    assume  $(axP, axQ) \in ?X$ 
    then obtain  $a \ x \ P \ Q$  where  $A: \forall y. P[x::=y] \sim Q[x::=y]$  and  $B: axP =$ 
 $a\langle x \rangle.P$  and  $C: axQ = a\langle x \rangle.Q$ 
    by auto
    have  $\bigwedge y. ((p::name \ prm) \cdot P)[(p \cdot x)::=y] \sim (p \cdot Q)[(p \cdot x)::=y]$ 
    proof –
      fix  $y$ 
      from  $A$  have  $P[x::=(rev \ p \cdot y)] \sim Q[x::=(rev \ p \cdot y)]$ 
      by blast
      hence  $(p \cdot (P[x::=(rev \ p \cdot y)])) \sim p \cdot (Q[x::=(rev \ p \cdot y)])$ 
      by(rule bisimClosed)
      thus  $(p \cdot P)[(p \cdot x)::=y] \sim (p \cdot Q)[(p \cdot x)::=y]$ 
      by(simp add: eqvts pt-pi-rev[OF pt-name-inst, OF at-name-inst])
    qed
    hence  $((p::name \ prm) \cdot axP, p \cdot axQ) \in ?X$  using  $B \ C$ 
    by auto
  }
  hence eqvt  $?X$  by(simp add: eqvt-def)
  from  $PSimQ$  have  $(a\langle x \rangle.P, a\langle x \rangle.Q) \in ?X$  by auto
  thus ?thesis
proof(coinduct rule: bisimCoinduct)
  case(cSim P Q)

```

```

    thus ?case using ⟨eqvt ?X⟩
    by(force intro: inputPres)
next
case(cSym P Q)
thus ?case
by(blast dest: bisimE)
qed
qed

```

lemma *outputPres*:

```

fixes P :: pi
and Q :: pi
and a :: name
and b :: name

```

assumes $P \sim Q$

shows $a\{b\}.P \sim a\{b\}.Q$

proof –

```

let ?X = {(a{b}.P, a{b}.Q) | a b P Q. P ~ Q}
from ⟨P ~ Q⟩ have (a{b}.P, a{b}.Q) ∈ ?X by auto
thus ?thesis

```

by(*coinduct rule: bisimCoinduct*) (*blast intro: outputPres dest: bisimE*)+

qed

lemma *matchPres*:

```

fixes P :: pi
and Q :: pi
and a :: name
and b :: name

```

assumes $P \sim Q$

shows $[a\curvearrowright]P \sim [a\curvearrowright]Q$

proof –

```

let ?X = {x. ∃ P Q a b. P ~ Q ∧ x = ([a\curvearrowright]P, [a\curvearrowright]Q)}
from assms have ([a\curvearrowright]P, [a\curvearrowright]Q) ∈ ?X by blast
thus ?thesis

```

by(*coinduct rule: bisimCoinduct*) (*blast intro: matchPres dest: bisimE*)+

qed

lemma *mismatchPres*:

```

fixes P :: pi
and Q :: pi
and a :: name
and b :: name

```

assumes $P \sim Q$

shows $[a \neq b]P \sim [a \neq b]Q$
proof –
let $?X = \{x. \exists P Q a b. P \sim Q \wedge x = ([a \neq b]P, [a \neq b]Q)\}$
from *assms* **have** $([a \neq b]P, [a \neq b]Q) \in ?X$ **by** *blast*
thus *?thesis*
by(*coinduct rule: bisimCoinduct*) (*blast intro: mismatchPres dest: bisimE*)+
qed

lemma *sumPres*:

fixes $P :: pi$
and $Q :: pi$
and $R :: pi$

assumes $P \sim Q$

shows $P \oplus R \sim Q \oplus R$

proof –
let $?X = \{(P \oplus R, Q \oplus R) \mid P Q R. P \sim Q\}$
from *assms* **have** $(P \oplus R, Q \oplus R) \in ?X$ **by** *blast*
thus *?thesis*
by(*coinduct rule: bisimCoinduct*) (*auto dest: bisimE intro: reflexive sumPres*)
qed

lemma *resPres*:

fixes $P :: pi$
and $Q :: pi$
and $x :: name$

assumes $P \sim Q$

shows $\langle \nu x \rangle P \sim \langle \nu x \rangle Q$

proof –
let $?X = \{x. \exists P Q. P \sim Q \wedge (\exists a. x = (\langle \nu a \rangle P, \langle \nu a \rangle Q))\}$
from *assms* **have** $(\langle \nu x \rangle P, \langle \nu x \rangle Q) \in ?X$ **by** *blast*
thus *?thesis*
proof(*coinduct rule: bisimCoinduct*)
case(*cSim xP xQ*)
moreover {
fix $P Q a$
assume $P \sim Q$
hence $P \rightsquigarrow[bisim] Q$ **by**(*rule bisimE*)
moreover **have** $\bigwedge P Q a. P \sim Q \implies (\langle \nu a \rangle P, \langle \nu a \rangle Q) \in ?X \cup bisim$ **by**
blast
moreover **have** $bisim \subseteq ?X \cup bisim$ **by** *blast*
moreover **have** *eqvt bisim* **by**(*rule eqvt*)
moreover **have** *eqvt (?X \cup bisim)* **using** *eqvts*
by(*auto simp add: eqvt-def*) *blast*
ultimately **have** $\langle \nu a \rangle P \rightsquigarrow[(?X \cup bisim)] \langle \nu a \rangle Q$
by(*rule Strong-Early-Sim-Pres.resPres*)

```

    }
    ultimately show ?case by auto
  next
    case(cSym xP xQ)
    thus ?case by(auto dest: bisimE)
  qed
qed

lemma parPres:
  fixes P :: pi
  and Q :: pi
  and R :: pi
  and T :: pi

  assumes P ~ Q

  shows P || R ~ Q || R
proof -
  let ?X = {(resChain lst (P || R), resChain lst (Q || R)) | lst P Q R. P ~ Q}
  have BC:  $\bigwedge P Q. P || Q = resChain [] (P || Q)$  by auto
  from assms have (P || R, Q || R)  $\in$  ?X by(blast intro: BC)
  thus ?thesis
proof(coinduct rule: bisimWeakCoinduct)
  case(cSim PR QR)
  moreover {
    fix lst P Q R
    assume P ~ Q
    have eqvt ?X using eqvts by(auto simp add: eqvt-def) blast
    moreover have Res:  $\bigwedge P Q x. (P, Q) \in ?X \implies (\langle \nu x \rangle P, \langle \nu x \rangle Q) \in ?X$ 
      by(auto, rule-tac x=x#lst in exI) auto
    moreover {
      from  $\langle P \sim Q \rangle$  have  $P \rightsquigarrow[bisim] Q$  by(rule bisimE)
      moreover note  $\langle P \sim Q \rangle$ 
      moreover have  $\bigwedge P Q R. P \sim Q \implies (P || R, Q || R) \in ?X$ 
        by(blast intro: BC)
      ultimately have  $P || R \rightsquigarrow[?X] Q || R$  using Res
        by(rule parPres)
    }
  }

  ultimately have  $resChain\ lst\ (P || R) \rightsquigarrow[?X] resChain\ lst\ (Q || R)$ 
    by(rule resChainI)
  }
  ultimately show ?case by auto
next
  case(cSym P Q)
  thus ?case by(auto dest: bisimE)
qed
qed

```

```

lemma bangRelBisimE:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $Rel :: (pi \times pi)$  set

  assumes  $A: (P, Q) \in \text{bangRel } Rel$ 
  and  $Sym: \bigwedge P Q. (P, Q) \in Rel \implies (Q, P) \in Rel$ 

  shows  $(Q, P) \in \text{bangRel } Rel$ 
proof –
  from  $A$  show ?thesis
proof(induct)
  fix  $P Q$ 
  assume  $(P, Q) \in Rel$ 
  hence  $(Q, P) \in Rel$  by(rule Sym)
  thus  $(!Q, !P) \in \text{bangRel } Rel$  by(rule BRBang)
next
  fix  $P Q R T$ 
  assume  $RRelT: (R, T) \in Rel$ 
  assume  $IH: (Q, P) \in \text{bangRel } Rel$ 
  from  $RRelT$  have  $(T, R) \in Rel$  by(rule Sym)
  thus  $(T \parallel Q, R \parallel P) \in \text{bangRel } Rel$  using  $IH$  by(rule BRPar)
next
  fix  $P Q a$ 
  assume  $(Q, P) \in \text{bangRel } Rel$ 
  thus  $(\langle \nu a \rangle Q, \langle \nu a \rangle P) \in \text{bangRel } Rel$  by(rule BRRes)
qed
qed

```

```

lemma bangPres:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 

  assumes  $PBiSimQ: P \sim Q$ 

  shows  $!P \sim !Q$ 
proof –
  let  $?X = \text{bangRel } \text{bisim}$ 
  from  $PBiSimQ$  have  $(!P, !Q) \in ?X$  by(rule BRBang)
  thus ?thesis
proof(coinduct rule: bisimWeakCoinduct)
  case(cSim bP bQ)
  {
    fix  $P Q$ 
    assume  $(P, Q) \in ?X$ 
    hence  $P \rightsquigarrow[?X] Q$ 
    proof(induct)
    fix  $P Q$ 
    assume  $P \sim Q$ 

```

```

    thus !P  $\rightsquigarrow$ [?X] !Q using bisimE(1) eqvt
      by(rule Strong-Early-Sim-Pres.bangPres)
  next
    fix P Q R T
    assume RBiSimT: R  $\sim$  T
    assume PBangRelQ: (P, Q)  $\in$  ?X
    assume PSimQ: P  $\rightsquigarrow$ [?X] Q
    from RBiSimT have R  $\rightsquigarrow$ [bisim] T by(blast dest: bisimE)
    thus R || P  $\rightsquigarrow$ [?X] T || Q using PSimQ RBiSimT PBangRelQ BRPar
  BRRes eqvt eqvtBangRel
    by(blast intro: Strong-Early-Sim-Pres.parCompose)
  next
    fix P Q a
    assume P  $\rightsquigarrow$ [?X] Q
    moreover from eqvtBangRel eqvt have eqvt ?X by blast
    ultimately show <v a>P  $\rightsquigarrow$ [?X] <v a>Q using BRRes by(blast intro:
  Strong-Early-Sim-Pres.resPres)
    qed
  }
  with (bP, bQ)  $\in$  ?X show ?case by blast
next
  case(cSym bP bQ)
  thus ?case by(metis bangRelSymmetric bisimE)
qed
qed
end

```

```

theory Strong-Early-Bisim-Subst-Pres
  imports Strong-Early-Bisim-Subst Strong-Early-Bisim-Pres
begin

```

```

lemma tauPres:
  fixes P :: pi
  and Q :: pi

  assumes P  $\sim^s$  Q

  shows  $\tau.(P) \sim^s \tau.(Q)$ 
using assms
by(force simp add: substClosed-def intro: Strong-Early-Bisim-Pres.tauPres)

```

```

lemma inputPres:
  fixes P :: pi
  and Q :: pi
  and a :: name
  and x :: name

  assumes P  $\sim^s$  Q

```

```

shows  $a\langle x \rangle.P \sim^s a\langle x \rangle.Q$ 
proof(auto simp add: substClosed-def)
  fix  $\sigma :: (\text{name} \times \text{name}) \text{ list}$ 
  {
    fix  $P Q a x \sigma$ 
    assume  $P \sim^s Q$ 
    then have  $P[\langle \sigma \rangle] \sim^s Q[\langle \sigma \rangle]$  by(rule partUnfold)
    then have  $\forall y. (P[\langle \sigma \rangle])[x::=y] \sim (Q[\langle \sigma \rangle])[x::=y]$ 
      apply(auto simp add: substClosed-def)
      by(erule-tac x=[(x, y)] in allE) auto
    moreover assume  $x \# \sigma$ 
    ultimately have  $(a\langle x \rangle.P)[\langle \sigma \rangle] \sim (a\langle x \rangle.Q)[\langle \sigma \rangle]$ 
      by(force intro: Strong-Early-Bisim-Pres.inputPres)
  }
note Goal = this

obtain  $y::\text{name}$  where  $y \# P$  and  $y \# Q$  and  $y \# \sigma$ 
  by(generate-fresh name) auto
from  $\langle P \sim^s Q \rangle$  have  $([(x, y)] \cdot P) \sim^s ([(x, y)] \cdot Q)$  by(rule eqvtI)
hence  $(a\langle y \rangle.([(x, y)] \cdot P))[\langle \sigma \rangle] \sim (a\langle y \rangle.([(x, y)] \cdot Q))[\langle \sigma \rangle]$  using  $\langle y \# \sigma \rangle$ 
by(rule Goal)
  moreover from  $\langle y \# P \rangle \langle y \# Q \rangle$  have  $a\langle x \rangle.P = a\langle y \rangle.([(x, y)] \cdot P)$  and
 $a\langle x \rangle.Q = a\langle y \rangle.([(x, y)] \cdot Q)$ 
  by(simp add: alphaInput)+

  ultimately show  $(a\langle x \rangle.P)[\langle \sigma \rangle] \sim (a\langle x \rangle.Q)[\langle \sigma \rangle]$  by simp
qed

lemma outputPres:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 

  assumes  $P \sim^s Q$ 

  shows  $a\{b\}.P \sim^s a\{b\}.Q$ 
using assms
by(force simp add: substClosed-def intro: Strong-Early-Bisim-Pres.outputPres)

lemma matchPres:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $a :: \text{name}$ 
  and  $b :: \text{name}$ 

  assumes  $P \sim^s Q$ 

  shows  $[a \frown b]P \sim^s [a \frown b]Q$ 
using assms

```

by(force simp add: substClosed-def intro: Strong-Early-Bisim-Pres.matchPres)

lemma mismatchPres:

fixes $P :: pi$
and $Q :: pi$
and $a :: name$
and $b :: name$

assumes $P \sim^s Q$

shows $[a \neq b]P \sim^s [a \neq b]Q$

using *assms*

by(force simp add: substClosed-def intro: Strong-Early-Bisim-Pres.mismatchPres)

lemma sumPres:

fixes $P :: pi$
and $Q :: pi$
and $R :: pi$

assumes $P \sim^s Q$

shows $P \oplus R \sim^s Q \oplus R$

using *assms*

by(force simp add: substClosed-def intro: Strong-Early-Bisim-Pres.sumPres)

lemma parPres:

fixes $P :: pi$
and $Q :: pi$
and $R :: pi$

assumes $P \sim^s Q$

shows $P \parallel R \sim^s Q \parallel R$

using *assms*

by(force simp add: substClosed-def intro: Strong-Early-Bisim-Pres.parPres)

lemma resPres:

fixes $P :: pi$
and $Q :: pi$
and $x :: name$

assumes $PeqQ: P \sim^s Q$

shows $\langle \nu x \rangle P \sim^s \langle \nu x \rangle Q$

proof(auto simp add: substClosed-def)

fix $s :: (name \times name) list$

have $Res: \bigwedge P Q x s. \llbracket P[\langle s \rangle] \sim Q[\langle s \rangle]; x \# s \rrbracket \implies (\langle \nu x \rangle P)[\langle s \rangle] \sim (\langle \nu x \rangle Q)[\langle s \rangle]$
by(force intro: Strong-Early-Bisim-Pres.resPres)

```

have  $\exists c::name. c \# (P, Q, s)$  by(blast intro: name-exists-fresh)
then obtain  $c::name$  where  $cFreshP: c \# P$  and  $cFreshQ: c \# Q$  and  $cFreshs:$ 
 $c \# s$ 
by(force simp add: fresh-prod)

from  $PeqQ$  have  $P[\langle [(x, c)] \cdot s \rangle] \sim Q[\langle [(x, c)] \cdot s \rangle]$  by(simp add: subst-
Closed-def)
hence  $([(x, c)] \cdot P[\langle [(x, c)] \cdot s \rangle]) \sim ([x, c] \cdot Q[\langle [(x, c)] \cdot s \rangle])$  by(rule
Strong-Early-Bisim.bisimClosed)
hence  $([(x, c)] \cdot P[\langle s \rangle]) \sim ([x, c] \cdot Q[\langle s \rangle])$  by simp
hence  $(\langle \nu c \rangle([(x, c)] \cdot P))[\langle s \rangle] \sim (\langle \nu c \rangle([x, c] \cdot Q))[\langle s \rangle]$  using  $cFreshs$ 
by(rule Res)

moreover from  $cFreshP$   $cFreshQ$  have  $\langle \nu x \rangle P = \langle \nu c \rangle([x, c] \cdot P)$  and
 $\langle \nu x \rangle Q = \langle \nu c \rangle([x, c] \cdot Q)$ 
by(simp add: alphaRes)+

ultimately show  $(\langle \nu x \rangle P)[\langle s \rangle] \sim (\langle \nu x \rangle Q)[\langle s \rangle]$  by simp
qed

lemma  $bangPres:$ 
fixes  $P :: pi$ 
and  $Q :: pi$ 

assumes  $P \sim^s Q$ 

shows  $!P \sim^s !Q$ 
using  $assms$ 
by(force simp add: substClosed-def intro: Strong-Early-Bisim-Pres.bangPres)

end

theory  $Early-Tau-Chain$ 
imports  $Early-Semantics$ 
begin

abbreviation  $tauChain :: pi \Rightarrow pi \Rightarrow bool$  ( $\langle \cdot \rangle \Longrightarrow_{\tau} \rightarrow [80, 80] 80$ )
where  $P \Longrightarrow_{\tau} P' \equiv (P, P') \in \{(P, P') \mid P P'. P \mapsto_{\tau} \prec P'\}^*$ 

lemma  $tauActTauChain:$ 
fixes  $P :: pi$ 
and  $P' :: pi$ 

assumes  $P \mapsto_{\tau} \prec P'$ 

shows  $P \Longrightarrow_{\tau} P'$ 
using  $assms$ 
by auto

```

lemma *tauChainAddTau*[*intro*]:

fixes $P :: pi$
and $P' :: pi$
and $P'' :: pi$

shows $P \Longrightarrow_{\tau} P' \Longrightarrow P' \mapsto_{\tau} \prec P'' \Longrightarrow P \Longrightarrow_{\tau} P''$
and $P \mapsto_{\tau} \prec P' \Longrightarrow P' \Longrightarrow_{\tau} P'' \Longrightarrow P \Longrightarrow_{\tau} P''$

by(*auto dest: tauActTauChain*)

lemma *tauChainInduct*[*consumes 1, case-names id ih*]:

fixes $P :: pi$
and $P' :: pi$

assumes $P \Longrightarrow_{\tau} P'$
and $F P$
and $\bigwedge P'' P'''. \llbracket P \Longrightarrow_{\tau} P''; P'' \mapsto_{\tau} \prec P'''; F P'' \rrbracket \Longrightarrow F P'''$

shows $F P'$

using *assms*

by(*drule-tac rtrancl-induct*) *auto*

lemma *eqvtChainI*:

fixes $P :: pi$
and $P' :: pi$
and $perm :: name prm$

assumes $P \Longrightarrow_{\tau} P'$

shows $(perm \cdot P) \Longrightarrow_{\tau} (perm \cdot P')$

using *assms*

proof(*induct rule: tauChainInduct*)

case *id*

thus *?case by simp*

next

case(*ih P'' P'''*)

have $P \Longrightarrow_{\tau} P''$ **and** $P'' \mapsto_{\tau} \prec P'''$ **by** *fact+*

hence $(perm \cdot P'') \mapsto_{\tau} \prec (perm \cdot P''')$ **by**(*drule-tac TransitionsEarly.eqvt*)

auto

moreover have $(perm \cdot P) \Longrightarrow_{\tau} (perm \cdot P'')$ **by** *fact*

ultimately show *?case by*(*force dest: tauActTauChain*)

qed

lemma *eqvtChainE*:

fixes $perm :: name prm$
and $P :: pi$
and $P' :: pi$

assumes *Trans*: $(perm \cdot P) \Longrightarrow_{\tau} (perm \cdot P')$

shows $P \Longrightarrow_{\tau} P'$
proof –
have $\text{rev perm} \cdot (\text{perm} \cdot P) = P$ **by** (*simp add: pt-rev-pi[OF pt-name-inst, OF at-name-inst]*)
moreover have $\text{rev perm} \cdot (\text{perm} \cdot P') = P'$ **by** (*simp add: pt-rev-pi[OF pt-name-inst, OF at-name-inst]*)
ultimately show *?thesis* **using** *assms*
by (*drule-tac perm=rev perm in eqvtChainI, simp*)
qed

lemma *eqvtChainEq*:

fixes $P :: pi$
and $P' :: pi$
and $\text{perm} :: name prm$

shows $P \Longrightarrow_{\tau} P' = (\text{perm} \cdot P) \Longrightarrow_{\tau} (\text{perm} \cdot P')$
by (*blast intro: eqvtChainE eqvtChainI*)

lemma *freshChain*:

fixes $P :: pi$
and $P' :: pi$
and $x :: name$

assumes $P \Longrightarrow_{\tau} P'$
and $x \# P$

shows $x \# P'$

using *assms*

proof (*induct rule: tauChainInduct*)

case *id*

thus *?case* **by** *simp*

next

case (*ih P' P''*)

have $x \# P$ **and** $x \# P \Longrightarrow x \# P'$ **by** *fact+*

hence $x \# P'$ **by** *simp*

moreover have $P' \mapsto_{\tau} P''$ **by** *fact*

ultimately show *?case* **by** (*force intro: freshTransition*)

qed

lemma *matchChain*:

fixes $b :: name$
and $P :: pi$
and $P' :: pi$

assumes $P \Longrightarrow_{\tau} P'$

and $P \neq P'$

shows $[b \curvearrowright]P \Longrightarrow_{\tau} P'$

```

using assms
proof(induct rule: tauChainInduct)
  case id
  thus ?case by simp
next
  case(ih P'' P''')
  have P''TransP''':  $P'' \mapsto_{\tau} \prec P'''$  by fact
  show  $[b \frown b]P \Longrightarrow_{\tau} P'''$ 
  proof(cases P = P'')
    assume  $P = P''$ 
    moreover with P''TransP''' have  $[b \frown b]P \mapsto_{\tau} \prec P'''$  by(force intro: Match)
    thus  $[b \frown b]P \Longrightarrow_{\tau} P'''$  by(rule tauActTauChain)
  next
    assume  $P \neq P''$ 
    moreover have  $P \neq P'' \Longrightarrow [b \frown b]P \Longrightarrow_{\tau} P''$  by fact
    ultimately show  $[b \frown b]P \Longrightarrow_{\tau} P'''$  using P''TransP''' by(blast)
  qed
qed

```

lemma *mismatchChain*:

```

fixes a :: name
and b :: name
and P :: pi
and P' :: pi

```

```

assumes PChain:  $P \Longrightarrow_{\tau} P'$ 
and aineqb:  $a \neq b$ 
and PineqP':  $P \neq P'$ 

```

shows $[a \neq b]P \Longrightarrow_{\tau} P'$

proof –

from *PChain PineqP'* **show** *?thesis*

```

proof(induct rule: tauChainInduct)

```

```

  case id

```

```

    thus ?case by simp

```

next

```

  case(ih P'' P''')

```

```

  have P''TransP''':  $P'' \mapsto_{\tau} \prec P'''$  by fact

```

```

  show  $[a \neq b]P \Longrightarrow_{\tau} P'''$ 

```

```

  proof(cases P = P'')

```

```

    assume  $P = P''$ 

```

```

    moreover with aineqb P''TransP''' have  $[a \neq b]P \mapsto_{\tau} \prec P'''$  by(force intro:

```

Mismatch)

```

    thus  $[a \neq b]P \Longrightarrow_{\tau} P'''$  by(rule tauActTauChain)

```

next

```

    assume  $P \neq P''$ 

```

```

    moreover have  $P \neq P'' \Longrightarrow [a \neq b]P \Longrightarrow_{\tau} P''$  by fact

```

```

    ultimately show  $[a \neq b]P \Longrightarrow_{\tau} P'''$  using P''TransP''' by(blast)

```

qed

qed
qed

lemma *sum1Chain*:

fixes $P :: pi$
and $P' :: pi$
and $Q :: pi$

assumes $P \Rightarrow_{\tau} P'$
and $P \neq P'$

shows $P \oplus Q \Rightarrow_{\tau} P'$

using *assms*

proof(*induct rule: tauChainInduct*)

case *id*

thus ?*case* by *simp*

next

case(*ih P'' P'''*)

have $P'' \text{Trans} P'''$: $P'' \mapsto_{\tau} < P'''$ by *fact*

show $P \oplus Q \Rightarrow_{\tau} P'''$

proof(*cases P = P''*)

assume $P = P''$

moreover with $P'' \text{Trans} P'''$ have $P \oplus Q \mapsto_{\tau} < P'''$ by(*force intro: Sum1*)

thus $P \oplus Q \Rightarrow_{\tau} P'''$ by(*force intro: tauActTauChain*)

next

assume $P \neq P''$

moreover have $P \neq P'' \Rightarrow P \oplus Q \Rightarrow_{\tau} P''$ by *fact*

ultimately show $P \oplus Q \Rightarrow_{\tau} P'''$ using $P'' \text{Trans} P'''$ by(*force dest: tauAct-*

TauChain)

qed

qed

lemma *sum2Chain*:

fixes $P :: pi$
and $Q :: pi$
and $Q' :: pi$

assumes $Q \Rightarrow_{\tau} Q'$
and $Q \neq Q'$

shows $P \oplus Q \Rightarrow_{\tau} Q'$

using *assms*

proof(*induct rule: tauChainInduct*)

case *id*

thus ?*case* by *simp*

next

case(*ih Q'' Q'''*)

have $Q'' \text{Trans} Q'''$: $Q'' \mapsto_{\tau} < Q'''$ by *fact*

show $P \oplus Q \Rightarrow_{\tau} Q'''$

```

proof(cases  $Q = Q'$ )
  assume  $Q=Q''$ 
  moreover with  $Q''\text{Trans}Q'''$  have  $P \oplus Q \mapsto_{\tau} \prec Q'''$  by(force intro: Sum2)
  thus  $P \oplus Q \implies_{\tau} Q'''$  by(force intro: tauActTauChain)
next
  assume  $Q \neq Q''$ 
  moreover have  $Q \neq Q'' \implies P \oplus Q \implies_{\tau} Q''$  by fact
  ultimately show  $P \oplus Q \implies_{\tau} Q'''$  using  $Q''\text{Trans}Q'''$  by blast
qed
qed

```

lemma *Par1Chain*:

```

fixes  $P :: pi$ 
and  $P' :: pi$ 
and  $Q :: pi$ 

```

assumes $P \implies_{\tau} P'$

shows $P \parallel Q \implies_{\tau} P' \parallel Q$

using *assms*

proof(*induct rule: tauChainInduct*)

case *id*

thus ?*case* **by simp**

next

case(*ih* $P'' P'$)

have $P''\text{Trans}P'$: $P'' \mapsto_{\tau} \prec P'$ **by fact**

have *IH*: $P \parallel Q \implies_{\tau} P'' \parallel Q$ **by fact**

have $P'' \parallel Q \mapsto_{\tau} \prec P' \parallel Q$ **using** $P''\text{Trans}P'$ **by**(force intro: Par1F)

thus $P \parallel Q \implies_{\tau} P' \parallel Q$ **using** *IH* **by**(force dest: tauActTauChain)

qed

lemma *Par2Chain*:

```

fixes  $P :: pi$ 
and  $Q :: pi$ 
and  $Q' :: pi$ 

```

assumes $Q \implies_{\tau} Q'$

shows $P \parallel Q \implies_{\tau} P \parallel Q'$

using *assms*

proof(*induct rule: tauChainInduct*)

case *id*

thus ?*case* **by simp**

next

case(*ih* $Q'' Q'$)

have $Q''\text{Trans}Q'$: $Q'' \mapsto_{\tau} \prec Q'$ **by fact**

have *IH*: $P \parallel Q \implies_{\tau} P \parallel Q''$ **by fact**

have $P \parallel Q'' \mapsto_{\tau} \prec P \parallel Q'$ using $Q''\text{Trans}Q'$ by(force intro: Par2F)
 thus $P \parallel Q \Longrightarrow_{\tau} P \parallel Q'$ using IH by(force dest: tauActTauChain)
 qed

lemma *chainPar*:

fixes $P :: pi$
 and $P' :: pi$
 and $Q :: pi$
 and $Q' :: pi$

assumes $P \Longrightarrow_{\tau} P'$
 and $Q \Longrightarrow_{\tau} Q'$

shows $P \parallel Q \Longrightarrow_{\tau} P' \parallel Q'$

proof –

from $\langle P \Longrightarrow_{\tau} P' \rangle$ have $P \parallel Q \Longrightarrow_{\tau} P' \parallel Q$ by(rule Par1Chain)
 moreover from $\langle Q \Longrightarrow_{\tau} Q' \rangle$ have $P' \parallel Q \Longrightarrow_{\tau} P' \parallel Q'$ by(rule Par2Chain)
 ultimately show ?thesis by auto

qed

lemma *ResChain*:

fixes $P :: pi$
 and $P' :: pi$
 and $a :: name$

assumes $P \Longrightarrow_{\tau} P'$

shows $\langle \nu a \rangle P \Longrightarrow_{\tau} \langle \nu a \rangle P'$

using *assms*

proof(*induct rule: tauChainInduct*)

case *id*

thus ?case by *simp*

next

case(*ih P'' P'''*)

have $P'' \mapsto_{\tau} \prec P'''$ by *fact*

hence $\langle \nu a \rangle P'' \mapsto_{\tau} \prec \langle \nu a \rangle P'''$ by(force intro: ResF)

moreover have $\langle \nu a \rangle P \Longrightarrow_{\tau} \langle \nu a \rangle P''$ by *fact*

ultimately show ?case by(force dest: tauActTauChain)

qed

lemma *substChain*:

fixes $P :: pi$
 and $x :: name$
 and $b :: name$
 and $P' :: pi$

assumes $P\text{Trans}: P[x::=b] \Longrightarrow_{\tau} P'$

shows $P[x::=b] \Longrightarrow_{\tau} P'[x::=b]$

```

proof(cases x=b)
  assume x = b
  with PTrans show ?thesis by simp
next
  assume x ≠ b
  hence x ≠ P[x::=b] by(simp add: fresh-fact2)
  with PTrans have x ≠ P' by(force intro: freshChain)
  hence P' = P'[x::=b] by(simp add: forget)
  with PTrans show ?thesis by simp
qed

lemma bangChain:
  fixes P :: pi
  and P' :: pi

  assumes PTrans: P || !P ⇒τ P'
  and P'ineq: P' ≠ P || !P

  shows !P ⇒τ P'
using assms
proof(induct rule: tauChainInduct)
  case id
  thus ?case by simp
next
  case(ih P' P'')
  show ?case
  proof(cases P' = P || !P)
    case True
    from ⟨P' ↦τ < P''⟩ ⟨P' = P || !P⟩ have !P ↦τ < P'' by(blast intro: Bang)
    thus ?thesis by auto
  next
    case False
    from ⟨P' ≠ P || !P⟩ have !P ⇒τ P' by(rule ih)
    with ⟨P' ↦τ < P''⟩ show ?thesis by(auto dest: tauActTauChain)
  qed
qed

end

theory Weak-Early-Step-Semantics
  imports Early-Tau-Chain
begin

lemma inputSupportDerivative:
  assumes P ↦a <x> < P'

  shows (supp P') - {x} ⊆ supp P
using assms
apply(nominal-induct rule: inputInduct)

```

```

apply(auto simp add: pi.supp abs-supp supp-atm)
apply(rule ccontr)
apply(simp add: fresh-def[symmetric])
apply(drule-tac fresh-fact1)
apply(rotate-tac 4)
apply assumption
apply(simp add: fresh-def)
apply force
apply(case-tac x # P)
apply(drule-tac fresh-fact1)
apply(rotate-tac 2)
apply assumption
apply(simp add: fresh-def)
apply force
apply(rotate-tac 2)
apply(drule-tac fresh-fact2)
apply(simp add: fresh-def)
by force

```

lemma *outputSupportDerivative:*

```

fixes P :: pi
and a :: name
and b :: name
and P' :: pi

```

assumes $P \mapsto a[b] \prec P'$

shows $(\text{supp } P') \subseteq ((\text{supp } P)::\text{name set})$

using *assms*

by(*nominal-induct rule: outputInduct*) (*auto simp add: pi.supp abs-supp*)

lemma *boundOutputSupportDerivative:*

```

assumes  $P \mapsto a \langle \nu x \rangle \prec P'$ 
and  $x \# P$ 

```

shows $(\text{supp } P') - \{x\} \subseteq \text{supp } P$

using *assms*

by(*nominal-induct rule: boundOutputInduct*) (*auto simp add: pi.supp abs-supp supp-atm*
dest: outputSupportDerivative)

lemma *tauSupportDerivative:*

assumes $P \mapsto \tau \prec P'$

shows $((\text{supp } P')::\text{name set}) \subseteq \text{supp } P$

using *assms*

proof(*nominal-induct rule: tauInduct*)

case(*Tau P*)

thus *?case by*(*force simp add: pi.supp*)

```

next
  case(Match P)
  thus ?case by(force simp add: pi.supp)
next
  case(Mismatch P)
  thus ?case by(force simp add: pi.supp)
next
  case(Sum1 P)
  thus ?case by(force simp add: pi.supp)
next
  case(Sum2 P)
  thus ?case by(force simp add: pi.supp)
next
  case(Par1 P)
  thus ?case by(force simp add: pi.supp)
next
  case(Par2 P)
  thus ?case by(force simp add: pi.supp)
next
  case(Comm1 P a b P' Q Q')
  from  $\langle P \mapsto a \langle b \rangle \prec P' \rangle$  have  $(\text{supp } P') - \{b\} \subseteq \text{supp } P$  by(rule inputSupportDerivative)
  moreover from  $\langle Q \mapsto a[b] \prec Q' \rangle$  have  $(\text{supp } Q')::\text{name set} \subseteq \text{supp } Q$  by(rule outputSupportDerivative)
  moreover from  $\langle Q \mapsto a[b] \prec Q' \rangle$  have  $b \in \text{supp } Q$ 
  by(nominal-induct rule: outputInduct) (auto simp add: pi.supp abs-supp supp-atm)
  ultimately show ?case by(auto simp add: pi.supp)
next
  case(Comm2 P a b P' Q Q')
  from  $\langle P \mapsto a[b] \prec P' \rangle$  have  $(\text{supp } P')::\text{name set} \subseteq \text{supp } P$  by(rule outputSupportDerivative)
  moreover from  $\langle Q \mapsto a \langle b \rangle \prec Q' \rangle$  have  $(\text{supp } Q') - \{b\} \subseteq \text{supp } Q$  by(rule inputSupportDerivative)
  moreover from  $\langle P \mapsto a[b] \prec P' \rangle$  have  $b \in \text{supp } P$ 
  by(nominal-induct rule: outputInduct) (auto simp add: pi.supp abs-supp supp-atm)
  ultimately show ?case by(auto simp add: pi.supp)
next
  case(Close1 P a x P' Q Q')
  thus ?case by(auto dest: inputSupportDerivative boundOutputSupportDerivative simp add: abs-supp pi.supp)
next
  case(Close2 P a x P' Q Q')
  thus ?case by(auto dest: inputSupportDerivative boundOutputSupportDerivative simp add: abs-supp pi.supp)
next
  case(Res P P' x)
  thus ?case by(force simp add: pi.supp abs-supp)
next
  case(Bang P P')

```

thus ?case by(force simp add: pi.supp)
qed

lemma tauChainSupportDerivative:

fixes $P :: pi$
and $P' :: pi$

assumes $P \Rightarrow_{\tau} P'$

shows $((supp P')::name set) \subseteq (supp P)$

using *assms*

by(*induct rule: tauChainInduct*) (*auto dest: tauSupportDerivative*)

definition outputTransition :: $pi \Rightarrow name \Rightarrow name \Rightarrow pi \Rightarrow bool$ ($\langle - \Rightarrow - \rangle \langle \nu - \rangle \prec$
 $\rightarrow [80, 80, 80, 80] 80$)

where $P \Rightarrow_{a \langle \nu x \rangle} \prec P' \equiv \exists P''' P''. P \Rightarrow_{\tau} P''' \wedge P''' \mapsto_{a \langle \nu x \rangle} \prec P'' \wedge P'' \Rightarrow_{\tau} P'$

definition freeTransition :: $pi \Rightarrow freeRes \Rightarrow pi \Rightarrow bool$ ($\langle - \Rightarrow - \rangle \prec \rightarrow [80, 80, 80]$
 80)

where $P \Rightarrow_{\alpha} \prec P' \equiv \exists P''' P''. P \Rightarrow_{\tau} P''' \wedge P''' \mapsto_{\alpha} \prec P'' \wedge P'' \Rightarrow_{\tau} P'$

lemma transitionI:

fixes $P :: pi$
and $P''' :: pi$
and $\alpha :: freeRes$
and $P'' :: pi$
and $P' :: pi$
and $a :: name$
and $x :: name$

shows $\llbracket P \Rightarrow_{\tau} P'''; P''' \mapsto_{\alpha} \prec P''; P'' \Rightarrow_{\tau} P' \rrbracket \Rightarrow P \Rightarrow_{\alpha} \prec P'$

and $\llbracket P \Rightarrow_{\tau} P'''; P''' \mapsto_{a \langle \nu x \rangle} \prec P''; P'' \Rightarrow_{\tau} P' \rrbracket \Rightarrow P \Rightarrow_{a \langle \nu x \rangle} \prec P'$

by(*auto simp add: outputTransition-def freeTransition-def*)

lemma transitionE:

fixes $P :: pi$
and $\alpha :: freeRes$
and $P' :: pi$
and $a :: name$
and $x :: name$

shows $P \Rightarrow_{\alpha} \prec P' \Rightarrow (\exists P'' P'''. P \Rightarrow_{\tau} P'' \wedge P'' \mapsto_{\alpha} \prec P''' \wedge P''' \Rightarrow_{\tau} P')$

and $P \Rightarrow_{a \langle \nu x \rangle} \prec P' \Rightarrow (\exists P'' P'''. P \Rightarrow_{\tau} P'' \wedge P'' \mapsto_{a \langle \nu x \rangle} \prec P''' \wedge P''' \Rightarrow_{\tau} P')$

by(*auto simp add: outputTransition-def freeTransition-def*)

lemma weakTransitionAlpha:

```

fixes  $P :: pi$ 
and  $a :: name$ 
and  $x :: name$ 
and  $P' :: pi$ 
and  $y :: name$ 

assumes  $PTrans: P \Longrightarrow a \langle \nu x \rangle \prec P'$ 
and  $y \# P$ 

shows  $P \Longrightarrow a \langle \nu y \rangle \prec ((x, y) \cdot P')$ 
proof (cases y=x)
  case True
    with  $PTrans$  show ?thesis by simp
  next
    case False
      from  $PTrans$  obtain  $P'' P'''$  where  $PChain: P \Longrightarrow_{\tau} P'''$ 
        and  $P'''Trans: P''' \mapsto a \langle \nu x \rangle \prec P''$ 
        and  $P''Chain: P'' \Longrightarrow_{\tau} P'$ 
        by (force dest: transitionE)
      note  $PChain$ 
      moreover from  $PChain \langle y \# P \rangle$  have  $y \# P'''$  by (rule freshChain)
      with  $P'''Trans$  have  $y \# P''$  using  $\langle y \neq x \rangle$  by (rule freshTransition)
      with  $P'''Trans$  have  $P''' \mapsto a \langle \nu y \rangle \prec ((x, y) \cdot P'')$  by (simp add: alphaBound-
Output name-swap)
      moreover from  $P''Chain$  have  $((x, y) \cdot P'') \Longrightarrow_{\tau} ((x, y) \cdot P')$ 
        by (rule eqvtChainI)
      ultimately show ?thesis by (rule transitionI)
    qed

lemma singleActionChain:
  fixes  $P :: pi$ 
  and  $Rs :: residual$ 

  shows  $P \mapsto a \langle \nu x \rangle \prec P' \Longrightarrow P \Longrightarrow a \langle \nu x \rangle \prec P'$ 
  and  $P \mapsto \alpha \prec P' \Longrightarrow P \Longrightarrow \alpha \prec P'$ 
  proof -
    have  $P \Longrightarrow_{\tau} P$  by simp
    moreover assume  $P \mapsto a \langle \nu x \rangle \prec P'$ 
    moreover have  $P' \Longrightarrow_{\tau} P'$  by simp
    ultimately show  $P \Longrightarrow a \langle \nu x \rangle \prec P'$ 
      by (rule transitionI)
    next
      have  $P \Longrightarrow_{\tau} P$  by simp
      moreover assume  $P \mapsto \alpha \prec P'$ 
      moreover have  $P' \Longrightarrow_{\tau} P'$  by simp
      ultimately show  $P \Longrightarrow \alpha \prec P'$ 
        by (rule transitionI)
    qed

```

lemma *Tau*:
fixes $P :: pi$

shows $\tau.(P) \Longrightarrow \tau \prec P$
proof –
have $\tau.(P) \Longrightarrow_{\tau} \tau.(P)$ **by** *simp*
moreover have $\tau.(P) \mapsto_{\tau} \tau \prec P$ **by**(*rule Early-Semantics.Tau*)
moreover have $P \Longrightarrow_{\tau} P$ **by** *simp*
ultimately show *?thesis* **by**(*rule transitionI*)
qed

lemma *Input*:
fixes $a :: name$
and $x :: name$
and $u :: name$
and $P :: pi$

shows $a\langle x \rangle.P \Longrightarrow a\langle u \rangle \prec P[x::=u]$
proof –
have $a\langle x \rangle.P \Longrightarrow_{\tau} a\langle x \rangle.P$ **by** *simp*
moreover have $a\langle x \rangle.P \mapsto a\langle u \rangle \prec P[x::=u]$ **by**(*rule Early-Semantics.Input*)
moreover have $P[x::=u] \Longrightarrow_{\tau} P[x::=u]$ **by** *simp*
ultimately show *?thesis* **by**(*rule transitionI*)
qed

lemma *Output*:
fixes $a :: name$
and $b :: name$
and $P :: pi$

shows $a\{b\}.P \Longrightarrow a[b] \prec P$
proof –
have $a\{b\}.P \Longrightarrow_{\tau} a\{b\}.P$ **by** *simp*
moreover have $a\{b\}.P \mapsto a[b] \prec P$ **by**(*rule Early-Semantics.Output*)
moreover have $P \Longrightarrow_{\tau} P$ **by** *simp*
ultimately show *?thesis* **by**(*rule transitionI*)
qed

lemma *Match*:
fixes $P :: pi$
and $b :: name$
and $x :: name$
and $a :: name$
and $P' :: pi$
and $\alpha :: freeRes$

shows $P \Longrightarrow b\langle \nu x \rangle \prec P' \Longrightarrow [a\curvearrowright a]P \Longrightarrow b\langle \nu x \rangle \prec P'$
and $P \Longrightarrow \alpha \prec P' \Longrightarrow [a\curvearrowright a]P \Longrightarrow \alpha \prec P'$
proof –

```

assume  $P \Rightarrow b\langle \nu x \rangle \prec P'$ 
then obtain  $P'' P'''$  where  $PChain: P \Rightarrow_{\tau} P'''$ 
                and  $P'''Trans: P''' \mapsto b\langle \nu x \rangle \prec P''$ 
                and  $P''Chain: P'' \Rightarrow_{\tau} P'$ 
  by(force dest: transitionE)
show  $[a \frown a]P \Rightarrow b\langle \nu x \rangle \prec P'$ 
proof(cases  $P = P'''$ )
  case True
    have  $[a \frown a]P \Rightarrow_{\tau} [a \frown a]P$  by simp
    moreover from  $\langle P = P''' \rangle P'''Trans$  have  $[a \frown a]P \mapsto b\langle \nu x \rangle \prec P''$ 
      by(rule-tac Early-Semantics.Match) auto
    ultimately show ?thesis using  $P''Chain$  by(rule transitionI)
  next
  case False
    from  $PChain \langle P \neq P''' \rangle$  have  $[a \frown a]P \Rightarrow_{\tau} P'''$  by(rule matchChain)
    thus ?thesis using  $P'''Trans P''Chain$  by(rule transitionI)
qed
next
assume  $P \Rightarrow \alpha \prec P'$ 
then obtain  $P'' P'''$  where  $PChain: P \Rightarrow_{\tau} P'''$ 
                and  $P'''Trans: P''' \mapsto \alpha \prec P''$ 
                and  $P''Chain: P'' \Rightarrow_{\tau} P'$ 
  by(force dest: transitionE)
show  $[a \frown a]P \Rightarrow \alpha \prec P'$ 
proof(cases  $P = P'''$ )
  case True
    have  $[a \frown a]P \Rightarrow_{\tau} [a \frown a]P$  by simp
    moreover from  $\langle P = P''' \rangle P'''Trans$  have  $[a \frown a]P \mapsto \alpha \prec P''$ 
      by(rule-tac Early-Semantics.Match) auto
    ultimately show ?thesis using  $P''Chain$  by(rule transitionI)
  next
  case False
    from  $PChain \langle P \neq P''' \rangle$  have  $[a \frown a]P \Rightarrow_{\tau} P'''$  by(rule matchChain)
    thus ?thesis using  $P'''Trans P''Chain$  by(rule transitionI)
qed
qed

lemma Mismatch:
  fixes  $P :: pi$ 
  and  $c :: name$ 
  and  $x :: name$ 
  and  $a :: name$ 
  and  $b :: name$ 
  and  $P' :: pi$ 
  and  $\alpha :: freeRes$ 

  shows  $\llbracket P \Rightarrow c\langle \nu x \rangle \prec P'; a \neq b \rrbracket \Rightarrow [a \neq b]P \Rightarrow c\langle \nu x \rangle \prec P'$ 
  and  $\llbracket P \Rightarrow \alpha \prec P'; a \neq b \rrbracket \Rightarrow [a \neq b]P \Rightarrow \alpha \prec P'$ 
proof -

```

```

assume  $P \Rightarrow c\langle \nu x \rangle \prec P'$ 
then obtain  $P'' P'''$  where  $PChain: P \Rightarrow_{\tau} P'''$ 
                and  $P'''Trans: P''' \mapsto c\langle \nu x \rangle \prec P''$ 
                and  $P''Chain: P'' \Rightarrow_{\tau} P'$ 
    by(force dest: transitionE)
assume  $a \neq b$ 
show  $[a \neq b]P \Rightarrow c\langle \nu x \rangle \prec P'$ 
proof(cases  $P = P'''$ )
    case True
        have  $[a \neq b]P \Rightarrow_{\tau} [a \neq b]P$  by simp
        moreover from  $\langle P = P''' \rangle \langle a \neq b \rangle P'''Trans$  have  $[a \neq b]P \mapsto c\langle \nu x \rangle \prec P''$ 
            by(rule-tac Early-Semantics.Mismatch) auto
        ultimately show ?thesis using  $P''Chain$  by(rule transitionI)
    next
        case False
            from  $PChain \langle a \neq b \rangle \langle P \neq P''' \rangle$  have  $[a \neq b]P \Rightarrow_{\tau} P'''$  by(rule mismatchChain)
            thus ?thesis using  $P'''Trans P''Chain$  by(rule transitionI)
    qed
next
assume  $P \Rightarrow \alpha \prec P'$ 
then obtain  $P'' P'''$  where  $PChain: P \Rightarrow_{\tau} P'''$ 
                and  $P'''Trans: P''' \mapsto \alpha \prec P''$ 
                and  $P''Chain: P'' \Rightarrow_{\tau} P'$ 
    by(force dest: transitionE)
assume  $a \neq b$ 
show  $[a \neq b]P \Rightarrow \alpha \prec P'$ 
proof(cases  $P = P'''$ )
    case True
        have  $[a \neq b]P \Rightarrow_{\tau} [a \neq b]P$  by simp
        moreover from  $\langle P = P''' \rangle \langle a \neq b \rangle P'''Trans$  have  $[a \neq b]P \mapsto \alpha \prec P''$ 
            by(rule-tac Early-Semantics.Mismatch) auto
        ultimately show ?thesis using  $P''Chain$  by(rule transitionI)
    next
        case False
            from  $PChain \langle a \neq b \rangle \langle P \neq P''' \rangle$  have  $[a \neq b]P \Rightarrow_{\tau} P'''$  by(rule mismatchChain)
            thus ?thesis using  $P'''Trans P''Chain$  by(rule transitionI)
    qed
qed

lemma Open:
fixes  $P :: pi$ 
and  $a :: name$ 
and  $b :: name$ 
and  $P' :: pi$ 

assumes  $PTrans: P \Rightarrow a[b] \prec P'$ 
and  $a \neq b$ 

shows  $\langle \nu b \rangle P \Rightarrow a\langle \nu b \rangle \prec P'$ 

```

proof –
from $PTrans$ **obtain** $P'' P'''$ **where** $PChain: P \Rightarrow_{\tau} P'''$
and $P'''Trans: P''' \mapsto a[b] \prec P''$
and $P''Chain: P'' \Rightarrow_{\tau} P'$
by(*force dest: transitionE*)
from $PChain$ **have** $\langle \nu b \rangle P \Rightarrow_{\tau} \langle \nu b \rangle P'''$ **by**(*rule ResChain*)
moreover from $P'''Trans \langle a \neq b \rangle$ **have** $\langle \nu b \rangle P''' \mapsto a \langle \nu b \rangle \prec P''$ **by**(*rule Open*)
ultimately show *?thesis* **using** $P''Chain$ **by**(*rule transitionI*)
qed

lemma *Sum1*:

fixes $P :: pi$
and $a :: name$
and $x :: name$
and $P' :: pi$
and $Q :: pi$
and $\alpha :: freeRes$

shows $P \Rightarrow a \langle \nu x \rangle \prec P' \Rightarrow P \oplus Q \Rightarrow a \langle \nu x \rangle \prec P'$
and $P \Rightarrow \alpha \prec P' \Rightarrow P \oplus Q \Rightarrow \alpha \prec P'$

proof –

assume $P \Rightarrow a \langle \nu x \rangle \prec P'$
then obtain $P'' P'''$ **where** $PChain: P \Rightarrow_{\tau} P'''$
and $P'''Trans: P''' \mapsto a \langle \nu x \rangle \prec P''$
and $P''Chain: P'' \Rightarrow_{\tau} P'$

by(*force dest: transitionE*)

show $P \oplus Q \Rightarrow a \langle \nu x \rangle \prec P'$

proof(*cases P = P'''*)

case *True*

have $P \oplus Q \Rightarrow_{\tau} P \oplus Q$ **by** *simp*

moreover from $P'''Trans \langle P = P''' \rangle$ **have** $P \oplus Q \mapsto a \langle \nu x \rangle \prec P''$ **by**(*blast intro: Sum1*)

ultimately show *?thesis* **using** $P''Chain$ **by**(*rule transitionI*)

next

case *False*

from $PChain \langle P \neq P''' \rangle$ **have** $P \oplus Q \Rightarrow_{\tau} P'''$ **by**(*rule sum1Chain*)

thus *?thesis* **using** $P'''Trans P''Chain$ **by**(*rule transitionI*)

qed

next

assume $P \Rightarrow \alpha \prec P'$

then obtain $P'' P'''$ **where** $PChain: P \Rightarrow_{\tau} P'''$
and $P'''Trans: P''' \mapsto \alpha \prec P''$
and $P''Chain: P'' \Rightarrow_{\tau} P'$

by(*force dest: transitionE*)

show $P \oplus Q \Rightarrow \alpha \prec P'$

proof(*cases P = P'''*)

case *True*

have $P \oplus Q \Rightarrow_{\tau} P \oplus Q$ **by** *simp*

```

moreover from  $P'''Trans \langle P = P''' \rangle$  have  $P \oplus Q \mapsto_{\alpha} \prec P''$  by(blast intro: Sum1)
ultimately show ?thesis using  $P''Chain$  by(rule transitionI)
next
  case False
    from  $PChain \langle P \neq P''' \rangle$  have  $P \oplus Q \Longrightarrow_{\tau} P'''$  by(rule sum1Chain)
    thus ?thesis using  $P'''Trans P''Chain$  by(rule transitionI)
  qed
qed

```

lemma *Sum2*:

```

fixes  $Q :: pi$ 
and  $a :: name$ 
and  $x :: name$ 
and  $Q' :: pi$ 
and  $P :: pi$ 
and  $\alpha :: freeRes$ 

shows  $Q \Longrightarrow_{a \langle \nu x \rangle} \prec Q' \Longrightarrow P \oplus Q \Longrightarrow_{a \langle \nu x \rangle} \prec Q'$ 
and  $Q \Longrightarrow_{\alpha} \prec Q' \Longrightarrow P \oplus Q \Longrightarrow_{\alpha} \prec Q'$ 
proof -
  assume  $Q \Longrightarrow_{a \langle \nu x \rangle} \prec Q'$ 
  then obtain  $Q'' Q'''$  where  $QChain: Q \Longrightarrow_{\tau} Q'''$ 
    and  $Q'''Trans: Q''' \mapsto_{a \langle \nu x \rangle} \prec Q''$ 
    and  $Q''Chain: Q'' \Longrightarrow_{\tau} Q'$ 
    by(force dest: transitionE)
  show  $P \oplus Q \Longrightarrow_{a \langle \nu x \rangle} \prec Q'$ 
  proof(cases Q = Q''')
    case True
      have  $P \oplus Q \Longrightarrow_{\tau} P \oplus Q$  by simp
      moreover from  $Q'''Trans \langle Q = Q''' \rangle$  have  $P \oplus Q \mapsto_{a \langle \nu x \rangle} \prec Q''$  by(blast intro: Sum2)
      ultimately show ?thesis using  $Q''Chain$  by(rule transitionI)
    next
      case False
        from  $QChain \langle Q \neq Q''' \rangle$  have  $P \oplus Q \Longrightarrow_{\tau} Q'''$  by(rule sum2Chain)
        thus ?thesis using  $Q'''Trans Q''Chain$  by(rule transitionI)
      qed
    next
      assume  $Q \Longrightarrow_{\alpha} \prec Q'$ 
      then obtain  $Q'' Q'''$  where  $QChain: Q \Longrightarrow_{\tau} Q'''$ 
        and  $Q'''Trans: Q''' \mapsto_{\alpha} \prec Q''$ 
        and  $Q''Chain: Q'' \Longrightarrow_{\tau} Q'$ 
        by(force dest: transitionE)
      show  $P \oplus Q \Longrightarrow_{\alpha} \prec Q'$ 
      proof(cases Q = Q''')
        case True
          have  $P \oplus Q \Longrightarrow_{\tau} P \oplus Q$  by simp
          moreover from  $Q'''Trans \langle Q = Q''' \rangle$  have  $P \oplus Q \mapsto_{\alpha} \prec Q''$  by(blast intro:

```

```

Sum2)
  ultimately show ?thesis using Q''Chain by(rule transitionI)
next
  case False
  from QChain ⟨Q ≠ Q'''⟩ have P ⊕ Q ⇒τ Q''' by(rule sum2Chain)
  thus ?thesis using Q'''Trans Q''Chain by(rule transitionI)
qed
qed

lemma Par1B:
  fixes P :: pi
  and a :: name
  and x :: name
  and P' :: pi
  and Q :: pi

  assumes PTrans: P ⇒a<νx> P'
  and x ‡ Q

  shows P ‖ Q ⇒a<νx> (P' ‖ Q)
proof -
  from PTrans obtain P'' P''' where PChain: P ⇒τ P'''
  and P'''Trans: P''' ↦a<νx> P'
  and P''Chain: P'' ⇒τ P'

  by(blast dest: transitionE)
  from PChain have P ‖ Q ⇒τ P''' ‖ Q by(rule Par1Chain)
  moreover from P'''Trans ⟨x ‡ Q⟩ have P''' ‖ Q ↦a<νx> (P' ‖ Q) by(rule
Early-Semantics.Par1B)
  moreover from P''Chain have P'' ‖ Q ⇒τ P' ‖ Q by(rule Par1Chain)
  ultimately show P ‖ Q ⇒a<νx> (P' ‖ Q) by(rule transitionI)
qed

lemma Par1F:
  fixes P :: pi
  and α :: freeRes
  and P' :: pi
  and Q :: pi

  assumes PTrans: P ⇒α P'

  shows P ‖ Q ⇒α (P' ‖ Q)
proof -
  from PTrans obtain P'' P''' where PChain: P ⇒τ P'''
  and P'''Trans: P''' ↦α P'
  and P''Chain: P'' ⇒τ P'

  by(blast dest: transitionE)
  from PChain have P ‖ Q ⇒τ P''' ‖ Q by(rule Par1Chain)
  moreover from P'''Trans have P''' ‖ Q ↦α (P' ‖ Q) by(rule Early-Semantics.Par1F)
  moreover from P''Chain have P'' ‖ Q ⇒τ P' ‖ Q by(rule Par1Chain)

```

ultimately show *?thesis* by(rule transitionI)
qed

lemma Par2B:

fixes $Q :: pi$
and $a :: name$
and $x :: name$
and $Q' :: pi$
and $P :: pi$

assumes $QTrans: Q \Longrightarrow a<\nu x> \prec Q'$
and $x \# P$

shows $P \parallel Q \Longrightarrow a<\nu x> \prec (P \parallel Q')$

proof -

from $QTrans$ obtain $Q'' Q'''$ where $QChain: Q \Longrightarrow_{\tau} Q'''$
and $Q'''Trans: Q''' \mapsto a<\nu x> \prec Q''$
and $Q''Chain: Q'' \Longrightarrow_{\tau} Q'$

by(blast dest: transitionE)

from $QChain$ have $P \parallel Q \Longrightarrow_{\tau} P \parallel Q'''$ by(rule Par2Chain)

moreover from $Q'''Trans \langle x \# P \rangle$ have $P \parallel Q''' \mapsto a<\nu x> \prec (P \parallel Q'')$ by(rule Early-Semantics.Par2B)

moreover from $Q''Chain$ have $P \parallel Q'' \Longrightarrow_{\tau} P \parallel Q'$ by(rule Par2Chain)

ultimately show $P \parallel Q \Longrightarrow a<\nu x> \prec (P \parallel Q')$ by(rule transitionI)

qed

lemma Par2F:

fixes $Q :: pi$
and $\alpha :: freeRes$
and $Q' :: pi$
and $P :: pi$

assumes $QTrans: Q \Longrightarrow \alpha \prec Q'$

shows $P \parallel Q \Longrightarrow \alpha \prec (P \parallel Q')$

proof -

from $QTrans$ obtain $Q'' Q'''$ where $QChain: Q \Longrightarrow_{\tau} Q'''$
and $Q'''Trans: Q''' \mapsto \alpha \prec Q''$
and $Q''Chain: Q'' \Longrightarrow_{\tau} Q'$

by(blast dest: transitionE)

from $QChain$ have $P \parallel Q \Longrightarrow_{\tau} P \parallel Q'''$ by(rule Par2Chain)

moreover from $Q'''Trans$ have $P \parallel Q''' \mapsto \alpha \prec (P \parallel Q'')$ by(rule Early-Semantics.Par2F)

moreover from $Q''Chain$ have $P \parallel Q'' \Longrightarrow_{\tau} P \parallel Q'$ by(rule Par2Chain)

ultimately show *?thesis* by(rule transitionI)

qed

lemma Comm1:

fixes $P :: pi$
and $a :: name$

```

and  $b :: name$ 
and  $P' :: pi$ 
and  $Q :: pi$ 
and  $Q' :: pi$ 

assumes  $PTrans: P \Rightarrow a < b > \prec P'$ 
and  $QTrans: Q \Rightarrow a[b] \prec Q'$ 

shows  $P \parallel Q \Rightarrow_{\tau} \prec P' \parallel Q'$ 
proof -
from  $PTrans$  obtain  $P'' P'''$  where  $PChain: P \Rightarrow_{\tau} P'''$ 
and  $P'''Trans: P''' \mapsto a < b > \prec P''$ 
and  $P''Chain: P'' \Rightarrow_{\tau} P'$ 
  by(blast dest: transitionE)
from  $QTrans$  obtain  $Q'' Q'''$  where  $QChain: Q \Rightarrow_{\tau} Q'''$ 
and  $Q'''Trans: Q''' \mapsto a[b] \prec Q''$ 
and  $Q''Chain: Q'' \Rightarrow_{\tau} Q'$ 
  by(blast dest: transitionE)

from  $PChain$   $QChain$  have  $P \parallel Q \Rightarrow_{\tau} P''' \parallel Q'''$  by(rule chainPar)
moreover from  $P'''Trans$   $Q'''Trans$  have  $P''' \parallel Q''' \mapsto_{\tau} \prec P'' \parallel Q''$ 
  by(rule Early-Semantics.Comm1)
moreover from  $P''Chain$   $Q''Chain$  have  $P'' \parallel Q'' \Rightarrow_{\tau} P' \parallel Q'$  by(rule chain-
Par)
ultimately show ?thesis by(rule transitionI)
qed

```

lemma *Comm2*:

```

fixes  $P :: pi$ 
and  $a :: name$ 
and  $b :: name$ 
and  $P' :: pi$ 
and  $Q :: pi$ 
and  $Q' :: pi$ 

```

```

assumes  $PTrans: P \Rightarrow a[b] \prec P'$ 
and  $QTrans: Q \Rightarrow a < b > \prec Q'$ 

```

```

shows  $P \parallel Q \Rightarrow_{\tau} \prec P' \parallel Q'$ 

```

proof -

```

from  $PTrans$  obtain  $P'' P'''$  where  $PChain: P \Rightarrow_{\tau} P'''$ 
and  $P'''Trans: P''' \mapsto a[b] \prec P''$ 
and  $P''Chain: P'' \Rightarrow_{\tau} P'$ 
  by(blast dest: transitionE)
from  $QTrans$  obtain  $Q'' Q'''$  where  $QChain: Q \Rightarrow_{\tau} Q'''$ 
and  $Q'''Trans: Q''' \mapsto a < b > \prec Q''$ 
and  $Q''Chain: Q'' \Rightarrow_{\tau} Q'$ 
  by(blast dest: transitionE)

```

from $PChain$ $QChain$ **have** $P \parallel Q \Longrightarrow_{\tau} P''' \parallel Q'''$ **by**(rule $chainPar$)
moreover from $P''''Trans$ $Q''''Trans$ **have** $P'''' \parallel Q'''' \longmapsto_{\tau} \prec P'' \parallel Q''$
by(rule $Early-Semantics.Comm2$)
moreover from $P''Chain$ $Q''Chain$ **have** $P'' \parallel Q'' \Longrightarrow_{\tau} P' \parallel Q'$ **by**(rule $chainPar$)
ultimately show $?thesis$ **by**(rule $transitionI$)
qed

lemma $Close1$:

fixes $P :: pi$
and $a :: name$
and $x :: name$
and $P' :: pi$
and $Q :: pi$
and $Q' :: pi$

assumes $PTrans$: $P \Longrightarrow a \langle x \rangle \prec P'$
and $QTrans$: $Q \Longrightarrow a \langle \nu x \rangle \prec Q'$
and $x \# P$

shows $P \parallel Q \Longrightarrow_{\tau} \prec \langle \nu x \rangle (P' \parallel Q')$

proof –

from $PTrans$ **obtain** $P''' P''$ **where** $PChain$: $P \Longrightarrow_{\tau} P'''$
and $P''''Trans$: $P'''' \longmapsto a \langle x \rangle \prec P''$
and $P''Chain$: $P'' \Longrightarrow_{\tau} P'$

by(blast dest: $transitionE$)

from $QTrans$ **obtain** $Q'' Q'''$ **where** $QChain$: $Q \Longrightarrow_{\tau} Q'''$
and $Q''''Trans$: $Q'''' \longmapsto a \langle \nu x \rangle \prec Q''$
and $Q''Chain$: $Q'' \Longrightarrow_{\tau} Q'$

by(blast dest: $transitionE$)

from $PChain$ $QChain$ **have** $P \parallel Q \Longrightarrow_{\tau} P''' \parallel Q'''$ **by**(rule $chainPar$)
moreover from $PChain$ $\langle x \# P \rangle$ **have** $x \# P''''$ **by**(rule $freshChain$)
with $P''''Trans$ $Q''''Trans$ **have** $P'''' \parallel Q'''' \longmapsto_{\tau} \prec \langle \nu x \rangle (P'' \parallel Q'')$
by(rule $Early-Semantics.Close1$)
moreover from $P''Chain$ $Q''Chain$ **have** $P'' \parallel Q'' \Longrightarrow_{\tau} P' \parallel Q'$ **by**(rule $chainPar$)
hence $\langle \nu x \rangle (P'' \parallel Q'') \Longrightarrow_{\tau} \langle \nu x \rangle (P' \parallel Q')$ **by**(rule $ResChain$)
ultimately show $?thesis$ **by**(rule $transitionI$)
qed

lemma $Close2$:

fixes $P :: pi$
and $a :: name$
and $x :: name$
and $P' :: pi$
and $Q :: pi$
and $Q' :: pi$

assumes $PTrans: P \Longrightarrow a\langle \nu x \rangle \prec P'$
and $QTrans: Q \Longrightarrow a\langle x \rangle \prec Q'$
and $xFreshQ: x \# Q$

shows $P \parallel Q \Longrightarrow_{\tau} \prec \langle \nu x \rangle (P' \parallel Q')$

proof –

from $PTrans$ **obtain** $P'' P'''$ **where** $PChain: P \Longrightarrow_{\tau} P''$
and $P'''Trans: P''' \mapsto a\langle \nu x \rangle \prec P''$
and $P''Chain: P'' \Longrightarrow_{\tau} P'$

by(blast dest: transitionE)

from $QTrans$ **obtain** $Q'' Q'''$ **where** $QChain: Q \Longrightarrow_{\tau} Q''$
and $Q'''Trans: Q''' \mapsto a\langle x \rangle \prec Q''$
and $Q''Chain: Q'' \Longrightarrow_{\tau} Q'$

by(blast dest: transitionE)

from $PChain$ $QChain$ **have** $P \parallel Q \Longrightarrow_{\tau} P'' \parallel Q''$ **by**(rule chainPar)
moreover from $QChain$ $\langle x \# Q \rangle$ **have** $x \# Q'''$ **by**(rule freshChain)

with $P'''Trans$ $Q'''Trans$ **have** $P''' \parallel Q''' \mapsto_{\tau} \prec \langle \nu x \rangle (P'' \parallel Q'')$
by(rule Early-Semantics.Close2)

moreover from $P''Chain$ $Q''Chain$ **have** $P'' \parallel Q'' \Longrightarrow_{\tau} P' \parallel Q'$ **by**(rule chain-Par)

hence $\langle \nu x \rangle (P'' \parallel Q'') \Longrightarrow_{\tau} \langle \nu x \rangle (P' \parallel Q')$ **by**(rule ResChain)

ultimately show ?thesis **by**(rule transitionI)

qed

lemma *ResF*:

fixes $P :: pi$
and $\alpha :: freeRes$
and $P' :: pi$
and $x :: name$

assumes $PTrans: P \Longrightarrow \alpha \prec P'$
and $x \# \alpha$

shows $\langle \nu x \rangle P \Longrightarrow \alpha \prec \langle \nu x \rangle P'$

proof –

from $PTrans$ **obtain** $P'' P'''$ **where** $PChain: P \Longrightarrow_{\tau} P''$
and $P'''Trans: P''' \mapsto \alpha \prec P''$
and $P''Chain: P'' \Longrightarrow_{\tau} P'$

by(blast dest: transitionE)

from $PChain$ **have** $\langle \nu x \rangle P \Longrightarrow_{\tau} \langle \nu x \rangle P''$ **by**(rule ResChain)

moreover from $P'''Trans$ $\langle x \# \alpha \rangle$ **have** $\langle \nu x \rangle P''' \mapsto \alpha \prec \langle \nu x \rangle P''$

by(rule Early-Semantics.ResF)

moreover from $P''Chain$ **have** $\langle \nu x \rangle P'' \Longrightarrow_{\tau} \langle \nu x \rangle P'$ **by**(rule ResChain)

ultimately show ?thesis **by**(rule transitionI)

qed

```

lemma ResB:
  fixes P :: pi
  and a :: name
  and x :: name
  and P' :: pi
  and y :: name

  assumes PTrans: P  $\Rightarrow$  a<math>\nux>  $\prec$  P'
  and y  $\neq$  a
  and y  $\neq$  x

  shows <math>\nuy>P  $\Rightarrow$  a<math>\nux>  $\prec$  (<math>\nuy>P')
proof -
  from PTrans obtain P'' P''' where PChain: P  $\Rightarrow_{\tau}$  P'''
    and P'''Trans: P'''  $\mapsto$  a<math>\nux>  $\prec$  P''
    and P''Chain: P''  $\Rightarrow_{\tau}$  P'
  by(blast dest: transitionE)

  from PChain have <math>\nuy>P  $\Rightarrow_{\tau}$  <math>\nuy>P''' by(rule ResChain)
  moreover from P'''Trans <math>\langley  $\neq$  a> <math>\langley  $\neq$  x> have <math>\nuy>P'''  $\mapsto$  a<math>\nux>  $\prec$ 
    (<math>\nuy>P'')
  by(rule Early-Semantics.ResB)
  moreover from P''Chain have <math>\nuy>P''  $\Rightarrow_{\tau}$  <math>\nuy>P' by(rule ResChain)
  ultimately show ?thesis by(rule transitionI)
qed

lemma Bang:
  fixes P :: pi
  and Rs :: residual

  shows P  $\parallel$  !P  $\Rightarrow$  a<math>\nux>  $\prec$  P'  $\Rightarrow$  !P  $\Rightarrow$  a<math>\nux>  $\prec$  P'
  and P  $\parallel$  !P  $\Rightarrow$   $\alpha$   $\prec$  P'  $\Rightarrow$  !P  $\Rightarrow$   $\alpha$   $\prec$  P'
proof -
  assume PTrans: P  $\parallel$  !P  $\Rightarrow$  a<math>\nux>  $\prec$  P'

  from PTrans obtain P'' P''' where PChain: P  $\parallel$  !P  $\Rightarrow_{\tau}$  P'''
    and P'''Trans: P'''  $\mapsto$  a<math>\nux>  $\prec$  P''
    and P''Chain: P''  $\Rightarrow_{\tau}$  P'
  by(force dest: transitionE)

  show !P  $\Rightarrow$  a<math>\nux>  $\prec$  P'
proof(cases P''' = P  $\parallel$  !P)
  case True
  have !P  $\Rightarrow_{\tau}$  !P by simp
  moreover from P'''Trans <math>\langleP''' = P  $\parallel$  !P> have !P  $\mapsto$  a<math>\nux>  $\prec$  P'' by(blast
intro: Early-Semantics.Bang)
  ultimately show ?thesis using P''Chain by(rule transitionI)

```

```

next
  case False
  from PChain  $\langle P''' \neq P \parallel !P \rangle$  have  $!P \Rightarrow_{\tau} P'''$  by(rule bangChain)
  with P'''Trans P'''Chain show ?thesis by(blast intro: transitionI)
qed
next
fix  $\alpha$  P' P
assume  $P \parallel !P \Rightarrow_{\alpha} \prec P'$ 

then obtain P'' P''' where PChain:  $P \parallel !P \Rightarrow_{\tau} P''$ 
  and P''Trans:  $P'' \mapsto_{\alpha} \prec P'''$ 
  and P'''Chain:  $P''' \Rightarrow_{\tau} P'$ 
  by(force dest: transitionE)

show  $!P \Rightarrow_{\alpha} \prec P'$ 
proof(cases P'' = P \parallel !P)
  assume  $P'' = P \parallel !P$ 
  moreover with P''Trans have  $!P \mapsto_{\alpha} \prec P'''$  by(blast intro: Bang)
  ultimately show ?thesis using PChain P'''Chain by(rule-tac transitionI,
auto)
next
  assume  $P'' \neq P \parallel !P$ 
  with PChain have  $!P \Rightarrow_{\tau} P''$  by(rule bangChain)
  with P''Trans P'''Chain show ?thesis by(blast intro: transitionI)
qed
qed

lemma tauTransitionChain:
  fixes P :: pi
  and P' :: pi

  assumes  $P \Rightarrow_{\tau} \prec P'$ 

  shows  $P \Rightarrow_{\tau} P'$ 
using assms
by(force dest: transitionE tauActTauChain)

lemma chainTransitionAppend:
  fixes P :: pi
  and P' :: pi
  and Rs :: residual
  and a :: name
  and x :: name
  and P'' :: pi
  and  $\alpha$  :: freeRes

  shows  $P \Rightarrow_{a\langle \nu x \rangle} \prec P'' \Rightarrow P'' \Rightarrow_{\tau} P' \Rightarrow P \Rightarrow_{a\langle \nu x \rangle} \prec P'$ 
  and  $P \Rightarrow_{\alpha} \prec P'' \Rightarrow P'' \Rightarrow_{\tau} P' \Rightarrow P \Rightarrow_{\alpha} \prec P'$ 
  and  $P \Rightarrow_{\tau} P'' \Rightarrow P'' \Rightarrow_{a\langle \nu x \rangle} \prec P' \Rightarrow P \Rightarrow_{a\langle \nu x \rangle} \prec P'$ 

```

and $P \Rightarrow_{\tau} P'' \Rightarrow P'' \Rightarrow_{\alpha} \prec P' \Rightarrow P \Rightarrow_{\alpha} \prec P'$
proof –
assume $PTrans: P \Rightarrow a\langle \nu x \rangle \prec P''$
assume $P''Chain: P'' \Rightarrow_{\tau} P'$

from $PTrans$ **obtain** $P''' P''''$ **where** $PChain: P \Rightarrow_{\tau} P''''$
and $P''''Trans: P'''' \mapsto a\langle \nu x \rangle \prec P'''$
and $P'''Chain: P''' \Rightarrow_{\tau} P''$
by(blast dest: transitionE)

from $P'''Chain P''Chain$ **have** $P''' \Rightarrow_{\tau} P'$ **by** auto
with $PChain P''''Trans$ **show** $P \Rightarrow a\langle \nu x \rangle \prec P'$ **by**(rule transitionI)

next
assume $PTrans: P \Rightarrow_{\alpha} \prec P''$
assume $P''Chain: P'' \Rightarrow_{\tau} P'$

from $PTrans$ **obtain** $P''' P''''$ **where** $PChain: P \Rightarrow_{\tau} P''''$
and $P''''Trans: P'''' \mapsto_{\alpha} \prec P'''$
and $P'''Chain: P''' \Rightarrow_{\tau} P''$
by(blast dest: transitionE)

from $P'''Chain P''Chain$ **have** $P''' \Rightarrow_{\tau} P'$ **by** auto
with $PChain P''''Trans$ **show** $P \Rightarrow_{\alpha} \prec P'$ **by**(rule transitionI)

next
assume $PChain: P \Rightarrow_{\tau} P''$
assume $P''Trans: P'' \Rightarrow a\langle \nu x \rangle \prec P'$

from $P''Trans$ **obtain** $P''' P''''$ **where** $P''Chain: P'' \Rightarrow_{\tau} P''''$
and $P''''Trans: P'''' \mapsto a\langle \nu x \rangle \prec P'''$
and $P'''Chain: P''' \Rightarrow_{\tau} P'$
by(blast dest: transitionE)

from $PChain P''Chain$ **have** $P \Rightarrow_{\tau} P''''$ **by** auto
thus $P \Rightarrow a\langle \nu x \rangle \prec P'$ **using** $P''''Trans P'''Chain$ **by**(rule transitionI)

next
assume $PChain: P \Rightarrow_{\tau} P''$
assume $P''Trans: P'' \Rightarrow_{\alpha} \prec P'$

from $P''Trans$ **obtain** $P''' P''''$ **where** $P''Chain: P'' \Rightarrow_{\tau} P''''$
and $P''''Trans: P'''' \mapsto_{\alpha} \prec P'''$
and $P'''Chain: P''' \Rightarrow_{\tau} P'$
by(blast dest: transitionE)

from $PChain P''Chain$ **have** $P \Rightarrow_{\tau} P''''$ **by** auto
thus $P \Rightarrow_{\alpha} \prec P'$ **using** $P''''Trans P'''Chain$ **by**(rule transitionI)

qed

lemma *freshBoundOutputTransition*:
fixes $P :: pi$

```

and  $a :: name$ 
and  $x :: name$ 
and  $P' :: pi$ 
and  $c :: name$ 

assumes  $PTrans: P \Longrightarrow a \langle \nu x \rangle \prec P'$ 
and  $c \# P$ 
and  $c \neq x$ 

shows  $c \# P'$ 
proof –
  from  $PTrans$  obtain  $P'' P'''$  where  $PChain: P \Longrightarrow_{\tau} P'''$ 
    and  $P'''Trans: P''' \longmapsto a \langle \nu x \rangle \prec P''$ 
    and  $P''Chain: P'' \Longrightarrow_{\tau} P'$ 
    by(blast dest: transitionE)

  from  $PChain$   $\langle c \# P \rangle$  have  $c \# P'''$  by(rule freshChain)
  with  $P'''Trans$  have  $c \# P''$  using  $\langle c \neq x \rangle$  by(rule Early-Semantics.freshTransition)
  with  $P''Chain$  show  $c \# P'$  by(rule freshChain)
qed

```

```

lemma freshTauTransition:
  fixes  $P :: pi$ 
  and  $c :: name$ 

  assumes  $PTrans: P \Longrightarrow_{\tau} \prec P'$ 
  and  $c \# P$ 

  shows  $c \# P'$ 
proof –
  from  $PTrans$  have  $P \Longrightarrow_{\tau} P'$  by(rule tauTransitionChain)
  thus ?thesis using  $\langle c \# P \rangle$  by(rule freshChain)
qed

```

```

lemma freshOutputTransition:
  fixes  $P :: pi$ 
  and  $a :: name$ 
  and  $b :: name$ 
  and  $P' :: pi$ 
  and  $c :: name$ 

  assumes  $PTrans: P \Longrightarrow a[b] \prec P'$ 
  and  $c \# P$ 

  shows  $c \# P'$ 
proof –
  from  $PTrans$  obtain  $P'' P'''$  where  $PChain: P \Longrightarrow_{\tau} P'''$ 
    and  $P'''Trans: P''' \longmapsto a[b] \prec P''$ 
    and  $P''Chain: P'' \Longrightarrow_{\tau} P'$ 

```

by(*blast dest: transitionE*)

from $PChain \langle c \# P \rangle$ **have** $c \# P'''$ **by**(*rule freshChain*)
with $P'''Trans$ **have** $c \# P''$ **by**(*rule Early-Semantics.freshTransition*)
with $P''Chain$ **show** $?thesis$ **by**(*rule freshChain*)

qed

lemma $eqvtI[eqvt]$:
fixes $P :: pi$
and $a :: name$
and $x :: name$
and $P' :: pi$
and $p :: name prm$
and $\alpha :: freeRes$

shows $P \Longrightarrow a \langle \nu x \rangle \prec P' \Longrightarrow (p \cdot P) \Longrightarrow (p \cdot a) \langle \nu(p \cdot x) \rangle \prec (p \cdot P')$
and $P \Longrightarrow \alpha \prec P' \Longrightarrow (p \cdot P) \Longrightarrow (p \cdot \alpha) \prec (p \cdot P')$

proof –
assume $P \Longrightarrow a \langle \nu x \rangle \prec P'$
then obtain $P'' P'''$ **where** $PChain: P \Longrightarrow_{\tau} P'''$
and $P'''Trans: P''' \mapsto a \langle \nu x \rangle \prec P''$
and $P''Chain: P'' \Longrightarrow_{\tau} P'$
by(*blast dest: transitionE*)

from $PChain$ **have** $(p \cdot P) \Longrightarrow_{\tau} (p \cdot P''')$ **by**(*rule eqvtChainI*)
moreover from $P'''Trans$ **have** $(p \cdot P''') \mapsto (p \cdot (a \langle \nu x \rangle \prec P''))$
by(*rule TransitionsEarly.eqvt*)
hence $(p \cdot P''') \mapsto (p \cdot a) \langle \nu(p \cdot x) \rangle \prec (p \cdot P'')$
by(*simp add: eqvts*)
moreover from $P''Chain$ **have** $(p \cdot P'') \Longrightarrow_{\tau} (p \cdot P')$ **by**(*rule eqvtChainI*)
ultimately show $(p \cdot P) \Longrightarrow (p \cdot a) \langle \nu(p \cdot x) \rangle \prec (p \cdot P')$
by(*rule transitionI*)

next
assume $P \Longrightarrow \alpha \prec P'$
then obtain $P'' P'''$ **where** $PChain: P \Longrightarrow_{\tau} P'''$
and $P'''Trans: P''' \mapsto \alpha \prec P''$
and $P''Chain: P'' \Longrightarrow_{\tau} P'$
by(*blast dest: transitionE*)

from $PChain$ **have** $(p \cdot P) \Longrightarrow_{\tau} (p \cdot P''')$ **by**(*rule eqvtChainI*)
moreover from $P'''Trans$ **have** $(p \cdot P''') \mapsto (p \cdot (\alpha \prec P''))$
by(*rule TransitionsEarly.eqvt*)
hence $(p \cdot P''') \mapsto (p \cdot \alpha) \prec (p \cdot P'')$
by(*simp add: eqvts*)
moreover from $P''Chain$ **have** $(p \cdot P'') \Longrightarrow_{\tau} (p \cdot P')$ **by**(*rule eqvtChainI*)
ultimately show $(p \cdot P) \Longrightarrow (p \cdot \alpha) \prec (p \cdot P')$
by(*rule transitionI*)

qed

```

lemma freshInputTransition:
  fixes  $P :: pi$ 
  and  $a :: name$ 
  and  $b :: name$ 
  and  $P' :: pi$ 
  and  $c :: name$ 

  assumes  $PTrans: P \Longrightarrow a \langle b \rangle \prec P'$ 
  and  $c \# P$ 
  and  $c \neq b$ 

  shows  $c \# P'$ 
proof -
  from  $PTrans$  obtain  $P'' P'''$  where  $PChain: P \Longrightarrow_{\tau} P''$ 
    and  $P'''Trans: P''' \longmapsto a \langle b \rangle \prec P''$ 
    and  $P''Chain: P'' \Longrightarrow_{\tau} P'$ 
  by(blast dest: transitionE)

  from  $PChain \langle c \# P \rangle$  have  $c \# P'''$  by(rule freshChain)
  with  $P'''Trans$  have  $c \# P''$  using  $\langle c \neq b \rangle$  by(rule Early-Semantics.freshInputTransition)
  with  $P''Chain$  show ?thesis by(rule freshChain)
qed

```

```

lemmas freshTransition = freshBoundOutputTransition freshOutputTransition
  freshInputTransition freshTauTransition

```

end

```

theory Weak-Early-Semantics
  imports Weak-Early-Step-Semantics
begin

```

```

definition weakFreeTransition ::  $pi \Rightarrow freeRes \Rightarrow pi \Rightarrow bool$  ( $\prec \Longrightarrow^{\wedge} - \prec \rightarrow$ ) [80,
  80, 80] 80)
  where  $P \Longrightarrow^{\wedge} \alpha \prec P' \equiv P \Longrightarrow \alpha \prec P' \vee (\alpha = \tau \wedge P = P')$ 

```

```

lemma weakTransitionI:

```

```

  fixes  $P :: pi$ 
  and  $\alpha :: freeRes$ 
  and  $P' :: pi$ 

```

```

  shows  $P \Longrightarrow \alpha \prec P' \Longrightarrow P \Longrightarrow^{\wedge} \alpha \prec P'$ 
  and  $P \Longrightarrow^{\wedge} \tau \prec P$ 

```

```

by(auto simp add: weakFreeTransition-def)

```

```

lemma transitionCases[consumes 1, case-names Step Stay]:

```

```

  fixes  $P :: pi$ 
  and  $\alpha :: freeRes$ 
  and  $P' :: pi$ 

```

```

assumes  $P \Longrightarrow \hat{\alpha} \prec P'$ 
and  $P \Longrightarrow \alpha \prec P' \Longrightarrow F \alpha P'$ 
and  $F (\tau) P$ 

shows  $F \alpha P'$ 
using assms
by(auto simp add: weakFreeTransition-def)

lemma singleActionChain:
  fixes  $P :: pi$ 
  and  $\alpha :: freeRes$ 
  and  $P' :: pi$ 

  assumes  $P \mapsto \alpha \prec P'$ 

  shows  $P \Longrightarrow \hat{\alpha} \prec P'$ 
using assms
by(auto dest: singleActionChain intro: weakTransitionI)

lemma Tau:
  fixes  $P :: pi$ 

  shows  $\tau.(P) \Longrightarrow \hat{\tau} \prec P$ 
by(auto intro: Weak-Early-Step-Semantics.Tau
  simp add: weakFreeTransition-def)

lemma Input:
  fixes  $a :: name$ 
  and  $x :: name$ 
  and  $u :: name$ 
  and  $P :: pi$ 

  shows  $a \langle x \rangle . P \Longrightarrow \hat{a} \langle u \rangle \prec P[x ::= u]$ 
by(auto intro: Weak-Early-Step-Semantics.Input
  simp add: weakFreeTransition-def)

lemma Output:
  fixes  $a :: name$ 
  and  $b :: name$ 
  and  $P :: pi$ 

  shows  $a \{ b \} . P \Longrightarrow \hat{a} [ b ] \prec P$ 
by(auto intro: Weak-Early-Step-Semantics.Output
  simp add: weakFreeTransition-def)

lemma Par1F:
  fixes  $P :: pi$ 
  and  $\alpha :: freeRes$ 

```

```

and P' :: pi
and Q :: pi

assumes P  $\Longrightarrow$   $\hat{\alpha} \prec P'$ 

shows P || Q  $\Longrightarrow$   $\hat{\alpha} \prec (P' || Q)$ 
using assms
by(auto intro: Weak-Early-Step-Semantics.Par1F
    simp add: weakFreeTransition-def residual.inject)

lemma Par2F:
  fixes Q :: pi
  and  $\alpha$  :: freeRes
  and Q' :: pi
  and P :: pi

  assumes QTrans: Q  $\Longrightarrow$   $\hat{\alpha} \prec Q'$ 

  shows P || Q  $\Longrightarrow$   $\hat{\alpha} \prec (P || Q')$ 
using assms
by(auto intro: Weak-Early-Step-Semantics.Par2F
    simp add: weakFreeTransition-def residual.inject)

lemma ResF:
  fixes P :: pi
  and  $\alpha$  :: freeRes
  and P' :: pi
  and x :: name

  assumes P  $\Longrightarrow$   $\hat{\alpha} \prec P'$ 
  and x  $\#$   $\alpha$ 

  shows  $\langle \nu x \rangle P \Longrightarrow \hat{\alpha} \prec \langle \nu x \rangle P'$ 
using assms
by(auto intro: Weak-Early-Step-Semantics.ResF
    simp add: weakFreeTransition-def residual.inject)

lemma Bang:
  fixes P :: pi
  and Rs :: residual

  assumes P || !P  $\Longrightarrow$   $\hat{\alpha} \prec P'$ 
  and P'  $\neq$  P || !P

  shows !P  $\Longrightarrow$   $\hat{\alpha} \prec P'$ 
using assms
by(auto intro: Weak-Early-Step-Semantics.Bang
    simp add: weakFreeTransition-def residual.inject)

```

```

lemma tauTransitionChain[simp]:
  fixes  $P :: pi$ 
  and  $P' :: pi$ 

  shows  $P \Longrightarrow \hat{\tau} \prec P' = P \Longrightarrow_{\tau} P'$ 
apply(auto dest: Weak-Early-Step-Semantics.tauTransitionChain
  simp add: weakFreeTransition-def)
by(erule rtrancl.cases) (auto intro: transitionI)

```

```

lemma tauStepTransitionChain[simp]:
  fixes  $P :: pi$ 
  and  $P' :: pi$ 

  assumes  $P \neq P'$ 

  shows  $P \Longrightarrow_{\tau} \prec P' = P \Longrightarrow_{\tau} P'$ 
using assms
apply(auto dest: Weak-Early-Step-Semantics.tauTransitionChain
  simp add: weakFreeTransition-def)
by(erule rtrancl.cases) (auto intro: transitionI)

```

```

lemma chainTransitionAppend:
  fixes  $P :: pi$ 
  and  $P' :: pi$ 
  and  $Rs :: residual$ 
  and  $a :: name$ 
  and  $x :: name$ 
  and  $P'' :: pi$ 
  and  $\alpha :: freeRes$ 

  shows  $P \Longrightarrow_{\tau} P'' \Longrightarrow P'' \Longrightarrow \hat{\alpha} \prec P' \Longrightarrow P \Longrightarrow \hat{\alpha} \prec P'$ 
  and  $P \Longrightarrow \hat{\alpha} \prec P'' \Longrightarrow P'' \Longrightarrow_{\tau} P' \Longrightarrow P \Longrightarrow \hat{\alpha} \prec P'$ 
by(auto intro: chainTransitionAppend simp add: weakFreeTransition-def dest: Weak-Early-Step-Semantics.tauT)

```

```

lemma freshTauTransition:
  fixes  $P :: pi$ 
  and  $c :: name$ 

  assumes  $P \Longrightarrow \hat{\tau} \prec P'$ 
  and  $c \# P$ 

  shows  $c \# P'$ 
using assms
by(auto intro: Weak-Early-Step-Semantics.freshTauTransition
  simp add: weakFreeTransition-def residual.inject)

```

```

lemma freshOutputTransition:
  fixes  $P :: pi$ 

```

```

and a :: name
and b :: name
and P' :: pi
and c :: name

assumes P  $\Longrightarrow$   $\hat{a}[b] \prec P'$ 
and c # P

shows c # P'
using assms
by(auto intro: Weak-Early-Step-Semantics.freshOutputTransition
    simp add: weakFreeTransition-def residual.inject)

lemma eqvI:
  fixes P :: pi
  and  $\alpha$  :: freeRes
  and P' :: pi
  and p :: name prm

  assumes P  $\Longrightarrow$   $\hat{\alpha} \prec P'$ 

  shows (p · P)  $\Longrightarrow$   $\hat{(p \cdot \alpha)} \prec (p \cdot P')$ 
using assms
by(auto intro: Weak-Early-Step-Semantics.eqvI
    simp add: weakFreeTransition-def residual.inject)

lemma freshInputTransition:
  fixes P :: pi
  and a :: name
  and b :: name
  and P' :: pi
  and c :: name

  assumes P  $\Longrightarrow$   $\hat{a \langle b \rangle} \prec P'$ 
  and c # P
  and c  $\neq$  b

  shows c # P'
using assms
by(auto intro: Weak-Early-Step-Semantics.freshInputTransition
    simp add: weakFreeTransition-def residual.inject)

lemmas freshTransition = freshBoundOutputTransition freshOutputTransition
    freshInputTransition freshTauTransition

end

theory Weak-Early-Sim
  imports Weak-Early-Semantics Strong-Early-Sim-Pres

```

begin

definition *weakSimulation* :: $pi \Rightarrow (pi \times pi) \text{ set} \Rightarrow pi \Rightarrow \text{bool}$ ($\langle - \rightsquigarrow \langle - \rangle - \rangle$ [80, 80, 80] 80)

where $P \rightsquigarrow \langle \text{Rel} \rangle Q \equiv (\forall a \ x \ Q'. \ Q \mapsto a \langle \nu x \rangle \prec Q' \wedge x \# P \longrightarrow (\exists P'. \ P \Longrightarrow a \langle \nu x \rangle \prec P' \wedge (P', Q') \in \text{Rel})) \wedge$
 $(\forall \alpha \ Q'. \ Q \mapsto \alpha \prec Q' \longrightarrow (\exists P'. \ P \Longrightarrow \hat{\alpha} \prec P' \wedge (P', Q') \in \text{Rel}))$

lemma *monotonic*:

fixes $A :: (pi \times pi) \text{ set}$
and $B :: (pi \times pi) \text{ set}$
and $P :: pi$
and $P' :: pi$

assumes $P \rightsquigarrow \langle A \rangle P'$
and $A \subseteq B$

shows $P \rightsquigarrow \langle B \rangle P'$

using *assms*

by(*simp add: weakSimulation-def*) *blast*

lemma *simCasesCont*[*consumes 1, case-names Bound Free*]:

fixes $P :: pi$
and $Q :: pi$
and $\text{Rel} :: (pi \times pi) \text{ set}$
and $C :: 'a::\text{fs-name}$

assumes *Eqvt*: *eqvt Rel*

and *Bound*: $\bigwedge a \ x \ Q'. \ [Q \mapsto a \langle \nu x \rangle \prec Q'; x \# P; x \# Q; x \neq a; x \# C] \Longrightarrow \exists P'. \ P \Longrightarrow a \langle \nu x \rangle \prec P' \wedge (P', Q') \in \text{Rel}$

and *Free*: $\bigwedge \alpha \ Q'. \ Q \mapsto \alpha \prec Q' \Longrightarrow \exists P'. \ P \Longrightarrow \hat{\alpha} \prec P' \wedge (P', Q') \in \text{Rel}$

shows $P \rightsquigarrow \langle \text{Rel} \rangle Q$

proof(*auto simp add: weakSimulation-def*)

fix $a \ x \ Q'$

assume *QTrans*: $Q \mapsto a \langle \nu x \rangle \prec Q'$ **and** $x \# P$

obtain $c::\text{name}$ **where** $c \# P$ **and** $c \# Q$ **and** $c \neq a$ **and** $c \# Q'$ **and** $c \# C$ **and** $c \neq x$

by(*generate-fresh name*) *auto*

from *QTrans* $\langle c \# Q' \rangle$ **have** $Q \mapsto a \langle \nu c \rangle \prec ([(x, c)] \cdot Q')$ **by**(*simp add: alphaBoundOutput*)

then obtain P' **where** *PTrans*: $P \Longrightarrow a \langle \nu c \rangle \prec P'$ **and** *P'RelQ'*: $(P', [(x, c)] \cdot Q') \in \text{Rel}$

using $\langle c \# P \rangle \langle c \# Q \rangle \langle c \neq a \rangle \langle c \# C \rangle$

by(*drule-tac Bound*) *auto*

from *PTrans* $\langle x \# P \rangle \langle c \neq x \rangle$ **have** $P \Longrightarrow a \langle \nu x \rangle \prec ([(x, c)] \cdot P')$

by(*force intro: weakTransitionAlpha simp add: name-swap*)
moreover from *Eqvt P'RelQ'* **have** $[(x, c)] \cdot P', [(x, c)] \cdot [(x, c)] \cdot Q' \in Rel$
by(*rule eqvtRelI*)
hence $[(x, c)] \cdot P', Q' \in Rel$ **by** *simp*
ultimately show $\exists P'. P \Longrightarrow a \langle \nu x \rangle \prec P' \wedge (P', Q') \in Rel$
by *blast*
next
fix $\alpha Q'$
assume $Q \mapsto \alpha \prec Q'$
thus $\exists P'. P \Longrightarrow \hat{\alpha} \prec P' \wedge (P', Q') \in Rel$
by(*rule Free*)
qed

lemma *simCases*[*case-names Bound Free*]:

fixes $P :: pi$
and $Q :: pi$
and $Rel :: (pi \times pi)$ *set*
and $C :: 'a::fs-name$

assumes $\bigwedge Q' a x. \llbracket Q \mapsto a \langle \nu x \rangle \prec Q'; x \# P \rrbracket \Longrightarrow \exists P'. P \Longrightarrow a \langle \nu x \rangle \prec P' \wedge (P', Q') \in Rel$
and $\bigwedge Q' \alpha. Q \mapsto \alpha \prec Q' \Longrightarrow \exists P'. P \Longrightarrow \hat{\alpha} \prec P' \wedge (P', Q') \in Rel$

shows $P \rightsquigarrow \langle Rel \rangle Q$
using *assms*
by(*auto simp add: weakSimulation-def*)

lemma *simE*:

fixes $P :: pi$
and $Rel :: (pi \times pi)$ *set*
and $Q :: pi$
and $a :: name$
and $x :: name$
and $Q' :: pi$

assumes $P \rightsquigarrow \langle Rel \rangle Q$

shows $Q \mapsto a \langle \nu x \rangle \prec Q' \Longrightarrow x \# P \Longrightarrow \exists P'. P \Longrightarrow a \langle \nu x \rangle \prec P' \wedge (P', Q') \in Rel$
and $Q \mapsto \alpha \prec Q' \Longrightarrow \exists P'. P \Longrightarrow \hat{\alpha} \prec P' \wedge (P', Q') \in Rel$
using *assms* **by**(*simp add: weakSimulation-def*)+

lemma *weakSimTauChain*:

fixes $P :: pi$
and $Rel :: (pi \times pi)$ *set*
and $Rel' :: (pi \times pi)$ *set*
and $Q :: pi$
and $Q' :: pi$

assumes $QChain: Q \Longrightarrow_{\tau} Q'$
and $PRelQ: (P, Q) \in Rel$
and $PSimQ: \bigwedge R S. (R, S) \in Rel \Longrightarrow R \rightsquigarrow \langle Rel \rangle S$

shows $\exists P'. P \Longrightarrow_{\tau} P' \wedge (P', Q') \in Rel$

proof –
from $QChain$ **show** $?thesis$
proof(*induct rule: tauChainInduct*)
case id
moreover **have** $P \Longrightarrow_{\tau} P$ **by** $simp$
ultimately **show** $?case$ **using** $PSimQ PRelQ$ **by** $blast$
next
case($ih Q' Q''$)
have $\exists P'. P \Longrightarrow_{\tau} P' \wedge (P', Q') \in Rel$ **by** $fact$
then **obtain** P' **where** $PChain: P \Longrightarrow_{\tau} P'$ **and** $P'Rel'Q': (P', Q') \in Rel$ **by**
 $blast$
from $P'Rel'Q'$ **have** $P' \rightsquigarrow \langle Rel \rangle Q'$ **by**($rule PSimQ$)
moreover **have** $Q'Trans: Q' \mapsto_{\tau} \prec Q''$ **by** $fact$
ultimately **obtain** P'' **where** $P'Trans: P' \Longrightarrow_{\tau} \prec P''$ **and** $P''RelQ'': (P'', Q'') \in Rel$
by($blast dest: simE$)
from $P'Trans$ **have** $P' \Longrightarrow_{\tau} P''$ **by** $simp$
with $PChain$ **have** $P \Longrightarrow_{\tau} P''$ **by** $auto$
with $P''RelQ''$ **show** $?case$ **by** $blast$

qed
qed

lemma $simE2$:
fixes $P :: pi$
and $Rel :: (pi \times pi) set$
and $Q :: pi$
and $a :: name$
and $x :: name$
and $Q' :: pi$

assumes $Sim: \bigwedge R S. (R, S) \in Rel \Longrightarrow R \rightsquigarrow \langle Rel \rangle S$
and $Eqvt: eqvt Rel$
and $PRelQ: (P, Q) \in Rel$

shows $Q \Longrightarrow a \langle \nu x \rangle \prec Q' \Longrightarrow x \# P \Longrightarrow \exists P'. P \Longrightarrow a \langle \nu x \rangle \prec P' \wedge (P', Q') \in Rel$
and $Q \Longrightarrow \hat{\alpha} \prec Q' \Longrightarrow \exists P'. P \Longrightarrow \hat{\alpha} \prec P' \wedge (P', Q') \in Rel$

proof –
assume $QTrans: Q \Longrightarrow a \langle \nu x \rangle \prec Q'$ **and** $x \# P$
from $QTrans$ **obtain** $Q'' Q'''$ **where** $QChain: Q \Longrightarrow_{\tau} Q'''$
and $Q'''Trans: Q''' \mapsto a \langle \nu x \rangle \prec Q''$
and $Q''Chain: Q'' \Longrightarrow_{\tau} Q'$
by($blast dest: transitionE$)

from $QChain\ PRelQ\ Sim$ **obtain** P''' **where** $PChain: P \Longrightarrow_{\tau} P'''$ **and** $P'''RelQ''': (P''', Q''') \in Rel$
by(blast dest: weakSimTauChain)

from $PChain\ \langle x \# P \rangle$ **have** $x \# P'''$ **by**(rule freshChain)

from $P'''RelQ'''$ **have** $P''' \rightsquigarrow_{\langle Rel \rangle} Q'''$ **by**(rule Sim)
with $Q'''Trans\ \langle x \# P''' \rangle$ **obtain** P'' **where** $P'''Trans: P''' \Longrightarrow_{a\langle \nu x \rangle} P''$
and $P''RelQ'': (P'', Q'') \in Rel$
by(blast dest: simE)

from $Q''Chain\ P''RelQ''\ Sim$ **obtain** P' **where** $P''Chain: P'' \Longrightarrow_{\tau} P'$ **and**
 $P'RelQ': (P', Q') \in Rel$
by(blast dest: weakSimTauChain)

from $PChain\ P'''Trans\ P''Chain$ **have** $P \Longrightarrow_{a\langle \nu x \rangle} P'$
by(blast dest: Weak-Early-Step-Semantics.chainTransitionAppend)

with $P'RelQ'$ **show** $\exists P'. P \Longrightarrow_{a\langle \nu x \rangle} P' \wedge (P', Q') \in Rel$ **by** blast

next

assume $Q \Longrightarrow^{\hat{\alpha}} P' \wedge (P', Q') \in Rel$
thus $\exists P'. P \Longrightarrow^{\hat{\alpha}} P' \wedge (P', Q') \in Rel$
proof(induct rule: transitionCases)

case Step

have $Q \Longrightarrow^{\alpha} P' \wedge (P', Q') \in Rel$ **by** fact

then obtain $Q''\ Q'''$ **where** $QChain: Q \Longrightarrow_{\tau} Q''$
and $Q''Trans: Q'' \mapsto^{\alpha} Q'''$
and $Q'''Chain: Q''' \Longrightarrow_{\tau} P'$
by(blast dest: transitionE)

from $QChain\ PRelQ\ Sim$ **have** $\exists P''. P \Longrightarrow_{\tau} P'' \wedge (P'', Q'') \in Rel$
by(rule weakSimTauChain)

then obtain P'' **where** $PChain: P \Longrightarrow_{\tau} P''$ **and** $P''RelQ'': (P'', Q'') \in Rel$

by blast

from $P''RelQ''$ **have** $P'' \rightsquigarrow_{\langle Rel \rangle} Q''$ **by**(rule Sim)

with $Q''Trans$ **obtain** P''' **where** $P''Trans: P'' \Longrightarrow^{\hat{\alpha}} P'''$
and $P'''RelQ''': (P''', Q''') \in Rel$
by(blast dest: simE)

have $\exists P'. P''' \Longrightarrow_{\tau} P' \wedge (P', Q') \in Rel$ **using** $Q'''Chain\ P'''RelQ'''\ Sim$
by(rule weakSimTauChain)

then obtain P' **where** $P'''Chain: P''' \Longrightarrow_{\tau} P'$ **and** $P'RelQ': (P', Q') \in Rel$

by blast

from $PChain\ P''Trans\ P'''Chain$ **have** $P \Longrightarrow^{\hat{\alpha}} P'$
by(blast dest: chainTransitionAppend)

with $P'RelQ'$ **show** ?case **by** blast

next

case Stay

have $P \Longrightarrow^{\hat{\tau}} P$ **by** simp

thus ?case **using** PRelQ **by** blast

qed
qed

lemma *eqvtI*:

fixes $P :: pi$
and $Q :: pi$
and $Rel :: (pi \times pi) \text{ set}$
and $perm :: \text{name prm}$

assumes $PSimQ: P \rightsquigarrow \langle Rel \rangle Q$
and $RelRel': Rel \subseteq Rel'$
and $EqvtRel': eqvt Rel'$

shows $(perm \cdot P) \rightsquigarrow \langle Rel' \rangle (perm \cdot Q)$

proof(*induct rule: simCases*)

case(*Bound* $Q' a x$)

have $xFreshP: x \# perm \cdot P$ **by** *fact*

have $QTrans: (perm \cdot Q) \mapsto a \langle \nu x \rangle \prec Q'$ **by** *fact*

hence $(rev\ perm \cdot (perm \cdot Q)) \mapsto rev\ perm \cdot (a \langle \nu x \rangle \prec Q')$ **by** (*rule eqvts*)

hence $Q \mapsto (rev\ perm \cdot a) \langle \nu (rev\ perm \cdot x) \rangle \prec (rev\ perm \cdot Q')$

by (*simp add: name-rev-per*)

moreover **from** $xFreshP$ **have** $(rev\ perm \cdot x) \# P$ **by** (*simp add: name-fresh-left*)

ultimately obtain P' **where** $PTrans: P \Longrightarrow (rev\ perm \cdot a) \langle \nu (rev\ perm \cdot x) \rangle \prec P'$

and $P'RelQ': (P', rev\ perm \cdot Q') \in Rel$ **using** $PSimQ$

by (*blast dest: simE*)

from $PTrans$ **have** $(perm \cdot P) \Longrightarrow (perm \cdot rev\ perm \cdot a) \langle \nu (perm \cdot rev\ perm \cdot x) \rangle \prec perm \cdot P'$

by (*rule eqvts*)

hence $(perm \cdot P) \Longrightarrow a \langle \nu x \rangle \prec (perm \cdot P')$ **by** (*simp add: name-per-rev*)

moreover **from** $P'RelQ' RelRel'$ **have** $(P', rev\ perm \cdot Q') \in Rel'$ **by** *blast*

with $EqvtRel'$ **have** $(perm \cdot P', perm \cdot (rev\ perm \cdot Q')) \in Rel'$

by (*rule eqvtRelI*)

hence $(perm \cdot P', Q') \in Rel'$ **by** (*simp add: name-per-rev*)

ultimately show *?case* **by** *blast*

next

case(*Free* $Q' \alpha$)

have $QTrans: (perm \cdot Q) \mapsto \alpha \prec Q'$ **by** *fact*

hence $(rev\ perm \cdot (perm \cdot Q)) \mapsto rev\ perm \cdot (\alpha \prec Q')$ **by** (*rule eqvts*)

hence $Q \mapsto (rev\ perm \cdot \alpha) \prec (rev\ perm \cdot Q')$ **by** (*simp add: name-rev-per*)

with $PSimQ$ **obtain** P' **where** $PTrans: P \Longrightarrow (rev\ perm \cdot \alpha) \prec P'$

and $PRel: (P', (rev\ perm \cdot Q')) \in Rel$

by (*blast dest: simE*)

from $PTrans$ **have** $(perm \cdot P) \Longrightarrow (perm \cdot rev\ perm \cdot \alpha) \prec perm \cdot P'$

by (*rule Weak-Early-Semantics.eqvtI*)

hence $L1: (perm \cdot P) \Longrightarrow \hat{\alpha} \prec (perm \cdot P')$ **by** (*simp add: name-per-rev*)
from $PRel\ EqvtRel'\ RelRel'$ **have** $((perm \cdot P'), (perm \cdot (rev\ perm \cdot Q')) \in Rel'$
by (*force intro: eqvtRelI*)
hence $((perm \cdot P'), Q') \in Rel'$ **by** (*simp add: name-per-rev*)
with $L1$ **show** $?case$ **by** *blast*
qed

lemma *reflexive:*

fixes $P :: pi$
and $Rel :: (pi \times pi)\ set$

assumes $Id \subseteq Rel$

shows $P \rightsquigarrow \langle Rel \rangle P$

using *assms*

by (*auto intro: Weak-Early-Step-Semantics.singleActionChain*
simp add: weakSimulation-def weakFreeTransition-def)

lemma *transitive:*

fixes $P :: pi$
and $Q :: pi$
and $R :: pi$
and $Rel :: (pi \times pi)\ set$
and $Rel' :: (pi \times pi)\ set$
and $Rel'' :: (pi \times pi)\ set$

assumes $QSimR: Q \rightsquigarrow \langle Rel' \rangle R$

and $Eqvt: eqvt\ Rel$

and $Eqvt'': eqvt\ Rel''$

and $Trans: Rel\ O\ Rel' \subseteq Rel''$

and $Sim: \bigwedge S\ T. (S, T) \in Rel \Longrightarrow S \rightsquigarrow \langle Rel \rangle T$

and $PRelQ: (P, Q) \in Rel$

shows $P \rightsquigarrow \langle Rel'' \rangle R$

proof –

from $Eqvt''$ **show** $?thesis$

proof (*induct rule: simCasesCont* [**where** $C=Q$])

case (*Bound a x R'*)

have $RTrans: R \mapsto a \langle \nu x \rangle \prec R'$ **by** *fact*

from $\langle x \# Q \rangle QSimR\ RTrans$ **obtain** Q' **where** $QTrans: Q \Longrightarrow a \langle \nu x \rangle \prec Q'$
and $Q'Rel'R': (Q', R') \in Rel'$

by (*blast dest: simE*)

from $Sim\ Eqvt\ PRelQ\ QTrans\ \langle x \# P \rangle$

obtain P' **where** $PTrans: P \Longrightarrow a \langle \nu x \rangle \prec P'$ **and** $P'RelQ': (P', Q') \in Rel$

by (*drule-tac simE2*) *auto*

moreover from $P' \text{Rel} Q' \ Q' \text{Rel}' R'$ **Trans have** $(P', R') \in \text{Rel}''$ **by blast**
ultimately show *?case by blast*
next
case(Free $\alpha \ R'$)
have $R \text{Trans}: R \mapsto \alpha \prec R'$ **by fact**
with $Q \text{Sim} R$ **obtain** Q' **where** $Q \text{Trans}: Q \Longrightarrow \hat{\alpha} \prec Q'$ **and** $Q' \text{Rel}' R': (Q', R') \in \text{Rel}'$
by(blast dest: simE)
from Sim Eqvt $P \text{Rel} Q \ Q \text{Trans}$ **have** $\exists P'. P \Longrightarrow \hat{\alpha} \prec P' \wedge (P', Q') \in \text{Rel}$
by(blast intro: simE2)
then obtain P' **where** $P \text{Trans}: P \Longrightarrow \hat{\alpha} \prec P'$ **and** $P' \text{Rel} Q': (P', Q') \in \text{Rel}$
by blast
from $P' \text{Rel} Q' \ Q' \text{Rel}' R'$ **Trans have** $(P', R') \in \text{Rel}''$ **by blast**
with $P \text{Trans}$ **show** *?case by blast*
qed
qed

lemma strongAppend:

fixes $P \quad :: \text{pi}$
and $Q \quad :: \text{pi}$
and $R \quad :: \text{pi}$
and $\text{Rel} \quad :: (\text{pi} \times \text{pi}) \text{ set}$
and $\text{Rel}' \quad :: (\text{pi} \times \text{pi}) \text{ set}$
and $\text{Rel}'' \quad :: (\text{pi} \times \text{pi}) \text{ set}$

assumes $P \text{Sim} Q: P \rightsquigarrow \langle \text{Rel} \rangle Q$
and $Q \text{Sim} R: Q \rightsquigarrow [\text{Rel}] R$
and $\text{Eqvt}'': \text{eqvt } \text{Rel}''$
and $\text{Trans}: \text{Rel} \circ \text{Rel}' \subseteq \text{Rel}''$

shows $P \rightsquigarrow \langle \text{Rel}'' \rangle R$

proof –

from Eqvt'' **show** *?thesis*

proof(induct rule: simCasesCont[where $C=Q$])

case(Bound $a \ x \ R'$)

have $R \text{Trans}: R \mapsto a \langle \nu x \rangle \prec R'$ **by fact**

from $Q \text{Sim} R \ R \text{Trans} \langle x \ \# \ Q \rangle$ **obtain** Q' **where** $Q \text{Trans}: Q \mapsto a \langle \nu x \rangle \prec Q'$
and $Q' \text{Rel}' R': (Q', R') \in \text{Rel}'$

by(blast dest: Strong-Early-Sim.elim)

with $P \text{Sim} Q \ Q \text{Trans} \langle x \ \# \ P \rangle$ **obtain** P' **where** $P \text{Trans}: P \Longrightarrow a \langle \nu x \rangle \prec P'$
and $P' \text{Rel} Q': (P', Q') \in \text{Rel}$

by(blast dest: simE)

moreover from $P' \text{Rel} Q' \ Q' \text{Rel}' R'$ **Trans have** $(P', R') \in \text{Rel}''$ **by blast**

ultimately show *?case by blast*

next

case(Free $\alpha \ R'$)

have $R \text{Trans}: R \mapsto \alpha \prec R'$ **by fact**

with $Q \text{Sim} R$ **obtain** Q' **where** $Q \text{Trans}: Q \mapsto \alpha \prec Q'$ **and** $Q' \text{Rel}' R': (Q', R') \in \text{Rel}'$

```

∈ Rel'
  by(blast dest: Strong-Early-Sim.elim)
  from PSimQ QTrans obtain P' where PTrans: P ⇒α P' and P'RelQ':
(P', Q') ∈ Rel
  by(blast dest: simE)
  from P'RelQ' Q'RelR' Trans have (P', R') ∈ Rel'' by blast
  with PTrans show ?case by blast
qed
qed

lemma strongSimWeakSim:
  fixes P :: pi
  and Q :: pi
  and Rel :: (pi × pi) set

  assumes PSimQ: P ∼[Rel] Q

  shows P ∼<Rel> Q
proof(induct rule: simCases)
  case(Bound Q' a x)
  have Q ↦a<νx> Q' by fact
  with PSimQ ⟨x ‡ P⟩ obtain P' where PTrans: P ↦a<νx> P' and P'RelQ':
(P', Q') ∈ Rel
  by(blast dest: Strong-Early-Sim.elim)
  from PTrans have P ⇒a<νx> P'
  by(force intro: Weak-Early-Step-Semantics.singleActionChain simp add: weak-
FreeTransition-def)
  with P'RelQ' show ?case by blast
next
  case(Free Q' α)
  have Q ↦α Q' by fact
  with PSimQ obtain P' where PTrans: P ↦α P' and P'RelQ': (P', Q') ∈
Rel
  by(blast dest: Strong-Early-Sim.elim)
  from PTrans have P ⇒α P' by(rule Weak-Early-Semantics.singleActionChain)
  with P'RelQ' show ?case by blast
qed

end

theory Weak-Early-Bisim
  imports Weak-Early-Sim Strong-Early-Bisim
begin

lemma monoAux: A ⊆ B ⇒ P ∼<A> Q ⟶ P ∼<B> Q
by(auto intro: Weak-Early-Sim.monotonic)

coinductive-set weakBisim :: (pi × pi) set
where

```

step: $\llbracket P \rightsquigarrow \langle \text{weakBisim} \rangle Q; (Q, P) \in \text{weakBisim} \rrbracket \implies (P, Q) \in \text{weakBisim}$
monos *monoAux*

abbreviation *weakEarlyBisimJudge* (**infixr** $\langle \approx \rangle$ 65) **where** $P \approx Q \equiv (P, Q) \in \text{weakBisim}$

lemma *weakBisimCoinductAux*[*case-names weakBisim, case-conclusion weakBisim step, consumes 1*]:

assumes $p: (P, Q) \in X$
and step: $\bigwedge P Q. (P, Q) \in X \implies P \rightsquigarrow \langle X \cup \text{weakBisim} \rangle Q \wedge (Q, P) \in X \cup \text{weakBisim}$

shows $P \approx Q$

proof –

have $aux: X \cup \text{weakBisim} = \{(P, Q). (P, Q) \in X \vee P \approx Q\}$ **by** *blast*

from p **show** *?thesis*

by(*coinduct, force dest: step simp add: aux*)

qed

lemma *weakBisimWeakCoinductAux*[*case-names weakBisim, case-conclusion weakBisim step, consumes 1*]:

assumes $p: (P, Q) \in X$
and step: $\bigwedge P Q. (P, Q) \in X \implies P \rightsquigarrow \langle X \rangle Q \wedge (Q, P) \in X$

shows $P \approx Q$

using p

proof(*coinduct rule: weakBisimCoinductAux*)

case (*weakBisim P*)

from *step[OF this]* **show** *?case using Weak-Early-Sim.monotonic* **by** *blast*

qed

lemma *weakBisimCoinduct*[*consumes 1, case-names cSim cSym*]:

fixes $P :: pi$

and $Q :: pi$

and $X :: (pi \times pi)$ *set*

assumes $(P, Q) \in X$

and $\bigwedge R S. (R, S) \in X \implies R \rightsquigarrow \langle X \cup \text{weakBisim} \rangle S$

and $\bigwedge R S. (R, S) \in X \implies (S, R) \in X$

shows $P \approx Q$

using *assms*

by(*coinduct rule: weakBisimCoinductAux*) *auto*

lemma *weakBisimWeakCoinduct*[*consumes 1, case-names cSim cSym*]:

fixes $P :: pi$

and $Q :: pi$

and $X :: (pi \times pi)$ *set*

```

assumes  $(P, Q) \in X$ 
and  $\bigwedge P Q. (P, Q) \in X \implies P \rightsquigarrow\langle X \rangle Q$ 
and  $\bigwedge P Q. (P, Q) \in X \implies (Q, P) \in X$ 

shows  $P \approx Q$ 
using assms
by(coinduct rule: weakBisimWeakCoinductAux) auto

lemma weakBisimE:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 

  assumes  $P \approx Q$ 

  shows  $P \rightsquigarrow\langle weakBisim \rangle Q$ 
  and  $Q \approx P$ 
using assms
by(auto dest: weakBisim.cases)

lemma weakBisimI:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 

  assumes  $P \rightsquigarrow\langle weakBisim \rangle Q$ 
  and  $Q \approx P$ 

  shows  $P \approx Q$ 
using assms
by(auto intro: weakBisim.intros)

lemma eqvt[simp]:
  shows eqvt weakBisim
proof(auto simp add: eqvt-def)
  let  $?X = \{x. \exists P Q (perm::name prm). P \approx Q \wedge x = (perm \cdot P, perm \cdot Q)\}$ 
  fix  $P Q$ 
  fix  $perm::name prm$ 
  assume  $PBiSimQ: P \approx Q$ 

  hence  $(perm \cdot P, perm \cdot Q) \in ?X$  by blast
  moreover have  $\bigwedge P Q perm::name prm. \llbracket P \rightsquigarrow\langle weakBisim \rangle Q \rrbracket \implies (perm \cdot P) \rightsquigarrow\langle ?X \rangle (perm \cdot Q)$ 
  proof –
    fix  $P Q$ 
    fix  $perm::name prm$ 
    assume  $P \rightsquigarrow\langle weakBisim \rangle Q$ 

    moreover have  $weakBisim \subseteq ?X$ 
    proof(auto)

```

```

fix P Q
assume P ≈ Q
moreover have P = ([::name prm) · P and Q = ([::name prm) · Q by
auto
ultimately show ∃ P' Q'. P' ≈ Q' ∧ (∃(perm::name prm). P = perm · P'
∧ Q = perm · Q')
by blast
qed

moreover have eqvt ?X
proof(auto simp add: eqvt-def)
fix P Q
fix perm1::name prm
fix perm2::name prm

assume P ≈ Q
moreover have perm1 · perm2 · P = (perm1 @ perm2) · P by(simp add:
pt2[OF pt-name-inst])
moreover have perm1 · perm2 · Q = (perm1 @ perm2) · Q by(simp add:
pt2[OF pt-name-inst])

ultimately show ∃ P' Q'. P' ≈ Q' ∧ (∃(perm::name prm). perm1 · perm2
· P = perm · P' ∧
                                                                    perm1 · perm2 · Q = perm · Q')
by blast
qed

ultimately show (perm · P) ~<?X> (perm · Q)
by(rule Weak-Early-Sim.eqvtI)
qed

ultimately show (perm · P) ≈ (perm · Q) by(coinduct rule: weakBisimWeak-
CoinductAux, blast dest: weakBisimE)
qed

lemma eqvtI[eqvt]:
fixes P :: pi
and Q :: pi
and perm :: name prm

assumes P ≈ Q

shows (perm · P) ≈ (perm · Q)
using assms
by(rule eqvtRelI[OF eqvt])

lemma strongBisimWeakBisim:
fixes P :: pi
and Q :: pi

```

```

assumes  $P \sim Q$ 

shows  $P \approx Q$ 
proof -
  from  $\langle P \sim Q \rangle$  show ?thesis
  proof (coinduct rule: weakBisimWeakCoinduct)
    case (cSim P Q)
      from  $\langle P \sim Q \rangle$  have  $P \rightsquigarrow[bisim] Q$  by (rule bisimE)
      thus  $P \rightsquigarrow\langle bisim \rangle Q$  by (rule strongSimWeakSim)
    next
      case (cSym P Q)
      thus ?case by (rule bisimE)
  qed
qed

lemma reflexive:
  fixes  $P :: pi$ 

  shows  $P \approx P$ 
proof -
  have  $(P, P) \in Id$  by simp
  thus ?thesis
    by (coinduct rule: weakBisimCoinduct) (auto intro: Weak-Early-Sim.reflexive)
qed

lemma symmetric:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 

  assumes  $P \approx Q$ 

  shows  $Q \approx P$ 
using assms
by (auto dest: weakBisimE)

lemma transitive:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $R :: pi$ 

  assumes  $P \approx Q$ 
  and  $Q \approx R$ 

  shows  $P \approx R$ 
proof -
  let ?X = weakBisim O weakBisim
  from assms have  $(P, R) \in ?X$  by blast
  thus ?thesis

```

```

proof(coinduct rule: weakBisimCoinduct)
  case(cSim P R)
    from  $\langle (P, R) \in ?X \rangle$  obtain  $Q$  where  $P \approx Q$  and  $Q \approx R$  by auto
    from  $\langle Q \approx R \rangle$  have  $Q \rightsquigarrow \langle \text{weakBisim} \rangle R$  by(rule weakBisimE)
    moreover have eqvt ?X by auto
    moreover have  $?X \subseteq ?X$  by simp
    ultimately show  $P \rightsquigarrow \langle (?X \cup \text{weakBisim}) \rangle R$  using weakBisimE(1)  $\langle P \approx$ 
 $Q \rangle$ 
    by(rule-tac Weak-Early-Sim.transitive) auto
  next
    case(cSym P R)
    thus ?case by(auto dest: symetric)
  qed
qed

```

lemma *weakBisimWeakUpto*[*case-names cSim cSym, consumes 1*]:

```

assumes  $p: (P, Q) \in X$ 
and Eqvt: eqvt X
and rSim:  $\bigwedge P Q. (P, Q) \in X \implies P \rightsquigarrow \langle (\text{weakBisim } O X O \text{bisim}) \rangle Q$ 
and rSym:  $\bigwedge P Q. (P, Q) \in X \implies (Q, P) \in X$ 

```

shows $P \approx Q$

proof –

let $?X = \text{weakBisim } O X O \text{weakBisim}$

let $?Y = \text{weakBisim } O X O \text{bisim}$

from *Eqvt eqvt* **have** *eqvt ?X* **by** *blast*

from *Strong-Early-Bisim.eqvt Eqvt eqvt* **have** *eqvt ?Y* **by** *blast*

from $\langle (P, Q) \in X \rangle$ **have** $(P, Q) \in ?X$ **by**(*blast intro: Strong-Early-Bisim.reflexive*
reflexive)

thus *?thesis*

proof(*coinduct rule: weakBisimWeakCoinduct*)

case(*cSim P Q*)

{

fix $P P' Q' Q$

assume $P \approx P'$ **and** $(P', Q') \in X$ **and** $Q' \approx Q$

from $\langle Q' \approx Q \rangle$ **have** $Q' \rightsquigarrow \langle \text{weakBisim} \rangle Q$ **by**(*rule weakBisimE*)

moreover **note** $\langle \text{eqvt } ?Y \rangle \langle \text{eqvt } ?X \rangle$

moreover **have** $?Y O \text{weakBisim} \subseteq ?X$ **by**(*blast dest: strongBisimWeakBisim*
transitive)

moreover {

fix $P Q$

assume $(P, Q) \in ?Y$

then **obtain** $P' Q'$ **where** $P \approx P'$ **and** $(P', Q') \in X$ **and** $Q' \sim Q$ **by** *auto*

from $\langle (P', Q') \in X \rangle$ **have** $P' \rightsquigarrow \langle ?Y \rangle Q'$ **by**(*rule rSim*)

moreover **from** $\langle Q' \sim Q \rangle$ **have** $Q' \rightsquigarrow [\text{bisim}] Q$ **by**(*rule bisimE*)

moreover **note** $\langle \text{eqvt } ?Y \rangle$

moreover **have** $?Y O \text{bisim} \subseteq ?Y$ **by**(*auto dest: Strong-Early-Bisim.transitive*)

ultimately **have** $P' \rightsquigarrow \langle ?Y \rangle Q$ **by**(*rule strongAppend*)

```

moreover note  $\langle P \approx P' \rangle$ 
moreover have  $weakBisim\ O\ ?Y \subseteq ?Y$  by(blast dest: transitive)
ultimately have  $P \rightsquigarrow \langle ?Y \rangle Q$  using  $weakBisimE(1)$  eqvt  $\langle eqvt\ ?Y \rangle$ 
by(rule-tac Weak-Early-Sim.transitive)
}
moreover from  $\langle (P', Q') \in X \rangle$  have  $(P', Q') \in ?Y$  by(blast intro: reflexive
Strong-Early-Bisim.reflexive)
ultimately have  $P' \rightsquigarrow \langle ?X \rangle Q$  by(rule Weak-Early-Sim.transitive)
moreover note  $\langle P \approx P' \rangle$ 
moreover have  $weakBisim\ O\ ?X \subseteq ?X$  by(blast dest: transitive)
ultimately have  $P \rightsquigarrow \langle ?X \rangle Q$  using  $weakBisimE(1)$  eqvt  $\langle eqvt\ ?X \rangle$ 
by(rule-tac Weak-Early-Sim.transitive)
}
with  $\langle (P, Q) \in ?X \rangle$  show  $?case$  by auto
next
case(cSym  $P\ Q$ )
thus  $?case$ 
apply auto
by(blast dest: bisimE rSym weakBisimE)
qed
qed

```

lemma *weakBisimUpto*[*case-names* *cSim* *cSym*, *consumes* 1]:

```

assumes  $p: (P, Q) \in X$ 
and  $Eqvt: eqvt\ X$ 
and  $rSim: \bigwedge R\ S. (R, S) \in X \implies R \rightsquigarrow \langle (weakBisim\ O\ (X \cup weakBisim)\ O\ bisim) \rangle S$ 
and  $rSym: \bigwedge R\ S. (R, S) \in X \implies (S, R) \in X$ 

```

shows $P \approx Q$

proof –

```

from  $p$  have  $(P, Q) \in X \cup weakBisim$  by simp
thus  $?thesis$  using  $Eqvt$ 
apply(coinduct rule: weakBisimWeakUpto)
apply(auto dest: rSim rSym weakBisimE)
apply(rule Weak-Early-Sim.monotonic)
apply(blast dest: weakBisimE)
apply(auto simp add: relcomp-unfold)
by(metis reflexive Strong-Early-Bisim.reflexive transitive)
qed

```

lemma *transitive-coinduct-weak*[*case-names* *cSim* *cSym*, *consumes* 2]:

```

assumes  $p: (P, Q) \in X$ 
and  $Eqvt: eqvt\ X$ 
and  $rSim: \bigwedge P\ Q. (P, Q) \in X \implies P \rightsquigarrow \langle (bisim\ O\ X\ O\ bisim) \rangle Q$ 
and  $rSym: \bigwedge P\ Q. (P, Q) \in X \implies (Q, P) \in bisim\ O\ X\ O\ bisim$ 

```

shows $P \approx Q$

```

proof –
  let ?X = bisim O X O bisim
  from  $\langle (P, Q) \in X \rangle$  have  $(P, Q) \in ?X$  by (blast intro: Strong-Early-Bisim.reflexive)
  thus ?thesis
  proof (coinduct rule: weakBisimWeakCoinduct)
    case (cSim P Q)
    {
      fix P P' Q' Q
      assume PBisimP':  $P \sim P'$ 
      assume P'SimQ':  $P' \rightsquigarrow_{\langle ?X \rangle} Q'$ 
      assume Q'SimQ:  $Q' \rightsquigarrow_{[bisim]} Q$ 

      have  $P \rightsquigarrow_{\langle ?X \rangle} Q$ 
      proof –
        have  $P' \rightsquigarrow_{\langle ?X \rangle} Q$ 
        proof –
          have  $?X \text{ O } bisim \subseteq ?X$  by (blast intro: Strong-Early-Bisim.transitive)
          moreover from Strong-Early-Bisim.eqvt Eqvt have eqvt ?X by blast
          ultimately show ?thesis using P'SimQ' Q'SimQ
          by (rule-tac strongAppend)
        qed
        moreover have eqvt bisim by (rule Strong-Early-Bisim.eqvt)
        moreover from Strong-Early-Bisim.eqvt Eqvt have eqvt ?X by blast
        moreover have  $bisim \text{ O } ?X \subseteq ?X$  by (blast intro: Strong-Early-Bisim.transitive)
        moreover have  $\bigwedge P Q. P \sim Q \implies P \rightsquigarrow_{\langle bisim \rangle} Q$  by (blast dest: Strong-Early-Bisim.bisimE strongSimWeakSim)
        ultimately show ?thesis using PBisimP' by (rule Weak-Early-Sim.transitive)
        qed
      }
    }
  thus ?case using  $\langle (P, Q) \in ?X \rangle$  rSim by (blast dest: Strong-Early-Bisim.bisimE)
  next
  case (cSym P Q)
  {
    fix P P' Q' Q
    assume  $P \sim P'$  and  $(P', Q') \in X$  and  $Q' \sim Q$ 
    from  $\langle (P', Q') \in X \rangle$  have  $(Q', P') \in ?X$  by (rule rSym)
    with  $\langle P \sim P' \rangle$   $\langle Q' \sim Q \rangle$  have  $(Q, P) \in ?X$ 
    apply auto
    apply (drule-tac Strong-Early-Bisim.bisimE(2))
    apply (drule Strong-Early-Bisim.transitive[where Q=P'])
    apply assumption
    apply (drule-tac Strong-Early-Bisim.bisimE(2))
    apply (drule Strong-Early-Bisim.transitive[where Q=Q'])
    apply assumption
    by auto
  }
  thus ?case using  $\langle (P, Q) \in ?X \rangle$  by auto
  qed
qed

```

end

theory *Weak-Early-Step-Sim*
imports *Weak-Early-Sim Strong-Early-Sim*
begin

definition *weakStepSimulation* :: $pi \Rightarrow (pi \times pi)$ set $\Rightarrow pi \Rightarrow bool$ ($\prec \rightsquigarrow \langle \langle - \rangle \rangle \rightarrow$
[80, 80, 80] 80) **where**
 $P \rightsquigarrow \langle \langle Rel \rangle \rangle Q \equiv (\forall Q' a x. Q \mapsto a \langle \nu x \rangle \prec Q' \longrightarrow x \# P \longrightarrow (\exists P'. P \Longrightarrow a \langle \nu x \rangle$
 $\prec P' \wedge (P', Q') \in Rel)) \wedge$
 $(\forall Q' \alpha. Q \mapsto \alpha \prec Q' \longrightarrow (\exists P'. P \Longrightarrow \alpha \prec P' \wedge (P', Q') \in$
Rel))

lemma *monotonic*:

fixes $A :: (pi \times pi)$ set
and $B :: (pi \times pi)$ set
and $P :: pi$
and $P' :: pi$

assumes $P \rightsquigarrow \langle \langle A \rangle \rangle P'$
and $A \subseteq B$

shows $P \rightsquigarrow \langle \langle B \rangle \rangle P'$

using *assms*

by(*simp add: weakStepSimulation-def*) *blast*

lemma *simCasesCont*[*consumes 1, case-names Bound Free*]:

fixes $P :: pi$
and $Q :: pi$
and $Rel :: (pi \times pi)$ set
and $C :: 'a::fs-name$

assumes *Eqvt*: *eqvt Rel*
and *Bound*: $\bigwedge a x Q'. \llbracket x \# C; Q \mapsto a \langle \nu x \rangle \prec Q' \rrbracket \Longrightarrow \exists P'. P \Longrightarrow a \langle \nu x \rangle$
 $\prec P' \wedge (P', Q') \in Rel$
and *Free*: $\bigwedge \alpha Q'. Q \mapsto \alpha \prec Q' \Longrightarrow \exists P'. P \Longrightarrow \alpha \prec P' \wedge (P', Q') \in Rel$

shows $P \rightsquigarrow \langle \langle Rel \rangle \rangle Q$

proof –

from *Free* **show** *?thesis*

proof(*auto simp add: weakStepSimulation-def*)

fix $Q' a x$

assume *xFreshP*: $(x::name) \# P$

assume *Trans*: $Q \mapsto a \langle \nu x \rangle \prec Q'$

have $\exists c::name. c \# (P, Q', x, C)$ **by**(*blast intro: name-exists-fresh*)

then obtain $c::name$ **where** *cFreshP*: $c \# P$ **and** *cFreshQ'*: $c \# Q'$ **and** *cFreshC*:

$c \# C$

and *cineqx*: $c \neq x$

by(*force simp add: fresh-prod*)

from *Trans cFreshQ'* **have** $Q \mapsto a\langle \nu c \rangle \prec ((x, c)] \cdot Q')$ **by**(*simp add: alphaBoundOutput*)

with *cFreshC* **have** $\exists P'. P \Longrightarrow a\langle \nu c \rangle \prec P' \wedge (P', [(x, c)] \cdot Q') \in Rel$
by(*rule Bound*)

then obtain P' **where** $PTrans: P \Longrightarrow a\langle \nu c \rangle \prec P'$ **and** $P'RelQ': (P', [(x, c)] \cdot Q') \in Rel$
by *blast*

from $PTrans \langle x \# P \rangle \langle c \neq x \rangle$ **have** $P \Longrightarrow a\langle \nu x \rangle \prec ((x, c)] \cdot P')$
by(*simp add: weakTransitionAlpha name-swap*)

moreover from *Eqvt P'RelQ'* **have** $((x, c)] \cdot P', [(x, c)] \cdot [(x, c)] \cdot Q') \in Rel$
by(*rule eqvtRelI*)

with $\langle c \neq x \rangle$ **have** $((x, c)] \cdot P', Q') \in Rel$
by *simp*

ultimately show $\exists P'. P \Longrightarrow a\langle \nu x \rangle \prec P' \wedge (P', Q') \in Rel$ **by** *blast*

qed

qed

lemma *simCases[consumes 0, case-names Bound Free]:*

fixes $P :: pi$
and $Q :: pi$
and $Rel :: (pi \times pi)$ *set*
and $C :: 'a::fs-name$

assumes $\bigwedge a x Q'. [Q \mapsto a\langle \nu x \rangle \prec Q'; x \# P] \Longrightarrow \exists P'. P \Longrightarrow a\langle \nu x \rangle \prec P' \wedge (P', Q') \in Rel$
and $\bigwedge \alpha Q'. Q \mapsto \alpha \prec Q' \Longrightarrow \exists P'. P \Longrightarrow \alpha \prec P' \wedge (P', Q') \in Rel$

shows $P \rightsquigarrow \langle \langle Rel \rangle \rangle Q$

using *assms*

by(*auto simp add: weakStepSimulation-def*)

lemma *simE:*

fixes $P :: pi$
and $Rel :: (pi \times pi)$ *set*
and $Q :: pi$
and $a :: name$
and $x :: name$
and $Q' :: pi$

assumes $P \rightsquigarrow \langle \langle Rel \rangle \rangle Q$

shows $Q \mapsto a\langle \nu x \rangle \prec Q' \Longrightarrow x \# P \Longrightarrow \exists P'. P \Longrightarrow a\langle \nu x \rangle \prec P' \wedge (P', Q') \in Rel$

and $Q \mapsto \alpha \prec Q' \Longrightarrow \exists P'. P \Longrightarrow \alpha \prec P' \wedge (P', Q') \in Rel$

using *assms by(simp add: weakStepSimulation-def)+*

lemma *simE2*:

fixes $P :: pi$
and $Rel :: (pi \times pi) \text{ set}$
and $Q :: pi$
and $a :: name$
and $x :: name$
and $Q' :: pi$

assumes $PSimQ: P \rightsquigarrow \langle Rel \rangle Q$
and $Sim: \bigwedge R S. (R, S) \in Rel \implies R \rightsquigarrow \langle Rel \rangle S$
and $Eqvt: eqvt Rel$
and $PRelQ: (P, Q) \in Rel$

shows $Q \implies a \langle \nu x \rangle \prec Q' \implies x \# P \implies \exists P'. P \implies a \langle \nu x \rangle \prec P' \wedge (P', Q') \in Rel$
and $Q \implies \alpha \prec Q' \implies \exists P'. P \implies \alpha \prec P' \wedge (P', Q') \in Rel$

proof –

assume $QTrans: Q \implies a \langle \nu x \rangle \prec Q'$
assume $x \# P$

from $QTrans$ **obtain** $Q'' Q'''$ **where** $QChain: Q \implies_{\tau} Q''$
and $Q''Trans: Q'' \mapsto a \langle \nu x \rangle \prec Q'''$
and $Q'''Chain: Q''' \implies_{\tau} Q'$

by(*blast dest: transitionE*)

from $QChain PRelQ Sim$ **have** $\exists P''. P \implies_{\tau} P'' \wedge (P'', Q'') \in Rel$
by(*rule weakSimTauChain*)

then obtain P'' **where** $PChain: P \implies_{\tau} P''$ **and** $P''RelQ'': (P'', Q'') \in Rel$ **by**
blast

from $PChain \langle x \# P \rangle$ **have** $xFreshP'': x \# P''$ **by**(*rule freshChain*)

from $P''RelQ''$ **have** $P'' \rightsquigarrow \langle Rel \rangle Q''$ **by**(*rule Sim*)

with $Q''Trans xFreshP''$ **obtain** P''' **where** $P'''Trans: P'' \implies a \langle \nu x \rangle \prec P'''$
and $P'''RelQ''': (P''', Q''') \in Rel$

by(*blast dest: Weak-Early-Sim.simE*)

have $\exists P'. P''' \implies_{\tau} P' \wedge (P', Q') \in Rel$ **using** $Q'''Chain P'''RelQ''' Sim$
by(*rule weakSimTauChain*)

then obtain P' **where** $P'''Chain: P''' \implies_{\tau} P'$ **and** $P'RelQ': (P', Q') \in Rel$ **by**
blast

from $PChain P''Trans P'''Chain$ **have** $P \implies a \langle \nu x \rangle \prec P'$
by(*blast dest: Weak-Early-Step-Semantics.chainTransitionAppend*)

with $P'RelQ'$ **show** $\exists P'. P \implies a \langle \nu x \rangle \prec P' \wedge (P', Q') \in Rel$
by *blast*

next

assume $Q \implies \alpha \prec Q'$

then obtain $Q'' Q'''$ **where** $QChain: Q \implies_{\tau} Q''$

and $Q''Trans: Q'' \mapsto_{\alpha} \prec Q'''$
and $Q'''Chain: Q''' \implies_{\tau} Q'$
by(blast dest: transitionE)
from $QChain\ Q''Trans\ Q'''Chain$ **show** $\exists P'. P \implies_{\alpha} \prec P' \wedge (P', Q') \in Rel$
proof(induct arbitrary: $\alpha\ Q''' \prec Q'$ rule: tauChainInduct)
case id
from $PSimQ\ \langle Q \mapsto_{\alpha} \prec Q'' \rangle$ **have** $\exists P'. P \implies_{\alpha} \prec P' \wedge (P', Q''') \in Rel$
by(blast dest: simE)
then obtain P''' **where** $PTrans: P \implies_{\alpha} \prec P'''$ **and** $P'RelQ''': (P''', Q''') \in Rel$
Rel
by blast

have $\exists P'. P''' \implies_{\tau} P' \wedge (P', Q') \in Rel$ **using** $\langle Q''' \implies_{\tau} Q' \rangle\ P'RelQ''' Sim$
by(rule Weak-Early-Sim.weakSimTauChain)
then obtain P' **where** $P'''Chain: P''' \implies_{\tau} P'$ **and** $P'RelQ': (P', Q') \in Rel$
by blast

from $P'''Chain\ PTrans$ **have** $P \implies_{\alpha} \prec P'$
by(blast dest: Weak-Early-Step-Semantics.chainTransitionAppend)

with $P'RelQ'$ **show** ?case **by** blast
next
case(ih $Q''''\ Q''\ \alpha\ Q''' \prec Q'$)
have $Q'' \implies_{\tau} Q''$ **by** simp
with $\langle Q'''' \mapsto_{\tau} \prec Q'' \rangle$ **obtain** P'' **where** $PChain: P \implies_{\tau} \prec P''$ **and**
 $P''RelQ'': (P'', Q'') \in Rel$
by(drule-tac ih) auto

from $P''RelQ''$ **have** $P'' \rightsquigarrow_{\langle Rel \rangle} Q''$ **by**(rule Sim)
hence $\exists P'''. P'' \implies_{\alpha} \prec P''' \wedge (P''', Q''') \in Rel$ **using** $\langle Q'' \mapsto_{\alpha} \prec Q''' \rangle$
by(rule Weak-Early-Sim.simE)
then obtain P''' **where** $P''Trans: P'' \implies_{\alpha} \prec P'''$
and $P'''RelQ''': (P''', Q''') \in Rel$

by blast
from $\langle Q''' \implies_{\tau} Q' \rangle\ P'''RelQ''' Sim$ **have** $\exists P'. P''' \implies_{\tau} P' \wedge (P', Q') \in Rel$
by(rule Weak-Early-Sim.weakSimTauChain)
then obtain P' **where** $P'''Chain: P''' \implies_{\tau} P'$
and $P'RelQ': (P', Q') \in Rel$

by blast
from $PChain\ P''Trans$ **have** $P \implies_{\alpha} \prec P'''$
apply(auto simp add: freeTransition-def weakFreeTransition-def)
apply(drule tauActTauChain, auto)
by(rule-tac $x=P'''aa$ in exI) auto
hence $P \implies_{\alpha} \prec P'$ **using** $P'''Chain$
by(rule Weak-Early-Step-Semantics.chainTransitionAppend)
with $P'RelQ'$ **show** ?case **by** blast
qed
qed

lemma *eqvtI*:
fixes $P :: pi$
and $Q :: pi$
and $Rel :: (pi \times pi)$ *set*
and $perm :: name$ *prm*

assumes $PSimQ: P \rightsquigarrow \langle Rel \rangle Q$
and $RelRel': Rel \subseteq Rel'$
and $EqvtRel': eqvt Rel'$

shows $(perm \cdot P) \rightsquigarrow \langle Rel' \rangle (perm \cdot Q)$

proof(*induct rule: simCases*)
case(*Bound a x Q'*)
have $xFreshP: x \# perm \cdot P$ **by** *fact*
have $QTrans: (perm \cdot Q) \mapsto a \langle \nu x \rangle \prec Q'$ **by** *fact*

hence $(rev\ perm \cdot (perm \cdot Q)) \mapsto rev\ perm \cdot (a \langle \nu x \rangle \prec Q')$ **by**(*rule eqvt*)
hence $Q \mapsto (rev\ perm \cdot a) \langle \nu(rev\ perm \cdot x) \rangle \prec (rev\ perm \cdot Q')$
by(*simp add: name-rev-per*)
moreover from $xFreshP$ **have** $(rev\ perm \cdot x) \# P$ **by**(*simp add: name-fresh-left*)
ultimately obtain P' **where** $PTrans: P \Longrightarrow (rev\ perm \cdot a) \langle \nu(rev\ perm \cdot x) \rangle \prec P'$
and $P'RelQ': (P', rev\ perm \cdot Q') \in Rel$ **using** $PSimQ$
by(*blast dest: simE*)

from $PTrans$ **have** $(perm \cdot P) \Longrightarrow (perm \cdot rev\ perm \cdot a) \langle \nu(perm \cdot rev\ perm \cdot x) \rangle \prec perm \cdot P'$
by(*rule Weak-Early-Step-Semantics.eqvtI*)
hence $L1: (perm \cdot P) \Longrightarrow a \langle \nu x \rangle \prec (perm \cdot P')$ **by**(*simp add: name-per-rev*)
from $P'RelQ' RelRel'$ **have** $(P', rev\ perm \cdot Q') \in Rel'$ **by** *blast*
with $EqvtRel'$ **have** $(perm \cdot P', perm \cdot (rev\ perm \cdot Q')) \in Rel'$
by(*rule eqvtRelI*)
hence $(perm \cdot P', Q') \in Rel'$ **by**(*simp add: name-per-rev*)
with $L1$ **show** *?case* **by** *blast*

next
case(*Free α Q'*)
have $QTrans: (perm \cdot Q) \mapsto \alpha \prec Q'$ **by** *fact*

hence $(rev\ perm \cdot (perm \cdot Q)) \mapsto rev\ perm \cdot (\alpha \prec Q')$ **by**(*rule eqvts*)
hence $Q \mapsto (rev\ perm \cdot \alpha) \prec (rev\ perm \cdot Q')$ **by**(*simp add: name-rev-per*)
with $PSimQ$ **obtain** P' **where** $PTrans: P \Longrightarrow (rev\ perm \cdot \alpha) \prec P'$
and $PRel: (P', (rev\ perm \cdot Q')) \in Rel$
by(*blast dest: simE*)

from $PTrans$ **have** $(perm \cdot P) \Longrightarrow (perm \cdot rev\ perm \cdot \alpha) \prec perm \cdot P'$
by(*rule Weak-Early-Step-Semantics.eqvtI*)
hence $L1: (perm \cdot P) \Longrightarrow \alpha \prec (perm \cdot P')$ **by**(*simp add: name-per-rev*)
from $PRel EqvtRel' RelRel'$ **have** $((perm \cdot P'), (perm \cdot (rev\ perm \cdot Q'))) \in Rel'$
by(*force intro: eqvtRelI*)

hence $((perm \cdot P'), Q') \in Rel'$ **by** $(simp \text{ add: name-per-rev})$
with $L1$ **show** $?case$ **by** $blast$
qed

lemma *reflexive*:

fixes $P :: pi$
and $Rel :: (pi \times pi)$ *set*

assumes $Id \subseteq Rel$

shows $P \rightsquigarrow \langle Rel \rangle P$

using *assms*

by $(auto \text{ intro: Weak-Early-Step-Semantics.singleActionChain}$
 $simp \text{ add: weakStepSimulation-def weakFreeTransition-def})$

lemma *transitive*:

fixes $P :: pi$
and $Q :: pi$
and $R :: pi$
and $Rel :: (pi \times pi)$ *set*
and $Rel' :: (pi \times pi)$ *set*
and $Rel'' :: (pi \times pi)$ *set*

assumes $PSimQ: P \rightsquigarrow \langle Rel \rangle Q$

and $QSimR: Q \rightsquigarrow \langle Rel' \rangle R$

and $Eqvt: eqvt \ Rel$

and $Eqvt'': eqvt \ Rel''$

and $Trans: Rel \ O \ Rel' \subseteq Rel''$

and $Sim: \bigwedge S \ T. (S, T) \in Rel \implies S \rightsquigarrow \langle Rel \rangle T$

and $PRelQ: (P, Q) \in Rel$

shows $P \rightsquigarrow \langle Rel'' \rangle R$

proof –

from $Eqvt''$ **show** $?thesis$

proof $(induct \text{ rule: simCasesCont[of - (P, Q)])$

case $(Bound \ a \ x \ R')$

have $x \# (P, Q)$ **by** *fact*

hence $xFreshP: x \# P$ **and** $xFreshQ: x \# Q$ **by** $(simp \text{ add: fresh-prod})+$

have $RTrans: R \mapsto a \langle \nu x \rangle \prec R'$ **by** *fact*

from $xFreshQ \ QSimR \ RTrans$ **obtain** Q' **where** $QTrans: Q \implies a \langle \nu x \rangle \prec Q'$

and $Q'Rel'R': (Q', R') \in Rel'$

by $(blast \text{ dest: simE})$

with $PSimQ \ Sim \ Eqvt \ PRelQ \ QTrans \ xFreshP$ **have** $\exists P'. P \implies a \langle \nu x \rangle \prec P'$
 $\wedge (P', Q') \in Rel$

by $(blast \text{ intro: simE2})$

then obtain P' **where** $PTrans: P \implies a \langle \nu x \rangle \prec P'$ **and** $P'RelQ': (P', Q') \in$

Rel **by** *blast*
moreover from $P' \text{Rel} Q' \ Q' \text{Rel}' R'$ *Trans* **have** $(P', R') \in \text{Rel}''$ **by** *blast*
ultimately show *?case* **by** *blast*
next
case(*Free* $\alpha \ R'$)
have $R \text{Trans}: R \mapsto \alpha \prec R'$ **by** *fact*
with $Q \text{Sim} R$ **obtain** Q' **where** $Q \text{Trans}: Q \implies \alpha \prec Q'$ **and** $Q' \text{Rel}' R': (Q', R') \in \text{Rel}'$
by(*blast dest: simE*)
from $P \text{Sim} Q \ \text{Sim} \ \text{Eqvt} \ P \text{Rel} Q \ Q \text{Trans}$ **have** $\exists P'. P \implies \alpha \prec P' \wedge (P', Q') \in \text{Rel}$
by(*blast intro: simE2*)
then obtain P' **where** $P \text{Trans}: P \implies \alpha \prec P'$ **and** $P' \text{Rel} Q': (P', Q') \in \text{Rel}$
by *blast*
from $P' \text{Rel} Q' \ Q' \text{Rel}' R'$ *Trans* **have** $(P', R') \in \text{Rel}''$ **by** *blast*
with $P \text{Trans}$ **show** *?case* **by** *blast*
qed
qed

lemma *strongSimWeakSim*:
fixes $P \ :: \ pi$
and $Q \ :: \ pi$
and $\text{Rel} \ :: \ (pi \times pi) \ \text{set}$

assumes $P \text{Sim} Q: P \rightsquigarrow[\text{Rel}] Q$

shows $P \rightsquigarrow\langle\langle \text{Rel} \rangle\rangle Q$

proof(*induct rule: simCases*)
case(*Bound* $a \ x \ Q'$)
have $Q \mapsto a \langle \nu x \rangle \prec Q'$ **and** $x \# P$ **by** *fact+*
with $P \text{Sim} Q$ **obtain** P' **where** $P \text{Trans}: P \mapsto a \langle \nu x \rangle \prec P'$ **and** $P' \text{Rel} Q': (P', Q') \in \text{Rel}$
by(*blast dest: Strong-Early-Sim.elim*)
from $P \text{Trans}$ **have** $P \implies a \langle \nu x \rangle \prec P'$
by(*force intro: Weak-Early-Step-Semantics.singleActionChain simp add: weak-FreeTransition-def*)
with $P' \text{Rel} Q'$ **show** *?case* **by** *blast*
next
case(*Free* $\alpha \ Q'$)
have $Q \mapsto \alpha \prec Q'$ **by** *fact*
with $P \text{Sim} Q$ **obtain** P' **where** $P \text{Trans}: P \mapsto \alpha \prec P'$ **and** $P' \text{Rel} Q': (P', Q') \in \text{Rel}$
by(*blast dest: Strong-Early-Sim.elim*)
from $P \text{Trans}$ **have** $P \implies \alpha \prec P'$ **by**(*rule Weak-Early-Step-Semantics.singleActionChain*)
with $P' \text{Rel} Q'$ **show** *?case* **by** *blast*
qed

lemma *weakSimWeakEqSim*:
fixes $P \ :: \ pi$

```

and  $Q :: pi$ 
and  $Rel :: (pi \times pi) set$ 

assumes  $P \rightsquigarrow \langle Rel \rangle Q$ 

shows  $P \rightsquigarrow \langle Rel \rangle Q$ 
using assms
by(force simp add: weakStepSimulation-def Weak-Early-Sim.weakSimulation-def weak-FreeTransition-def)

end

theory Weak-Early-Cong
imports Weak-Early-Bisim Weak-Early-Step-Sim Strong-Early-Bisim
begin

definition weakCongruence ::  $pi \Rightarrow pi \Rightarrow bool$  (infixr  $\simeq$  65)
where  $P \simeq Q \equiv P \rightsquigarrow \langle weakBisim \rangle Q \wedge Q \rightsquigarrow \langle weakBisim \rangle P$ 

lemma weakCongISym[consumes 1, case-names cSym cSim]:
fixes  $P :: pi$ 
and  $Q :: pi$ 

assumes  $Prop P Q$ 
and  $\bigwedge R S. Prop R S \implies Prop S R$ 
and  $\bigwedge R S. Prop R S \implies (F R) \rightsquigarrow \langle weakBisim \rangle (F S)$ 

shows  $F P \simeq F Q$ 
using assms
by(auto simp add: weakCongruence-def)

lemma weakCongISym2[consumes 1, case-names cSim]:
fixes  $P :: pi$ 
and  $Q :: pi$ 

assumes  $P \simeq Q$ 
and  $\bigwedge R S. R \simeq S \implies (F R) \rightsquigarrow \langle weakBisim \rangle (F S)$ 

shows  $F P \simeq F Q$ 
using assms
by(auto simp add: weakCongruence-def)

lemma weakCongEE:
fixes  $P :: pi$ 
and  $Q :: pi$ 
and  $s :: (name \times name) list$ 

assumes  $P \simeq Q$ 

```

```

  shows  $P \rightsquigarrow\langle\langle \text{weakBisim} \rangle\rangle Q$ 
  and  $Q \rightsquigarrow\langle\langle \text{weakBisim} \rangle\rangle P$ 
using assms
by(auto simp add: weakCongruence-def)

lemma weakCongI:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 

  assumes  $P \rightsquigarrow\langle\langle \text{weakBisim} \rangle\rangle Q$ 
  and  $Q \rightsquigarrow\langle\langle \text{weakBisim} \rangle\rangle P$ 

  shows  $P \simeq Q$ 
using assms
by(auto simp add: weakCongruence-def)

lemma eqvtI[eqvt]:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $p :: name prm$ 

  assumes  $P \simeq Q$ 

  shows  $(p \cdot P) \simeq (p \cdot Q)$ 
using assms
by(auto simp add: weakCongruence-def intro: eqvtI)

lemma strongBisimWeakCong:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 

  assumes  $P \sim Q$ 

  shows  $P \simeq Q$ 
proof -
  have  $\bigwedge P Q. P \rightsquigarrow[bisim] Q \implies P \rightsquigarrow\langle\langle \text{weakBisim} \rangle\rangle Q$ 
  proof -
    fix  $P Q$ 
    assume  $P \rightsquigarrow[bisim] Q$ 
    hence  $P \rightsquigarrow\langle\langle \text{bisim} \rangle\rangle Q$  by(rule Weak-Early-Step-Sim.strongSimWeakSim)
    moreover have  $bisim \subseteq \text{weakBisim}$ 
    by(auto intro: strongBisimWeakBisim)
    ultimately show  $P \rightsquigarrow\langle\langle \text{weakBisim} \rangle\rangle Q$  by(rule Weak-Early-Step-Sim.monotonic)
  qed
  with assms show ?thesis
  by(blast intro: weakCongI dest: Strong-Early-Bisim.bisimE)
qed

lemma congruenceWeakBisim:

```

```

fixes  $P :: pi$ 
and  $Q :: pi$ 

assumes  $P \simeq Q$ 

shows  $P \approx Q$ 
using assms
proof –
  let  $?X = \{(P, Q) \mid P \ Q. P \simeq Q\}$ 
  from assms have  $(P, Q) \in ?X$  by simp
  thus ?thesis
  proof(induct rule: weakBisimCoinduct)
    case(cSim P Q)
    {
      fix  $P \ Q$ 
      assume  $P \simeq Q$ 
      hence  $P \rightsquigarrow\langle\langle weakBisim \rangle\rangle Q$  by(simp add: weakCongruence-def)
      hence  $P \rightsquigarrow\langle\langle ?X \cup weakBisim \rangle\rangle Q$  by(rule-tac Weak-Early-Step-Sim.monotonic)
    }
  auto
  hence  $P \rightsquigarrow\langle\langle ?X \cup weakBisim \rangle\rangle Q$  by(rule weakSimWeakEqSim)
  }
  with  $\langle(P, Q) \in ?X\rangle$  show ?case by auto
next
  case(cSym P Q)
  thus ?case by(auto simp add: weakCongruence-def)
qed
qed

lemma reflexive:
  fixes  $P :: pi$ 

  shows  $P \simeq P$ 
proof –
  from Weak-Early-Bisim.reflexive have  $\bigwedge P. P \rightsquigarrow\langle\langle weakBisim \rangle\rangle P$ 
  by(blast intro: Weak-Early-Step-Sim.reflexive)
  thus ?thesis
  by(force simp add: substClosed-def weakCongruence-def)
qed

lemma symetric:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 

  assumes  $P \simeq Q$ 

  shows  $Q \simeq P$ 
using assms
by(force simp add: substClosed-def weakCongruence-def)

```

```

lemma transitive:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $R :: pi$ 

  assumes  $P \simeq Q$ 
  and  $Q \simeq R$ 

  shows  $P \simeq R$ 
proof –
  have Goal:  $\bigwedge P Q R. \llbracket P \rightsquigarrow\langle\langle weakBisim \rangle\rangle Q; Q \rightsquigarrow\langle\langle weakBisim \rangle\rangle R; P \approx Q \rrbracket \implies$ 
 $P \rightsquigarrow\langle\langle weakBisim \rangle\rangle R$ 
  using Weak-Early-Bisim.eqvt Weak-Early-Bisim.weakBisimE Weak-Early-Bisim.transitive
  by(blast intro: Weak-Early-Step-Sim.transitive)
  from assms show ?thesis
  apply(simp add: weakCongruence-def) using assms
  by(blast intro: Goal dest: congruenceWeakBisim symmetric)
qed

end

theory Weak-Early-Bisim-Subst
  imports Weak-Early-Bisim Strong-Early-Bisim-Subst
begin

consts weakBisimSubst ::  $(pi \times pi)$  set
abbreviation weakEarlyBisimSubstJudge (infixr  $\langle \approx^s \rangle$  65) where  $P \approx^s Q \equiv (P,$ 
 $Q) \in (substClosed\ weakBisim)$ 

lemma congBisim:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 

  assumes  $P \approx^s Q$ 

  shows  $P \approx Q$ 
using assms substClosedSubset
by blast

lemma strongBisimWeakBisim:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 

  assumes  $P \sim^s Q$ 

  shows  $P \approx^s Q$ 
using assms
by(auto simp add: substClosed-def intro: strongBisimWeakBisim)

```

```

lemma eqvt:
  shows eqvt (substClosed weakBisim)
by(rule eqvtSubstClosed[OF Weak-Early-Bisim.eqvt])

lemma eqvtI[eqvt]:
  fixes P :: pi
  and Q :: pi
  and perm :: name prm

  assumes  $P \approx^s Q$ 

  shows  $(perm \cdot P) \approx^s (perm \cdot Q)$ 
using assms
by(rule eqvtRelI[OF eqvt])

lemma reflexive:
  fixes P :: pi

  shows  $P \approx^s P$ 
by(force simp add: substClosed-def intro: Weak-Early-Bisim.reflexive)

lemma symetric:
  fixes P :: pi
  and Q :: pi

  assumes  $P \approx^s Q$ 

  shows  $Q \approx^s P$ 
using assms
by(force simp add: substClosed-def intro: Weak-Early-Bisim.symetric)

lemma transitive:
  fixes P :: pi
  and Q :: pi
  and R :: pi

  assumes  $P \approx^s Q$ 
  and  $Q \approx^s R$ 

  shows  $P \approx^s R$ 
using assms
by(force simp add: substClosed-def intro: Weak-Early-Bisim.transitive)

lemma partUnfold:
  fixes P :: pi
  and Q :: pi
  and s :: (name  $\times$  name) list

  assumes  $P \approx^s Q$ 

```

```

  shows  $P[\langle s \rangle] \approx^s Q[\langle s \rangle]$ 
using assms
proof(auto simp add: substClosed-def)
  fix  $s'$ 
  assume  $\forall s. P[\langle s \rangle] \approx Q[\langle s \rangle]$ 
  hence  $P[\langle (s@s') \rangle] \approx Q[\langle (s@s') \rangle]$  by blast
  moreover have  $P[\langle (s@s') \rangle] = (P[\langle s \rangle])[\langle s' \rangle]$ 
    by(induct s', auto)
  moreover have  $Q[\langle (s@s') \rangle] = (Q[\langle s \rangle])[\langle s' \rangle]$ 
    by(induct s', auto)

  ultimately show  $(P[\langle s \rangle])[\langle s' \rangle] \approx (Q[\langle s \rangle])[\langle s' \rangle]$ 
    by simp
qed

end

theory Weak-Early-Cong-Subst
  imports Weak-Early-Cong Weak-Early-Bisim-Subst Strong-Early-Bisim-Subst
begin

const congruenceSubst ::  $(\pi \times \pi)$  set

definition weakCongruenceSubst (infix  $\langle \simeq^s \rangle$  65) where  $P \simeq^s Q \equiv \forall \sigma. P[\langle \sigma \rangle] \simeq Q[\langle \sigma \rangle]$ 

lemma unfoldE:
  fixes  $P :: \pi$ 
  and  $Q :: \pi$ 
  and  $s :: (\text{name} \times \text{name})$  list

  assumes  $P \simeq^s Q$ 

  shows  $P[\langle s \rangle] \rightsquigarrow \langle \text{weakBisim} \rangle Q[\langle s \rangle]$ 
  and  $Q[\langle s \rangle] \rightsquigarrow \langle \text{weakBisim} \rangle P[\langle s \rangle]$ 
proof -
  from assms show  $P[\langle s \rangle] \rightsquigarrow \langle \text{weakBisim} \rangle Q[\langle s \rangle]$  by(simp add: weakCongruenceSubst-def weakCongruence-def)
next
  from assms show  $Q[\langle s \rangle] \rightsquigarrow \langle \text{weakBisim} \rangle P[\langle s \rangle]$  by(simp add: weakCongruenceSubst-def weakCongruence-def)
qed

lemma unfoldI:
  fixes  $P :: \pi$ 
  and  $Q :: \pi$ 

  assumes  $\bigwedge s. P[\langle s \rangle] \rightsquigarrow \langle \text{weakBisim} \rangle Q[\langle s \rangle]$ 

```

```

and  $\bigwedge s. Q[\langle s \rangle] \rightsquigarrow \ll \text{weakBisim} \gg P[\langle s \rangle]$ 

shows  $P \simeq^s Q$ 
using assms
by(simp add: weakCongruenceSubst-def weakCongruence-def)

lemma weakCongWeakEq:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 

  assumes  $P \simeq^s Q$ 

  shows  $P \simeq Q$ 
using assms
apply(simp add: weakCongruenceSubst-def weakCongruence-def)
apply(erule-tac x=[] in allE)
by auto

lemma eqvtI:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $p :: name prm$ 

  assumes  $P \simeq^s Q$ 

  shows  $(p \cdot P) \simeq^s (p \cdot Q)$ 
proof(simp add: weakCongruenceSubst-def, rule allI)
  fix  $s$ 
  from assms have  $P[\langle \text{rev } p \cdot s \rangle] \simeq Q[\langle \text{rev } p \cdot s \rangle]$  by(auto simp add: weakCongruenceSubst-def)
  thus  $(p \cdot P)[\langle s \rangle] \simeq (p \cdot Q)[\langle s \rangle]$  by(drule-tac p=p in Weak-Early-Cong.eqvtI (simp add: eqvts name-per-rev))
qed

lemma strongEqWeakCong:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 

  assumes  $P \sim^s Q$ 

  shows  $P \simeq^s Q$ 
using assms
by(auto intro: strongBisimWeakCong simp add: substClosed-def weakCongruence-Subst-def)

lemma congSubstBisimSubst:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 

```

```

assumes  $P \simeq^s Q$ 

shows  $P \approx^s Q$ 
using assms
by(auto intro: congruenceWeakBisim simp add: substClosed-def weakCongruenceSubst-def)

lemma reflexive:
  fixes  $P :: pi$ 

  shows  $P \simeq^s P$ 
proof –
  from Weak-Early-Bisim.reflexive have  $\bigwedge P. P \rightsquigarrow \langle \langle \text{weakBisim} \rangle \rangle P$ 
    by(blast intro: Weak-Early-Step-Sim.reflexive)
  thus ?thesis
    by(force simp add: weakCongruenceSubst-def weakCongruence-def)
qed

lemma symetric:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 

  assumes  $P \simeq^s Q$ 

  shows  $Q \simeq^s P$ 
using assms by(auto simp add: weakCongruenceSubst-def weakCongruence-def)

lemma transitive:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $R :: pi$ 

  assumes  $P \simeq^s Q$ 
  and  $Q \simeq^s R$ 

  shows  $P \simeq^s R$ 
using assms by(auto simp add: weakCongruenceSubst-def intro: Weak-Early-Cong.transitive)

lemma partUnfold:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $s :: (name \times name) list$ 

  assumes  $P \simeq^s Q$ 

  shows  $P[\langle s \rangle] \simeq^s Q[\langle s \rangle]$ 
using assms
proof(auto simp add: weakCongruenceSubst-def)
  fix  $s'$ 

```

```

assume  $\forall s. P[\langle s \rangle] \simeq Q[\langle s \rangle]$ 
hence  $P[\langle (s@s') \rangle] \simeq Q[\langle (s@s') \rangle]$  by blast
moreover have  $P[\langle (s@s') \rangle] = (P[\langle s \rangle])[\langle s' \rangle]$ 
  by(induct s', auto)
moreover have  $Q[\langle (s@s') \rangle] = (Q[\langle s \rangle])[\langle s' \rangle]$ 
  by(induct s', auto)

  ultimately show  $(P[\langle s \rangle])[\langle s' \rangle] \simeq (Q[\langle s \rangle])[\langle s' \rangle]$ 
    by simp
qed

end

theory Weak-Early-Step-Sim-Pres
  imports Weak-Early-Step-Sim
begin

lemma tauPres:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $Rel :: (pi \times pi)$  set
  and  $Rel' :: (pi \times pi)$  set

  assumes  $PRelQ: (P, Q) \in Rel$ 

  shows  $\tau.(P) \rightsquigarrow \langle Rel \rangle \tau.(Q)$ 
proof(induct rule: simCases)
  case(Bound a x Q')
  have  $\tau.(Q) \mapsto a \prec \nu x \prec Q'$  by fact
  hence False by(induct rule: tauCases', auto)
  thus ?case by simp
next
  case(Free  $\alpha$  Q')
  have  $\tau.(Q) \mapsto (\alpha \prec Q')$  by fact
  thus ?case
  proof(induct rule: tauCases', auto simp add: pi.inject residual.inject)
  have  $\tau.(P) \Longrightarrow \tau \prec P$  by(rule Weak-Early-Step-Semantics.Tau)
  with  $PRelQ$  show  $\exists P'. \tau.(P) \Longrightarrow \tau \prec P' \wedge (P', Q) \in Rel$  by blast
  qed
qed

lemma inputPres:
  fixes  $P :: pi$ 
  and  $x :: name$ 
  and  $Q :: pi$ 
  and  $a :: name$ 
  and  $Rel :: (pi \times pi)$  set

  assumes  $PRelQ: \forall y. (P[x::=y], Q[x::=y]) \in Rel$ 

```

```

and      Eqvt: eqvt Rel

shows  $a\langle x \rangle.P \rightsquigarrow_{\langle Rel \rangle} a\langle x \rangle.Q$ 
using Eqvt
proof(induct rule: simCasesCont[where C=(x, a, P, Q)])
  case(Bound b y Q')
  from  $\langle y \# (x, a, P, Q) \rangle$  have  $y \neq x \ y \neq a \ y \# P \ y \# Q$  by simp+
  from  $\langle a\langle x \rangle.Q \mapsto b\langle \nu y \rangle \prec Q' \rangle \ \langle y \neq a \rangle \ \langle y \neq x \rangle \ \langle y \# Q \rangle$  show ?case
    by(erule-tac inputCases') auto
next
case(Free  $\alpha \ Q'$ )
from  $\langle a\langle x \rangle.Q \mapsto \alpha \prec Q' \rangle$ 
show ?case
proof(induct rule: inputCases)
  case(cInput u)
  have  $a\langle x \rangle.P \implies (a\langle u \rangle) \prec (P[x::=u])$ 
  by(rule Weak-Early-Step-Semantics.Input)
  moreover from PRelQ have  $(P[x::=u], Q[x::=u]) \in Rel$  by auto
  ultimately show ?case by blast
qed
qed

lemma outputPres:
  fixes P    :: pi
  and Q      :: pi
  and a      :: name
  and b      :: name
  and Rel    :: (pi  $\times$  pi) set
  and Rel'   :: (pi  $\times$  pi) set

  assumes PRelQ: (P, Q)  $\in$  Rel

  shows  $a\{b\}.P \rightsquigarrow_{\langle Rel \rangle} a\{b\}.Q$ 
proof(induct rule: simCases)
  case(Bound c x Q')
  have  $a\{b\}.Q \mapsto c\langle \nu x \rangle \prec Q'$  by fact
  hence False by(induct rule: outputCases', auto)
  thus ?case by simp
next
case(Free  $\alpha \ Q'$ )
have  $a\{b\}.Q \mapsto \alpha \prec Q'$  by fact
thus  $\exists P'. a\{b\}.P \implies \alpha \prec P' \wedge (P', Q') \in Rel$ 
proof(induct rule: outputCases', auto simp add: pi.inject residual.inject)
  have  $a\{b\}.P \implies a[b] \prec P$  by(rule Weak-Early-Step-Semantics.Output)
  with PRelQ show  $\exists P'. a\{b\}.P \implies a[b] \prec P' \wedge (P', Q) \in Rel$  by blast
qed
qed

lemma matchPres:

```

```

fixes  $P$   ::  $pi$ 
and    $Q$   ::  $pi$ 
and    $a$   ::  $name$ 
and    $b$   ::  $name$ 
and    $Rel$  ::  $(pi \times pi)$  set
and    $Rel'$  ::  $(pi \times pi)$  set

assumes  $PSimQ$ :  $P \rightsquigarrow_{\langle Rel \rangle} Q$ 
and      $RelRel'$ :  $Rel \subseteq Rel'$ 

shows  $[a \frown b]P \rightsquigarrow_{\langle Rel' \rangle} [a \frown b]Q$ 
proof(induct rule: simCases)
  case( $Bound\ c\ x\ Q'$ )
    have  $x \# [a \frown b]P$  by fact
    hence  $xFreshP$ :  $(x::name) \# P$  by simp
    have  $[a \frown b]Q \mapsto_{c \langle \nu x \rangle} \prec Q'$  by fact
    thus ?case
  proof(induct rule: matchCases)
    case  $Match$ 
      have  $Q \mapsto_{c \langle \nu x \rangle} \prec Q'$  by fact
      with  $PSimQ\ xFreshP$  obtain  $P'$  where  $PTrans$ :  $P \Longrightarrow_{c \langle \nu x \rangle} \prec P'$ 
        and  $P'RelQ'$ :  $(P', Q') \in Rel$ 
      by(blast dest: simE)
      from  $PTrans$  have  $[a \frown a]P \Longrightarrow_{c \langle \nu x \rangle} \prec P'$  by(rule Weak-Early-Step-Semantics.Match)
      moreover from  $P'RelQ'\ RelRel'$  have  $(P', Q') \in Rel'$  by blast
      ultimately show ?case by blast
    qed
  next
    case( $Free\ \alpha\ Q'$ )
      have  $[a \frown b]Q \mapsto_{\alpha} \prec Q'$  by fact
      thus ?case
    proof(induct rule: matchCases)
      case  $Match$ 
        have  $Q \mapsto_{\alpha} \prec Q'$  by fact
        with  $PSimQ$  obtain  $P'$  where  $PTrans$ :  $P \Longrightarrow_{\alpha} \prec P'$  and  $P'Rel$ :  $(P', Q') \in Rel$ 
        by(blast dest: simE)
        from  $PTrans$  have  $[a \frown a]P \Longrightarrow_{\alpha} \prec P'$  by(rule Weak-Early-Step-Semantics.Match)
        with  $RelRel'\ P'Rel$  show ?case by blast
      qed
    qed
  qed

lemma mismatchPres:
  fixes  $P$   ::  $pi$ 
  and    $Q$   ::  $pi$ 
  and    $a$   ::  $name$ 
  and    $b$   ::  $name$ 
  and    $Rel$  ::  $(pi \times pi)$  set
  and    $Rel'$  ::  $(pi \times pi)$  set

```

```

assumes  $PSimQ: P \rightsquigarrow \langle Rel \rangle Q$ 
and  $RelRel': Rel \subseteq Rel'$ 

shows  $[a \neq b]P \rightsquigarrow \langle Rel' \rangle [a \neq b]Q$ 
proof(induct rule: simCases)
  case(Bound c x Q')
    have  $x \# [a \neq b]P$  by fact
    hence  $xFreshP: (x::name) \# P$  by simp
    have  $[a \neq b]Q \mapsto c \langle \nu x \rangle \prec Q'$  by fact
    thus ?case
    proof(induct rule: mismatchCases)
      case Mismatch
        have  $a \neq b$  by fact
        have  $Q \mapsto c \langle \nu x \rangle \prec Q'$  by fact
        with  $PSimQ$   $xFreshP$  obtain  $P'$  where  $PTrans: P \Longrightarrow c \langle \nu x \rangle \prec P'$ 
          and  $P'RelQ': (P', Q') \in Rel$ 
        by(blast dest: simE)
        from  $PTrans$   $a \neq b$  have  $[a \neq b]P \Longrightarrow c \langle \nu x \rangle \prec P'$  by(rule Weak-Early-Step-Semantics.Mismatch)
        moreover from  $P'RelQ'$   $RelRel'$  have  $(P', Q') \in Rel'$  by blast
        ultimately show ?case by blast
      qed
    next
      case(Free  $\alpha$  Q')
        have  $[a \neq b]Q \mapsto \alpha \prec Q'$  by fact
        thus ?case
        proof(induct rule: mismatchCases)
          case Mismatch
            have  $Q \mapsto \alpha \prec Q'$  by fact
            with  $PSimQ$  obtain  $P'$  where  $PTrans: P \Longrightarrow \alpha \prec P'$  and  $P'Rel: (P', Q') \in$ 
               $Rel$ 
            by(blast dest: simE)
            from  $PTrans$   $\langle a \neq b \rangle$  have  $[a \neq b]P \Longrightarrow \alpha \prec P'$  by(rule Weak-Early-Step-Semantics.Mismatch)
            with  $RelRel'$   $P'Rel$  show ?case by blast
          qed
        qed
    lemma sumPres:
      fixes  $P :: pi$ 
      and  $Q :: pi$ 
      and  $R :: pi$ 

      assumes  $PSimQ: P \rightsquigarrow \langle Rel \rangle Q$ 
      and  $RelRel': Rel \subseteq Rel'$ 
      and  $C: Id \subseteq Rel'$ 

      shows  $P \oplus R \rightsquigarrow \langle Rel' \rangle Q \oplus R$ 
      proof(induct rule: simCases)
        case(Bound a x Q')

```

```

have x # P ⊕ R by fact
hence xFreshP: (x::name) # P and xFreshR: x # R by simp+
have Q ⊕ R ⟶ a⟨νx⟩ < Q' by fact
thus ?case
proof(induct rule: sumCases)
  case Sum1
  have Q ⟶ a⟨νx⟩ < Q' by fact
  with xFreshP PSimQ obtain P' where PTrans: P ⟹ a⟨νx⟩ < P' and
P'RelQ': (P', Q') ∈ Rel
  by(blast dest: simE)
  from PTrans have P ⊕ R ⟹ a⟨νx⟩ < P' by(rule Weak-Early-Step-Semantics.Sum1)
  moreover from P'RelQ' RelRel' have (P', Q') ∈ Rel' by blast
  ultimately show ?case by blast
next
  case Sum2
  from ⟨R ⟶ a⟨νx⟩ < Q'⟩ have P ⊕ R ⟶ a⟨νx⟩ < Q' by(rule Early-Semantics.Sum2)
  hence P ⊕ R ⟹ a⟨νx⟩ < Q' by(rule Weak-Early-Step-Semantics.singleActionChain)
  moreover from C have (Q', Q') ∈ Rel' by blast
  ultimately show ?case by blast
qed
next
case(Free α Q')
have Q ⊕ R ⟶ α < Q' by fact
thus ?case
proof(induct rule: sumCases)
  case Sum1
  have Q ⟶ α < Q' by fact
  with PSimQ obtain P' where PTrans: P ⟹ α < P' and PRel: (P', Q') ∈
Rel
  by(blast dest: simE)
  from PTrans have P ⊕ R ⟹ α < P' by(rule Weak-Early-Step-Semantics.Sum1)
  with RelRel' PRel show ?case by blast
next
  case Sum2
  from ⟨R ⟶ α < Q'⟩ have P ⊕ R ⟶ α < Q' by(rule Early-Semantics.Sum2)
  hence P ⊕ R ⟹ α < Q' by(rule Weak-Early-Step-Semantics.singleActionChain)
  moreover from C have (Q', Q') ∈ Rel' by blast
  ultimately show ?case by blast
qed
qed

lemma parPres:
  fixes P    :: pi
  and Q     :: pi
  and R     :: pi
  and T     :: pi
  and Rel  :: (pi × pi) set
  and Rel' :: (pi × pi) set
  and Rel'' :: (pi × pi) set

```

assumes $PSimQ: P \rightsquigarrow \langle Rel \rangle Q$
and $PRelQ: (P, Q) \in Rel$
and $Par: \bigwedge S T U. (S, T) \in Rel \implies (S \parallel U, T \parallel U) \in Rel'$
and $Res: \bigwedge S T x. (S, T) \in Rel' \implies (\langle \nu x \rangle S, \langle \nu x \rangle T) \in Rel'$

shows $P \parallel R \rightsquigarrow \langle Rel' \rangle Q \parallel R$

proof –

show *?thesis*

proof(*induct rule: simCases*)

case(*Bound a x Q'*)

have $x \# P \parallel R$ **by** *fact*

hence $xFreshP: x \# P$ **and** $xFreshR: x \# R$ **by** *simp+*

have $Q \parallel R \mapsto a \langle \nu x \rangle \prec Q'$ **by** *fact*

thus *?case*

proof(*induct rule: parCasesB*)

case(*cPar1 Q'*)

have $QTrans: Q \mapsto a \langle \nu x \rangle \prec Q'$ **by** *fact*

from $xFreshP$ $PSimQ$ $QTrans$ **obtain** P' **where** $PTrans: P \implies a \langle \nu x \rangle \prec P'$
and $P'RelQ': (P', Q') \in Rel$

by(*blast dest: simE*)

from $PTrans$ $xFreshR$ **have** $P \parallel R \implies a \langle \nu x \rangle \prec (P' \parallel R)$ **by**(*rule Weak-Early-Step-Semantics.Par1B*)

moreover from $P'RelQ'$ **have** $(P' \parallel R, Q' \parallel R) \in Rel'$ **by**(*rule Par*)

ultimately show *?case* **by** *blast*

next

case(*cPar2 R'*)

from $\langle R \mapsto a \langle \nu x \rangle \prec R' \rangle \langle x \# P \rangle$ **have** $P \parallel R \mapsto a \langle \nu x \rangle \prec (P \parallel R')$

by(*rule Early-Semantics.Par2B*)

hence $P \parallel R \implies a \langle \nu x \rangle \prec (P \parallel R')$ **by**(*rule Weak-Early-Step-Semantics.singleActionChain*)

moreover from $PRelQ$ **have** $(P \parallel R', Q \parallel R') \in Rel'$ **by**(*rule Par*)

ultimately show *?case* **by** *blast*

qed

next

case(*Free α QR'*)

have $Q \parallel R \mapsto \alpha \prec QR'$ **by** *fact*

thus *?case*

proof(*induct rule: parCasesF[of - - - - (P, R)]*)

case(*cPar1 Q'*)

have $Q \mapsto \alpha \prec Q'$ **by** *fact*

with $PSimQ$ **obtain** P' **where** $PTrans: P \implies \alpha \prec P'$ **and** $PRel: (P', Q') \in Rel$

by(*blast dest: simE*)

from $PTrans$ **have** $Trans: P \parallel R \implies \alpha \prec P' \parallel R$ **by**(*rule Weak-Early-Step-Semantics.Par1F*)

moreover from $PRel$ **have** $(P' \parallel R, Q' \parallel R) \in Rel'$ **by**(*blast intro: Par*)

ultimately show *?case* **by** *blast*

next

case(*cPar2 R'*)

from $\langle R \mapsto \alpha \prec R' \rangle$ **have** $P \parallel R \mapsto \alpha \prec (P \parallel R')$

by(*rule Early-Semantics.Par2F*)

hence $P \parallel R \Longrightarrow \alpha \prec (P \parallel R')$ **by** (*rule Weak-Early-Step-Semantics.singleActionChain*)
moreover from $P \text{Rel} Q$ **have** $(P \parallel R', Q \parallel R') \in \text{Rel}'$ **by** (*rule Par*)
ultimately show *?case by blast*

next
case (*cComm1 Q' R' a b*)
have $Q \text{Trans}: Q \mapsto a \langle b \rangle \prec Q'$ **and** $R \text{Trans}: R \mapsto a[b] \prec R'$ **by** *fact+*

from $P \text{Sim} Q$ $Q \text{Trans}$ **obtain** P' **where** $P \text{Trans}: P \Longrightarrow a \langle b \rangle \prec P'$
and $P' \text{Rel} Q': (P', Q') \in \text{Rel}$
by (*blast dest: simE*)

from $R \text{Trans}$ **have** $R \Longrightarrow a[b] \prec R'$ **by** (*rule Weak-Early-Step-Semantics.singleActionChain*)
with $P \text{Trans}$ **have** $P \parallel R \Longrightarrow \tau \prec P' \parallel R'$ **by** (*rule Weak-Early-Step-Semantics.Comm1*)
moreover from $P' \text{Rel} Q'$ **have** $(P' \parallel R', Q' \parallel R') \in \text{Rel}'$ **by** (*rule Par*)
ultimately show *?case by blast*

next
case (*cComm2 Q' R' a b*)
have $Q \text{Trans}: Q \mapsto a[b] \prec Q'$ **and** $R \text{Trans}: R \mapsto a \langle b \rangle \prec R'$ **by** *fact+*

from $P \text{Sim} Q$ $Q \text{Trans}$ **obtain** P' **where** $P \text{Trans}: P \Longrightarrow a[b] \prec P'$
and $P' \text{Rel} Q': (P', Q') \in \text{Rel}$
by (*blast dest: simE*)

from $R \text{Trans}$ **have** $R \Longrightarrow a \langle b \rangle \prec R'$ **by** (*rule Weak-Early-Step-Semantics.singleActionChain*)
with $P \text{Trans}$ **have** $P \parallel R \Longrightarrow \tau \prec P' \parallel R'$ **by** (*rule Weak-Early-Step-Semantics.Comm2*)
moreover from $P' \text{Rel} Q'$ **have** $(P' \parallel R', Q' \parallel R') \in \text{Rel}'$ **by** (*rule Par*)
ultimately show *?case by blast*

next
case (*cClose1 Q' R' a x*)
have $Q \text{Trans}: Q \mapsto a \langle x \rangle \prec Q'$ **and** $R \text{Trans}: R \mapsto a \langle \nu x \rangle \prec R'$ **by** *fact+*
have $x \# (P, R)$ **by** *fact*
hence $x \text{Fresh} P: x \# P$ **and** $x \text{Fresh} R: x \# R$ **by** (*simp add: fresh-prod*)+

from $P \text{Sim} Q$ $Q \text{Trans}$ **obtain** P' **where** $P \text{Trans}: P \Longrightarrow a \langle x \rangle \prec P'$
and $P' \text{Rel} Q': (P', Q') \in \text{Rel}$
by (*blast dest: simE*)

from $R \text{Trans}$ **have** $R \Longrightarrow a \langle \nu x \rangle \prec R'$ **by** (*rule Weak-Early-Step-Semantics.singleActionChain*)
with $P \text{Trans}$ **have** $\text{Trans}: P \parallel R \Longrightarrow \tau \prec \langle \nu x \rangle (P' \parallel R')$ **using** $\langle x \# P \rangle$
by (*rule Weak-Early-Step-Semantics.Close1*)
moreover from $P' \text{Rel} Q'$ **have** $\langle \nu x \rangle (P' \parallel R'), \langle \nu x \rangle (Q' \parallel R') \in \text{Rel}'$
by (*blast intro: Par Res*)
ultimately show *?case by blast*

next
case (*cClose2 Q' R' a x*)
have $Q \text{Trans}: Q \mapsto a \langle \nu x \rangle \prec Q'$ **and** $R \text{Trans}: R \mapsto a \langle x \rangle \prec R'$ **by** *fact+*
have $x \# (P, R)$ **by** *fact*
hence $x \text{Fresh} R: x \# R$ **and** $x \text{Fresh} P: x \# P$ **by** (*simp add: fresh-prod*)+

from $P \text{Sim} Q$ $Q \text{Trans}$ $x \text{Fresh} P$ **obtain** P' **where** $P \text{Trans}: P \Longrightarrow a \langle \nu x \rangle \prec P'$

and $P'RelQ': (P', Q') \in Rel$

by(blast dest: simE)

from $RTrans$ **have** $R \Longrightarrow a\langle x \rangle \prec R'$ **by**(rule Weak-Early-Step-Semantics.singleActionChain)

with $PTrans$ **have** $Trans: P \parallel R \Longrightarrow \tau \prec \langle \nu x \rangle (P' \parallel R')$ **using** $\langle x \# R \rangle$

by(rule Weak-Early-Step-Semantics.Close2)

moreover from $P'RelQ'$ **have** $(\langle \nu x \rangle (P' \parallel R'), \langle \nu x \rangle (Q' \parallel R')) \in Rel'$

by(blast intro: Par Res)

ultimately show ?case **by** blast

qed

qed

qed

lemma resPres:

fixes $P :: pi$

and $Q :: pi$

and $Rel :: (pi \times pi)$ set

and $x :: name$

and $Rel' :: (pi \times pi)$ set

assumes $PSimQ: P \rightsquigarrow \langle Rel \rangle Q$

and $C1: \bigwedge R S x. (R, S) \in Rel \implies (\langle \nu x \rangle R, \langle \nu x \rangle S) \in Rel'$

and $RelRel': Rel \subseteq Rel'$

and $EqvtRel: eqvt Rel$

and $EqvtRel': eqvt Rel'$

shows $\langle \nu x \rangle P \rightsquigarrow \langle Rel' \rangle \langle \nu x \rangle Q$

proof –

from $EqvtRel'$ **show** ?thesis

proof(induct rule: simCasesCont[of - (P, x)])

case(Bound a y Q')

have $Trans: \langle \nu x \rangle Q \mapsto a\langle \nu y \rangle \prec Q'$ **by** fact

have $y \# (P, x)$ **by** fact

hence $y \text{ineq}x: y \neq x$ **and** $y \text{Fresh}P: y \# P$ **by**(simp add: fresh-prod)+

from $Trans$ $y \text{ineq}x$ **show** ?case

proof(induct rule: resCasesB)

case(Open Q')

have $QTrans: Q \mapsto a[x] \prec Q'$ **and** $a \text{ineq}x: a \neq x$ **by** fact+

from $PSimQ$ $QTrans$ **obtain** P' **where** $PTrans: P \Longrightarrow a[x] \prec P'$

and $P'RelQ': (P', Q') \in Rel$

by(blast dest: simE)

from $PTrans$ $a \text{ineq}x$ **have** $\langle \nu x \rangle P \Longrightarrow a\langle \nu x \rangle \prec P'$ **by**(rule Weak-Early-Step-Semantics.Open)

hence $\langle \nu x \rangle P \Longrightarrow a\langle \nu y \rangle \prec ([(y, x)] \cdot P')$ **using** $\langle y \# P \rangle \langle y \neq x \rangle$

by(force simp add: weakTransitionAlpha abs-fresh name-swap)

moreover from $EqvtRel$ $P'RelQ'$ $RelRel'$ **have** $([(y, x)] \cdot P', [(y, x)] \cdot Q') \in Rel'$

```

    by(blast intro: eqvtRelI)
  ultimately show ?case by blast
next
case(Res Q')
have QTrans:  $Q \mapsto a \langle \nu y \rangle \prec Q'$  and xineqa:  $x \neq a$  by fact+

from PSimQ yFreshP QTrans obtain P' where PTrans:  $P \Longrightarrow a \langle \nu y \rangle \prec P'$ 
and P'RelQ':  $(P', Q') \in \text{Rel}$ 

  by(blast dest: simE)
  from PTrans xineqa yineqx yFreshP have ResTrans:  $\langle \nu x \rangle P \Longrightarrow a \langle \nu y \rangle \prec$ 
( $\langle \nu x \rangle P'$ )
  by(blast intro: Weak-Early-Step-Semantics.ResB)
  moreover from P'RelQ' have  $((\langle \nu x \rangle P'), (\langle \nu x \rangle Q')) \in \text{Rel}'$ 
  by(rule C1)
  ultimately show ?case by blast
qed
next
case(Free  $\alpha$  Q')
have QTrans:  $\langle \nu x \rangle Q \mapsto \alpha \prec Q'$  by fact
have  $\exists c::\text{name}. c \# (P, Q, Q', \alpha)$  by(blast intro: name-exists-fresh)
  then obtain c::name where cFreshQ:  $c \# Q$  and cFreshAlpha:  $c \# \alpha$  and
cFreshQ':  $c \# Q'$  and cFreshP:  $c \# P$ 
  by(force simp add: fresh-prod)
  from cFreshP have  $\langle \nu x \rangle P = \langle \nu c \rangle ([x, c] \cdot P)$  by(simp add: alphaRes)
  moreover have  $\exists P'. \langle \nu c \rangle ([x, c] \cdot P) \Longrightarrow \alpha \prec P' \wedge (P', Q') \in \text{Rel}'$ 
  proof -
    from QTrans cFreshQ have  $\langle \nu c \rangle ([x, c] \cdot Q) \mapsto \alpha \prec Q'$  by(simp add:
alphaRes)
    moreover have  $c \# \alpha$  by(rule cFreshAlpha)
    moreover from PSimQ EqvtRel have  $([x, c] \cdot P) \rightsquigarrow \langle \text{Rel} \rangle ([x, c] \cdot Q)$ 
      by(blast intro: eqvtI)
    ultimately show ?thesis
      apply(induct rule: resCasesF, auto simp add: residual.inject pi.inject
name-abs-eq)
      by(blast intro: Weak-Early-Step-Semantics.ResF C1 dest: simE)
  qed

  ultimately show ?case by force
qed
qed

lemma resChainI:
  fixes P :: pi
  and Q :: pi
  and Rel ::  $(\text{pi} \times \text{pi}) \text{ set}$ 
  and lst :: name list

  assumes eqvtRel: eqvt Rel
  and Res:  $\bigwedge R S x. (R, S) \in \text{Rel} \Longrightarrow (\langle \nu x \rangle R, \langle \nu x \rangle S) \in \text{Rel}$ 

```

and $P\text{Rel}Q: P \rightsquigarrow\langle\text{Rel}\rangle Q$

shows $(\text{resChain } lst) P \rightsquigarrow\langle\text{Rel}\rangle (\text{resChain } lst) Q$

proof –

show *?thesis*

proof(*induct lst*)

from $P\text{Rel}Q$ **show** $\text{resChain } [] P \rightsquigarrow\langle\text{Rel}\rangle \text{resChain } [] Q$ **by** *simp*

next

fix $a \text{ } lst$

assume $IH: (\text{resChain } lst P) \rightsquigarrow\langle\text{Rel}\rangle (\text{resChain } lst Q)$

moreover from Res **have** $\bigwedge P Q a. (P, Q) \in \text{Rel} \implies \langle\nu a\rangle P, \langle\nu a\rangle Q \in \text{Rel}$

by *simp*

moreover have $\text{Rel} \subseteq \text{Rel}$ **by** *simp*

ultimately have $\langle\nu a\rangle(\text{resChain } lst P) \rightsquigarrow\langle\text{Rel}\rangle \langle\nu a\rangle(\text{resChain } lst Q)$ **using** *eqvtRel*

by(*rule-tac resPres*)

thus $\text{resChain } (a \# lst) P \rightsquigarrow\langle\text{Rel}\rangle \text{resChain } (a \# lst) Q$

by *simp*

qed

qed

lemma *bangPres*:

fixes $P :: pi$

and $Q :: pi$

and $\text{Rel} :: (pi \times pi) \text{ set}$

assumes $P\text{Rel}Q: (P, Q) \in \text{Rel}$

and $\text{Sim}: \bigwedge R S. (R, S) \in \text{Rel} \implies R \rightsquigarrow\langle\text{Rel}'\rangle S$

and $C1: \text{Rel} \subseteq \text{Rel}'$

and $\text{eqvtRel}: \text{eqvt } \text{Rel}'$

shows $!P \rightsquigarrow\langle\text{bangRel } \text{Rel}'\rangle !Q$

proof –

let $?Sim = \lambda P Rs. (\forall a x Q'. Rs = a \langle\nu x\rangle \prec Q' \longrightarrow x \# P \longrightarrow (\exists P'. P \implies a \langle\nu x\rangle \prec P' \wedge (P', Q') \in \text{bangRel } \text{Rel}')) \wedge$

$(\forall \alpha Q'. Rs = \alpha \prec Q' \longrightarrow (\exists P'. P \implies \alpha \prec P' \wedge (P', Q') \in \text{bangRel } \text{Rel}'))$

from eqvtRel **have** $\text{EqvtBangRel}: \text{eqvt}(\text{bangRel } \text{Rel}')$ **by**(*rule eqvtBangRel*)

from $C1$ **have** $B\text{RelRel}': \bigwedge P Q. (P, Q) \in \text{bangRel } \text{Rel} \implies (P, Q) \in \text{bangRel } \text{Rel}'$

by(*auto intro: bangRelSubset*)

{

fix $Pa Rs$

assume $!Q \longmapsto Rs$ **and** $(Pa, !Q) \in \text{bangRel } \text{Rel}$

hence $?Sim Pa Rs$ **using** $P\text{Rel}Q$

proof(*nominal-induct avoiding: Pa P rule: bangInduct*)

```

case(Par1B  $a\ x\ Q'\ Pa\ P$ )
have  $QTrans: Q \mapsto a\langle \nu x \rangle \prec Q'$  by fact
have  $(Pa, Q \parallel !Q) \in \text{bangRel Rel}$  and  $x \# Pa$  by fact+
thus  $?Sim\ Pa\ (a\langle \nu x \rangle \prec (Q' \parallel !Q))$ 
proof(induct rule: BRParCases)
  case(BRPar  $P\ R$ )
    have  $PRelQ: (P, Q) \in Rel$  by fact
    have  $PBRQ: (R, !Q) \in \text{bangRel Rel}$  by fact
    have  $x \# P \parallel R$  by fact
    hence  $xFreshP: x \# P$  and  $xFreshR: x \# R$  by simp+
    show ?case
    proof(auto simp add: residual.inject alpha')
      from  $PRelQ$  have  $P \rightsquigarrow \langle Rel' \rangle Q$  by(rule Sim)

    with  $QTrans\ xFreshP$  obtain  $P'$  where  $PTrans: P \Longrightarrow a\langle \nu x \rangle \prec P'$  and
 $P'RelQ': (P', Q') \in Rel'$ 
      by(blast dest: simE)

    from  $PTrans\ xFreshR$  have  $P \parallel R \Longrightarrow a\langle \nu x \rangle \prec (P' \parallel R)$ 
      by(force intro: Weak-Early-Step-Semantics.Par1B)
    moreover from  $P'RelQ'\ PBRQ\ BRelRel'$  have  $(P' \parallel R, Q' \parallel !Q) \in$ 
 $\text{bangRel Rel}'$  by(blast intro: Rel.BRPar)
    ultimately show  $\exists P'. P \parallel R \Longrightarrow a\langle \nu x \rangle \prec P' \wedge (P', Q' \parallel !Q) \in \text{bangRel}$ 
 $Rel'$  by blast
    next
      fix  $y$ 
      assume  $(y::name) \# Q'$  and  $y \# P$  and  $y \# R$  and  $y \# Q$ 
      from  $QTrans\ \langle y \# Q' \rangle$  have  $Q \mapsto a\langle \nu y \rangle \prec ((x, y) \cdot Q')$ 
      by(simp add: alphaBoundOutput)
      moreover from  $PRelQ$  have  $P \rightsquigarrow \langle Rel' \rangle Q$  by(rule Sim)
      ultimately obtain  $P'$  where  $PTrans: P \Longrightarrow a\langle \nu y \rangle \prec P'$  and  $P'RelQ':$ 
 $(P', [(x, y)] \cdot Q') \in Rel'$ 
      using  $\langle y \# P \rangle$ 
      by(blast dest: simE)
      from  $PTrans\ \langle y \# R \rangle$  have  $P \parallel R \Longrightarrow a\langle \nu y \rangle \prec (P' \parallel R)$  by(force intro:
 $\text{Weak-Early-Step-Semantics.Par1B}$ )
      moreover from  $P'RelQ'\ PBRQ\ BRelRel'$  have  $(P' \parallel R, [(x, y)] \cdot Q') \parallel$ 
 $!Q) \in \text{bangRel Rel}'$  by(metis Rel.BRPar)
      with  $\langle x \# Q \rangle\ \langle y \# Q \rangle$  have  $(P' \parallel R, [(y, x)] \cdot Q') \parallel ![(y, x)] \cdot Q) \in$ 
 $\text{bangRel Rel}'$ 
      by(simp add: name-fresh-fresh name-swap)
      ultimately show  $\exists P'. P \parallel R \Longrightarrow a\langle \nu y \rangle \prec P' \wedge (P', [(y, x)] \cdot Q') \parallel$ 
 $![(y, x)] \cdot Q) \in \text{bangRel Rel}'$ 
      by blast
    qed
  qed
next
  case(Par1F  $\alpha\ Q'\ Pa\ P$ )
  have  $QTrans: Q \mapsto \alpha \prec Q'$  by fact

```

```

have (Pa, Q || !Q) ∈ bangRel Rel by fact
thus ?case
proof(induct rule: BRParCases)
  case(BRPar P R)
  have PRelQ: (P, Q) ∈ Rel and BR: (R, !Q) ∈ bangRel Rel by fact+
  show ?case
  proof(auto simp add: residual.inject)
    from PRelQ have P ~«Rel'» Q by(rule Sim)
    with QTrans obtain P' where PTrans: P ⇒α < P' and RRel: (P',
Q') ∈ Rel'
    by(blast dest: simE)

    from PTrans have P || R ⇒α < P' || R by(rule Weak-Early-Step-Semantics.Par1F)
    moreover from RRel BR BRelRel' have (P' || R, Q' || !Q) ∈ bangRel
Rel' by(metis Rel.BRPar)
    ultimately show ∃ P'. P || R ⇒α < P' ∧ (P', Q' || !Q) ∈ bangRel Rel'
by blast
  qed
qed
next
case(Par2B a x Q' Pa P)
  hence IH: ∧Pa. (Pa, !Q) ∈ bangRel Rel ⇒ ?Sim Pa (a<νx> < Q') by
simp
  have (Pa, Q || !Q) ∈ bangRel Rel and x ‡ Pa by fact+
  thus ?Sim Pa (a<νx> < (Q || Q'))
  proof(induct rule: BRParCases)
    case(BRPar P R)
    have PRelQ: (P, Q) ∈ Rel and RBRQ: (R, !Q) ∈ bangRel Rel by fact+
    have x ‡ P || R by fact
    hence xFreshP: x ‡ P and xFreshR: x ‡ R by simp+

    from EqvtBangRel show ?Sim (P || R) (a<νx> < (Q || Q'))
    proof(auto simp add: residual.inject alpha')
      from RBRQ have ?Sim R (a<νx> < Q') by(rule IH)
      with xFreshR obtain R' where RTrans: R ⇒a<νx> < R' and R'BRQ':
(R', Q') ∈ (bangRel Rel')
      by(metis simE)
      from RTrans xFreshP have P || R ⇒a<νx> < (P || R') by(auto intro:
Weak-Early-Step-Semantics.Par2B)
      moreover from PRelQ R'BRQ' C1 have (P || R', Q || Q') ∈ (bangRel
Rel') by(blast dest: Rel.BRPar)
      ultimately show ∃ P'. P || R ⇒a<νx> < P' ∧ (P', Q || Q') ∈ bangRel
Rel' by blast
    next
    fix y
    assume (y::name) ‡ Q and y ‡ Q' and y ‡ P and y ‡ R
    from RBRQ have ?Sim R (a<νx> < Q') by(rule IH)
    with ⟨y ‡ Q'⟩ have ?Sim R (a<νy> < ([x, y] · Q')) by(simp add:
alphaBoundOutput)

```

with $\langle y \# R \rangle$ **obtain** R' **where** $RTrans: R \Longrightarrow a \langle \nu y \rangle \prec R'$ **and** $R'BRQ'$:
 $(R', ((x, y)] \cdot Q')) \in (bangRel Rel')$
by $(metis simE)$
from $RTrans \langle y \# P \rangle$ **have** $P \parallel R \Longrightarrow a \langle \nu y \rangle \prec (P \parallel R')$ **by** $(auto intro: Weak-Early-Step-Semantics.Par2B)$
moreover from $PRelQ R'BRQ' C1$ **have** $(P \parallel R', Q \parallel ((x, y)] \cdot Q')) \in (bangRel Rel')$ **by** $(blast dest: Rel.BRPar)$
with $\langle y \# Q \rangle \langle x \# Q \rangle$ **have** $(P \parallel R', ((y, x)] \cdot Q) \parallel ((y, x)] \cdot Q')) \in (bangRel Rel')$
by $(simp add: name-swap name-fresh-fresh)$
ultimately show $\exists P'. P \parallel R \Longrightarrow a \langle \nu y \rangle \prec P' \wedge (P', ((y, x)] \cdot Q) \parallel ((y, x)] \cdot Q')) \in bangRel Rel'$ **by** $blast$
qed
qed
next
case $(Par2F \alpha Q' Pa P)$
hence $IH: \bigwedge Pa. (Pa, !Q) \in bangRel Rel \Longrightarrow ?Sim Pa (\alpha \prec Q')$ **by** $simp$
have $(Pa, Q \parallel !Q) \in bangRel Rel$ **by** $fact$
thus $?case$
proof $(induct rule: BRParCases)$
case $(BRPar P R)$
have $PRelQ: (P, Q) \in Rel$ **and** $RBRQ: (R, !Q) \in bangRel Rel$ **by** $fact+$
show $?case$
proof $(auto simp add: residual.inject)$
from $RBRQ IH$ **have** $\exists R'. R \Longrightarrow \alpha \prec R' \wedge (R', Q') \in bangRel Rel'$
by $(metis simE)$
then obtain R' **where** $RTrans: R \Longrightarrow \alpha \prec R'$ **and** $R'RelQ': (R', Q') \in bangRel Rel'$
by $blast$
from $RTrans$ **have** $P \parallel R \Longrightarrow \alpha \prec P \parallel R'$ **by** $(rule Weak-Early-Step-Semantics.Par2F)$
moreover from $PRelQ R'RelQ' C1$ **have** $(P \parallel R', Q \parallel Q') \in bangRel Rel'$ **by** $(blast dest: Rel.BRPar)$
ultimately show $\exists P'. P \parallel R \Longrightarrow \alpha \prec P' \wedge (P', Q \parallel Q') \in bangRel Rel'$
by $blast$
qed
qed
next
case $(Comm1 a Q' b Q'' Pa P)$
hence $IH: \bigwedge Pa. (Pa, !Q) \in bangRel Rel \Longrightarrow ?Sim Pa (a[b] \prec Q'')$ **by** $simp$
have $QTrans: Q \mapsto a \langle b \rangle \prec Q'$ **by** $fact$
have $(Pa, Q \parallel !Q) \in bangRel Rel$ **by** $fact$
thus $?case$
proof $(induct rule: BRParCases)$
case $(BRPar P R)$
have $PRelQ: (P, Q) \in Rel$ **and** $RBRQ: (R, !Q) \in bangRel Rel$ **by** $fact+$
show $?case$
proof $(auto simp add: residual.inject)$
from $PRelQ$ **have** $P \rightsquigarrow \langle Rel' \rangle Q$ **by** $(rule Sim)$

with $QTrans$ **obtain** P' **where** $PTrans: P \Longrightarrow a \langle b \rangle \prec P'$ **and** $P'RelQ': (P', Q') \in Rel'$
by (*blast dest: simE*)

from $IH RBRQ$ **have** $RTrans: \exists R'. R \Longrightarrow a[b] \prec R' \wedge (R', Q') \in \text{bangRel } Rel'$
by (*metis simE*)
then obtain R' **where** $RTrans: R \Longrightarrow a[b] \prec R'$ **and** $R'RelQ'': (R', Q'') \in \text{bangRel } Rel'$
by *blast*

from $PTrans RTrans$ **have** $P \parallel R \Longrightarrow \tau \prec P' \parallel R'$ **by** (*rule Weak-Early-Step-Semantics.Comm1*)
moreover from $P'RelQ' R'RelQ''$ **have** $(P' \parallel R', Q' \parallel Q'') \in \text{bangRel } Rel'$ **by** (*rule Rel.BRPar*)
ultimately show $\exists P'. P \parallel R \Longrightarrow \tau \prec P' \wedge (P', Q' \parallel Q'') \in \text{bangRel } Rel'$
by *blast*
qed
qed
next

case ($Comm2 a b Q' Q''$)
hence $IH: \bigwedge Pa. (Pa, !Q) \in \text{bangRel } Rel \Longrightarrow ?Sim Pa (a \langle b \rangle \prec Q')$ **by** *simp*
have $QTrans: Q \mapsto a[b] \prec Q'$ **by** *fact*
have $(Pa, Q \parallel !Q) \in \text{bangRel } Rel$ **by** *fact*
thus *?case*
proof (*induct rule: BRParCases*)
case ($BRPar P R$)
have $PRelQ: (P, Q) \in Rel$ **and** $RBRQ: (R, !Q) \in \text{bangRel } Rel$ **by** *fact+*
show *?case*
proof (*auto simp add: residual.inject*)
from $PRelQ$ **have** $P \rightsquigarrow \langle Rel' \rangle Q$ **by** (*rule Sim*)
with $QTrans$ **obtain** P' **where** $PTrans: P \Longrightarrow a[b] \prec P'$ **and** $P'RelQ': (P', Q') \in Rel'$
by (*blast dest: simE*)

from $IH RBRQ$ **have** $RTrans: \exists R'. R \Longrightarrow a \langle b \rangle \prec R' \wedge (R', Q'') \in \text{bangRel } Rel'$
by (*metis simE*)
then obtain R' **where** $RTrans: R \Longrightarrow a \langle b \rangle \prec R'$ **and** $R'RelQ'': (R', Q'') \in \text{bangRel } Rel'$
by *blast*

from $PTrans RTrans$ **have** $P \parallel R \Longrightarrow \tau \prec P' \parallel R'$ **by** (*rule Weak-Early-Step-Semantics.Comm2*)
moreover from $P'RelQ' R'RelQ''$ **have** $(P' \parallel R', Q' \parallel Q'') \in \text{bangRel } Rel'$ **by** (*rule Rel.BRPar*)
ultimately show $\exists P'. P \parallel R \Longrightarrow \tau \prec P' \wedge (P', Q' \parallel Q'') \in \text{bangRel } Rel'$
by *blast*
qed
qed
next

case(*Close1* $a\ x\ Q'\ Q''\ Pa\ P$)
hence $IH: \bigwedge Pa. (Pa, !Q) \in \text{bangRel Rel} \longrightarrow ?\text{Sim Pa } (a\langle\nu x\rangle \prec Q'')$ **by**
simp
have $QTrans: Q \longmapsto a\langle x\rangle \prec Q'$ **by fact**
have $xFreshQ: x \# Q$ **by fact**
have $(Pa, Q \parallel !Q) \in \text{bangRel Rel}$ **by fact**
moreover have $xFreshPa: x \# Pa$ **by fact**
ultimately show *?case*
proof(*induct rule: BRParCases*)
case(*BRPar P R*)
have $PRelQ: (P, Q) \in \text{Rel}$ **and** $RBRQ: (R, !Q) \in \text{bangRel Rel}$ **by fact+**
have $x \# P \parallel R$ **by fact**
hence $xFreshP: x \# P$ **and** $xFreshR: x \# R$ **by simp+**
show *?case*
proof(*auto simp add: residual.inject*)
from $PRelQ$ **have** $P \rightsquigarrow \langle \text{Rel}' \rangle Q$ **by**(*rule Sim*)
with $QTrans\ xFreshP$ **obtain** P' **where** $PTrans: P \Longrightarrow a\langle x\rangle \prec P'$ **and**
 $P'RelQ': (P', Q') \in \text{Rel}'$
by(*blast dest: simE*)

from $RBRQ\ xFreshR\ IH$ **have** $\exists R'. R \Longrightarrow a\langle\nu x\rangle \prec R' \wedge (R', Q') \in$
 $\text{bangRel Rel}'$
by(*metis simE*)
then obtain R' **where** $RTrans: R \Longrightarrow a\langle\nu x\rangle \prec R'$ **and** $R'RelQ'': (R',$
 $Q'') \in \text{bangRel Rel}'$
by *blast*

from $PTrans\ RTrans\ xFreshP$ **have** $P \parallel R \Longrightarrow \tau \prec \langle\nu x\rangle(P' \parallel R')$
by(*rule Weak-Early-Step-Semantics.Close1*)
moreover from $P'RelQ'\ R'RelQ''$ **have** $(\langle\nu x\rangle(P' \parallel R'), \langle\nu x\rangle(Q' \parallel$
 $Q'')) \in \text{bangRel Rel}'$
by(*force intro: Rel.BRPar BRRes*)
ultimately show $\exists P'. P \parallel R \Longrightarrow \tau \prec P' \wedge (P', \langle\nu x\rangle(Q' \parallel Q'')) \in$
 $\text{bangRel Rel}'$ **by** *blast*
qed
qed
next
case(*Close2* $a\ x\ Q'\ Q''\ Pa\ P$)
hence $IH: \bigwedge Pa. (Pa, !Q) \in \text{bangRel Rel} \Longrightarrow ?\text{Sim Pa } (a\langle x\rangle \prec Q'')$ **by** *simp*
have $QTrans: Q \longmapsto a\langle\nu x\rangle \prec Q'$ **by fact**
have $xFreshQ: x \# Q$ **by fact**
have $(Pa, Q \parallel !Q) \in \text{bangRel Rel}$ **and** $x \# Pa$ **by fact+**
thus *?case*
proof(*induct rule: BRParCases*)
case(*BRPar P R*)
have $PRelQ: (P, Q) \in \text{Rel}$ **and** $RBRQ: (R, !Q) \in \text{bangRel Rel}$ **by fact+**
have $x \# P \parallel R$ **by fact**
hence $xFreshP: x \# P$ **and** $xFreshR: x \# R$ **by simp+**
show *?case*

```

proof(auto simp add: residual.inject)
  from PRelQ have  $P \rightsquigarrow_{\langle \text{Rel}' \rangle} Q$  by(rule Sim)
  with QTrans xFreshP obtain  $P'$  where  $P \Longrightarrow_{a \langle \nu x \rangle} P'$  and
P'RelQ':  $(P', Q') \in \text{Rel}'$ 
    by(blast dest: simE)

  from RBRQ IH have  $\exists R'. R \Longrightarrow_{a \langle x \rangle} R' \wedge (R', Q') \in \text{bangRel Rel}'$ 
    by auto
    then obtain  $R'$  where  $R \Longrightarrow_{a \langle x \rangle} R'$  and  $R' \text{Rel}Q''$ :  $(R', Q'') \in \text{bangRel Rel}'$ 
    by blast

  from PTrans RTrans xFreshR have  $P \parallel R \Longrightarrow_{\tau} \langle \nu x \rangle (P' \parallel R')$ 
    by(rule Weak-Early-Step-Semantics.Close2)
    moreover from  $P' \text{Rel}Q' R' \text{Rel}Q''$  have  $(\langle \nu x \rangle (P' \parallel R'), \langle \nu x \rangle (Q' \parallel Q'')) \in \text{bangRel Rel}'$ 
    by(force intro: Rel.BRPar BRRes)
    ultimately show  $\exists P'. P \parallel R \Longrightarrow_{\tau} P' \wedge (P', \langle \nu x \rangle (Q' \parallel Q'')) \in \text{bangRel Rel}'$ 
    by blast
  qed
qed
next
  case(Bang Rs Pa P)
  hence  $IH: \bigwedge Pa. (Pa, Q \parallel !Q) \in \text{bangRel Rel} \Longrightarrow ?\text{Sim Pa Rs}$  by simp
  have  $(Pa, !Q) \in \text{bangRel Rel}$  by fact
  thus ?case
  proof(induct rule: BRBangCases)
    case(BRBang P)
    have  $P \text{Rel}Q: (P, Q) \in \text{Rel}$  by fact
    hence  $(!P, !Q) \in \text{bangRel Rel}$  by(rule Rel.BRBang)
    with  $P \text{Rel}Q$  have  $(P \parallel !P, Q \parallel !Q) \in \text{bangRel Rel}$  by(rule BRPar)
    with  $IH$  have  $?\text{Sim} (P \parallel !P) Rs$  by simp
    thus ?case by(force intro: Weak-Early-Step-Semantics.Bang)
  qed
qed
}

  moreover from  $P \text{Rel}Q$  have  $(!P, !Q) \in \text{bangRel Rel}$  by(rule BRBang)
  ultimately show  $?\text{thesis}$  by(auto simp add: weakStepSimulation-def)
qed

end

theory Weak-Early-Sim-Pres
  imports Weak-Early-Sim
begin

lemma tauPres:
  fixes  $P$  :: pi

```

```

and Q :: pi
and Rel :: (pi × pi) set
and Rel' :: (pi × pi) set

assumes PRelQ: (P, Q) ∈ Rel

shows τ.(P) ~><Rel> τ.(Q)
proof(induct rule: simCases)
  case(Bound Q' a x)
  have τ.(Q) ⟶ a<νx> < Q' by fact
  hence False by(induct rule: tauCases', auto)
  thus ?case by simp
next
  case(Free Q' α)
  have τ.(Q) ⟶ (α < Q') by fact
  thus ?case
  proof(induct rule: tauCases', auto simp only: pi.inject residual.inject)
    have τ.(P) ⟶ τ < P by(rule Tau)
    with PRelQ show ∃ P'. τ.(P) ⟶ τ < P' ∧ (P', Q) ∈ Rel by blast
  qed
qed

lemma inputPres:
  fixes P :: pi
  and x :: name
  and Q :: pi
  and a :: name
  and Rel :: (pi × pi) set

  assumes PRelQ: ∀ y. (P[x::=y], Q[x::=y]) ∈ Rel
  and Eqvt: eqvt Rel

  shows a<x>.P ~><Rel> a<x>.Q
using Eqvt
proof(induct rule: simCasesCont[where C=(x, a, P, Q)])
  case(Bound b y Q')
  from ⟨y # (x, a, P, Q)⟩ have y ≠ x y ≠ a y # P y # Q by simp+
  from ⟨a<x>.Q ⟶ b<νy> < Q'⟩ ⟨y ≠ a⟩ ⟨y ≠ x⟩ ⟨y # Q⟩ show ?case
  by(erule-tac inputCases') auto
next
  case(Free α Q')
  from ⟨a<x>.Q ⟶ α < Q'⟩
  show ?case
  proof(induct rule: inputCases)
    case(cInput u)
    have a<x>.P ⟶ (a<u>) < P[x::=u]
    by(rule Input)
    moreover from PRelQ have (P[x::=u], Q[x::=u]) ∈ Rel by auto
    ultimately show ?case by blast
  qed

```

qed
qed

lemma *outputPres*:

fixes $P :: pi$
and $Q :: pi$
and $a :: name$
and $b :: name$
and $Rel :: (pi \times pi) set$

assumes $PRelQ: (P, Q) \in Rel$

shows $a\{b\}.P \rightsquigarrow_{\langle Rel \rangle} a\{b\}.Q$

proof(*induct rule: simCases*)

case(*Bound Q' c x*)

have $a\{b\}.Q \mapsto_{c\langle \nu x \rangle} \prec Q'$ by *fact*

hence *False* by(*induct rule: outputCases', auto*)

thus ?*case* by *simp*

next

case(*Free Q' α*)

have $a\{b\}.Q \mapsto_{\alpha} \prec Q'$ by *fact*

thus $\exists P'. a\{b\}.P \Longrightarrow \hat{\alpha} \prec P' \wedge (P', Q') \in Rel$

proof(*induct rule: outputCases', auto simp add: pi.inject residual.inject*)

have $a\{b\}.P \Longrightarrow \hat{a[b]} \prec P$ by(*rule Output*)

with $PRelQ$ show $\exists P'. a\{b\}.P \Longrightarrow \hat{a[b]} \prec P' \wedge (P', Q) \in Rel$ by *blast*

qed

qed

lemma *matchPres*:

fixes $P :: pi$
and $Q :: pi$
and $a :: name$
and $b :: name$
and $Rel :: (pi \times pi) set$
and $Rel' :: (pi \times pi) set$

assumes $PSimQ: P \rightsquigarrow_{\langle Rel \rangle} Q$

and $RelRel': Rel \subseteq Rel'$

and $RelStay: \bigwedge R S c. (R, S) \in Rel \Longrightarrow ([c\curvearrowright]R, S) \in Rel$

shows $[a\curvearrowright]P \rightsquigarrow_{\langle Rel' \rangle} [a\curvearrowright]Q$

proof(*induct rule: simCases*)

case(*Bound Q' c x*)

have $x \# [a\curvearrowright]P$ by *fact*

hence $xFreshP: (x::name) \# P$ by *simp*

have $[a\curvearrowright]Q \mapsto_{c\langle \nu x \rangle} \prec Q'$ by *fact*

thus ?*case*

proof(*induct rule: matchCases*)

case *Match*

```

have  $Q \mapsto c \langle \nu x \rangle \prec Q'$  by fact
with  $PSimQ$   $xFreshP$  obtain  $P'$  where  $PTrans: P \Longrightarrow c \langle \nu x \rangle \prec P'$ 
and  $P'RelQ': (P', Q') \in Rel$ 
  by(blast dest: simE)
from  $PTrans$  have  $[a \frown a]P \Longrightarrow c \langle \nu x \rangle \prec P'$  by(rule Weak-Early-Step-Semantics.Match)
moreover from  $P'RelQ' RelRel'$  have  $(P', Q') \in Rel'$  by blast
ultimately show ?case by blast
qed
next
case(Free Q'  $\alpha$ )
have  $[a \frown b]Q \mapsto \alpha \prec Q'$  by fact
thus ?case
proof(induct rule: matchCases)
  case Match
  have  $Q \mapsto \alpha \prec Q'$  by fact
  with  $PSimQ$  obtain  $P'$  where  $P \Longrightarrow \hat{\alpha} \prec P'$  and  $(P', Q') \in Rel$ 
  by(blast dest: simE)
  thus ?case
proof(induct rule: transitionCases)
  case Step
  have  $P \Longrightarrow \alpha \prec P'$  by fact
  hence  $[a \frown a]P \Longrightarrow \alpha \prec P'$  by(rule Weak-Early-Step-Semantics.Match)
  with  $RelRel' \langle (P', Q') \in Rel \rangle$  show ?case by(force simp add: weakFreeTransition-def)
  next
  case Stay
  have  $[a \frown a]P \Longrightarrow \hat{\tau} \prec [a \frown a]P$  by(simp add: weakFreeTransition-def)
  moreover from  $\langle (P', Q') \in Rel \rangle$  have  $([a \frown a]P, Q') \in Rel$  by(blast intro: RelStay)
  ultimately show ?case using  $RelRel'$  by blast
qed
qed
qed

```

lemma *mismatchPres:*

```

fixes  $P :: pi$ 
and  $Q :: pi$ 
and  $a :: name$ 
and  $b :: name$ 
and  $Rel :: (pi \times pi)$  set
and  $Rel' :: (pi \times pi)$  set

```

```

assumes  $PSimQ: P \rightsquigarrow \langle Rel \rangle Q$ 
and  $RelRel': Rel \subseteq Rel'$ 
and  $RelStay: \bigwedge R S c d. \llbracket (R, S) \in Rel; c \neq d \rrbracket \Longrightarrow \llbracket [c \neq d]R, S \rrbracket \in Rel$ 

```

shows $[a \neq b]P \rightsquigarrow \langle Rel' \rangle [a \neq b]Q$

```

proof(induct rule: simCases)
  case(Bound Q' c x)

```

```

have  $x \# [a \neq b]P$  by fact
hence  $xFreshP: (x::name) \# P$  by simp
have  $[a \neq b]Q \mapsto c \langle \nu x \rangle \prec Q'$  by fact
thus ?case
proof(induct rule: mismatchCases)
  case Mismatch
  have  $a \neq b$  by fact
  have  $Q \mapsto c \langle \nu x \rangle \prec Q'$  by fact
  with  $PSimQ\ xFreshP$  obtain  $P'$  where  $PTrans: P \Longrightarrow c \langle \nu x \rangle \prec P'$ 
    and  $P'RelQ': (P', Q') \in Rel$ 
    by(blast dest: simE)
  from  $PTrans\ a \neq b$  have  $[a \neq b]P \Longrightarrow c \langle \nu x \rangle \prec P'$  by(rule Weak-Early-Step-Semantics.Mismatch)
  moreover from  $P'RelQ'\ RelRel'$  have  $(P', Q') \in Rel'$  by blast
  ultimately show ?case by blast
qed
next
case(Free  $Q'\ \alpha$ )
have  $[a \neq b]Q \mapsto \alpha \prec Q'$  by fact
thus ?case
proof(induct rule: mismatchCases)
  case Mismatch
  have  $a \neq b$  by fact
  have  $Q \mapsto \alpha \prec Q'$  by fact
  with  $PSimQ$  obtain  $P'$  where  $P \Longrightarrow \hat{\alpha} \prec P'$  and  $(P', Q') \in Rel$ 
    by(blast dest: simE)
  thus ?case
proof(induct rule: transitionCases)
  case Step
  have  $P \Longrightarrow \alpha \prec P'$  by fact
  hence  $[a \neq b]P \Longrightarrow \alpha \prec P'$  using  $a \neq b$  by(rule Weak-Early-Step-Semantics.Mismatch)
  with  $RelRel' \langle (P', Q') \in Rel \rangle$  show ?case by(force simp add: weakFreeTransition-def)
next
case Stay
  have  $[a \neq b]P \Longrightarrow \hat{\tau} \prec [a \neq b]P$  by(simp add: weakFreeTransition-def)
  moreover from  $\langle (P, Q') \in Rel \rangle\ a \neq b$  have  $([a \neq b]P, Q') \in Rel$  by(blast
intro: RelStay)
  ultimately show ?case using  $RelRel'$  by blast
qed
qed
qed

lemma parCompose:
  fixes  $P$     ::  $\pi$ 
  and  $Q$      ::  $\pi$ 
  and  $R$      ::  $\pi$ 
  and  $S$      ::  $\pi$ 
  and  $Rel$    ::  $(\pi \times \pi)$  set
  and  $Rel'$  ::  $(\pi \times \pi)$  set

```

and $Rel'' :: (pi \times pi)$ set
assumes $PSimQ: P \rightsquigarrow_{\langle Rel \rangle} Q$
and $RSimT: R \rightsquigarrow_{\langle Rel' \rangle} S$
and $PRelQ: (P, Q) \in Rel$
and $RRel'T: (R, S) \in Rel'$
and $Par: \bigwedge P' Q' R' S'. \llbracket (P', Q') \in Rel; (R', S') \in Rel' \rrbracket \implies (P' \parallel R', Q' \parallel S') \in Rel''$
and $Res: \bigwedge T U x. (T, U) \in Rel'' \implies (\langle \nu x \rangle T, \langle \nu x \rangle U) \in Rel''$

shows $P \parallel R \rightsquigarrow_{\langle Rel'' \rangle} Q \parallel S$
proof –
show *?thesis*
proof(*induct rule: simCases*)
case(*Bound Q' a x*)
have $x \# P \parallel R$ **by** *fact*
hence $xFreshP: x \# P$ **and** $xFreshR: x \# R$ **by** *simp+*
have $Q \parallel S \mapsto a \langle \nu x \rangle \prec Q'$ **by** *fact*
thus *?case*
proof(*induct rule: parCasesB*)
case(*cPar1 Q'*)
have $QTrans: Q \mapsto a \langle \nu x \rangle \prec Q'$ **and** $xFreshT: x \# S$ **by** *fact+*
from $xFreshP$ $PSimQ$ $QTrans$ **obtain** P' **where** $PTrans: P \implies a \langle \nu x \rangle \prec P'$
and $P'RelQ': (P', Q') \in Rel$
by(*blast dest: simE*)
from $PTrans$ $xFreshR$ **have** $P \parallel R \implies a \langle \nu x \rangle \prec (P' \parallel R)$ **by**(*rule Weak-Early-Step-Semantics.Par1B*)
moreover from $P'RelQ'$ $RRel'T$ **have** $(P' \parallel R, Q' \parallel S) \in Rel''$ **by**(*rule Par*)
ultimately show *?case by blast*
next
case(*cPar2 S'*)
have $STrans: S \mapsto a \langle \nu x \rangle \prec S'$ **and** $xFreshQ: x \# Q$ **by** *fact+*
from $xFreshR$ $RSimT$ $STrans$ **obtain** R' **where** $RTrans: R \implies a \langle \nu x \rangle \prec R'$
and $R'Rel'T': (R', S') \in Rel'$
by(*blast dest: simE*)
from $RTrans$ $xFreshP$ $xFreshR$ **have** $ParTrans: P \parallel R \implies a \langle \nu x \rangle \prec (P \parallel R')$
by(*blast intro: Weak-Early-Step-Semantics.Par2B*)
moreover from $PRelQ$ $R'Rel'T'$ **have** $(P \parallel R', Q \parallel S') \in Rel''$ **by**(*rule Par*)
ultimately show *?case by blast*
qed
next
case(*Free QT' α*)
have $Q \parallel S \mapsto \alpha \prec QT'$ **by** *fact*
thus *?case*
proof(*induct rule: parCasesF[of - - - - (P, R)]*)
case(*cPar1 Q'*)
have $Q \mapsto \alpha \prec Q'$ **by** *fact*
with $PSimQ$ **obtain** P' **where** $PTrans: P \implies \hat{\alpha} \prec P'$ **and** $PRel: (P', Q') \in Rel$

by(*blast dest: simE*)
from $PTrans$ **have** $Trans: P \parallel R \Longrightarrow \hat{\alpha} \prec P' \parallel R$ **by**(*rule Weak-Early-Semantics.Par1F*)
moreover from $PRel RRel'T$ **have** $(P' \parallel R, Q' \parallel S) \in Rel''$ **by**(*blast intro:*
Par)
ultimately show *?case by blast*
next
case(*cPar2 S'*)
have $S \mapsto \alpha \prec S'$ **by** *fact*
with $RSimT$ **obtain** R' **where** $RTrans: R \Longrightarrow \hat{\alpha} \prec R'$ **and** $RRel: (R', S') \in Rel'$
 $\in Rel'$
by(*blast dest: simE*)
from $RTrans$ **have** $Trans: P \parallel R \Longrightarrow \hat{\alpha} \prec P \parallel R'$ **by**(*rule Weak-Early-Semantics.Par2F*)
moreover from $PRelQ RRel$ **have** $(P \parallel R', Q \parallel S') \in Rel''$ **by**(*blast intro:*
Par)
ultimately show *?case by blast*
next
case(*cComm1 Q' S' a b*)
have $QTrans: Q \mapsto a \langle b \rangle \prec Q'$ **and** $STrans: S \mapsto a[b] \prec S'$ **by** *fact+*

from $PSimQ QTrans$ **obtain** P' **where** $PTrans: P \Longrightarrow a \langle b \rangle \prec P'$
and $P'RelQ': (P', Q') \in Rel$
by(*fastforce dest: simE simp add: weakFreeTransition-def*)

from $RSimT STrans$ **obtain** R' **where** $RTrans: R \Longrightarrow a[b] \prec R'$
and $RRel: (R', S') \in Rel'$
by(*fastforce dest: simE simp add: weakFreeTransition-def*)

from $PTrans RTrans$ **have** $P \parallel R \Longrightarrow \tau \prec P' \parallel R'$ **by**(*rule Weak-Early-Step-Semantics.Comm1*)
hence $P \parallel R \Longrightarrow \hat{\tau} \prec P' \parallel R'$
by(*auto simp add: trancl-into-rtrancl dest: Weak-Early-Step-Semantics.tauTransitionChain*)

moreover from $P'RelQ' RRel$ **have** $(P' \parallel R', Q' \parallel S') \in Rel''$ **by**(*rule Par*)
ultimately show *?case by blast*
next
case(*cComm2 Q' S' a b*)
have $QTrans: Q \mapsto a[b] \prec Q'$ **and** $STrans: S \mapsto a \langle b \rangle \prec S'$ **by** *fact+*

from $PSimQ QTrans$ **obtain** P' **where** $PTrans: P \Longrightarrow a[b] \prec P'$
and $PRel: (P', Q') \in Rel$
by(*fastforce dest: simE simp add: weakFreeTransition-def*)

from $RSimT STrans$ **obtain** R' **where** $RTrans: R \Longrightarrow a \langle b \rangle \prec R'$
and $R'Rel'T': (R', S') \in Rel'$
by(*fastforce dest: simE simp add: weakFreeTransition-def*)

from $PTrans RTrans$ **have** $P \parallel R \Longrightarrow \tau \prec P' \parallel R'$ **by**(*rule Weak-Early-Step-Semantics.Comm2*)
hence $P \parallel R \Longrightarrow \hat{\tau} \prec P' \parallel R'$
by(*auto simp add: trancl-into-rtrancl dest: Weak-Early-Step-Semantics.tauTransitionChain*)
moreover from $PRel R'Rel'T'$ **have** $(P' \parallel R', Q' \parallel S') \in Rel''$ **by**(*rule Par*)

ultimately show *?case by blast*
next
case(*cClose1* $Q' S' a x$)
have $QTrans: Q \mapsto a \langle x \rangle \prec Q'$ **and** $STrans: S \mapsto a \langle \nu x \rangle \prec S'$ **by** *fact+*
have $x \# (P, R)$ **by** *fact*
hence $xFreshP: x \# P$ **and** $xFreshR: x \# R$ **by**(*simp add: fresh-prod*)

from $PSimQ$ $QTrans$ **obtain** P' **where** $PTrans: P \Longrightarrow a \langle x \rangle \prec P'$
and $P'RelQ': (P', Q') \in Rel$
by(*fastforce dest: simE simp add: weakFreeTransition-def*)

from $RSimT$ $STrans$ $xFreshR$ **obtain** R' **where** $RTrans: R \Longrightarrow a \langle \nu x \rangle \prec R'$
and $R'Rel'T': (R', S') \in Rel'$
by(*blast dest: simE*)

from $PTrans$ $RTrans$ $xFreshP$ **have** $Trans: P \parallel R \Longrightarrow \tau \prec \langle \nu x \rangle (P' \parallel R')$
by(*rule Weak-Early-Step-Semantics.Close1*)
hence $P \parallel R \Longrightarrow \hat{\tau} \prec \langle \nu x \rangle (P' \parallel R')$
by(*auto simp add: trancl-into-rtrancl dest: Weak-Early-Step-Semantics.tauTransitionChain*)
moreover from $P'RelQ' R'Rel'T'$ **have** $\langle \nu x \rangle (P' \parallel R'), \langle \nu x \rangle (Q' \parallel S')$
 $\in Rel''$
by(*blast intro: Par Res*)
ultimately show *?case by blast*
next
case(*cClose2* $Q' S' a x$)
have $QTrans: Q \mapsto a \langle \nu x \rangle \prec Q'$ **and** $STrans: S \mapsto a \langle x \rangle \prec S'$ **by** *fact+*
have $x \# (P, R)$ **by** *fact*
hence $xFreshR: x \# R$ **and** $xFreshP: x \# P$ **by**(*simp add: fresh-prod*)

from $PSimQ$ $QTrans$ $xFreshP$ **obtain** P' **where** $PTrans: P \Longrightarrow a \langle \nu x \rangle \prec P'$
and $P'RelQ': (P', Q') \in Rel$
by(*blast dest: simE*)

from $RSimT$ $STrans$ **obtain** R' **where** $RTrans: R \Longrightarrow a \langle x \rangle \prec R'$
and $R'Rel'T': (R', S') \in Rel'$
by(*fastforce dest: simE simp add: weakFreeTransition-def*)
from $PTrans$ $RTrans$ $xFreshR$ **have** $Trans: P \parallel R \Longrightarrow \tau \prec \langle \nu x \rangle (P' \parallel R')$
by(*rule Weak-Early-Step-Semantics.Close2*)
hence $P \parallel R \Longrightarrow \hat{\tau} \prec \langle \nu x \rangle (P' \parallel R')$
by(*auto simp add: trancl-into-rtrancl dest: Weak-Early-Step-Semantics.tauTransitionChain*)
moreover from $P'RelQ' R'Rel'T'$ **have** $\langle \nu x \rangle (P' \parallel R'), \langle \nu x \rangle (Q' \parallel S')$
 $\in Rel''$
by(*blast intro: Par Res*)
ultimately show *?case by blast*
qed
qed
qed

lemma *parPres*:

```

fixes  $P :: pi$ 
and  $Q :: pi$ 
and  $R :: pi$ 
and  $a :: name$ 
and  $Rel :: (pi \times pi) \text{ set}$ 
and  $Rel' :: (pi \times pi) \text{ set}$ 

assumes  $PSimQ: P \rightsquigarrow_{\langle Rel \rangle} Q$ 
and  $PRelQ: (P, Q) \in Rel$ 
and  $Par: \bigwedge S T U. (S, T) \in Rel \implies (S \parallel U, T \parallel U) \in Rel'$ 
and  $Res: \bigwedge S T x. (S, T) \in Rel' \implies (\langle \nu x \rangle S, \langle \nu x \rangle T) \in Rel'$ 

shows  $P \parallel R \rightsquigarrow_{\langle Rel' \rangle} Q \parallel R$ 
proof –
  note  $PSimQ$ 
  moreover have  $RSimR: R \rightsquigarrow_{\langle Id \rangle} R$  by(auto intro: reflexive)
  moreover note  $PRelQ$  moreover have  $(R, R) \in Id$  by auto
  moreover from  $Par$  have  $\bigwedge P Q R T. [(P, Q) \in Rel; (R, T) \in Id] \implies (P \parallel R, Q \parallel T) \in Rel'$ 
    by auto
  ultimately show ?thesis using  $Res$  by(rule parCompose)
qed

lemma  $resPres$ :
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $Rel :: (pi \times pi) \text{ set}$ 
  and  $x :: name$ 
  and  $Rel' :: (pi \times pi) \text{ set}$ 

  assumes  $PSimQ: P \rightsquigarrow_{\langle Rel \rangle} Q$ 
  and  $ResRel: \bigwedge R S y. (R, S) \in Rel \implies (\langle \nu y \rangle R, \langle \nu y \rangle S) \in Rel'$ 
  and  $RelRel': Rel \subseteq Rel'$ 
  and  $EqvtRel: eqvt Rel$ 
  and  $EqvtRel': eqvt Rel'$ 

  shows  $\langle \nu x \rangle P \rightsquigarrow_{\langle Rel' \rangle} \langle \nu x \rangle Q$ 
proof –
  from  $EqvtRel'$  show ?thesis
  proof(induct rule: simCasesCont[where  $C=(P, x)$ ])
    case(Bound a y Q')
    have  $Trans: \langle \nu x \rangle Q \mapsto a \langle \nu y \rangle \prec Q'$  by fact
    have  $y \# (P, x)$  by fact
    hence  $y \text{ineq}x: y \neq x$  and  $y \text{Fresh}P: y \# P$  by(simp add: fresh-prod)+
    from  $Trans$   $y \text{ineq}x$  show ?case
    proof(induct rule: resCasesB)
      case(Open Q')
      have  $Q \text{Trans}: Q \mapsto a[x] \prec Q'$  and  $a \text{ineq}x: a \neq x$  by fact+

```

```

from PSimQ QTrans obtain  $P'$  where  $PTrans: P \Longrightarrow \hat{a}[x] \prec P'$ 
and  $P'RelQ': (P', Q') \in Rel$ 
  by(blast dest: simE)

from  $PTrans$  aineqx have  $\langle \nu x \rangle P \Longrightarrow a \langle \nu x \rangle \prec P'$ 
  by(force intro: Weak-Early-Step-Semantics.Open simp add: weakFreeTransition-def)
with  $\langle y \# P \rangle \langle y \neq x \rangle$  have  $\langle \nu x \rangle P \Longrightarrow a \langle \nu y \rangle \prec ((y, x) \cdot P')$ 
  by(force intro: weakTransitionAlpha simp add: abs-fresh name-swap)
moreover from  $EqvtRel P'RelQ' RelRel'$  have  $((y, x) \cdot P', [(y, x)] \cdot Q') \in Rel'$ 
  by(blast intro: eqvtRelI)
ultimately show ?case by blast
next
case(Res Q')
have  $QTrans: Q \mapsto a \langle \nu y \rangle \prec Q'$  and xineqa: x ≠ a by fact+

from PSimQ yFreshP QTrans obtain  $P'$  where  $PTrans: P \Longrightarrow a \langle \nu y \rangle \prec P'$ 
and  $P'RelQ': (P', Q') \in Rel$ 
  by(blast dest: simE)
from  $PTrans$  xineqa yineqx yFreshP have  $ResTrans: \langle \nu x \rangle P \Longrightarrow a \langle \nu y \rangle \prec \langle \nu x \rangle P'$ 
  by(blast intro: Weak-Early-Step-Semantics.ResB)
moreover from  $P'RelQ'$  have  $((\langle \nu x \rangle P'), (\langle \nu x \rangle Q')) \in Rel'$ 
  by(rule ResRel)
ultimately show ?case by blast
qed
next
case(Free α Q')
have  $QTrans: \langle \nu x \rangle Q \mapsto \alpha \prec Q'$  by fact
have  $\exists c::name. c \# (P, Q, Q', \alpha)$  by(blast intro: name-exists-fresh)
then obtain  $c::name$  where  $cFreshQ: c \# Q$  and  $cFreshAlpha: c \# \alpha$  and
 $cFreshQ': c \# Q'$  and  $cFreshP: c \# P$ 
  by(force simp add: fresh-prod)
from  $cFreshP$  have  $\langle \nu x \rangle P = \langle \nu c \rangle ((x, c) \cdot P)$  by(simp add: alphaRes)
moreover have  $\exists P'. \langle \nu c \rangle ((x, c) \cdot P) \Longrightarrow \alpha \prec P' \wedge (P', Q') \in Rel'$ 
proof –
  from  $QTrans$   $cFreshQ$  have  $\langle \nu c \rangle ((x, c) \cdot Q) \mapsto \alpha \prec Q'$  by(simp add: alphaRes)
  moreover have  $c \# \alpha$  by(rule cFreshAlpha)
  moreover from  $PSimQ EqvtRel$  have  $((x, c) \cdot P) \rightsquigarrow_{\langle Rel \rangle} ((x, c) \cdot Q)$ 
  by(blast intro: eqvtI)
  ultimately show ?thesis
  apply(induct rule: resCasesF, auto simp add: residual.inject pi.inject name-abs-eq)
  by(blast intro: ResF ResRel dest: simE)
qed

ultimately show ?case by force

```

qed
qed

lemma *resChainI*:

fixes $P :: pi$
and $Q :: pi$
and $Rel :: (pi \times pi)$ set
and $lst :: name$ list

assumes *eqvtRel*: $eqvt$ Rel

and $Res: \bigwedge R S y. (R, S) \in Rel \implies (\langle \nu y \rangle R, \langle \nu y \rangle S) \in Rel$
and $PRelQ: P \rightsquigarrow \langle Rel \rangle Q$

shows $(resChain\ lst)\ P \rightsquigarrow \langle Rel \rangle (resChain\ lst)\ Q$

proof –

show *?thesis*

proof(*induct lst*)

from $PRelQ$ **show** $resChain\ []\ P \rightsquigarrow \langle Rel \rangle resChain\ []\ Q$ **by** *simp*

next

fix $a\ lst$

assume $IH: (resChain\ lst)\ P \rightsquigarrow \langle Rel \rangle (resChain\ lst)\ Q$

moreover from Res **have** $\bigwedge P Q a. (P, Q) \in Rel \implies (\langle \nu a \rangle P, \langle \nu a \rangle Q) \in Rel$

by *simp*

moreover have $Rel \subseteq Rel$ **by** *simp*

ultimately have $\langle \nu a \rangle (resChain\ lst)\ P \rightsquigarrow \langle Rel \rangle \langle \nu a \rangle (resChain\ lst)\ Q$ **using** *eqvtRel*

by(*rule-tac resPres*)

thus $resChain\ (a \# lst)\ P \rightsquigarrow \langle Rel \rangle resChain\ (a \# lst)\ Q$

by *simp*

qed

qed

lemma *bangPres*:

fixes $P :: pi$
and $Q :: pi$
and $Rel :: (pi \times pi)$ set

assumes $PRelQ: (P, Q) \in Rel$

and $Sim: \bigwedge R S. (R, S) \in Rel \implies R \rightsquigarrow \langle Rel \rangle S$

and $ParComp: \bigwedge R S T U. [(R, S) \in Rel; (T, U) \in Rel] \implies (R \parallel T, S \parallel U) \in Rel'$

and $Res: \bigwedge R S x. (R, S) \in Rel' \implies (\langle \nu x \rangle R, \langle \nu x \rangle S) \in Rel'$

and $RelStay: \bigwedge R S. (R \parallel !R, S) \in Rel' \implies (!R, S) \in Rel'$

and $BangRelRel': (bangRel\ Rel) \subseteq Rel'$

and $eqvtRel': eqvt\ Rel'$

shows $!P \rightsquigarrow \langle \text{Rel}' \rangle !Q$
proof –
let $?Sim = \lambda P Rs. (\forall a x Q'. Rs = a \langle \nu x \rangle \prec Q' \longrightarrow x \# P \longrightarrow (\exists P'. P \Longrightarrow a \langle \nu x \rangle \prec P' \wedge (P', Q') \in \text{Rel}')) \wedge$
 $(\forall \alpha Q'. Rs = \alpha \prec Q' \longrightarrow (\exists P'. P \Longrightarrow \hat{\alpha} \prec P' \wedge (P', Q') \in \text{Rel}'))$
{
fix $Rs P$
assume $!Q \mapsto Rs$ **and** $(P, !Q) \in \text{bangRel Rel}$
hence $?Sim P Rs$ **using** $P\text{Rel}Q$
proof(*nominal-induct avoiding: P rule: bangInduct*)
case($\text{Par1B } a \ x \ Q'$)
have $Q\text{Trans}: Q \mapsto a \langle \nu x \rangle \prec Q'$ **and** $x\text{Fresh}Q: x \# Q$ **by** fact+
have $(P, Q \parallel !Q) \in \text{bangRel Rel}$ **and** $x \# P$ **by** fact+
thus $?case$
proof(*induct rule: BRParCases*)
case($\text{BRPar } P \ R$)
have $P\text{Rel}Q: (P, Q) \in \text{Rel}$ **and** $R\text{BangRel}T: (R, !Q) \in \text{bangRel Rel}$ **by**
 fact+
have $x \# P \parallel R$ **by** fact
hence $x\text{Fresh}P: x \# P$ **and** $x\text{Fresh}R: x \# R$ **by** simp+
from $P\text{Rel}Q$ **have** $P\text{Sim}Q: P \rightsquigarrow \langle \text{Rel} \rangle Q$ **by**(*rule Sim*)
from $\langle x \# P \rangle \langle x \# Q \rangle$ **show** $?case$
proof(*auto simp add: residual.inject alpha' name-fresh-fresh*)
from $P\text{Sim}Q \ Q\text{Trans} \ x\text{Fresh}P$ **obtain** P' **where** $P\text{Trans}: P \Longrightarrow a \langle \nu x \rangle \prec P'$
 $\prec P'$
and $P'\text{Rel}Q': (P', Q') \in \text{Rel}$
by(*blast dest: simE*)
from $P\text{Trans} \ x\text{Fresh}R$ **have** $P \parallel R \Longrightarrow a \langle \nu x \rangle \prec (P' \parallel R)$
by(*rule Weak-Early-Step-Semantics.Par1B*)
moreover from $P'\text{Rel}Q' \ R\text{BangRel}T \ \text{BangRelRel}'$ **have** $(P' \parallel R, Q' \parallel !Q) \in \text{Rel}'$
by(*blast intro: Rel.BRPar*)
ultimately show $\exists P'. P \parallel R \Longrightarrow a \langle \nu x \rangle \prec P' \wedge (P', Q' \parallel !Q) \in \text{Rel}'$
by blast
next
fix y
assume $(y::\text{name}) \# Q'$ **and** $y \# P$ **and** $y \# R$
from $Q\text{Trans} \langle y \# Q' \rangle$ **have** $Q \mapsto a \langle \nu y \rangle \prec ([(x, y)] \cdot Q')$ **by**(*simp add: alphaBoundOutput*)
with $P\text{Sim}Q \langle y \# P \rangle$ **obtain** P' **where** $P\text{Trans}: P \Longrightarrow a \langle \nu y \rangle \prec P'$
and $P'\text{Rel}Q': (P', [(x, y)] \cdot Q') \in \text{Rel}$
by(*blast dest: simE*)
from $P\text{Trans} \langle y \# R \rangle$ **have** $P \parallel R \Longrightarrow a \langle \nu y \rangle \prec (P' \parallel R)$ **by**(*rule Weak-Early-Step-Semantics.Par1B*)
moreover from $P'\text{Rel}Q' \ R\text{BangRel}T \ \text{BangRelRel}'$ **have** $(P' \parallel R, [(y, x)] \cdot Q') \parallel !Q) \in \text{Rel}'$
by(*fastforce intro: Rel.BRPar simp add: name-swap*)
ultimately show $\exists P'. P \parallel R \Longrightarrow a \langle \nu y \rangle \prec P' \wedge (P', [(y, x)] \cdot Q') \parallel$

```

!Q) ∈ Rel' by blast
  qed
  qed
  next
  case(Par1F α Q' P)
  have QTrans: Q ⟶ α < Q' by fact
  have (P, Q || !Q) ∈ bangRel Rel by fact
  thus ?case
  proof(induct rule: BRParCases)
    case(BRPar P R)
    have PRelQ: (P, Q) ∈ Rel and RBangRelQ: (R, !Q) ∈ bangRel Rel by
fact+
    show ?case
    proof(auto simp add: residual.inject)
      from PRelQ have P ~<Rel> Q by(rule Sim)
      with QTrans obtain P' where PTrans: P ⟶ α < P' and P'RelQ': (P',
Q') ∈ Rel
      by(blast dest: simE)

      from PTrans have P || R ⟶ α < P' || R by(rule Weak-Early-Semantics.Par1F)
      moreover from P'RelQ' RBangRelQ have (P' || R, Q' || !Q) ∈ bangRel
Rel
      by(rule Rel.BRPar)
      ultimately show ∃ P'. P || R ⟶ α < P' ∧ (P', Q' || !Q) ∈ Rel' using
BangRelRel' by blast
    qed
    qed
    next
    case(Par2B a x Q' P)
    hence IH: ∧P. (P, !Q) ∈ bangRel Rel ⟹ ?Sim P (a<νx> < Q') by simp
    have xFreshQ: x ‡ Q by fact
    have (P, Q || !Q) ∈ bangRel Rel and x ‡ P by fact+
    thus ?case
    proof(induct rule: BRParCases)
      case(BRPar P R)
      have PRelQ: (P, Q) ∈ Rel and RBangRelQ: (R, !Q) ∈ bangRel Rel by
fact+
      have x ‡ P || R by fact
      hence xFreshP: x ‡ P and xFreshR: x ‡ R by simp+
      show ?case using ⟨x ‡ Q⟩
      proof(auto simp add: residual.inject alpha' name-fresh-fresh)
        from IH RBangRelQ have ?Sim R (a<νx> < Q') by blast
        with xFreshR obtain R' where RTrans: R ⟶ a<νx> < R' and
R'BangRelQ': (R', Q') ∈ Rel'
        by(blast dest: simE)
        from RTrans xFreshP have P || R ⟶ a<νx> < (P || R')
        by(auto intro: Weak-Early-Step-Semantics.Par2B)
        moreover from PRelQ R'BangRelQ' have (P || R', Q || Q') ∈ Rel'
        by(rule ParComp)
      qed
    qed
  qed

```

ultimately show $\exists P'. P \parallel R \Longrightarrow a \langle \nu x \rangle \prec P' \wedge (P', Q \parallel Q') \in \text{Rel}'$ **by**
blast
next
fix y
assume $(y::\text{name}) \# Q'$ **and** $y \# R$ **and** $y \# P$
from $IH \text{RBangRel}Q$ **have** $?Sim R (a \langle \nu x \rangle \prec Q')$ **by** *blast*
with $\langle y \# Q' \rangle$ **have** $?Sim R (a \langle \nu y \rangle \prec ((x, y)] \cdot Q'))$ **by** (*simp add: alphaBoundOutput*)
with $\langle y \# R \rangle$ **obtain** R' **where** $RTrans: R \Longrightarrow a \langle \nu y \rangle \prec R'$ **and**
 $R' \text{BangRel}Q': (R', [(x, y)] \cdot Q') \in \text{Rel}'$
by (*blast dest: simE*)
from $RTrans \langle y \# P \rangle$ **have** $P \parallel R \Longrightarrow a \langle \nu y \rangle \prec (P \parallel R')$
by (*auto intro: Weak-Early-Step-Semantics.Par2B*)
moreover from $P \text{Rel}Q R' \text{BangRel}Q'$ **have** $(P \parallel R', Q \parallel ((y, x)] \cdot Q'))$
 $\in \text{Rel}'$
by (*fastforce intro: ParComp simp add: name-swap*)
ultimately show $\exists P'. P \parallel R \Longrightarrow a \langle \nu y \rangle \prec P' \wedge (P', Q \parallel ((y, x)] \cdot Q'))$
 $\in \text{Rel}'$ **by** *blast*
qed
qed
next
case ($Par2F \alpha Q' P$)
hence $IH: \bigwedge P. (P, !Q) \in \text{bangRel Rel} \Longrightarrow ?Sim P (\alpha \prec Q')$ **by** *simp*
have $(P, Q \parallel !Q) \in \text{bangRel Rel}$ **by** *fact*
thus *?case*
proof (*induct rule: BRParCases*)
case ($BRPar P R$)
have $P \text{Rel}Q: (P, Q) \in \text{Rel}$ **and** $R \text{BangRel}Q: (R, !Q) \in \text{bangRel Rel}$ **by**
fact+
show *?case*
proof (*auto simp add: residual.inject*)
from $R \text{BangRel}Q$ **have** $?Sim R (\alpha \prec Q')$ **by** (*rule IH*)
then obtain R' **where** $RTrans: R \Longrightarrow \hat{\alpha} \prec R'$ **and** $R' \text{Rel}Q': (R', Q') \in$
 Rel'
by (*blast dest: simE*)
from $RTrans$ **have** $P \parallel R \Longrightarrow \hat{\alpha} \prec P \parallel R'$ **by** (*rule Weak-Early-Semantics.Par2F*)
moreover from $P \text{Rel}Q R' \text{Rel}Q'$ **have** $(P \parallel R', Q \parallel Q') \in \text{Rel}'$ **by** (*rule*
 $ParComp$)
ultimately show $\exists P'. P \parallel R \Longrightarrow \hat{\alpha} \prec P' \wedge (P', Q \parallel Q') \in \text{Rel}'$ **by** *blast*
qed
qed
next
case ($Comm1 a Q' b Q'' P$)
hence $IH: \bigwedge P. (P, !Q) \in \text{bangRel Rel} \Longrightarrow ?Sim P (a[b] \prec Q'')$ **by** *simp*
have $QTrans: Q \mapsto a \langle b \rangle \prec Q'$ **by** *fact*
have $(P, Q \parallel !Q) \in \text{bangRel Rel}$ **by** *fact*
thus *?case*
proof (*induct rule: BRParCases*)
case ($BRPar P R$)

```

      have PRelQ: (P, Q) ∈ Rel and RBangRelQ: (R, !Q) ∈ bangRel Rel by
fact+
    show ?case
    proof(auto simp add: residual.inject)
      from PRelQ have P ~<Rel> Q by(rule Sim)
      with QTrans obtain P' where PTrans: P ⇒a<b> < P' and P'RelQ':
(P', Q') ∈ Rel
      by(fastforce dest: simE simp add: weakFreeTransition-def)

      from RBangRelQ have ?Sim R (a[b] < Q'') by(rule IH)
      then obtain R' where RTrans: R ⇒a[b] < R'
      and R'RelQ'': (R', Q'') ∈ Rel'
      by(fastforce dest: simE simp add: weakFreeTransition-def)
      from PTrans RTrans have P || R ⇒τ < (P' || R')
      by(rule Weak-Early-Step-Semantics.Comm1)
      hence P || R ⇒τ P' || R'
      by(auto simp add: trancl-into-rtrancl dest: Weak-Early-Step-Semantics.tauTransitionChain)
      moreover from P'RelQ' R'RelQ'' have (P' || R', Q' || Q'') ∈ Rel'
      by(rule ParComp)
      ultimately show ∃ P'. (P || R, P') ∈ {(P, P'). P ↦ τ < P'}* ∧ (P',
Q' || Q'') ∈ Rel'
      by auto
    qed
  qed
next
case(Comm2 a b Q' Q'' P)
  hence IH: ∧P. (P, !Q) ∈ bangRel Rel ⇒ ?Sim P (a<b> < Q'') by simp
  have QTrans: Q ↦a[b] < Q' by fact
  have (P, Q || !Q) ∈ bangRel Rel by fact
  thus ?case
  proof(induct rule: BRParCases)
    case(BRPar P R)
      have PRelQ: (P, Q) ∈ Rel and RBangRelQ: (R, !Q) ∈ bangRel Rel by
fact+
      show ?case
      proof(auto simp add: residual.inject)
        from PRelQ have P ~<Rel> Q by(rule Sim)
        with QTrans obtain P' where PTrans: P ⇒a[b] < P' and P'RelQ':
(P', Q') ∈ Rel
        by(fastforce dest: simE simp add: weakFreeTransition-def)

        from RBangRelQ have ?Sim R (a<b> < Q'') by(rule IH)
        then obtain R' where RTrans: R ⇒a<b> < R' and R'BangRelQ'':
(R', Q'') ∈ Rel'
        by(fastforce dest: simE simp add: weakFreeTransition-def)

        from PTrans RTrans have P || R ⇒τ < (P' || R')
        by(rule Weak-Early-Step-Semantics.Comm2)
        hence P || R ⇒τ P' || R'

```

by(*auto simp add: trancl-into-rtrancl dest: Weak-Early-Step-Semantics.tauTransitionChain*)
moreover from $P' \text{Rel} Q' \ R' \text{BangRel} Q''$ **have** $(P' \parallel R', Q' \parallel Q'') \in \text{Rel}'$
by(*rule ParComp*)
ultimately show $\exists P'. (P \parallel R, P') \in \{(P, P'). P \mapsto \tau \prec P'\}^* \wedge (P',$
 $Q' \parallel Q'') \in \text{Rel}'$ **by** *auto*
qed
qed
next
case(*Close1 a x Q' Q'' P*)
hence $IH: \bigwedge P. (P, !Q) \in \text{bangRel Rel} \implies ?\text{Sim } P (a \langle \nu x \rangle \prec Q'')$ **by** *simp*
have $Q \text{Trans}: Q \mapsto a \langle x \rangle \prec Q'$ **by** *fact*
have $(P, Q \parallel !Q) \in \text{bangRel Rel}$ **and** $x \# P$ **by** *fact+*
thus *?case*
proof(*induct rule: BRParCases*)
case(*BRPar P R*)
have $P \text{Rel} Q: (P, Q) \in \text{Rel}$ **and** $R \text{BangRel} Q: (R, !Q) \in \text{bangRel Rel}$ **by**
fact+
have $x \# P \parallel R$ **by** *fact*
hence $x \text{Fresh} R: x \# R$ **and** $x \text{Fresh} P: x \# P$ **by** *simp+*
show *?case*
proof(*auto simp add: residual.inject*)
from $P \text{Rel} Q$ **have** $P \rightsquigarrow \langle \text{Rel} \rangle Q$ **by**(*rule Sim*)
with $Q \text{Trans}$ **obtain** P' **where** $P \text{Trans}: P \implies a \langle x \rangle \prec P'$ **and** $P' \text{Rel} Q':$
 $(P', Q') \in \text{Rel}$
by(*fastforce dest: simE simp add: weakFreeTransition-def*)

from $R \text{BangRel} Q$ **have** $?\text{Sim } R (a \langle \nu x \rangle \prec Q'')$ **by**(*rule IH*)
with $x \text{Fresh} R$ **obtain** R' **where** $R \text{Trans}: R \implies a \langle \nu x \rangle \prec R'$
and $R' \text{Rel} Q'': (R', Q'') \in \text{Rel}'$
by(*blast dest: simE*)

from $P \text{Trans} \ R \text{Trans} \ x \text{Fresh} P$ **have** $P \parallel R \implies \tau \prec \langle \nu x \rangle (P' \parallel R')$
by(*rule Weak-Early-Step-Semantics.Close1*)
moreover from $P' \text{Rel} Q' \ R' \text{Rel} Q''$ **have** $(\langle \nu x \rangle (P' \parallel R'), \langle \nu x \rangle (Q' \parallel$
 $Q'')) \in \text{Rel}'$
by(*force intro: ParComp Res*)
ultimately show $\exists P'. (P \parallel R, P') \in \{(P, P'). P \mapsto \tau \prec P'\}^* \wedge (P',$
 $\langle \nu x \rangle (Q' \parallel Q'')) \in \text{Rel}'$ **by** *auto*
qed
qed
next
case(*Close2 a x Q' Q'' P*)
hence $IH: \bigwedge P. (P, !Q) \in \text{bangRel Rel} \implies ?\text{Sim } P (a \langle x \rangle \prec Q'')$ **by** *simp*
have $Q \text{Trans}: Q \mapsto a \langle \nu x \rangle \prec Q'$ **by** *fact*
have $(P, Q \parallel !Q) \in \text{bangRel Rel}$ **and** $x \# P$ **by** *fact+*
thus *?case*
proof(*induct rule: BRParCases*)
case(*BRPar P R*)
have $P \text{Rel} Q: (P, Q) \in \text{Rel}$ **and** $R \text{BangRel} Q: (R, !Q) \in \text{bangRel Rel}$ **by**

```

fact+
  have  $x \# P \parallel R$  by fact
  hence  $xFreshP: x \# P$  and  $xFreshR: x \# R$  by simp+
  show ?case
  proof(auto simp add: residual.inject)
    from  $PRelQ$  have  $P \rightsquigarrow \langle Rel \rangle Q$  by(rule Sim)
    with  $QTrans$   $xFreshP$  obtain  $P'$  where  $PTrans: P \Longrightarrow a \langle \nu x \rangle \prec P'$ 
      and  $P'RelQ': (P', Q') \in Rel$ 
    by(blast dest: simE)

    from  $RBangRelQ$  have ? $Sim$   $R (a \langle x \rangle \prec Q')$  by(rule IH)
    with  $xFreshR$  obtain  $R'$  where  $RTrans: R \Longrightarrow a \langle x \rangle \prec R'$ 
      and  $R'RelQ'': (R', Q'') \in Rel'$ 
    by(fastforce simp add: weakFreeTransition-def)
    from  $PTrans$   $RTrans$   $xFreshR$  have  $P \parallel R \Longrightarrow \tau \prec \langle \nu x \rangle (P' \parallel R')$ 
    by(rule Weak-Early-Step-Semantics.Close2)
    moreover from  $P'RelQ'$   $R'RelQ''$  have  $(\langle \nu x \rangle (P' \parallel R'), \langle \nu x \rangle (Q' \parallel Q'')) \in Rel'$ 
    by(force intro: ParComp Res)
    ultimately show  $\exists P'. (P \parallel R, P') \in \{(P, P'). P \mapsto \tau \prec P'\}^* \wedge (P', \langle \nu x \rangle (Q' \parallel Q'')) \in Rel'$  by auto
  qed
qed
next
  case(Bang Rs)
  hence  $IH: \bigwedge P. (P, Q \parallel !Q) \in bangRel\ Rel \Longrightarrow ?Sim\ P\ Rs$  by simp
  have  $(P, !Q) \in bangRel\ Rel$  by fact
  thus ?case
  proof(induct rule: BRBangCases)
    case(BRBang P)
    have  $PRelQ: (P, Q) \in Rel$  by fact
    hence  $(!P, !Q) \in bangRel\ Rel$  by(rule Rel.BRBang)
    with  $PRelQ$  have  $(P \parallel !P, Q \parallel !Q) \in bangRel\ Rel$  by(rule Rel.BRPar)
    hence  $IH: ?Sim (P \parallel !P) Rs$  by(rule IH)
    show ?case
  proof(intro conjI allI impI)
    fix  $Q' a x$ 
    assume  $Rs = a \langle \nu x \rangle \prec Q'$  and  $x \# !P$ 
    then obtain  $P'$  where  $PTrans: (P \parallel !P) \Longrightarrow a \langle \nu x \rangle \prec P'$ 
      and  $P'RelQ': (P', Q') \in Rel'$  using  $IH$ 
    by(auto simp add: residual.inject)
    from  $PTrans$  have  $!P \Longrightarrow a \langle \nu x \rangle \prec P'$ 
    by(force intro: Weak-Early-Step-Semantics.Bang simp add: weakFree-Transition-def)
    with  $P'RelQ'$  show  $\exists P'. !P \Longrightarrow a \langle \nu x \rangle \prec P' \wedge (P', Q') \in Rel'$  by blast
  next
    fix  $Q' \alpha$ 
    assume  $Rs = \alpha \prec Q'$ 
    then obtain  $P'$  where  $PTrans: (P \parallel !P) \Longrightarrow \hat{\alpha} \prec P'$ 

```

```

      and P'RelQ': (P', Q') ∈ Rel' using IH
    by auto
  from PTrans show ∃ P'. !P ⇒ α < P' ∧ (P', Q') ∈ Rel' using P'RelQ'
  proof(induct rule: transitionCases)
    case Step
    have P || !P ⇒ α < P' by fact
    hence !P ⇒ α < P' by(rule Weak-Early-Step-Semantics.Bang)
    with P'RelQ' show ?case by(force simp add: weakFreeTransition-def)
  next
    case Stay
    have !P ⇒ τ < !P by(simp add: weakFreeTransition-def)
    moreover assume (P || !P, Q') ∈ Rel'
    hence (!P, Q') ∈ Rel' by(blast intro: RelStay)
    ultimately show ?case by blast
  qed
  qed
  qed
  qed
}
moreover from PRelQ have (!P, !Q) ∈ bangRel Rel by(rule Rel.BRBang)
ultimately show ?thesis by(auto simp add: weakSimulation-def)
qed

```

lemma *bangRelSim*:

```

  fixes P :: pi
  and Q :: pi
  and Rel :: (pi × pi) set
  and Rel'l :: (pi × pi) set

  assumes PBangRelQ: (P, Q) ∈ bangRel Rel
  and Sim: ⋀ R S. (R, S) ∈ Rel ⇒ R ↗<Rel> S

  and ParComp: ⋀ R S T U. [(R, S) ∈ Rel; (T, U) ∈ Rel] ⇒ (R || T, S
  || U) ∈ Rel'
  and Res: ⋀ R S x. (R, S) ∈ Rel' ⇒ (<νx>R, <νx>S) ∈ Rel'

  and RelStay: ⋀ R S. (R || !R, S) ∈ Rel' ⇒ (!R, S) ∈ Rel'
  and BangRelRel': (bangRel Rel) ⊆ Rel'
  and eqvtRel': eqvt Rel'
  and Eqvt: eqvt Rel

```

shows $P \rightsquigarrow \langle \text{Rel}' \rangle Q$

proof –

```

  from PBangRelQ show ?thesis
  proof(induct rule: bangRel.induct)
    case(BRBang P Q)
    have PRelQ: (P, Q) ∈ Rel by fact
    thus ?case using ParComp Res BangRelRel' eqvtRel' Eqvt RelStay Sim
    by(rule-tac bangPres)
  qed

```

```

next
  case(BRPar P Q R T)
  have (P, Q) ∈ Rel by fact
  moreover hence P ~<Rel> Q by(rule Sim)
  moreover have R ~<Rel'> T by fact
  moreover have (R, T) ∈ bangRel Rel by fact
  ultimately show ?case using ParComp eqvRel' Res Eqv BangRelRel'
    by(blast intro: parCompose)
next
  case(BRRes P Q x)
  have P ~<Rel'> Q by fact
  thus ?case using BangRelRel' eqvRel' Res by(blast intro: resPres)
qed
qed

end

```

```

theory Strong-Early-Late-Comp
  imports Strong-Late-Bisim-Subst-SC Strong-Early-Bisim-Subst
begin

```

abbreviation *TransitionsLate-judge* ($\langle \cdot \mapsto_l \cdot \rangle [80, 80] 80$) **where** $P \mapsto_l Rs \equiv \text{transitions } P \text{ } Rs$

abbreviation *TransitionsEarly-judge* ($\langle \cdot \mapsto_e \cdot \rangle [80, 80] 80$) **where** $P \mapsto_e Rs \equiv \text{TransitionsEarly } P \text{ } Rs$

abbreviation *Transitions-InputjudgeLate* ($\langle \cdot \langle \cdot \rangle \prec_l \cdot \rangle [80, 80] 80$) **where** $a \langle x \rangle \prec_l P' \equiv (\text{Late-Semantics.BoundR } (\text{Late-Semantics.InputS } a) \ x \ P')$

abbreviation *Transitions-OutputjudgeLate* ($\langle \cdot [-] \prec_l \cdot \rangle [80, 80] 80$) **where** $a[b] \prec_l P' \equiv (\text{Late-Semantics.FreeR } (\text{Late-Semantics.OutputR } a \ b) \ P')$

abbreviation *Transitions-BoundOutputjudgeLate* ($\langle \cdot \langle \nu \cdot \rangle \prec_l \cdot \rangle [80, 80] 80$) **where** $a \langle \nu x \rangle \prec_l P' \equiv (\text{Late-Semantics.BoundR } (\text{Late-Semantics.BoundOutputS } a) \ x \ P')$

abbreviation *Transitions-TaujudgeLate* ($\langle \tau \prec_l \cdot \rangle 80$) **where** $\tau \prec_l P' \equiv (\text{Late-Semantics.FreeR } \text{Late-Semantics.TauR } P')$

abbreviation *Transitions-InputjudgeEarly* ($\langle \cdot \langle \cdot \rangle \prec_e \cdot \rangle [80, 80] 80$) **where** $a \langle x \rangle \prec_e P' \equiv (\text{Early-Semantics.FreeR } (\text{Early-Semantics.InputR } a \ x) \ P')$

abbreviation *Transitions-OutputjudgeEarly* ($\langle \cdot [-] \prec_e \cdot \rangle [80, 80] 80$) **where** $a[b] \prec_e P' \equiv (\text{Early-Semantics.FreeR } (\text{Early-Semantics.OutputR } a \ b) \ P')$

abbreviation *Transitions-BoundOutputjudgeEarly* ($\langle \cdot \langle \nu \cdot \rangle \prec_e \cdot \rangle [80, 80] 80$) **where** $a \langle \nu x \rangle \prec_e P' \equiv (\text{Early-Semantics.BoundOutputR } a \ x \ P')$

abbreviation *Transitions-TaujudgeEarly* ($\langle \tau \prec_e \cdot \rangle 80$) **where** $\tau \prec_e P' \equiv (\text{Early-Semantics.FreeR } \text{Early-Semantics.TauR } P')$

lemma *earlyLateOutput*:

```

  fixes P :: pi
  and a :: name
  and b :: name
  and P' :: pi

```

```

assumes  $P \mapsto_e a[b] \prec_e P'$ 

shows  $P \mapsto_l a[b] \prec_l P'$ 
using assms
proof(nominal-induct rule: Early-Semantics.outputInduct)
  case(Output a b P)
    show  $?case$  by(rule Late-Semantics.Output)
  next
    case(Match P a b P' c)
    have  $P \mapsto_l a[b] \prec_l P'$  by fact
    thus  $?case$  by(rule Late-Semantics.Match)
  next
    case(Mismatch P a b P' c d)
    from  $\langle P \mapsto_l a[b] \prec_l P' \rangle \langle c \neq d \rangle$ 
    show  $?case$  by(rule Late-Semantics.Mismatch)
  next
    case(Sum1 P a b P' Q)
    have  $P \mapsto_l a[b] \prec_l P'$  by fact
    thus  $?case$  by(rule Late-Semantics.Sum1)
  next
    case(Sum2 Q a b Q' P)
    have  $Q \mapsto_l a[b] \prec_l Q'$  by fact
    thus  $?case$  by(rule Late-Semantics.Sum2)
  next
    case(Par1 P a b P' Q)
    have  $P \mapsto_l a[b] \prec_l P'$  by fact
    thus  $?case$  by(rule Late-Semantics.Par1F)
  next
    case(Par2 Q a b Q' P)
    have  $Q \mapsto_l a[b] \prec_l Q'$  by fact
    thus  $?case$  by(rule Late-Semantics.Par2F)
  next
    case(Res P a b P' x)
    have  $P \mapsto_l a[b] \prec_l P'$  and  $x \neq a$  and  $x \neq b$  by fact+
    thus  $?case$  by(force intro: Late-Semantics.ResF)
  next
    case(Bang P a b P')
    have  $P \parallel !P \mapsto_l a[b] \prec_l P'$  by fact
    thus  $?case$  by(rule Late-Semantics.Bang)
qed

lemma lateEarlyOutput:
  fixes  $P :: pi$ 
  and  $a :: name$ 
  and  $b :: name$ 
  and  $P' :: pi$ 

  assumes  $P \mapsto_l a[b] \prec_l P'$ 

```

```

  shows  $P \mapsto_e a[b] \prec_e P'$ 
using assms
proof(nominal-induct rule: Late-Semantics.outputInduct)
  case(Output a b P)
  thus ?case by(rule Early-Semantics.Output)
next
  case(Match P a b P' c)
  have  $P \mapsto_e a[b] \prec_e P'$  by fact
  thus ?case by(rule Early-Semantics.Match)
next
  case(Mismatch P a b P' c d)
  have  $P \mapsto_e a[b] \prec_e P'$  and  $c \neq d$  by fact+
  thus ?case by(rule Early-Semantics.Mismatch)
next
  case(Sum1 P a b P' Q)
  have  $P \mapsto_e a[b] \prec_e P'$  by fact
  thus ?case by(rule Early-Semantics.Sum1)
next
  case(Sum2 Q a b Q' P)
  have  $Q \mapsto_e a[b] \prec_e Q'$  by fact
  thus ?case by(rule Early-Semantics.Sum2)
next
  case(Par1 P a b P' Q)
  have  $P \mapsto_e a[b] \prec_e P'$  by fact
  thus ?case by(rule Early-Semantics.Par1F)
next
  case(Par2 Q a b Q' P)
  have  $Q \mapsto_e a[b] \prec_e Q'$  by fact
  thus ?case by(rule Early-Semantics.Par2F)
next
  case(Res P a b P' x)
  have  $P \mapsto_e a[b] \prec_e P'$  and  $x \neq a$  and  $x \neq b$  by fact+
  thus ?case by(force intro: Early-Semantics.ResF)
next
  case(Bang P a b P')
  have  $P \parallel !P \mapsto_e a[b] \prec_e P'$  by fact
  thus ?case by(rule Early-Semantics.Bang)
qed

```

lemma *outputEq*:

```

  fixes  $P :: pi$ 
  and  $a :: name$ 
  and  $b :: name$ 
  and  $P' :: pi$ 

```

```

  shows  $P \mapsto_e a[b] \prec_e P' = P \mapsto_l a[b] \prec_l P'$ 
by(auto intro: lateEarlyOutput earlyLateOutput)

```

```

lemma lateEarlyBoundOutput:
  fixes  $P :: pi$ 
  and  $a :: name$ 
  and  $x :: name$ 
  and  $P' :: pi$ 

  assumes  $P \mapsto_l a \langle \nu x \rangle \prec_l P'$ 

  shows  $P \mapsto_e a \langle \nu x \rangle \prec_e P'$ 
proof -
  have Goal:  $\bigwedge P a x P'. [P \mapsto_l a \langle \nu x \rangle \prec_l P'; x \# P] \implies P \mapsto_e a \langle \nu x \rangle \prec_e P'$ 
  proof -
    fix  $P a x P'$ 
    assume  $P \mapsto_l a \langle \nu x \rangle \prec_l P'$  and  $x \# P$ 
    thus  $P \mapsto_e a \langle \nu x \rangle \prec_e P'$ 
    proof(nominal-induct rule: Late-Semantics.boundOutputInduct)
      case(Match  $P a x P' b$ )
        have  $P \mapsto_e a \langle \nu x \rangle \prec_e P'$  by fact
        thus ?case by(rule Early-Semantics.Match)
      next
        case(Mismatch  $P a x P' b c$ )
        have  $P \mapsto_e a \langle \nu x \rangle \prec_e P'$  and  $b \neq c$  by fact+
        thus ?case by(rule Early-Semantics.Mismatch)
      next
        case(Open  $P a x P'$ )
        have  $P \mapsto_l a[x] \prec_l P'$  by fact
        hence  $P \mapsto_e a[x] \prec_e P'$  by(rule lateEarlyOutput)
        moreover have  $a \neq x$  by fact
        ultimately show ?case by(rule Early-Semantics.Open)
      next
        case(Sum1  $P Q a x P'$ )
        have  $P \mapsto_e a \langle \nu x \rangle \prec_e P'$  by fact
        thus ?case by(rule Early-Semantics.Sum1)
      next
        case(Sum2  $P Q a x Q'$ )
        have  $Q \mapsto_e a \langle \nu x \rangle \prec_e Q'$  by fact
        thus ?case by(rule Early-Semantics.Sum2)
      next
        case(Par1  $P P' Q a x$ )
        have  $P \mapsto_e a \langle \nu x \rangle \prec_e P'$  and  $x \# Q$  by fact+
        thus ?case by(rule Early-Semantics.Par1B)
      next
        case(Par2  $P Q Q' a x$ )
        have  $Q \mapsto_e a \langle \nu x \rangle \prec_e Q'$  and  $x \# P$  by fact+
        thus ?case by(rule Early-Semantics.Par2B)
      next
        case(Res  $P P' a x y$ )
        have  $P \mapsto_e a \langle \nu x \rangle \prec_e P'$  and  $y \neq a$  and  $y \neq x$  by fact+
        thus ?case by(force intro: Early-Semantics.ResB)
  
```

```

next
  case(Bang  $P$   $a$   $x$   $P'$ )
  have  $P \parallel !P \mapsto_e a < \nu x > \prec_e P'$  by fact
  thus ?case by(rule Early-Semantics.Bang)
qed
qed

have  $\exists c :: \text{name}. c \# (P, P', x)$  by(blast intro: name-exists-fresh)
then obtain  $c :: \text{name}$  where  $c \text{Fresh} P$ :  $c \# P$  and  $c \text{Fresh} P'$ :  $c \# P'$  and  $c \neq x$ 
  by(force simp add: fresh-prod)
from assms  $c \text{Fresh} P'$  have  $P \mapsto_l a < \nu c > \prec_l [(x, c)] \cdot P'$ 
  by(simp add: Late-Semantics.alphaBoundResidual)
hence  $P \mapsto_e a < \nu c > \prec_e [(x, c)] \cdot P'$  using  $c \text{Fresh} P$ 
  by(rule Goal)
moreover from  $c \text{Fresh} P' \langle c \neq x \rangle$  have  $x \# [(x, c)] \cdot P'$  by(simp add: name-fresh-left
name-calc)
ultimately show ?thesis by(simp add: Early-Semantics.alphaBoundOutput name-swap)
qed

```

lemma *earlyLateBoundOutput*:

```

fixes  $P :: pi$ 
and  $a :: \text{name}$ 
and  $x :: \text{name}$ 
and  $P' :: pi$ 

assumes  $P \mapsto_e a < \nu x > \prec_e P'$ 

shows  $P \mapsto_l a < \nu x > \prec_l P'$ 
proof –
have Goal:  $\bigwedge P a x P'. [P \mapsto_e a < \nu x > \prec_e P'; x \# P] \implies P \mapsto_l a < \nu x > \prec_l P'$ 
proof –
  fix  $P a x P'$ 
  assume  $P \mapsto_e a < \nu x > \prec_e P'$  and  $x \# P$ 
  thus  $P \mapsto_l a < \nu x > \prec_l P'$ 
  proof(nominal-induct rule: Early-Semantics.boundOutputInduct)
    case(Match  $P a x P' b$ )
    have  $P \mapsto_l a < \nu x > \prec_e P'$  by fact
    thus ?case by(rule Late-Semantics.Match)
  next
    case(Mismatch  $P a x P' b c$ )
    have  $P \mapsto_l a < \nu x > \prec_e P'$  and  $b \neq c$  by fact+
    thus ?case by(rule Late-Semantics.Mismatch)
  next
    case(Open  $P a x P'$ )
    have  $P \mapsto_e a[x] \prec_e P'$  by fact
    hence  $P \mapsto_l a[x] \prec_l P'$  by(rule earlyLateOutput)
    moreover have  $a \neq x$  by fact
    ultimately show ?case by(rule Late-Semantics.Open)
  next

```

```

    case(Sum1 P Q a x P')
    have P  $\mapsto_l a<\nu x> \prec_l P'$  by fact
    thus ?case by(rule Late-Semantics.Sum1)
  next
    case(Sum2 P Q a x Q')
    have Q  $\mapsto_l a<\nu x> \prec_l Q'$  by fact
    thus ?case by(rule Late-Semantics.Sum2)
  next
    case(Par1 P P' Q a x)
    have P  $\mapsto_l a<\nu x> \prec_l P'$  and  $x \# Q$  by fact+
    thus ?case by(rule Late-Semantics.Par1B)
  next
    case(Par2 P Q Q' a x)
    have Q  $\mapsto_l a<\nu x> \prec_l Q'$  and  $x \# P$  by fact+
    thus ?case by(rule Late-Semantics.Par2B)
  next
    case(Res P P' a x y)
    have P  $\mapsto_l a<\nu x> \prec_l P'$  and  $y \neq a$  and  $y \neq x$  by fact+
    thus ?case by(force intro: Late-Semantics.ResB)
  next
    case(Bang P a x P')
    have P  $\parallel !P \mapsto_l a<\nu x> \prec P'$  by fact
    thus ?case by(rule Late-Semantics.Bang)
qed
qed

have  $\exists c::name. c \# (P, P', x)$  by(blast intro: name-exists-fresh)
then obtain  $c::name$  where  $cFreshP: c \# P$  and  $cFreshP': c \# P'$  and  $c \neq x$ 
  by(force simp add: fresh-prod)
from assms cFreshP' have P  $\mapsto_e a<\nu c> \prec_e ((x, c) \cdot P')$ 
  by(simp add: Early-Semantics.alphaBoundOutput)
hence P  $\mapsto_l a<\nu c> \prec_l ((x, c) \cdot P')$  using cFreshP
  by(rule Goal)
moreover from cFreshP'  $\langle c \neq x \rangle$  have  $x \# ((x, c) \cdot P')$  by(simp add: name-fresh-left
name-calc)
ultimately show ?thesis by(simp add: Late-Semantics.alphaBoundResidual name-swap)
qed

lemma BoundOutputEq:
  fixes P :: pi
  and a :: name
  and x :: name
  and P' :: pi

  shows P  $\mapsto_e a<\nu x> \prec_e P' = P \mapsto_l a<\nu x> \prec_l P'$ 
by(auto intro: earlyLateBoundOutput lateEarlyBoundOutput)

lemma lateEarlyInput:
  fixes P :: pi

```

```

and a :: name
and x :: name
and P' :: pi
and u :: name

assumes PTrans: P  $\mapsto_l$  a<x>  $\prec_l$  P'

shows P  $\mapsto_e$  a<u>  $\prec_e$  (P'[x::=u])
proof -
  have Goal:  $\bigwedge P a x P' u. \llbracket P \mapsto_l a<x> \prec_l P'; x \# P \rrbracket \implies P \mapsto_e a<u> \prec_e$ 
    (P'[x::=u])
  proof -
    fix P a x P' u
    assume P  $\mapsto_l$  a<x>  $\prec_l$  P' and x  $\#$  P
    thus P  $\mapsto_e$  a<u>  $\prec_e$  (P'[x::=u])
    proof(nominal-induct avoiding: u rule: Late-Semantics.inputInduct)
      case(Input a x P)
      thus ?case by(rule Early-Semantics.Input)
    next
      case(Match P a x P' b u)
      have P  $\mapsto_e$  a<u>  $\prec_e$  (P'[x::=u]) by fact
      thus ?case by(rule Early-Semantics.Match)
    next
      case(Mismatch P a x P' b c u)
      have P  $\mapsto_e$  a<u>  $\prec_e$  (P'[x::=u]) by fact
      moreover have b  $\neq$  c by fact
      ultimately show ?case by(rule Early-Semantics.Mismatch)
    next
      case(Sum1 P Q a x P')
      have P  $\mapsto_e$  a<u>  $\prec_e$  (P'[x::=u]) by fact
      thus ?case by(rule Early-Semantics.Sum1)
    next
      case(Sum2 P Q a x Q')
      have Q  $\mapsto_e$  a<u>  $\prec_e$  (Q'[x::=u]) by fact
      thus ?case by(rule Early-Semantics.Sum2)
    next
      case(Par1 P P' Q a x)
      have P  $\mapsto_e$  a<u>  $\prec_e$  (P'[x::=u]) by fact
      hence P  $\parallel$  Q  $\mapsto_e$  a<u>  $\prec_e$  (P'[x::=u]  $\parallel$  Q) by(rule Early-Semantics.Par1F)
      moreover have x  $\#$  Q by fact
      ultimately show ?case by(simp add: forget)
    next
      case(Par2 P Q Q' a x)
      have Q  $\mapsto_e$  a<u>  $\prec_e$  (Q'[x::=u]) by fact
      hence P  $\parallel$  Q  $\mapsto_e$  a<u>  $\prec_e$  (P  $\parallel$  Q'[x::=u]) by(rule Early-Semantics.Par2F)
      moreover have x  $\#$  P by fact
      ultimately show ?case by(simp add: forget)
    next
      case(Res P P' a x y u)

```

have $P \mapsto_e a \langle u \rangle \prec_e (P'[x::=u])$ **and** $y \neq a$ **and** $y \text{ fresh } u$ **by** *fact+*
hence $\langle \nu y \rangle P \mapsto_e a \langle u \rangle \prec_e \langle \nu y \rangle (P'[x::=u])$ **by** (*force intro: Early-Semantics.ResF*)
moreover $y \neq x$ **by** *fact*
ultimately show *?case using yinequ by simp*
next
case (*Bang P a x P' u*)
have $P \parallel !P \mapsto_e a \langle u \rangle \prec_e (P'[x::=u])$ **by** *fact*
thus *?case by (rule Early-Semantics.Bang)*
qed
qed

have $\exists c::\text{name}. c \# (P, P')$ **by** (*blast intro: name-exists-fresh*)
then obtain $c::\text{name}$ **where** $c \text{ fresh } P$ **and** $c \text{ fresh } P'$
by (*force simp add: fresh-prod*)
from *assms cFreshP'* **have** $P \mapsto_l a \langle c \rangle \prec_l ((x, c) \cdot P')$
by (*simp add: Late-Semantics.alphaBoundResidual*)
hence $P \mapsto_e a \langle u \rangle \prec_e ((x, c) \cdot P')[c::=u]$ **using** $c \text{ fresh } P$
by (*rule Goal*)
with $c \text{ fresh } P'$ **show** *?thesis by (simp add: renaming name-swap)*
qed

lemma *earlyLateInput:*

fixes $P :: pi$
and $a :: \text{name}$
and $x :: \text{name}$
and $P' :: pi$
and $u :: \text{name}$
and $C :: 'a::\text{fs-name}$

assumes $P \mapsto_e a \langle u \rangle \prec_e P'$
and $x \# P$

shows $\exists P''. P \mapsto_l a \langle x \rangle \prec_l P'' \wedge P' = P''[x::=u]$

proof –

{
fix $P a u P'$
assume $P \mapsto_e a \langle u \rangle \prec_e P'$
hence $\exists P'' x. P \mapsto_l a \langle x \rangle \prec_l P'' \wedge P' = P''[x::=u]$
proof (*nominal-induct rule: Early-Semantics.inputInduct*)
case (*cInput a x P u*)
have $a \langle x \rangle \cdot P \mapsto_l a \langle x \rangle \prec P$ **by** (*rule Late-Semantics.Input*)
thus *?case by blast*
next
case (*cMatch P a u P' b*)
have $\exists P'' x. P \mapsto_l a \langle x \rangle \prec P'' \wedge P' = P''[x::=u]$ **by** *fact*
then obtain $P'' x$ **where** $P \text{ Trans } P \mapsto_l a \langle x \rangle \prec P''$ **and** $P' \text{ eq } P''$: $P' = P''[x::=u]$ **by** *blast*
from $P \text{ Trans}$ **have** $[b \frown b] P \mapsto_l a \langle x \rangle \prec P''$ **by** (*rule Late-Semantics.Match*)
with $P' \text{ eq } P''$ **show** *?case by blast*
}

```

next
  case(cMismatch P a u P' b c)
  have  $\exists P'' x. P \mapsto_l a \langle x \rangle \prec P'' \wedge P' = P''[x::=u]$  by fact
  then obtain  $P'' x$  where  $PTrans: P \mapsto_l a \langle x \rangle \prec P''$  and  $P'eqP'': P' = P''[x::=u]$  by blast
  have  $b \neq c$  by fact
  with  $PTrans$  have  $[b \neq c]P \mapsto_l a \langle x \rangle \prec P''$  by (rule Late-Semantics.Mismatch)
  with  $P'eqP''$  show ?case by blast
next
  case(cSum1 P a u P' Q)
  have  $\exists P'' x. P \mapsto_l a \langle x \rangle \prec P'' \wedge P' = P''[x::=u]$  by fact
  then obtain  $P'' x$  where  $PTrans: P \mapsto_l a \langle x \rangle \prec P''$  and  $P'eqP'': P' = P''[x::=u]$  by blast
  from  $PTrans$  have  $P \oplus Q \mapsto_l a \langle x \rangle \prec P''$  by (rule Late-Semantics.Sum1)
  with  $P'eqP''$  show ?case by blast
next
  case(cSum2 Q a u Q' P)
  have  $\exists Q'' x. Q \mapsto_l a \langle x \rangle \prec Q'' \wedge Q' = Q''[x::=u]$  by fact
  then obtain  $Q'' x$  where  $QTrans: Q \mapsto_l a \langle x \rangle \prec Q''$  and  $Q'eqQ'': Q' = Q''[x::=u]$  by blast
  from  $QTrans$  have  $P \oplus Q \mapsto_l a \langle x \rangle \prec Q''$  by (rule Late-Semantics.Sum2)
  with  $Q'eqQ''$  show ?case by blast
next
  case(cPar1 P a u P' Q)
  have  $\exists P'' x. P \mapsto_l a \langle x \rangle \prec P'' \wedge P' = P''[x::=u]$  by fact
  then obtain  $P'' x$  where  $PTrans: P \mapsto_l a \langle x \rangle \prec P''$  and  $P'eqP'': P' = P''[x::=u]$  by blast
  have  $\exists c::name. c \# (Q, P'')$  by (blast intro: name-exists-fresh)
  then obtain  $c::name$  where  $cFreshQ: c \# Q$  and  $cFreshP'': c \# P''$  by (force simp add: fresh-prod)
  from  $PTrans$   $cFreshP''$  have  $P \mapsto_l a \langle c \rangle \prec [(x, c)] \cdot P''$  by (simp add: Late-Semantics.alphaBoundResidual)
  hence  $P \parallel Q \mapsto_l a \langle c \rangle \prec ([ (x, c) ] \cdot P'') \parallel Q$  using  $\langle c \# Q \rangle$  by (rule Late-Semantics.Par1B)
  moreover from  $cFreshQ$   $cFreshP''$   $P'eqP''$  have  $P' \parallel Q = ([ (x, c) ] \cdot P'') \parallel Q[c::=u]$ 
  by (simp add: forget renaming name-swap)
  ultimately show ?case by blast
next
  case(cPar2 Q a u Q' P)
  have  $\exists Q'' x. Q \mapsto_l a \langle x \rangle \prec Q'' \wedge Q' = Q''[x::=u]$  by fact
  then obtain  $Q'' x$  where  $QTrans: Q \mapsto_l a \langle x \rangle \prec Q''$  and  $Q'eqQ'': Q' = Q''[x::=u]$  by blast
  have  $\exists c::name. c \# (P, Q'')$  by (blast intro: name-exists-fresh)
  then obtain  $c::name$  where  $cFreshP: c \# P$  and  $cFreshQ'': c \# Q''$  by (force simp add: fresh-prod)
  from  $QTrans$   $cFreshQ''$  have  $Q \mapsto_l a \langle c \rangle \prec [(x, c)] \cdot Q''$  by (simp add: Late-Semantics.alphaBoundResidual)
  hence  $P \parallel Q \mapsto_l a \langle c \rangle \prec P \parallel ([ (x, c) ] \cdot Q'')$  using  $\langle c \# P \rangle$  by (rule

```

Late-Semantics.Par2B)
moreover from $cFreshP$ $cFreshQ''$ $Q'eqQ''$ **have** $P \parallel Q' = (P \parallel [(x, c)] \cdot Q'')[c::=u]$
by (*simp add: forget renaming name-swap*)
ultimately show *?case by blast*
next
case ($cRes P a u P' y$)
have $\exists P'' x. P \mapsto_l a \langle x \rangle \prec P'' \wedge P' = P''[x::=u]$ **by** *fact*
then obtain $P'' x$ **where** $PTrans: P \mapsto_l a \langle x \rangle \prec P''$ **and** $P'eqP'': P' = P''[x::=u]$ **by** *blast*
have $y \neq u$ **by** *fact*
have $\exists c::name. c \# (y, P'')$ **by** (*blast intro: name-exists-fresh*)
then obtain $c::name$ **where** $c \neq y$ **and** $cFreshP'': c \# P''$ **by** (*force simp add: fresh-prod*)
from $PTrans$ $cFreshP''$ **have** $P \mapsto_l a \langle c \rangle \prec [(x, c)] \cdot P''$ **by** (*simp add: Late-Semantics.alphaBoundResidual*)
moreover have $y \neq a$ **by** *fact*
ultimately have $\langle \nu y \rangle P \mapsto_l a \langle c \rangle \prec \langle \nu y \rangle ([[(x, c)] \cdot P'')$ **using** $cineqy$
by (*force intro: Late-Semantics.ResB*)
moreover from $cineqy$ $cFreshP''$ $P'eqP''$ $y \neq u$ **have** $\langle \nu y \rangle P' = (\langle \nu y \rangle ([[(x, c)] \cdot P''))[c::=u]$
by (*simp add: renaming name-swap*)
ultimately show *?case by blast*
next
case ($cBang P a u P'$)
have $\exists P'' x. P \parallel !P \mapsto_l a \langle x \rangle \prec P'' \wedge P' = P''[x::=u]$ **by** *fact*
then obtain $P'' x$ **where** $PTrans: P \parallel !P \mapsto_l a \langle x \rangle \prec P''$ **and** $P'eqP'': P' = P''[x::=u]$ **by** *blast*
from $PTrans$ **have** $!P \mapsto_l a \langle x \rangle \prec P''$ **by** (*rule Late-Semantics.Bang*)
with $P'eqP''$ **show** *?case by blast*
qed
}
with $assms$ **obtain** $P'' y$ **where** $PTrans: P \mapsto_l a \langle y \rangle \prec P''$ **and** $P'eqP'': P' = P''[y::=u]$ **by** *blast*
show *?thesis*
proof (*cases x=y*)
case *True*
from $PTrans$ $P'eqP''$ $\langle x = y \rangle$ **show** *?thesis by blast*
next
case *False*
from $PTrans$ $\langle x \neq y \rangle$ $\langle x \# P \rangle$ **have** $x \# P''$ **by** (*fastforce dest: freshBoundDerivative simp add: residual.inject*)
with $PTrans$ **have** $P \mapsto_l a \langle x \rangle \prec_l ([[(x, y)] \cdot P'')$
by (*simp add: Late-Semantics.alphaBoundResidual*)
moreover from $\langle x \# P'' \rangle$ **have** $P''[y::=u] = ([[(x, y)] \cdot P'')[x::=u]$ **by** (*simp add: renaming name-swap*)
ultimately show *?thesis using P'eqP'' by blast*
qed
qed

```

lemma lateEarlyTau:
  fixes P :: pi
  and P' :: pi

  assumes P  $\mapsto_l \tau \prec_l P'$ 

  shows P  $\mapsto_e \tau \prec_e P'$ 
using assms
proof(nominal-induct rule: Late-Semantics.tauInduct)
  case(Tau P)
  thus ?case by(rule Early-Semantics.Tau)
next
  case(Match P P' a)
  have P  $\mapsto_e \tau \prec_e P'$  by fact
  thus  $[a \frown a]P \mapsto_e \tau \prec_e P'$  by(rule Early-Semantics.Match)
next
  case(Mismatch P P' a b)
  have P  $\mapsto_e \tau \prec_e P'$  by fact
  moreover have  $a \neq b$  by fact
  ultimately show  $[a \neq b]P \mapsto_e \tau \prec_e P'$  by(rule Early-Semantics.Mismatch)
next
  case(Sum1 P P' Q)
  have P  $\mapsto_e \tau \prec_e P'$  by fact
  thus  $P \oplus Q \mapsto_e \tau \prec_e P'$  by(rule Early-Semantics.Sum1)
next
  case(Sum2 Q Q' P)
  have Q  $\mapsto_e \tau \prec_e Q'$  by fact
  thus  $P \oplus Q \mapsto_e \tau \prec_e Q'$  by(rule Early-Semantics.Sum2)
next
  case(Par1 P P' Q)
  have P  $\mapsto_e \tau \prec_e P'$  by fact
  thus  $P \parallel Q \mapsto_e \tau \prec_e P' \parallel Q$  by(rule Early-Semantics.Par1F)
next
  case(Par2 Q Q' P)
  have Q  $\mapsto_e \tau \prec_e Q'$  by fact
  thus  $P \parallel Q \mapsto_e \tau \prec_e P \parallel Q'$  by(rule Early-Semantics.Par2F)
next
  case(Comm1 P a x P' Q b Q')
  have P  $\mapsto_e a < b > \prec_e P'[x ::= b]$ 
  proof -
    have P  $\mapsto_l a < x > \prec P'$  by fact
    thus ?thesis by(rule lateEarlyInput)
  qed
  moreover have Q  $\mapsto_e a[b] \prec_e Q'$ 
  proof -
    have Q  $\mapsto_l a[b] \prec_l Q'$  by fact
    thus ?thesis by(rule lateEarlyOutput)
  qed
qed

```

```

ultimately show ?case by(rule Early-Semantics.Comm1)
next
case(Comm2 P a b P' Q x Q')
have P  $\mapsto_e a[b]$   $\prec_e P'$ 
proof -
  have P  $\mapsto_l a[b]$   $\prec_l P'$  by fact
  thus ?thesis by(rule lateEarlyOutput)
qed
moreover have Q  $\mapsto_e a<b>$   $\prec_e Q'[x::=b]$ 
proof -
  have Q  $\mapsto_l a<x>$   $\prec_l Q'$  by fact
  thus ?thesis by(rule lateEarlyInput)
qed
ultimately show ?case by(rule Early-Semantics.Comm2)
next
case(Close1 P a x P' Q y Q')
have P  $\mapsto_e a<y>$   $\prec_e P'[x::=y]$ 
proof -
  have P  $\mapsto_l a<x>$   $\prec P'$  by fact
  thus ?thesis by(rule lateEarlyInput)
qed
moreover have Q  $\mapsto_e a<\nu y>$   $\prec Q'$ 
proof -
  have Q  $\mapsto_l a<\nu y>$   $\prec_l Q'$  by fact
  thus ?thesis by(rule lateEarlyBoundOutput)
qed
moreover have y  $\# P$  by fact
ultimately show ?case by(rule Early-Semantics.Close1)
next
case(Close2 P a y P' Q x Q')
have P  $\mapsto_e a<\nu y>$   $\prec P'$ 
proof -
  have P  $\mapsto_l a<\nu y>$   $\prec_l P'$  by fact
  thus ?thesis by(rule lateEarlyBoundOutput)
qed
moreover have Q  $\mapsto_e a<y>$   $\prec_e Q'[x::=y]$ 
proof -
  have Q  $\mapsto_l a<x>$   $\prec_l Q'$  by fact
  thus ?thesis by(rule lateEarlyInput)
qed
moreover have y  $\# Q$  by fact
ultimately show ?case by(rule Early-Semantics.Close2)
next
case(Res P P' x)
have P  $\mapsto_e \tau$   $\prec_e P'$  by fact
thus ?case by(force intro: Early-Semantics.ResF)
next
case(Bang P P')
have P  $\parallel !P$   $\mapsto_e \tau$   $\prec_e P'$  by fact

```

```

thus ?case by(rule Early-Semantics.Bang)
qed

lemma earlyLateTau:
  fixes  $P :: pi$ 
  and  $P' :: pi$ 

  assumes  $P \mapsto_e \tau \prec_e P'$ 

  shows  $P \mapsto_l \tau \prec_l P'$ 
using assms
proof(nominal-induct rule: Early-Semantics.tauInduct)
  case(Tau P)
  thus ?case by(rule Late-Semantics.Tau)
next
  case(Match P P' a)
  have  $P \mapsto_l \tau \prec_l P'$  by fact
  thus ?case by(rule Late-Semantics.Match)
next
  case(Mismatch P P' a b)
  have  $P \mapsto_l \tau \prec_l P'$  by fact
  moreover have  $a \neq b$  by fact
  ultimately show ?case by(rule Late-Semantics.Mismatch)
next
  case(Sum1 P P' Q)
  have  $P \mapsto_l \tau \prec_l P'$  by fact
  thus ?case by(rule Late-Semantics.Sum1)
next
  case(Sum2 Q Q' P)
  have  $Q \mapsto_l \tau \prec_l Q'$  by fact
  thus ?case by(rule Late-Semantics.Sum2)
next
  case(Par1 P P' Q)
  have  $P \mapsto_l \tau \prec_l P'$  by fact
  thus ?case by(rule Late-Semantics.Par1F)
next
  case(Par2 Q Q' P)
  have  $Q \mapsto_l \tau \prec_l Q'$  by fact
  thus ?case by(rule Late-Semantics.Par2F)
next
  case(Comm1 P a b P' Q Q')
  have  $P \mapsto_e a \langle b \rangle \prec_e P'$  by fact
  moreover obtain  $x :: name$  where  $x \# P$  by(generate-fresh name) auto
  ultimately obtain  $P''$  where  $PTrans: P \mapsto_l a \langle x \rangle \prec P''$  and  $P'eqP'': P' =$ 
 $P''[x ::= b]$ 
  by(blast dest: earlyLateInput)
  have  $Q \mapsto_e a[b] \prec_e Q'$  by fact
  hence  $Q \mapsto_l a[b] \prec_l Q'$  by(rule earlyLateOutput)
  with  $PTrans P'eqP''$  show ?case

```

```

    by(blast intro: Late-Semantics.Comm1)
next
case(Comm2 P a b P' Q Q')
have P  $\mapsto_e a[b]$   $\prec_e P'$  by fact
hence QTrans: P  $\mapsto_l a[b]$   $\prec_l P'$  by(rule earlyLateOutput)
have Q  $\mapsto_e a<b>$   $\prec_e Q'$  by fact
moreover obtain x::name where x # Q by(generate-fresh name) auto
ultimately obtain Q'' x where Q  $\mapsto_l a<x>$   $\prec Q''$  and Q' = Q''[x::=b]
  by(blast dest: earlyLateInput)
with QTrans show ?case
  by(blast intro: Late-Semantics.Comm2)
next
case(Close1 P a x P' Q Q')
have P  $\mapsto_e a<x>$   $\prec_e P'$  and x # P by fact+
then obtain P'' where P  $\mapsto_l a<x>$   $\prec P''$  and P' = P''[x::=x]
  by(blast dest: earlyLateInput)

moreover have Q  $\mapsto_e a<\nu x>$   $\prec_e Q'$  by fact
hence Q  $\mapsto_l a<\nu x>$   $\prec_l Q'$  by(rule earlyLateBoundOutput)
moreover have x # P by fact
ultimately show ?case
  by(blast intro: Late-Semantics.Close1)
next
case(Close2 P a x P' Q Q')
have P  $\mapsto_e a<\nu x>$   $\prec_e P'$  by fact
hence PTrans: P  $\mapsto_l a<\nu x>$   $\prec_l P'$  by(rule earlyLateBoundOutput)

have Q  $\mapsto_e a<x>$   $\prec_e Q'$  and x # Q by fact+
then obtain Q'' y where Q  $\mapsto_l a<x>$   $\prec Q''$  and Q' = Q''[x::=x]
  by(blast dest: earlyLateInput)
moreover have x # Q by fact
ultimately show ?case using PTrans
  by(blast intro: Late-Semantics.Close2)
next
case(Res P P' x)
have P  $\mapsto_l \tau$   $\prec_l P'$  by fact
thus ?case by(force intro: Late-Semantics.ResF)
next
case(Bang P P')
have P || !P  $\mapsto_l \tau$   $\prec_l P'$  by fact
thus ?case by(force intro: Late-Semantics.Bang)
qed

lemma tauEq:
  fixes P :: pi
  and P' :: pi

  shows P  $\mapsto_e$  (Early-Semantics.FreeR Early-Semantics.TauR P') = P  $\mapsto_\tau$   $\prec_l$ 
P'
```

by(auto intro: earlyLateTau lateEarlyTau)

abbreviation *simLate-judge* ($\langle \cdot \rightsquigarrow_l[-] \cdot \rangle \rightarrow [80, 80, 80] 80$) **where** $P \rightsquigarrow_l[Rel] Q \equiv$
Strong-Late-Sim.simulation $P Rel Q$

abbreviation *simEarly-judge* ($\langle \cdot \rightsquigarrow_e[-] \cdot \rangle \rightarrow [80, 80, 80] 80$) **where** $P \rightsquigarrow_e[Rel] Q \equiv$
Strong-Early-Sim.strongSimEarly $P Rel Q$

lemma *lateEarlySim*:

fixes $P :: pi$

and $Q :: pi$

and $Rel :: (pi \times pi) set$

assumes $PSimQ: P \rightsquigarrow_l[Rel] Q$

shows $P \rightsquigarrow_e[Rel] Q$

proof(*induct rule: Strong-Early-Sim.simCases*)

case(*Bound* $a x Q'$)

have $Q \mapsto_e a \langle \nu x \rangle \prec_e Q'$ **by** *fact*

hence $Q \mapsto_l a \langle \nu x \rangle \prec_l Q'$ **by**(*rule earlyLateBoundOutput*)

moreover **have** $x \# P$ **by** *fact*

ultimately obtain P' **where** $PTrans: P \mapsto_l a \langle \nu x \rangle \prec_l P'$ **and** $P'RelQ': (P', Q') \in Rel$ **using** $PSimQ$

by(*force dest: Strong-Late-Sim.simE simp add: derivative-def*)

from $PTrans$ **have** $P \mapsto_e a \langle \nu x \rangle \prec_e P'$ **by**(*rule lateEarlyBoundOutput*)

with $P'RelQ'$ **show** *?case* **by** *blast*

next

case(*Free* $\alpha Q'$)

have $Q \mapsto_e \alpha \langle \nu x \rangle \prec_e Q'$ **by** *fact*

thus *?case*

proof(*nominal-induct* α *rule: freeRes.strong-induct*)

case(*InputR* $a u$)

obtain $x::name$ **where** $x \# Q$ **and** $x \# P$ **by**(*generate-fresh name*) *auto*

with $\langle Q \mapsto_e a \langle u \rangle \prec_e Q' \rangle$ **obtain** Q'' **where** $QTrans: Q \mapsto_l a \langle x \rangle \prec_l Q''$

and $Q'eqQ'': Q' = Q''[x::=u]$

by(*blast dest: earlyLateInput*)

from $PSimQ QTrans \langle x \# P \rangle$ **obtain** P' **where** $PTrans: P \mapsto_l a \langle x \rangle \prec_l P'$

and $P'RelQ': (P'[x::=u], Q''[x::=u]) \in Rel$

by(*force dest: Strong-Late-Sim.simE simp add: derivative-def*)

from $PTrans$ **have** $P \mapsto_e a \langle u \rangle \prec_e P'[x::=u]$ **by**(*rule lateEarlyInput*)

with $P'RelQ' Q'eqQ''$ **show** $\exists P'. P \mapsto_e a \langle u \rangle \prec_e P' \wedge (P', Q') \in Rel$ **by**

blast

next

case(*OutputR* $a b$)

from $\langle Q \mapsto_e a[b] \prec_e Q' \rangle$ **have** $Q \mapsto_l a[b] \prec_l Q'$ **by**(*rule earlyLateOutput*)

with $PSimQ$ **obtain** P' **where** $PTrans: P \mapsto_l a[b] \prec_l P'$ **and** $P'RelQ': (P', Q') \in Rel$

by(*blast dest: Strong-Late-Sim.simE*)

from $PTrans$ **have** $P \mapsto_e a[b] \prec_e P'$ **by** (*rule lateEarlyOutput*)
with $P'RelQ'$ **show** $\exists P'. P \mapsto_e a[b] \prec_e P' \wedge (P', Q') \in Rel$ **by** *blast*
next
case $TauR$
from $\langle Q \mapsto_e \tau \prec_e Q' \rangle$ **have** $Q \mapsto_l \tau \prec_l Q'$ **by** (*rule earlyLateTau*)
with $PSimQ$ **obtain** P' **where** $PTrans: P \mapsto_l \tau \prec_l P'$ **and** $P'RelQ': (P', Q') \in Rel$
by (*blast dest: Strong-Late-Sim.simE*)
from $PTrans$ **have** $P \mapsto_e \tau \prec_e P'$ **by** (*rule lateEarlyTau*)
with $P'RelQ'$ **show** $\exists P'. P \mapsto_e \tau \prec_e P' \wedge (P', Q') \in Rel$ **by** *blast*
qed
qed

abbreviation *bisimLate-judge* $(\prec \sim_l \rightarrow [80, 80] 80)$ **where** $P \sim_l Q \equiv (P, Q) \in Strong-Late-Bisim.bisim$
abbreviation *bisimEarly-judge* $(\prec \sim_e \rightarrow [80, 80] 80)$ **where** $P \sim_e Q \equiv (P, Q) \in Strong-Early-Bisim.bisim$

lemma *lateEarlyBisim*:

fixes $P :: pi$
and $Q :: pi$

assumes $P \sim_l Q$

shows $P \sim_e Q$

using *assms*

by (*coinduct rule: Strong-Early-Bisim.weak-coinduct*)

(*auto dest: Strong-Late-Bisim.bisimE Strong-Late-Bisim.symmetric intro: lateEarlySim*)

abbreviation *congLate-judge* $(\prec \sim^s_l \rightarrow [80, 80] 80)$ **where** $P \sim^s_l Q \equiv (P, Q) \in (substClosed Strong-Late-Bisim.bisim)$

abbreviation *congEarly-judge* $(\prec \sim^s_e \rightarrow [80, 80] 80)$ **where** $P \sim^s_e Q \equiv (P, Q) \in (substClosed Strong-Early-Bisim.bisim)$

lemma *lateEarlyCong*:

fixes $P :: pi$
and $Q :: pi$

assumes $P \sim^s_l Q$

shows $P \sim^s_e Q$

using *assms*

by (*auto simp add: substClosed-def intro: lateEarlyBisim*)

```

lemma earlyCongStructCong:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 

  assumes  $P \equiv_s Q$ 

  shows  $P \sim_e^s Q$ 
using assms lateEarlyCong bisimSubstStructCong
by blast

lemma earlyBisimStructCong:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 

  assumes  $P \equiv_s Q$ 

  shows  $P \sim_e Q$ 
using assms lateEarlyBisim structCongBisim
by blast

end

theory Strong-Early-Bisim-SC
  imports Strong-Early-Bisim Strong-Late-Bisim-SC Strong-Early-Late-Comp
begin

lemma resComm:
  fixes  $P :: pi$ 

  shows  $\langle \nu a \rangle \langle \nu b \rangle P \sim_e \langle \nu b \rangle \langle \nu a \rangle P$ 
proof –
  have  $\langle \nu a \rangle \langle \nu b \rangle P \sim_l \langle \nu b \rangle \langle \nu a \rangle P$  by(rule Strong-Late-Bisim-SC.resComm)
  thus ?thesis by(rule lateEarlyBisim)
qed

lemma matchId:
  fixes  $a :: name$ 
  and  $P :: pi$ 

  shows  $[a \frown a]P \sim_e P$ 
proof –
  have  $[a \frown a]P \sim_l P$  by(rule Strong-Late-Bisim-SC.matchId)

```

```

thus ?thesis by(rule lateEarlyBisim)
qed

```

```

lemma mismatchId:

```

```

  fixes a :: name
  and b :: name
  and P :: pi

```

```

  assumes a ≠ b

```

```

  shows [a≠b]P ∼e P

```

```

proof –

```

```

  from assms have [a≠b]P ∼l P by(rule Strong-Late-Bisim-SC.mismatchId)
  thus ?thesis by(rule lateEarlyBisim)

```

```

qed

```

```

lemma mismatchNil:

```

```

  fixes a :: name
  and P :: pi

```

```

  shows [a≠a]P ∼e 0

```

```

proof –

```

```

  have [a≠a]P ∼l 0 by(rule Strong-Late-Bisim-SC.mismatchNil)
  thus ?thesis by(rule lateEarlyBisim)

```

```

qed

```

```

lemma sumSym:

```

```

  fixes P :: pi
  and Q :: pi

```

```

  shows P ⊕ Q ∼e Q ⊕ P

```

```

proof –

```

```

  have P ⊕ Q ∼l Q ⊕ P by(rule Strong-Late-Bisim-SC.sumSym)
  thus ?thesis by(rule lateEarlyBisim)

```

```

qed

```

```

lemma sumAssoc:

```

```

  fixes P :: pi
  and Q :: pi
  and R :: pi

```

```

  shows (P ⊕ Q) ⊕ R ∼e P ⊕ (Q ⊕ R)

```

```

proof –

```

```

  have (P ⊕ Q) ⊕ R ∼l P ⊕ (Q ⊕ R) by(rule Strong-Late-Bisim-SC.sumAssoc)
  thus ?thesis by(rule lateEarlyBisim)

```

qed

lemma *sumZero*:

fixes $P :: pi$

shows $P \oplus \mathbf{0} \sim_e P$

proof –

have $P \oplus \mathbf{0} \sim_l P$ by(rule *Strong-Late-Bisim-SC.sumZero*)

thus ?thesis by(rule *lateEarlyBisim*)

qed

lemma *parZero*:

fixes $P :: pi$

shows $P \parallel \mathbf{0} \sim_e P$

proof –

have $P \parallel \mathbf{0} \sim_l P$ by(rule *Strong-Late-Bisim-SC.parZero*)

thus ?thesis by(rule *lateEarlyBisim*)

qed

lemma *parSym*:

fixes $P :: pi$

and $Q :: pi$

shows $P \parallel Q \sim_e Q \parallel P$

proof –

have $P \parallel Q \sim_l Q \parallel P$ by(rule *Strong-Late-Bisim-SC.parSym*)

thus ?thesis by(rule *lateEarlyBisim*)

qed

lemma *scopeExtPar*:

fixes $P :: pi$

and $Q :: pi$

and $x :: name$

assumes $x \# P$

shows $\langle \nu x \rangle (P \parallel Q) \sim_e P \parallel \langle \nu x \rangle Q$

proof –

from *assms* have $\langle \nu x \rangle (P \parallel Q) \sim_l P \parallel \langle \nu x \rangle Q$ by(rule *Strong-Late-Bisim-SC.scopeExtPar*)

thus ?thesis by(rule *lateEarlyBisim*)

qed

lemma *scopeExtPar'*:

fixes $P :: pi$

and $Q :: pi$

and $x :: name$

```

assumes  $xFreshQ: x \# Q$ 

shows  $\langle \nu x \rangle (P \parallel Q) \sim_e \langle \nu x \rangle P \parallel Q$ 
proof –
  from assms have  $\langle \nu x \rangle (P \parallel Q) \sim_l \langle \nu x \rangle P \parallel Q$  by(rule Strong-Late-Bisim-SC.scopeExtPar')
  thus ?thesis by(rule lateEarlyBisim)
qed

lemma parAssoc:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $R :: pi$ 

  shows  $(P \parallel Q) \parallel R \sim_e P \parallel (Q \parallel R)$ 
proof –
  have  $(P \parallel Q) \parallel R \sim_l P \parallel (Q \parallel R)$  by(rule Strong-Late-Bisim-SC.parAssoc)
  thus ?thesis by(rule lateEarlyBisim)
qed

lemma freshRes:
  fixes  $P :: pi$ 
  and  $a :: name$ 

  assumes  $aFreshP: a \# P$ 

  shows  $\langle \nu a \rangle P \sim_e P$ 
proof –
  from aFreshP have  $\langle \nu a \rangle P \sim_l P$  by(rule Strong-Late-Bisim-SC.scopeFresh)
  thus ?thesis by(rule lateEarlyBisim)
qed

lemma scopeExtSum:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $x :: name$ 

  assumes  $x \# P$ 

  shows  $\langle \nu x \rangle (P \oplus Q) \sim_e P \oplus \langle \nu x \rangle Q$ 
proof –
  from  $\langle x \# P \rangle$  have  $\langle \nu x \rangle (P \oplus Q) \sim_l P \oplus \langle \nu x \rangle Q$  by(rule Strong-Late-Bisim-SC.scopeExtSum)
  thus ?thesis by(rule lateEarlyBisim)
qed

lemma bangSC:
  fixes  $P$ 

  shows  $!P \sim_e P \parallel !P$ 

```

```

proof –
  have  $!P \sim_l P \parallel !P$  by(rule Strong-Late-Bisim-SC.bangSC)
  thus ?thesis by(rule lateEarlyBisim)
qed

end

```

```

theory Weak-Early-Bisim-SC
  imports Weak-Early-Bisim Strong-Early-Bisim-SC
begin

```

```

lemma weakBisimStructCong:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 

  assumes  $P \equiv_s Q$ 

  shows  $P \approx Q$ 
using assms
by(metis earlyBisimStructCong strongBisimWeakBisim)

```

```

lemma matchId:
  fixes  $a :: name$ 
  and  $P :: pi$ 

  shows  $[a \frown a]P \approx P$ 
proof –
  have  $[a \frown a]P \sim_e P$  by(rule Strong-Early-Bisim-SC.matchId)
  thus ?thesis by(rule strongBisimWeakBisim)
qed

```

```

lemma mismatchId:
  fixes  $a :: name$ 
  and  $b :: name$ 
  and  $P :: pi$ 

  assumes  $a \neq b$ 

  shows  $[a \neq b]P \approx P$ 
proof –
  from  $\langle a \neq b \rangle$  have  $[a \neq b]P \sim_e P$  by(rule Strong-Early-Bisim-SC.mismatchId)
  thus ?thesis by(rule strongBisimWeakBisim)
qed

```

```

lemma mismatchNil:
  fixes  $a :: name$ 
  and  $P :: pi$ 

```

shows $[a \neq a]P \approx \mathbf{0}$
proof –
have $[a \neq a]P \sim_e \mathbf{0}$ **by**(rule *Strong-Early-Bisim-SC.mismatchNil*)
thus *?thesis* **by**(rule *strongBisimWeakBisim*)
qed

lemma *resComm*:
fixes $P :: pi$

shows $\langle \nu a \rangle \langle \nu b \rangle P \approx \langle \nu b \rangle \langle \nu a \rangle P$
proof –
have $\langle \nu a \rangle \langle \nu b \rangle P \sim_e \langle \nu b \rangle \langle \nu a \rangle P$ **by**(rule *Strong-Early-Bisim-SC.resComm*)
thus *?thesis* **by**(rule *strongBisimWeakBisim*)
qed

lemma *sumSym*:
fixes $P :: pi$
and $Q :: pi$

shows $P \oplus Q \approx Q \oplus P$
proof –
have $P \oplus Q \sim_e Q \oplus P$ **by**(rule *Strong-Early-Bisim-SC.sumSym*)
thus *?thesis* **by**(rule *strongBisimWeakBisim*)
qed

lemma *sumAssoc*:
fixes $P :: pi$
and $Q :: pi$
and $R :: pi$

shows $(P \oplus Q) \oplus R \approx P \oplus (Q \oplus R)$
proof –
have $(P \oplus Q) \oplus R \sim_e P \oplus (Q \oplus R)$ **by**(rule *Strong-Early-Bisim-SC.sumAssoc*)
thus *?thesis* **by**(rule *strongBisimWeakBisim*)
qed

lemma *sumZero*:
fixes $P :: pi$

shows $P \oplus \mathbf{0} \approx P$
proof –
have $P \oplus \mathbf{0} \sim_e P$ **by**(rule *Strong-Early-Bisim-SC.sumZero*)
thus *?thesis* **by**(rule *strongBisimWeakBisim*)
qed

lemma *parZero*:

fixes $P :: pi$

shows $P \parallel \mathbf{0} \approx P$

proof –

have $P \parallel \mathbf{0} \sim_e P$ **by**(*rule Strong-Early-Bisim-SC.parZero*)

thus *?thesis* **by**(*rule strongBisimWeakBisim*)

qed

lemma *parSym*:

fixes $P :: pi$

and $Q :: pi$

shows $P \parallel Q \approx Q \parallel P$

proof –

have $P \parallel Q \sim_e Q \parallel P$ **by**(*rule Strong-Early-Bisim-SC.parSym*)

thus *?thesis* **by**(*rule strongBisimWeakBisim*)

qed

lemma *scopeExtPar*:

fixes $P :: pi$

and $Q :: pi$

and $x :: name$

assumes $x \# P$

shows $\langle \nu x \rangle (P \parallel Q) \approx P \parallel \langle \nu x \rangle Q$

proof –

from $\langle x \# P \rangle$ **have** $\langle \nu x \rangle (P \parallel Q) \sim_e P \parallel \langle \nu x \rangle Q$ **by**(*rule Strong-Early-Bisim-SC.scopeExtPar*)

thus *?thesis* **by**(*rule strongBisimWeakBisim*)

qed

lemma *scopeExtPar'*:

fixes $P :: pi$

and $Q :: pi$

and $x :: name$

assumes $x \# Q$

shows $\langle \nu x \rangle (P \parallel Q) \approx (\langle \nu x \rangle P) \parallel Q$

proof –

from $\langle x \# Q \rangle$ **have** $\langle \nu x \rangle (P \parallel Q) \sim_e (\langle \nu x \rangle P) \parallel Q$ **by**(*rule Strong-Early-Bisim-SC.scopeExtPar'*)

thus *?thesis* **by**(*rule strongBisimWeakBisim*)

qed

lemma *parAssoc*:

```

fixes  $P :: pi$ 
and  $Q :: pi$ 
and  $R :: pi$ 

shows  $(P \parallel Q) \parallel R \approx P \parallel (Q \parallel R)$ 
proof –
  have  $(P \parallel Q) \parallel R \sim_e P \parallel (Q \parallel R)$  by(rule Strong-Early-Bisim-SC.parAssoc)
  thus ?thesis by(rule strongBisimWeakBisim)
qed

lemma freshRes:
  fixes  $P :: pi$ 
  and  $a :: name$ 

  assumes  $a \# P$ 

  shows  $\langle \nu a \rangle P \approx P$ 
proof –
  from  $\langle a \# P \rangle$  have  $\langle \nu a \rangle P \sim_e P$  by(rule Strong-Early-Bisim-SC.freshRes)
  thus ?thesis by(rule strongBisimWeakBisim)
qed

lemma scopeExtSum:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $x :: name$ 

  assumes  $x \# P$ 

  shows  $\langle \nu x \rangle (P \oplus Q) \approx P \oplus \langle \nu x \rangle Q$ 
proof –
  from  $\langle x \# P \rangle$  have  $\langle \nu x \rangle (P \oplus Q) \sim_e P \oplus \langle \nu x \rangle Q$  by(rule Strong-Early-Bisim-SC.scopeExtSum)
  thus ?thesis by(rule strongBisimWeakBisim)
qed

lemma bangSC:
  fixes  $P$ 

  shows  $!P \approx P \parallel !P$ 
proof –
  have  $!P \sim_e P \parallel !P$  by(rule Strong-Early-Bisim-SC.bangSC)
  thus ?thesis by(rule strongBisimWeakBisim)
qed

end

theory Weak-Early-Bisim-Pres
  imports Strong-Early-Bisim-Pres Weak-Early-Sim-Pres Weak-Early-Bisim-SC Weak-Early-Bisim

```

begin

lemma *tauPres*:

fixes $P :: pi$

and $Q :: pi$

assumes $P \approx Q$

shows $\tau.(P) \approx \tau.(Q)$

proof –

let $?X = \{(\tau.(P), \tau.(Q)) \mid P Q. P \approx Q\}$

from $\langle P \approx Q \rangle$ **have** $(\tau.(P), \tau.(Q)) \in ?X$ **by** *auto*

thus *?thesis*

proof(*coinduct rule: weakBisimCoinduct*)

case(*cSim P Q*)

thus *?case*

by(*force intro: Weak-Early-Sim-Pres.tauPres*)

next

case(*cSym P Q*)

thus *?case* **by**(*force dest: Weak-Early-Bisim.symmetric simp add: pi.inject*)

qed

qed

lemma *outputPres*:

fixes $P :: pi$

and $Q :: pi$

and $a :: name$

and $b :: name$

assumes $P \approx Q$

shows $a\{b\}.P \approx a\{b\}.Q$

proof –

let $?X = \{(a\{b\}.(P), a\{b\}.(Q)) \mid P Q a b. P \approx Q\}$

from $\langle P \approx Q \rangle$ **have** $(a\{b\}.(P), a\{b\}.(Q)) \in ?X$ **by** *auto*

thus *?thesis*

proof(*coinduct rule: weakBisimCoinduct*)

case(*cSim P Q*)

thus *?case*

by(*force intro: Weak-Early-Sim-Pres.outputPres*)

next

case(*cSym P Q*)

thus *?case* **by**(*force dest: Weak-Early-Bisim.symmetric simp add: pi.inject*)

qed

qed

lemma *inputPres*:

```

fixes  $P :: pi$ 
and  $Q :: pi$ 
and  $a :: name$ 
and  $x :: name$ 

assumes  $PSimQ: \forall y. P[x::=y] \approx Q[x::=y]$ 

shows  $a\langle x \rangle.P \approx a\langle x \rangle.Q$ 
proof –
let  $?X = \{(a\langle x \rangle.P, a\langle x \rangle.Q) \mid a\ x\ P\ Q. \forall y. P[x::=y] \approx Q[x::=y]\}$ 
{
  fix  $axP\ axQ\ p$ 
  assume  $(axP, axQ) \in ?X$ 
  then obtain  $a\ x\ P\ Q$  where  $A: \forall y. P[x::=y] \approx Q[x::=y]$  and  $B: axP =$ 
 $a\langle x \rangle.P$  and  $C: axQ = a\langle x \rangle.Q$ 
  by auto
  have  $\bigwedge y. ((p::name\ prm) \cdot P)[(p \cdot x)::=y] \approx (p \cdot Q)[(p \cdot x)::=y]$ 
  proof –
    fix  $y$ 
    from  $A$  have  $P[x::=(rev\ p \cdot y)] \approx Q[x::=(rev\ p \cdot y)]$ 
    by blast
    hence  $(p \cdot (P[x::=(rev\ p \cdot y)])) \approx p \cdot (Q[x::=(rev\ p \cdot y)])$ 
    by (rule eqvts)
    thus  $(p \cdot P)[(p \cdot x)::=y] \approx (p \cdot Q)[(p \cdot x)::=y]$ 
    by (simp add: eqvts pt-pi-rev[OF pt-name-inst, OF at-name-inst])
  qed
  hence  $((p::name\ prm) \cdot axP, p \cdot axQ) \in ?X$  using  $B\ C$ 
  by auto
}
hence eqvt  $?X$  by (simp add: eqvt-def)

from  $PSimQ$  have  $(a\langle x \rangle.P, a\langle x \rangle.Q) \in ?X$  by auto
thus ?thesis
proof (coinduct rule: weakBisimCoinduct)
  case (cSim P Q)
  thus ?case using  $\langle eqvt\ ?X \rangle$ 
  by (force intro: Weak-Early-Sim-Pres.inputPres)
next
  case (cSym P Q)
  thus ?case
  by (blast dest: weakBisimE)
qed
qed

lemma resPres:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $x :: name$ 

```

```

assumes  $P \approx Q$ 

shows  $\langle \nu x \rangle P \approx \langle \nu x \rangle Q$ 
proof –
let  $?X = \{(\langle \nu x \rangle P, \langle \nu x \rangle Q) \mid x P Q. P \approx Q\}$ 
from  $\langle P \approx Q \rangle$  have  $(\langle \nu x \rangle P, \langle \nu x \rangle Q) \in ?X$  by blast
thus ?thesis
proof(coinduct rule: weakBisimCoinduct)
  case(cSim xP xQ)
  {
    fix  $P Q x$ 
    assume  $P \approx Q$ 
    hence  $P \rightsquigarrow \langle \text{weakBisim} \rangle Q$  by(rule weakBisimE)
    moreover have  $\bigwedge P Q x. P \approx Q \implies (\langle \nu x \rangle P, \langle \nu x \rangle Q) \in ?X \cup \text{weakBisim}$ 
by blast
    moreover have  $\text{weakBisim} \subseteq ?X \cup \text{weakBisim}$  by blast
    moreover have eqvt weakBisim by simp
    moreover have eqvt (?X  $\cup$  weakBisim)
      by(auto simp add: eqvt-def dest: Weak-Early-Bisim.eqvtI)+
    ultimately have  $\langle \nu x \rangle P \rightsquigarrow \langle (?X \cup \text{weakBisim}) \rangle \langle \nu x \rangle Q$ 
      by(rule Weak-Early-Sim-Pres.resPres)
  }
  with  $\langle (xP, xQ) \in ?X \rangle$  show ?case by blast
next
  case(cSym xP xQ)
  thus ?case by(blast dest: Weak-Early-Bisim.symmetric)
qed
qed

```

lemma *matchPres*:

```

fixes  $P :: pi$ 
and  $Q :: pi$ 
and  $a :: name$ 
and  $b :: name$ 

```

assumes $P \approx Q$

shows $[a \frown b]P \approx [a \frown b]Q$

```

proof –
let  $?X = \{([a \frown b]P, [a \frown b]Q) \mid a b P Q. P \approx Q\}$ 
from  $\langle P \approx Q \rangle$  have  $([a \frown b]P, [a \frown b]Q) \in ?X$  by blast
thus ?thesis
proof(coinduct rule: weakBisimCoinduct)
  case(cSim abP abQ)
  {
    fix  $P Q a b$ 
    assume  $P \approx Q$ 
    hence  $P \rightsquigarrow \langle \text{weakBisim} \rangle Q$  by(rule weakBisimE)
    moreover have  $\text{weakBisim} \subseteq (?X \cup \text{weakBisim})$  by blast
  }

```

```

moreover have  $\bigwedge P Q a. P \approx Q \implies [a \frown a]P \approx Q$ 
by (metis (full-types) strongBisimWeakBisim Strong-Early-Bisim-SC.matchId
Weak-Early-Bisim.transitive)
ultimately have  $[a \frown b]P \rightsquigarrow \langle (?X \cup \text{weakBisim}) \rangle [a \frown b]Q$ 
by(rule Weak-Early-Sim-Pres.matchPres)
}
with  $\langle (abP, abQ) \in ?X \rangle$  show ?case by blast
next
case(cSym abP abQ)
thus ?case by(blast dest: Weak-Early-Bisim.symmetric)
qed
qed

```

lemma *mismatchPres*:

```

fixes  $P :: pi$ 
and  $Q :: pi$ 
and  $a :: name$ 
and  $b :: name$ 

assumes  $P \approx Q$ 

shows  $[a \neq b]P \approx [a \neq b]Q$ 
proof –
let  $?X = \{([a \neq b]P, [a \neq b]Q) \mid a b P Q. P \approx Q\}$ 
from  $\langle P \approx Q \rangle$  have  $([a \neq b]P, [a \neq b]Q) \in ?X$  by blast
thus ?thesis
proof(coinduct rule: weakBisimCoinduct)
case(cSim abP abQ)
{
fix  $P Q a b$ 
assume  $P \approx Q$ 
hence  $P \rightsquigarrow \langle \text{weakBisim} \rangle Q$  by(rule weakBisimE)
moreover have  $\text{weakBisim} \subseteq (?X \cup \text{weakBisim})$  by blast
moreover have  $\bigwedge P Q a b. \llbracket P \approx Q; a \neq b \rrbracket \implies [a \neq b]P \approx Q$ 
by (metis (full-types) strongBisimWeakBisim Strong-Early-Bisim-SC.mismatchId
Weak-Early-Bisim.transitive)
ultimately have  $[a \neq b]P \rightsquigarrow \langle (?X \cup \text{weakBisim}) \rangle [a \neq b]Q$ 
by(rule Weak-Early-Sim-Pres.mismatchPres)
}
with  $\langle (abP, abQ) \in ?X \rangle$  show ?case by blast
next
case(cSym abP abQ)
thus ?case by(blast dest: Weak-Early-Bisim.symmetric)
qed
qed

```

lemma *parPres*:

```

fixes  $P :: pi$ 
and  $Q :: pi$ 

```

```

and  $R :: pi$ 

assumes  $P \approx Q$ 

shows  $P \parallel R \approx Q \parallel R$ 
proof –
let  $?X = \{(resChain\ lst\ (P \parallel R),\ resChain\ lst\ (Q \parallel R)) \mid lst\ P\ R\ Q.\ P \approx Q\}$ 
have  $BC: \bigwedge P\ Q.\ P \parallel Q = resChain\ []\ (P \parallel Q)$  by auto
from  $\langle P \approx Q \rangle$  have  $(P \parallel R,\ Q \parallel R) \in ?X$  by (blast intro: BC)
thus ?thesis
proof (coinduct rule: weakBisimCoinduct)
  case (cSym PR QR)
  {
    fix  $P\ Q\ R\ lst$ 
    assume  $P \approx Q$ 
    moreover hence  $P \rightsquigarrow \langle weakBisim \rangle Q$  by (rule weakBisimE)
    moreover have  $\bigwedge P\ Q\ R.\ P \approx Q \implies (P \parallel R,\ Q \parallel R) \in ?X$  using  $BC$ 
      by blast
    moreover {
      fix  $PR\ QR\ x$ 
      assume  $(PR,\ QR) \in ?X$ 
      then obtain  $lst\ P\ Q\ R$  where  $P \approx Q$  and  $A: PR = resChain\ lst\ (P \parallel R)$ 
and  $B: QR = resChain\ lst\ (Q \parallel R)$ 
      by auto
      from  $A$  have  $\langle \nu x \rangle PR = resChain\ (x\#\lst)\ (P \parallel R)$  by auto
      moreover from  $B$  have  $\langle \nu x \rangle QR = resChain\ (x\#\lst)\ (Q \parallel R)$  by auto
      ultimately have  $(\langle \nu x \rangle PR,\ \langle \nu x \rangle QR) \in ?X$  using  $\langle P \approx Q \rangle$ 
      by blast
    }
    note  $Res = this$ 
    ultimately have  $P \parallel R \rightsquigarrow \langle ?X \rangle Q \parallel R$ 
      by (rule-tac Weak-Early-Sim-Pres.parPres)
    moreover have eqvt ?X
      by (auto simp add: eqvt-def) (blast intro: eqvts)
    ultimately have  $resChain\ lst\ (P \parallel R) \rightsquigarrow \langle ?X \rangle resChain\ lst\ (Q \parallel R)$  using
Res
      by (rule-tac Weak-Early-Sim-Pres.resChainI)
    hence  $resChain\ lst\ (P \parallel R) \rightsquigarrow \langle (?X \cup weakBisim) \rangle resChain\ lst\ (Q \parallel R)$ 
      by (force intro: Weak-Early-Sim.monotonic)
  }
  with  $\langle (PR,\ QR) \in ?X \rangle$  show  $PR \rightsquigarrow \langle (?X \cup weakBisim) \rangle QR$ 
    by blast
  next
    case (cSym PR QR)
    thus ?case by (blast dest: Weak-Early-Bisim.symmetric)
  qed
qed

lemma bangPres:

```

```

fixes  $P :: pi$ 
and  $Q :: pi$ 

assumes  $PBisimQ: P \approx Q$ 

shows  $!P \approx !Q$ 
proof –
  let  $?X = (bangRel\ weakBisim)$ 
  let  $?Y = Strong\text{-}Early\text{-}Bisim.bisim\ O\ (bangRel\ weakBisim)\ O\ Strong\text{-}Early\text{-}Bisim.bisim$ 

  from  $Weak\text{-}Early\text{-}Bisim.eqvt\ Strong\text{-}Early\text{-}Bisim.eqvt$  have  $eqvtY: eqvt\ ?Y$  by( $blast$ 
   $intro: eqvtBangRel$ )
  have  $XsubY: ?X \subseteq ?Y$  by( $auto\ intro: Strong\text{-}Early\text{-}Bisim.reflexive$ )

  have  $RelStay: \bigwedge P\ Q. (P \parallel !P, Q) \in ?Y \implies (!P, Q) \in ?Y$ 
  proof( $auto$ )
    fix  $P\ Q\ R\ T$ 
    assume  $PBisimQ: P \parallel !P \sim_e Q$ 
    and  $QBRR: (Q, R) \in bangRel\ weakBisim$ 
    and  $RBisimT: R \sim_e T$ 
    have  $!P \sim_e Q$ 
    proof –
      have  $!P \sim_e P \parallel !P$  by( $rule\ Strong\text{-}Early\text{-}Bisim\text{-}SC.bangSC$ )
      thus  $?thesis$  using  $PBisimQ$  by( $rule\ Strong\text{-}Early\text{-}Bisim.transitive$ )
    qed
    with  $QBRR\ RBisimT$  show  $(!P, T) \in ?Y$  by  $blast$ 
  qed

  have  $ParCompose: \bigwedge P\ Q\ R\ T. [P \approx Q; (R, T) \in ?Y] \implies (P \parallel R, Q \parallel T) \in$ 
   $?Y$ 
  proof –
    fix  $P\ Q\ R\ T$ 
    assume  $PBisimQ: P \approx Q$ 
    and  $RYT: (R, T) \in ?Y$ 
    thus  $(P \parallel R, Q \parallel T) \in ?Y$ 
    proof( $auto$ )
      fix  $T'\ R'$ 
      assume  $T'BisimT: T' \sim_e T$  and  $RBisimR': R \sim_e R'$ 
      and  $R'BRT': (R', T') \in bangRel\ weakBisim$ 
      have  $P \parallel R \sim_e P \parallel R'$ 
      proof –
        from  $RBisimR'$  have  $R \parallel P \sim_e R' \parallel P$  by( $rule\ Strong\text{-}Early\text{-}Bisim\text{-}Pres.parPres$ )
        moreover have  $P \parallel R \sim_e R \parallel P$  and  $R' \parallel P \sim_e P \parallel R'$  by( $rule$ 
         $Strong\text{-}Early\text{-}Bisim\text{-}SC.parSym$ )+
        ultimately show  $?thesis$  by( $blast\ intro: Strong\text{-}Early\text{-}Bisim.transitive$ )
      qed
      moreover from  $PBisimQ\ R'BRT'$  have  $(P \parallel R', Q \parallel T') \in bangRel\ weakBisim$ 
by( $rule\ BRPar$ )
      moreover have  $Q \parallel T' \sim_e Q \parallel T$ 

```

proof –
from $T' \text{Bisim} T$ **have** $T' \parallel Q \sim_e T \parallel Q$ **by**(rule *Strong-Early-Bisim-Pres.parPres*)
moreover have $Q \parallel T' \sim_e T' \parallel Q$ **and** $T \parallel Q \sim_e Q \parallel T$ **by**(rule *Strong-Early-Bisim-SC.parSym*)
ultimately show $?thesis$ **by**(blast intro: *Strong-Early-Bisim.transitive*)
qed
ultimately show $?thesis$ **by** blast
qed
qed

have *ResCong*: $\bigwedge P Q x. (P, Q) \in ?Y \implies (\langle \nu x \rangle P, \langle \nu x \rangle Q) \in ?Y$
by(auto intro: *BRRes Strong-Early-Bisim-Pres.resPres transitive*)

have *Sim*: $\bigwedge P Q. (P, Q) \in ?X \implies P \rightsquigarrow \langle ?Y \rangle Q$
proof –
fix $P Q$
assume $(P, Q) \in ?X$
thus $P \rightsquigarrow \langle ?Y \rangle Q$
proof(*induct*)
case(*BRBang P Q*)
have $P \approx Q$ **by** fact
moreover hence $P \rightsquigarrow \langle \text{weakBisim} \rangle Q$ **by**(blast dest: *weakBisimE*)
moreover have $\bigwedge P Q. P \approx Q \implies P \rightsquigarrow \langle \text{weakBisim} \rangle Q$ **by**(blast dest: *weakBisimE*)
moreover from *Strong-Early-Bisim.eqt Weak-Early-Bisim.eqt* **have** *eqt*
 $?Y$ **by**(blast intro: *eqtBangRel*)

ultimately show $!P \rightsquigarrow \langle ?Y \rangle !Q$ **using** *ParCompose ResCong RelStay XsubY*
by(rule-tac *Weak-Early-Sim-Pres.bangPres, simp-all*)
next
case(*BRPar P Q R T*)
have *PBiSimQ*: $P \approx Q$ **by** fact
moreover have *RBangRelT*: $(R, T) \in ?X$ **by** fact
have *RSimT*: $R \rightsquigarrow \langle ?Y \rangle T$ **by** fact
moreover from *PBiSimQ* **have** $P \rightsquigarrow \langle \text{weakBisim} \rangle Q$ **by**(blast dest: *weakBisimE*)
moreover from *RBangRelT* **have** $(R, T) \in ?Y$ **by**(blast intro: *Strong-Early-Bisim.reflexive*)
ultimately show $P \parallel R \rightsquigarrow \langle ?Y \rangle Q \parallel T$ **using** *ParCompose ResCong eqt*
 $eqtY$
by(rule-tac *Weak-Early-Sim-Pres.parCompose*)
next
case(*BRRes P Q x*)
have $P \rightsquigarrow \langle ?Y \rangle Q$ **by** fact
thus $\langle \nu x \rangle P \rightsquigarrow \langle ?Y \rangle \langle \nu x \rangle Q$ **using** *ResCong eqtY XsubY*
by(rule-tac *Weak-Early-Sim-Pres.resPres, simp-all*)
qed
qed

from *PBiSimQ* **have** $(!P, !Q) \in ?X$ **by**(rule *BRBang*)

```

moreover from Weak-Early-Bisim.eqt have eqvt (bangRel weakBisim) by(rule eqvtBangRel)
ultimately show ?thesis
  apply(coinduct rule: Weak-Early-Bisim.transitive-coinduct-weak)
  apply(blast intro: Sim)
  by(blast dest: bangRelSymetric Weak-Early-Bisim.symetric intro: Strong-Early-Bisim.reflexive)
qed

```

```

lemma bangRelSubWeakBisim:
  shows bangRel weakBisim  $\subseteq$  weakBisim
proof(auto)
  fix a b
  assume  $(a, b) \in$  bangRel weakBisim
  thus  $a \approx b$ 
  proof(induct)
    fix P Q
    assume  $P \approx Q$ 
    thus  $!P \approx !Q$  by(rule bangPres)
  next
    fix P Q R T
    assume  $R \approx T$  and  $P \approx Q$ 
    thus  $R \parallel P \approx T \parallel Q$  by(metis parPres parSym Weak-Early-Bisim.transitive)
  next
    fix P Q
    fix a::name
    assume  $P \approx Q$ 
    thus  $\langle \nu a \rangle P \approx \langle \nu a \rangle Q$  by(rule resPres)
  qed
qed

end

```

```

theory Weak-Early-Cong-Pres
  imports Weak-Early-Cong Weak-Early-Step-Sim-Pres Weak-Early-Bisim-Pres
begin

```

```

lemma tauPres:
  fixes P :: pi
  and Q :: pi

  assumes  $P \simeq Q$ 

  shows  $\tau.(P) \simeq \tau.(Q)$ 
proof –
  from assms have  $P \approx Q$  by(rule congruenceWeakBisim)
  thus ?thesis by(force intro: Weak-Early-Step-Sim-Pres.tauPres simp add: weak-Congruence-def dest: weakBisimE(2))
qed

```

```

lemma outputPres:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 

  assumes  $P \simeq Q$ 

  shows  $a\{b\}.P \simeq a\{b\}.Q$ 
proof –
  from assms have  $P \approx Q$  by(rule congruenceWeakBisim)
  thus ?thesis by(force intro: Weak-Early-Step-Sim-Pres.outputPres simp add: weakCongruence-def dest: weakBisimE(2))
qed

lemma matchPres:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $a :: name$ 
  and  $b :: name$ 

  assumes  $P \simeq Q$ 

  shows  $[a\frown b]P \simeq [a\frown b]Q$ 
using assms
by(auto simp add: weakCongruence-def intro: Weak-Early-Step-Sim-Pres.matchPres)

lemma mismatchPres:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $a :: name$ 
  and  $b :: name$ 

  assumes  $P \simeq Q$ 

  shows  $[a\neq b]P \simeq [a\neq b]Q$ 
using assms
by(auto simp add: weakCongruence-def intro: Weak-Early-Step-Sim-Pres.mismatchPres)

lemma sumPres:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $R :: pi$ 

  assumes  $P \simeq Q$ 

  shows  $P \oplus R \simeq Q \oplus R$ 
using assms
by(auto simp add: weakCongruence-def intro: Weak-Early-Step-Sim-Pres.sumPres Weak-Early-Bisim.reflexive)

```

```

lemma parPres:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $R :: pi$ 

  assumes  $P \simeq Q$ 

  shows  $P \parallel R \simeq Q \parallel R$ 
proof –
  have  $\bigwedge P Q R. [P \rightsquigarrow\langle\langle weakBisim \rangle\rangle Q; P \approx Q] \implies P \parallel R \rightsquigarrow\langle\langle weakBisim \rangle\rangle Q \parallel R$ 
  proof –
    fix  $P Q R$ 
    assume  $P \rightsquigarrow\langle\langle weakBisim \rangle\rangle Q$  and  $P \approx Q$ 
    thus  $P \parallel R \rightsquigarrow\langle\langle weakBisim \rangle\rangle Q \parallel R$ 
    using Weak-Early-Bisim-Pres.parPres Weak-Early-Bisim-Pres.resPres Weak-Early-Bisim.reflexive
    Weak-Early-Bisim.eqvt
    by(blast intro: Weak-Early-Step-Sim-Pres.parPres)
  qed
  moreover from assms have  $P \approx Q$  by(rule congruenceWeakBisim)
  ultimately show ?thesis using assms
  by(auto simp add: weakCongruence-def dest: weakBisimE)
qed

```

```

lemma resPres:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $x :: name$ 

  assumes  $P eqQ: P \simeq Q$ 

  shows  $\langle \nu x \rangle P \simeq \langle \nu x \rangle Q$ 
proof –
  have  $\bigwedge P Q x. P \rightsquigarrow\langle\langle weakBisim \rangle\rangle Q \implies \langle \nu x \rangle P \rightsquigarrow\langle\langle weakBisim \rangle\rangle \langle \nu x \rangle Q$ 
  proof –
    fix  $P Q x$ 
    assume  $P \rightsquigarrow\langle\langle weakBisim \rangle\rangle Q$ 
    with Weak-Early-Bisim.eqvt Weak-Early-Bisim-Pres.resPres show  $\langle \nu x \rangle P$ 
     $\rightsquigarrow\langle\langle weakBisim \rangle\rangle \langle \nu x \rangle Q$ 
    by(blast intro: Weak-Early-Step-Sim-Pres.resPres)
  qed
  with assms show ?thesis by(simp add: weakCongruence-def)
qed

```

```

lemma bangPres:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 

  assumes  $P \simeq Q$ 

```

```

  shows  $!P \simeq !Q$ 
using assms
proof(induct rule: weakCongISym2)
  case(cSim P Q)
  let  $?X = \{(P, Q) \mid P \ Q. P \simeq Q\}$ 
  from  $\langle P \simeq Q \rangle$  have  $(P, Q) \in ?X$  by auto
  moreover have  $\bigwedge P \ Q. (P, Q) \in ?X \implies P \rightsquigarrow\langle\langle\text{weakBisim}\rangle\rangle Q$  by(auto simp
add: weakCongruence-def)
  moreover from congruenceWeakBisim have  $?X \subseteq \text{weakBisim}$  by auto
  ultimately have  $!P \rightsquigarrow\langle\langle\text{bangRel weakBisim}\rangle\rangle !Q$  using Weak-Early-Bisim.eqvt
  by(rule Weak-Early-Step-Sim-Pres.bangPres)
  moreover have  $\text{bangRel weakBisim} \subseteq \text{weakBisim}$  by(rule bangRelSubWeak-
Bisim)
  ultimately show  $!P \rightsquigarrow\langle\langle\text{weakBisim}\rangle\rangle !Q$ 
  by(rule Weak-Early-Step-Sim.monotonic)
qed

end

theory Weak-Early-Cong-Subst-Pres
  imports Weak-Early-Cong-Subst Weak-Early-Cong-Pres
begin

lemma weakCongStructCong:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 

  assumes  $P \equiv_s Q$ 

  shows  $P \simeq^s Q$ 
using assms
by(metis earlyCongStructCong strongEqWeakCong)

lemma tauPres:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 

  assumes  $P \simeq^s Q$ 

  shows  $\tau.(P) \simeq^s \tau.(Q)$ 
using assms
by(auto simp add: weakCongruenceSubst-def intro: Weak-Early-Cong-Pres.tauPres)

lemma inputPres:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $a :: name$ 
  and  $x :: name$ 

```

```

assumes  $PeqQ: P \simeq^s Q$ 

shows  $a\langle x \rangle.P \simeq^s a\langle x \rangle.Q$ 
proof(auto simp add: weakCongruenceSubst-def)
  fix  $s::(\text{name} \times \text{name}) \text{ list}$ 

  from congruenceWeakBisim have  $\bigwedge P Q a x s. [P[\langle s \rangle] \simeq^s Q[\langle s \rangle]; x \# s] \implies (a\langle x \rangle.P)[\langle s \rangle] \simeq (a\langle x \rangle.Q)[\langle s \rangle]$ 
    apply(auto simp add: weakCongruenceSubst-def weakCongruence-def)
    apply(rule Weak-Early-Step-Sim-Pres.inputPres, auto)
    apply(erule-tac x=[(x, y)] in allE, auto)
    apply(rule Weak-Early-Step-Sim-Pres.inputPres, auto)
    by(erule-tac x=[(x, y)] in allE, auto)

  then obtain  $c::\text{name}$  where  $cFreshP: c \# P$  and  $cFreshQ: c \# Q$  and  $cFreshs: c \# s$ 
    by(force intro: name-exists-fresh[of (P, Q, s)])

  from  $PeqQ$  have  $P[\langle [(x, c)] \cdot s \rangle] \simeq^s Q[\langle [(x, c)] \cdot s \rangle]$  by(rule partUnfold)
  hence  $([(x, c)] \cdot P[\langle [(x, c)] \cdot s \rangle]) \simeq^s ([[(x, c)] \cdot Q[\langle [(x, c)] \cdot s \rangle]])$  by(rule Weak-Early-Cong-Subst.eqvtI)
  hence  $([(x, c)] \cdot P)[\langle s \rangle] \simeq^s ([[(x, c)] \cdot Q][\langle s \rangle])$  by simp
  hence  $(a\langle c \rangle.([(x, c)] \cdot P))[\langle s \rangle] \simeq (a\langle c \rangle.([(x, c)] \cdot Q))[\langle s \rangle]$  using  $cFreshs$ 
by(rule Input)

  moreover from  $cFreshP$   $cFreshQ$  have  $a\langle x \rangle.P = a\langle c \rangle.([(x, c)] \cdot P)$  and
 $a\langle x \rangle.Q = a\langle c \rangle.([(x, c)] \cdot Q)$ 
    by(simp add: Agent.alphaInput)+

  ultimately show  $(a\langle x \rangle.P)[\langle s \rangle] \simeq (a\langle x \rangle.Q)[\langle s \rangle]$  by simp
qed

lemma outputPres:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 

  assumes  $P \simeq^s Q$ 

  shows  $a\{b\}.P \simeq^s a\{b\}.Q$ 
using assms
by(auto simp add: weakCongruenceSubst-def intro: Weak-Early-Cong-Pres.outputPres)

lemma matchPres:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $a :: \text{name}$ 
  and  $b :: \text{name}$ 

```

```

assumes  $P \simeq^s Q$ 

shows  $[a \frown b]P \simeq^s [a \frown b]Q$ 
using assms
by(auto simp add: weakCongruenceSubst-def intro: Weak-Early-Cong-Pres.matchPres)

lemma mismatchPres:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $a :: name$ 
  and  $b :: name$ 

  assumes  $P \simeq^s Q$ 

  shows  $[a \neq b]P \simeq^s [a \neq b]Q$ 
using assms
by(auto simp add: weakCongruenceSubst-def intro: Weak-Early-Cong-Pres.mismatchPres)

lemma sumPres:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $R :: pi$ 

  assumes  $P \simeq^s Q$ 

  shows  $P \oplus R \simeq^s Q \oplus R$ 
using assms
by(auto simp add: weakCongruenceSubst-def intro: Weak-Early-Cong-Pres.sumPres)

lemma parPres:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $R :: pi$ 

  assumes  $P \simeq^s Q$ 

  shows  $P \parallel R \simeq^s Q \parallel R$ 
using assms
by(auto simp add: weakCongruenceSubst-def intro: Weak-Early-Cong-Pres.parPres)

lemma resPres:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $x :: name$ 

  assumes  $PeqQ: P \simeq^s Q$ 

  shows  $\langle \nu x \rangle P \simeq^s \langle \nu x \rangle Q$ 
proof(auto simp add: weakCongruenceSubst-def)

```

fix $s::(\text{name} \times \text{name}) \text{ list}$

have $\text{Goal}: \bigwedge P Q x s. \llbracket P[\langle s \rangle] \rightsquigarrow \langle\langle \text{weakBisim} \rangle\rangle Q[\langle s \rangle]; x \# s \rrbracket \implies (\langle \nu x \rangle P)[\langle s \rangle] \rightsquigarrow \langle\langle \text{weakBisim} \rangle\rangle (\langle \nu x \rangle Q)[\langle s \rangle]$

by($\text{force intro: Weak-Early-Step-Sim-Pres.resPres Weak-Early-Bisim-Pres.resPres Weak-Early-Bisim.eqvt}$)

then obtain $c::\text{name}$ **where** $c\text{Fresh}P: c \# P$ **and** $c\text{Fresh}Q: c \# Q$ **and** $c\text{Fresh}s: c \# s$

by($\text{force intro: name-exists-fresh[of (P, Q, s)]}$)

from $P \text{ eq } Q$ **have** $P[\langle [(x, c)] \cdot s \rangle] \rightsquigarrow \langle\langle \text{weakBisim} \rangle\rangle Q[\langle [(x, c)] \cdot s \rangle]$ **and**

$Q[\langle [(x, c)] \cdot s \rangle] \rightsquigarrow \langle\langle \text{weakBisim} \rangle\rangle P[\langle [(x, c)] \cdot s \rangle]$

by($\text{force simp add: weakCongruenceSubst-def weakCongruence-def}$) $+$

hence $([(x, c)] \cdot (P[\langle [(x, c)] \cdot s \rangle])) \rightsquigarrow \langle\langle \text{weakBisim} \rangle\rangle ((x, c) \cdot (Q[\langle [(x, c)] \cdot s \rangle]))$ **and**

$((x, c) \cdot (Q[\langle [(x, c)] \cdot s \rangle])) \rightsquigarrow \langle\langle \text{weakBisim} \rangle\rangle ((x, c) \cdot (P[\langle [(x, c)] \cdot s \rangle]))$

by($\text{blast intro: Weak-Early-Step-Sim.eqvtI Weak-Early-Bisim.eqvt}$) $+$

hence $([(x, c)] \cdot P)[\langle s \rangle] \rightsquigarrow \langle\langle \text{weakBisim} \rangle\rangle ((x, c) \cdot Q)[\langle s \rangle]$ **and**

$((x, c) \cdot Q)[\langle s \rangle] \rightsquigarrow \langle\langle \text{weakBisim} \rangle\rangle ((x, c) \cdot P)[\langle s \rangle]$ **by** $\text{simp}+$

with $c\text{Fresh}s$ **have** $(\langle \nu c \rangle([(x, c)] \cdot P))[\langle s \rangle] \rightsquigarrow \langle\langle \text{weakBisim} \rangle\rangle (\langle \nu c \rangle([(x, c)] \cdot Q))[\langle s \rangle]$ **and**

$(\langle \nu c \rangle([(x, c)] \cdot Q))[\langle s \rangle] \rightsquigarrow \langle\langle \text{weakBisim} \rangle\rangle (\langle \nu c \rangle([(x, c)] \cdot P))[\langle s \rangle]$

by(blast intro: Goal) $+$

moreover from $c\text{Fresh}P$ $c\text{Fresh}Q$ **have** $\langle \nu x \rangle P = \langle \nu c \rangle([(x, c)] \cdot P)$ **and**

$\langle \nu x \rangle Q = \langle \nu c \rangle([(x, c)] \cdot Q)$

by($\text{simp add: alphaRes}$) $+$

ultimately show $(\langle \nu x \rangle P)[\langle s \rangle] \simeq (\langle \nu x \rangle Q)[\langle s \rangle]$

by($\text{simp add: weakCongruence-def}$)

qed

lemma bangPres :

fixes $P :: pi$

and $Q :: pi$

assumes $P \simeq^s Q$

shows $!P \simeq^s !Q$

using assms

by($\text{auto simp add: weakCongruenceSubst-def intro: Weak-Early-Cong-Pres.bangPres}$)

end

```

theory Strong-Late-Expansion-Law
  imports Strong-Late-Bisim-SC
begin

nominal-primrec summands :: pi  $\Rightarrow$  pi set where
  summands 0 = {}
| summands ( $\tau$ .(P)) = { $\tau$ .(P)}
|  $x \# a \Rightarrow$  summands ( $a <x>$ .P) = { $a <x>$ .P}
| summands ( $a\{b\}$ .P) = { $a\{b\}$ .P}
| summands ( $[a \frown b]$ P) = {}
| summands ( $[a \neq b]$ P) = {}
| summands (P  $\oplus$  Q) = (summands P)  $\cup$  (summands Q)
| summands (P  $\parallel$  Q) = {}
| summands ( $<\nu x>$ P) = (if ( $\exists a P'$ .  $a \neq x \wedge P = a\{x\}$ .P') then ({ $<\nu x>$ P}) else
  {})
| summands (!P) = {}
apply(auto simp add: fresh-singleton name-fresh-abs fresh-set-empty fresh-singleton
pi.fresh)
apply(finite-guess)+
by(fresh-guess)+

lemma summandsInput[simp]:
  fixes a :: name
  and x :: name
  and P :: pi

  shows summands ( $a <x>$ .P) = { $a <x>$ .P}
proof –
  obtain y where yineqa:  $y \neq a$  and yFreshP:  $y \# P$ 
  by(force intro: name-exists-fresh[of (a, P)] simp add: fresh-prod)
  from yFreshP have  $a <x>$ .P =  $a <y>$ .( $[(x, y)] \cdot P$ ) by(simp add: alphaInput)
  with yineqa show ?thesis by simp
qed

lemma finiteSummands:
  fixes P :: pi

  shows finite(summands P)
by(induct P rule: pi.induct) auto

lemma boundSummandDest[dest]:
  fixes x :: name
  and y :: name
  and P' :: pi
  and P :: pi

  assumes  $<\nu x>x\{y\}$ .P'  $\in$  summands P

  shows False

```

using *assms*
by(*induct P rule: pi.induct, auto simp add: if-split pi.inject name-abs-eq name-calc*)

lemma *summandFresh*:

fixes $P :: pi$
and $Q :: pi$
and $x :: name$

assumes $P \in \text{summands } Q$
and $x \# Q$

shows $x \# P$

using *assms*

by(*nominal-induct Q avoiding: P rule: pi.strong-induct, auto simp add: if-split*)

nominal-primrec *hnf* :: $pi \Rightarrow bool$ **where**

$hnf \mathbf{0} = True$
 $hnf (\tau.(P)) = True$
 $x \# a \Longrightarrow hnf (a<x>.P) = True$
 $hnf (a\{b\}.P) = True$
 $hnf ([a\wedge b]P) = False$
 $hnf ([a\neq b]P) = False$
 $hnf (P \oplus Q) = ((hnf P) \wedge (hnf Q) \wedge P \neq \mathbf{0} \wedge Q \neq \mathbf{0})$
 $hnf (P \parallel Q) = False$
 $hnf (<\nu x>P) = (\exists a P'. a \neq x \wedge P = a\{x\}.P')$
 $hnf (!P) = False$

apply(*auto simp add: fresh-bool*)

apply(*finite-guess*)**+**

by(*fresh-guess*)**+**

lemma *hnfInput[simp]*:

fixes $a :: name$
and $x :: name$
and $P :: pi$

shows $hnf (a<x>.P)$

proof –

obtain y **where** *yineqa: $y \neq a$* **and** *yFreshP: $y \# P$*

by(*force intro: name-exists-fresh[of (a, P)] simp add: fresh-prod*)

from *yFreshP* **have** $a<x>.P = a<y>.([x, y] \cdot P)$ **by**(*simp add: alphaInput*)

with *yineqa* **show** *?thesis* **by** *simp*

qed

lemma *summandTransition*:

fixes $P :: pi$
and $a :: name$
and $x :: name$
and $b :: name$
and $P' :: pi$

assumes *hnf P*

shows $P \mapsto_{\tau} \prec P' = (\tau.(P') \in \text{summands } P)$

and $P \mapsto a \langle x \rangle \prec P' = (a \langle x \rangle . P' \in \text{summands } P)$

and $P \mapsto a[b] \prec P' = (a\{b\} . P' \in \text{summands } P)$

and $a \neq x \implies P \mapsto a \langle \nu x \rangle \prec P' = (\langle \nu x \rangle a\{x\} . P' \in \text{summands } P)$

proof –

from *assms* **show** $P \mapsto_{\tau} \prec P' = (\tau.(P') \in \text{summands } P)$

proof(*induct P rule: pi.induct*)

case *PiNil*

show *?case* **by** *auto*

next

case(*Output a b P*)

show *?case* **by** *auto*

next

case(*Tau P*)

have $\tau.(P) \mapsto_{\tau} \prec P' \implies \tau.(P') \in \text{summands } (\tau.(P))$

by(*auto elim: tauCases simp add: pi.inject residual.inject*)

moreover **have** $\tau.(P') \in \text{summands } (\tau.(P)) \implies \tau.(P) \mapsto_{\tau} \prec P'$

by(*auto simp add: pi.inject intro: transitions.Tau*)

ultimately **show** *?case* **by** *blast*

next

case(*Input a x P*)

show *?case* **by** *auto*

next

case(*Match a b P*)

have *hnf* ($[a \sim b]P$) **by** *fact*

hence *False* **by** *simp*

thus *?case* **by** *simp*

next

case(*Mismatch a b P*)

have *hnf* ($[a \neq b]P$) **by** *fact*

hence *False* **by** *simp*

thus *?case* **by** *simp*

next

case(*Sum P Q*)

have *hnf* ($P \oplus Q$) **by** *fact*

hence *Phnf: hnf P* **and** *Qhnf: hnf Q* **by** *simp+*

have *IHP*: $P \mapsto_{\tau} \prec P' = (\tau.(P') \in \text{summands } P)$

proof –

have *hnf P* $\implies P \mapsto_{\tau} \prec P' = (\tau.(P') \in \text{summands } P)$ **by** *fact*

with *Phnf* **show** *?thesis* **by** *simp*

qed

have *IHQ*: $Q \mapsto_{\tau} \prec P' = (\tau.(P') \in \text{summands } Q)$

proof –

have *hnf Q* $\implies Q \mapsto_{\tau} \prec P' = (\tau.(P') \in \text{summands } Q)$ **by** *fact*

```

    with Qhmf show ?thesis by simp
  qed

  from IHP IHQ have  $P \oplus Q \mapsto \tau \prec P' \implies \tau.(P') \in \text{summands } (P \oplus Q)$ 
    by(erule-tac sumCases, auto)
  moreover from IHP IHQ have  $\tau.(P') \in \text{summands } (P \oplus Q) \implies P \oplus Q \mapsto \tau \prec P'$ 
    by(auto dest: Sum1 Sum2)
  ultimately show ?case by blast
next
  case(Par P Q)
  have hnf (P || Q) by fact
  hence False by simp
  thus ?case by simp
next
  case(Res x P)
  thus ?case by(auto elim: resCasesF)
next
  case(Bang P)
  have hnf (!P) by fact
  hence False by simp
  thus ?case by simp
qed
next
  from assms show  $P \mapsto a\langle x \rangle \prec P' = (a\langle x \rangle.P' \in \text{summands } P)$ 
  proof(induct P rule: pi.induct)
    case PiNil
    show ?case by auto
  next
    case(Output c b P)
    show ?case by auto
  next
    case(Tau P)
    show ?case by auto
  next
    case(Input b y P)
    have  $b\langle y \rangle.P \mapsto a\langle x \rangle \prec P' \implies a\langle x \rangle.P' \in \text{summands } (b\langle y \rangle.P)$ 
      by(auto elim: inputCases' simp add: pi.inject residual.inject)
    moreover have  $a\langle x \rangle.P' \in \text{summands } (b\langle y \rangle.P) \implies b\langle y \rangle.P \mapsto a\langle x \rangle \prec P'$ 
      apply(auto simp add: pi.inject name-abs-eq intro: Late-Semantics.Input)
      apply(subgoal-tac  $b\langle x \rangle \prec [(x, y)] \cdot P = (b\langle y \rangle \prec [(x, y)] \cdot [(x, y)] \cdot P)$ )
      apply(auto intro: Late-Semantics.Input)
      by(simp add: alphaBoundResidual name-swap)
    ultimately show ?case by blast
  next
    case(Match a b P)
    have hnf ([a  $\frown$  b]P) by fact
    hence False by simp

```

```

    thus ?case by simp
next
  case(Mismatch a b P)
  have hnf ([a≠b]P) by fact
  hence False by simp
  thus ?case by simp
next
  case(Sum P Q)
  have hnf (P ⊕ Q) by fact
  hence Phnf: hnf P and Qhnf: hnf Q by simp+

  have IHP: P ⟶ a⟨x⟩ ◁ P' = (a⟨x⟩.P' ∈ summands P)
  proof -
    have hnf P ⟹ P ⟶ a⟨x⟩ ◁ P' = (a⟨x⟩.P' ∈ summands P) by fact
    with Phnf show ?thesis by simp
  qed

  have IHQ: Q ⟶ a⟨x⟩ ◁ P' = (a⟨x⟩.P' ∈ summands Q)
  proof -
    have hnf Q ⟹ Q ⟶ a⟨x⟩ ◁ P' = (a⟨x⟩.P' ∈ summands Q) by fact
    with Qhnf show ?thesis by simp
  qed

  from IHP IHQ have P ⊕ Q ⟶ a⟨x⟩ ◁ P' ⟹ a⟨x⟩.P' ∈ summands (P ⊕
Q)
    by(erule-tac sumCases, auto)
  moreover from IHP IHQ have a⟨x⟩.P' ∈ summands (P ⊕ Q) ⟹ P ⊕ Q
⟶ a⟨x⟩ ◁ P'
    by(auto dest: Sum1 Sum2)
  ultimately show ?case by blast
next
  case(Par P Q)
  have hnf (P || Q) by fact
  hence False by simp
  thus ?case by simp
next
  case(Res y P)
  have hnf (<νy>P) by fact
  thus ?case by(auto simp add: if-split)
next
  case(Bang P)
  have hnf (!P) by fact
  hence False by simp
  thus ?case by simp
qed
next
  from assms show P ⟶ a[b] ◁ P' = (a{b}.P' ∈ summands P)
  proof(induct P rule: pi.induct)
    case PiNil

```

```

  show ?case by auto
next
case(Output c d P)
have  $c\{d\}.P \mapsto a[b] \prec P' \implies a\{b\}.P' \in \text{summands } (c\{d\}.P)$ 
  by(auto elim: outputCases simp add: residual.inject pi.inject)
moreover have  $a\{b\}.P' \in \text{summands } (c\{d\}.P) \implies c\{d\}.P \mapsto a[b] \prec P'$ 
  by(auto simp add: pi.inject intro: transitions.Output)
ultimately show ?case by blast
next
case(Tau P)
show ?case by auto
next
case(Input c x P)
show ?case by auto
next
case(Match a b P)
have hnf ( $[a \frown b]P$ ) by fact
hence False by simp
thus ?case by simp
next
case(Mismatch a b P)
have hnf ( $[a \neq b]P$ ) by fact
hence False by simp
thus ?case by simp
next
case(Sum P Q)
have hnf ( $P \oplus Q$ ) by fact
hence Phnf: hnf P and Qhnf: hnf Q by simp+

have IHP:  $P \mapsto a[b] \prec P' = (a\{b\}.P' \in \text{summands } P)$ 
proof -
  have hnf P  $\implies P \mapsto a[b] \prec P' = (a\{b\}.P' \in \text{summands } P)$  by fact
  with Phnf show ?thesis by simp
qed

have IHQ:  $Q \mapsto a[b] \prec P' = (a\{b\}.P' \in \text{summands } Q)$ 
proof -
  have hnf Q  $\implies Q \mapsto a[b] \prec P' = (a\{b\}.P' \in \text{summands } Q)$  by fact
  with Qhnf show ?thesis by simp
qed

from IHP IHQ have  $P \oplus Q \mapsto a[b] \prec P' \implies a\{b\}.P' \in \text{summands } (P \oplus Q)$ 
  by(erule-tac sumCases, auto)
moreover from IHP IHQ have  $a\{b\}.P' \in \text{summands } (P \oplus Q) \implies P \oplus Q$ 
 $\mapsto a[b] \prec P'$ 
  by(auto dest: Sum1 Sum2)
ultimately show ?case by blast
next
case(Par P Q)

```

```

    have hnf (P || Q) by fact
    hence False by simp
    thus ?case by simp
  next
    case(Res x P)
    have hnf (<νx>P) by fact
    thus ?case by(force elim: resCasesF outputCases simp add: if-split resid-
ual.inject)
  next
    case(Bang P)
    have hnf (!P) by fact
    hence False by simp
    thus ?case by simp
  qed
next
assume a≠x
with assms show P ↦ a<νx> < P' = (<νx>a{x}.P' ∈ summands P)
proof(nominal-induct P avoiding: x P' rule: pi.strong-induct)
  case PiNil
  show ?case by auto
next
  case(Output a b P)
  show ?case by auto
next
  case(Tau P)
  show ?case by auto
next
  case(Input a x P)
  show ?case by auto
next
  case(Match a b P)
  have hnf ([a↔b]P) by fact
  hence False by simp
  thus ?case by simp
next
  case(Mismatch a b P)
  have hnf ([a≠b]P) by fact
  hence False by simp
  thus ?case by simp
next
  case(Sum P Q)
  have hnf (P ⊕ Q) by fact
  hence Phnf: hnf P and Qhnf: hnf Q by simp+
  have aineqx: a ≠ x by fact

  have IHP: P ↦ a<νx> < P' = (<νx>a{x}.P' ∈ summands P)
  proof -
    have ∧x P'. [[hnf P; a ≠ x]] ⇒ P ↦ a<νx> < P' = (<νx>a{x}.P' ∈
summands P) by fact

```

with *Phnf aineqx* **show** *?thesis* **by** *simp*
qed

have *IHQ*: $Q \mapsto a \langle \nu x \rangle \prec P' = (\langle \nu x \rangle a\{x\}.P' \in \text{summands } Q)$
proof –
have $\bigwedge x Q'. \llbracket \text{hnf } Q; a \neq x \rrbracket \implies Q \mapsto a \langle \nu x \rangle \prec P' = (\langle \nu x \rangle a\{x\}.P' \in \text{summands } Q)$ **by** *fact*
with *Qhnf aineqx* **show** *?thesis* **by** *simp*
qed

from *IHP IHQ* **have** $P \oplus Q \mapsto a \langle \nu x \rangle \prec P' \implies \langle \nu x \rangle a\{x\}.P' \in \text{summands } (P \oplus Q)$
by(*erule-tac sumCases, auto*)
moreover from *IHP IHQ* **have** $\langle \nu x \rangle a\{x\}.P' \in \text{summands } (P \oplus Q) \implies P \oplus Q \mapsto a \langle \nu x \rangle \prec P'$
by(*auto dest: Sum1 Sum2*)
ultimately show *?case* **by** *blast*
next
case(*Par P Q*)
have *hnf* ($P \parallel Q$) **by** *fact*
hence *False* **by** *simp*
thus *?case* **by** *simp*
next
case(*Res y P*)
have *Phnf*: *hnf* ($\langle \nu y \rangle P$) **by** *fact*
then obtain $b P''$ **where** *bineqy*: $b \neq y$ **and** *PeqP''*: $P = b\{y\}.P''$
by *auto*
have $y \# x$ **by** *fact* **hence** *xineqy*: $x \neq y$ **by** *simp*
have *yFreshP'*: $y \# P'$ **by** *fact*
have *aineqx*: $a \neq x$ **by** *fact*
have $\langle \nu y \rangle P \mapsto a \langle \nu x \rangle \prec P' \implies (\langle \nu x \rangle a\{x\}.P' \in \text{summands } (\langle \nu y \rangle P))$
proof –
assume *Trans*: $\langle \nu y \rangle P \mapsto a \langle \nu x \rangle \prec P'$
hence *aeqb*: $a = b$ **using** *xineqy bineqy PeqP''*
by(*induct rule: resCasesB', auto elim: outputCases simp add: residual.inject alpha' abs-fresh pi.inject*)

have *Goal*: $\bigwedge x P'. \llbracket \langle \nu y \rangle b\{y\}.P'' \mapsto b \langle \nu x \rangle \prec P'; x \neq y; x \neq b; x \# P'' \rrbracket$
 \implies

$\langle \nu x \rangle b\{x\}.P' \in \text{summands}(\langle \nu y \rangle b\{y\}.P'')$

proof –
fix $x P'$
assume *xFreshP''*: $(x::\text{name}) \# P''$ **and** *xineqb*: $x \neq b$
assume $\langle \nu y \rangle b\{y\}.P'' \mapsto b \langle \nu x \rangle \prec P'$ **and** *xineqy*: $x \neq y$
moreover from $\langle x \neq b \rangle \langle x \# P'' \rangle \langle x \neq y \rangle$ **have** $x \# b\{y\}.P''$ **by** *simp*
ultimately show $\langle \nu x \rangle b\{x\}.P' \in \text{summands } (\langle \nu y \rangle b\{y\}.P'')$
proof(*induct rule: resCasesB*)
case(*cOpen a P'''*)
have *BoundOutputS* $b = \text{BoundOutputS } a$ **by** *fact* **hence** *beqa*: $b = a$ **by**

simp
have *Trans*: $b\{y\}.P'' \mapsto a[y] \prec P'''$ **by** *fact*
with *PeqP''* **have** $P''eqP'''$: $P'' = P'''$
by(*force elim: outputCases simp add: residual.inject*)
with *bineqy xineqy xFreshP''* **have** $y \# b\{x\}.((x, y)) \cdot P'''$
by(*simp add: name-fresh-abs name-calc name-fresh-left*)
with *bineqy Phnf PeqP'' P''eqP''' xineqb* **show** *?case*
by(*simp only: alphaRes, simp add: name-calc*)
next
case(*cRes P'''*)
have $b\{y\}.P'' \mapsto b\langle \nu x \rangle \prec P'''$ **by** *fact*
hence *False* **by** *auto*
thus *?case* **by** *simp*
qed
qed
obtain *z* **where** *zineqx*: $z \neq x$ **and** *zineqy*: $z \neq y$ **and** *zFreshP''*: $z \# P''$
and *zineqb*: $z \neq b$ **and** *zFreshP'*: $z \# P'$
by(*force intro: name-exists-fresh[of (x, y, b, P'', P')] simp add: fresh-prod*)

from *zFreshP' aeqb PeqP'' Trans* **have** *Trans'*: $\langle \nu y \rangle b\{y\}.P'' \mapsto b\langle \nu z \rangle \prec$
 $[(z, x)] \cdot P'$
by(*simp add: alphaBoundResidual name-swap*)
hence $\langle \nu z \rangle b\{z\}.([(z, x)] \cdot P') \in \text{summands}(\langle \nu y \rangle b\{y\}.P'')$ **using** *zineqy*
zineqb zFreshP''
by(*rule Goal*)
moreover from *bineqy zineqx zFreshP' aineqx aeqb* **have** $x \# b\{z\}.([(z, x)] \cdot$
 $P')$
by(*simp add: name-fresh-left name-calc*)
ultimately have $\langle \nu x \rangle b\{x\}.P' \in \text{summands}(\langle \nu y \rangle b\{y\}.P'')$ **using** *zineqb*
by(*simp add: alphaRes name-calc*)
with *aeqb PeqP''* **show** *?thesis* **by** *blast*
qed
moreover have $\langle \nu x \rangle a\{x\}.P' \in \text{summands}(\langle \nu y \rangle P) \implies \langle \nu y \rangle P \mapsto a\langle \nu x \rangle$
 $\prec P'$
proof –
assume $\langle \nu x \rangle a\{x\}.P' \in \text{summands}(\langle \nu y \rangle P)$
with *PeqP''* **have** *Summ*: $\langle \nu x \rangle a\{x\}.P' \in \text{summands}(\langle \nu y \rangle b\{y\}.P'')$ **by**
simp
moreover with *bineqy xineqy* **have** *aeqb*: $a = b$
by(*auto simp add: if-split pi.inject name-abs-eq name-fresh-fresh*)
from *bineqy xineqy yFreshP'* **have** $y \# b\{x\}.P'$ **by**(*simp add: name-calc*)
with *Summ aeqb bineqy aineqx* **have** $\langle \nu y \rangle b\{y\}.([(x, y)] \cdot P') \in \text{sum-}$
mands($\langle \nu y \rangle b\{y\}.P''$)
by(*simp only: alphaRes, simp add: name-calc*)
with *aeqb PeqP''* **have** $\langle \nu y \rangle P \mapsto a\langle \nu y \rangle \prec [(x, y)] \cdot P'$
by(*auto intro: Open Output simp add: if-split pi.inject name-abs-eq*)
moreover from *yFreshP'* **have** $x \# [(x, y)] \cdot P'$ **by**(*simp add: name-fresh-left*
name-calc)
ultimately show *?thesis* **by**(*simp add: alphaBoundResidual name-swap*)

qed
ultimately show *?case by blast*
next
case(*Bang P*)
have *hnf (!P) by fact*
hence *False by simp*
thus *?case by simp*
qed
qed

definition *expandSet* :: *pi* \Rightarrow *pi* \Rightarrow *pi set* **where**

$$\begin{aligned}
& \text{expandSet } P \ Q \equiv \{ \tau.(P' \parallel Q) \mid P'. \tau.(P') \in \text{summands } P \} \cup \\
& \quad \{ \tau.(P \parallel Q') \mid Q'. \tau.(Q') \in \text{summands } Q \} \cup \\
& \quad \{ a\{b\}.(P' \parallel Q) \mid a \ b \ P'. a\{b\}.P' \in \text{summands } P \} \cup \\
& \quad \{ a\{b\}.(P \parallel Q') \mid a \ b \ Q'. a\{b\}.Q' \in \text{summands } Q \} \cup \\
& \quad \{ a\langle x \rangle.(P' \parallel Q) \mid a \ x \ P'. a\langle x \rangle.P' \in \text{summands } P \wedge x \# Q \} \\
\cup \\
& \quad \{ a\langle x \rangle.(P \parallel Q') \mid a \ x \ Q'. a\langle x \rangle.Q' \in \text{summands } Q \wedge x \# \\
P \} \cup \\
& \quad \{ \langle \nu x \rangle a\{x\}.(P' \parallel Q) \mid a \ x \ P'. \langle \nu x \rangle a\{x\}.P' \in \text{summands } P \\
\wedge x \# Q \} \cup \\
& \quad \{ \langle \nu x \rangle a\{x\}.(P \parallel Q') \mid a \ x \ Q'. \langle \nu x \rangle a\{x\}.Q' \in \text{summands } \\
Q \wedge x \# P \} \cup \\
& \quad \{ \tau.(P'[x::=b] \parallel Q') \mid x \ P' \ b \ Q'. \exists a. a\langle x \rangle.P' \in \text{summands } P \\
\wedge a\{b\}.Q' \in \text{summands } Q \} \cup \\
& \quad \{ \tau.(P' \parallel (Q'[x::=b])) \mid b \ P' \ x \ Q'. \exists a. a\{b\}.P' \in \text{summands } \\
P \wedge a\langle x \rangle.Q' \in \text{summands } Q \} \cup \\
& \quad \{ \tau.(\langle \nu y \rangle (P'[x::=y] \parallel Q')) \mid x \ P' \ y \ Q'. \exists a. a\langle x \rangle.P' \in \\
\text{summands } P \wedge \langle \nu y \rangle a\{y\}.Q' \in \text{summands } Q \wedge y \# P \} \cup \\
& \quad \{ \tau.(\langle \nu y \rangle (P' \parallel (Q'[x::=y]))) \mid y \ P' \ x \ Q'. \exists a. \langle \nu y \rangle a\{y\}.P' \\
\in \text{summands } P \wedge a\langle x \rangle.Q' \in \text{summands } Q \wedge y \# Q \}
\end{aligned}$$

lemma *finiteExpand*:

fixes *P* :: *pi*

and *Q* :: *pi*

shows *finite(expandSet P Q)*

proof –

have *finite* $\{ \tau.(P' \parallel Q) \mid P'. \tau.(P') \in \text{summands } P \}$

by(*induct P rule: pi.induct, auto simp add: pi.inject Collect-ex-eq conj-disj-distribL*
Collect-disj-eq UN-Un-distrib)

moreover have *finite* $\{ \tau.(P \parallel Q') \mid Q'. \tau.(Q') \in \text{summands } Q \}$

by(*induct Q rule: pi.induct, auto simp add: pi.inject Collect-ex-eq conj-disj-distribL*
Collect-disj-eq UN-Un-distrib)

moreover have *finite* $\{ a\{b\}.(P' \parallel Q) \mid a \ b \ P'. a\{b\}.P' \in \text{summands } P \}$

by(*induct P rule: pi.induct, auto simp add: pi.inject Collect-ex-eq conj-disj-distribL*
Collect-disj-eq UN-Un-distrib)

moreover have *finite* $\{ a\{b\}.(P \parallel Q') \mid a \ b \ Q'. a\{b\}.Q' \in \text{summands } Q \}$

by(*induct Q rule: pi.induct, auto simp add: pi.inject Collect-ex-eq conj-disj-distribL*)

Collect-disj-eq UN-Un-distrib

moreover have finite $\{a\langle x \rangle.(P' \parallel Q) \mid a x P'. a\langle x \rangle.P' \in \text{summands } P \wedge x \# Q\}$

proof –

have $Aux: \bigwedge a x P Q. (x::\text{name}) \# Q \implies \{a'\langle x' \rangle.(P' \parallel Q) \mid a' x' P'. a'\langle x' \rangle.P' = a\langle x \rangle.P \wedge x' \# Q\} = \{a\langle x \rangle.(P \parallel Q)\}$

by(*auto simp add: pi.inject name-abs-eq name-fresh-fresh*)

thus *?thesis*

by(*nominal-induct P avoiding: Q rule: pi.strong-induct, auto simp add: Collect-ex-eq conj-disj-distribL conj-disj-distribR Collect-disj-eq UN-Un-distrib*)

qed

moreover have finite $\{a\langle x \rangle.(P \parallel Q') \mid a x Q'. a\langle x \rangle.Q' \in \text{summands } Q \wedge x \# P\}$

proof –

have $Aux: \bigwedge a x P Q. (x::\text{name}) \# P \implies \{a'\langle x' \rangle.(P \parallel Q') \mid a' x' Q'. a'\langle x' \rangle.Q' = a\langle x \rangle.Q \wedge x' \# P\} = \{a\langle x \rangle.(P \parallel Q)\}$

by(*auto simp add: pi.inject name-abs-eq name-fresh-fresh*)

thus *?thesis*

by(*nominal-induct Q avoiding: P rule: pi.strong-induct, auto simp add: Collect-ex-eq conj-disj-distribL conj-disj-distribR Collect-disj-eq UN-Un-distrib*)

qed

moreover have finite $\{\langle \nu x \rangle a\{x\}.(P' \parallel Q) \mid a x P'. \langle \nu x \rangle a\{x\}.P' \in \text{summands } P \wedge x \# Q\}$

proof –

have $Aux: \bigwedge a x P Q. \llbracket x \# Q; a \neq x \rrbracket \implies \{\langle \nu x' \rangle a'\{x'\}.(P' \parallel Q) \mid a' x' P'. \langle \nu x' \rangle a'\{x'\}.P' = \langle \nu x \rangle a\{x\}.P \wedge x' \# Q\} = \{\langle \nu x \rangle a\{x\}.(P \parallel Q)\}$

by(*auto simp add: pi.inject name-abs-eq name-fresh-fresh*)

thus *?thesis*

by(*nominal-induct P avoiding: Q rule: pi.strong-induct, auto simp add: Collect-ex-eq conj-disj-distribL conj-disj-distribR Collect-disj-eq UN-Un-distrib*)

qed

moreover have finite $\{\langle \nu x \rangle a\{x\}.(P \parallel Q') \mid a x Q'. \langle \nu x \rangle a\{x\}.Q' \in \text{summands } Q \wedge x \# P\}$

proof –

have $Aux: \bigwedge a x P Q. \llbracket x \# P; a \neq x \rrbracket \implies \{\langle \nu x' \rangle a'\{x'\}.(P \parallel Q') \mid a' x' Q'. \langle \nu x' \rangle a'\{x'\}.Q' = \langle \nu x \rangle a\{x\}.Q \wedge x' \# P\} = \{\langle \nu x \rangle a\{x\}.(P \parallel Q)\}$

by(*auto simp add: pi.inject name-abs-eq name-fresh-fresh*)

thus *?thesis*

by(*nominal-induct Q avoiding: P rule: pi.strong-induct, auto simp add: Collect-ex-eq conj-disj-distribL conj-disj-distribR Collect-disj-eq UN-Un-distrib*)

qed

moreover have finite $\{\tau.(P'[x::=b] \parallel Q') \mid x P' b Q'. \exists a. a\langle x \rangle.P' \in \text{summands } P \wedge a\{b\}.Q' \in \text{summands } Q\}$

proof –

have $Aux: \bigwedge a x P b Q. \{\tau.(P'[x'::=b] \parallel Q') \mid a' x' P' b' Q'. a' <x'>.P' = a <x>.P \wedge a'\{b'\}.Q' = a\{b\}.Q\} = \{\tau.(P[x::=b] \parallel Q)\}$

by (*auto simp add: name-abs-eq pi.inject renaming*)

have $\bigwedge a x P Q b::'a::\{\}. \text{finite } \{\tau.(P'[x'::=b] \parallel Q') \mid a' x' P' b' Q'. a' <x'>.P' = a <x>.P \wedge a'\{b'\}.Q' \in \text{summands } Q\}$

apply (*induct rule: pi.induct, simp-all*)

apply (*case-tac a=name1*)

apply (*simp add: Aux*)

apply (*simp add: pi.inject*)

by (*simp add: Collect-ex-eq conj-disj-distribL conj-disj-distribR*
Collect-disj-eq UN-Un-distrib)

hence $\text{finite } \{\tau.(P'[x::=b] \parallel Q') \mid a x P' b Q'. a <x>.P' \in \text{summands } P \wedge a\{b\}.Q' \in \text{summands } Q\}$

by (*nominal-induct P avoiding: Q rule: pi.strong-induct,*
auto simp add: Collect-ex-eq conj-disj-distribL conj-disj-distribR
Collect-disj-eq UN-Un-distrib name-abs-eq)

thus *?thesis*

apply (*rule-tac finite-subset*)

defer

by *blast+*

qed

moreover have $\text{finite } \{\tau.(P' \parallel (Q'[x::=b])) \mid b P' x Q'. \exists a. a\{b\}.P' \in \text{summands } P \wedge a <x>.Q' \in \text{summands } Q\}$

proof –

have $Aux: \bigwedge a x P b Q. \{\tau.(P' \parallel (Q'[x'::=b])) \mid a' b' P' x' Q'. a'\{b'\}.P' = a\{b\}.P \wedge a' <x'>.Q' = a <x>.Q\} = \{\tau.(P \parallel (Q[x::=b]))\}$

by (*auto simp add: name-abs-eq pi.inject renaming*)

have $\bigwedge a b P Q x::'a::\{\}. \text{finite } \{\tau.(P' \parallel (Q'[x::=b])) \mid a' b' P' x Q'. a'\{b'\}.P' = a\{b\}.P \wedge a' <x>.Q' \in \text{summands } Q\}$

apply (*induct rule: pi.induct, simp-all*)

apply (*case-tac a=name1*)

apply (*simp add: Aux*)

apply (*simp add: pi.inject*)

by (*simp add: Collect-ex-eq conj-disj-distribL conj-disj-distribR*
Collect-disj-eq UN-Un-distrib)

hence $\text{finite } \{\tau.(P' \parallel (Q'[x::=b])) \mid a b P' x Q'. a\{b\}.P' \in \text{summands } P \wedge a <x>.Q' \in \text{summands } Q\}$

by (*nominal-induct P avoiding: Q rule: pi.strong-induct,*
auto simp add: Collect-ex-eq conj-disj-distribL conj-disj-distribR
Collect-disj-eq UN-Un-distrib name-abs-eq)

thus *?thesis*

apply (*rule-tac finite-subset*) **defer by** *blast+*

qed

moreover have $\text{finite } \{\tau.(<\nu y>(P'[x::=y] \parallel Q')) \mid x P' y Q'. \exists a. a <x>.P' \in \text{summands } P \wedge <\nu y>a\{y\}.Q' \in \text{summands } Q \wedge y \# P\}$

proof –

have $Aux: \bigwedge a x P y Q. y \# P \wedge y \neq a \implies \{\tau.(<\nu y'>(P'[x'::=y'] \parallel Q')) \mid a' x' P' y' Q'. a' <x'>.P' = a <x>.P \wedge <\nu y'>a'\{y'\}.Q' = <\nu y>a\{y\}.Q \wedge y' \#$

$a <x>.P\} = \{\tau.(\langle \nu y \rangle (P[x::=y] \parallel Q))\}$
apply(*auto simp add: pi.inject name-abs-eq name-fresh-abs name-calc fresh-fact2 fresh-fact1 eqts forget*)
apply(*subst name-swap, simp add: injPermSubst fresh-fact1 fresh-fact2*) +
by(*simp add: name-swap injPermSubst*) +

have *BC*: $\bigwedge a x P Q. \text{finite } \{\tau.(\langle \nu y \rangle (P'[x::=y] \parallel Q')) \mid a' x' P' y Q'. a' <x'>.P' = a <x>.P \wedge \langle \nu y \rangle a'\{y\}.Q' \in \text{summands } Q \wedge y \# a <x>.P\}$
proof –
fix *a x P Q*
show *finite* $\{\tau.(\langle \nu y \rangle (P'[x::=y] \parallel Q')) \mid a' x' P' y Q'. a' <x'>.P' = a <x>.P \wedge \langle \nu y \rangle a'\{y\}.Q' \in \text{summands } Q \wedge y \# a <x>.P\}$
apply(*nominal-induct Q avoiding: a P rule: pi.strong-induct, simp-all*)
apply(*simp add: Collect-ex-eq conj-disj-distribL conj-disj-distribR Collect-disj-eq UN-Un-distrib*)

apply(*clarsimp*)
apply(*case-tac a=aa*)
apply(*insert Aux, auto*)
by(*simp add: pi.inject name-abs-eq name-calc*)
qed

have *IH*: $\bigwedge P P' Q. \{\tau.(\langle \nu y \rangle (P''[x::=y] \parallel Q')) \mid a x P'' y Q'. (a <x>.P'' \in \text{summands } P \vee a <x>.P'' \in \text{summands } P') \wedge \langle \nu y \rangle a\{y\}.Q' \in \text{summands } Q \wedge y \# P \wedge y \# P'\} = \{\tau.(\langle \nu y \rangle (P''[x::=y] \parallel Q')) \mid a x P'' y Q'. a <x>.P'' \in \text{summands } P \wedge \langle \nu y \rangle a\{y\}.Q' \in \text{summands } Q \wedge y \# P \wedge y \# P'\} \cup \{\tau.(\langle \nu y \rangle (P''[x::=y] \parallel Q')) \mid a x P'' y Q'. a <x>.P'' \in \text{summands } P' \wedge \langle \nu y \rangle a\{y\}.Q' \in \text{summands } Q \wedge y \# P \wedge y \# P'\}$
by *blast*
have *IH'*: $\bigwedge P Q P'. \{\tau.(\langle \nu y \rangle (P''[x::=y] \parallel Q')) \mid a x P'' y Q'. a <x>.P'' \in \text{summands } P \wedge \langle \nu y \rangle a\{y\}.Q' \in \text{summands } Q \wedge y \# P \wedge y \# P'\} \subseteq \{\tau.(\langle \nu y \rangle (P''[x::=y] \parallel Q')) \mid a x P'' y Q'. a <x>.P'' \in \text{summands } P \wedge \langle \nu y \rangle a\{y\}.Q' \in \text{summands } Q \wedge y \# P\}$
by *blast*
have *IH''*: $\bigwedge P Q P'. \{\tau.(\langle \nu y \rangle (P''[x::=y] \parallel Q')) \mid a x P'' y Q'. a <x>.P'' \in \text{summands } P' \wedge \langle \nu y \rangle a\{y\}.Q' \in \text{summands } Q \wedge y \# P \wedge y \# P'\} \subseteq \{\tau.(\langle \nu y \rangle (P''[x::=y] \parallel Q')) \mid a x P'' y Q'. a <x>.P'' \in \text{summands } P' \wedge \langle \nu y \rangle a\{y\}.Q' \in \text{summands } Q \wedge y \# P\}$
by *blast*
have *finite* $\{\tau.(\langle \nu y \rangle (P'[x::=y] \parallel Q')) \mid a x P' y Q'. a <x>.P' \in \text{summands } P \wedge \langle \nu y \rangle a\{y\}.Q' \in \text{summands } Q \wedge y \# P\}$
apply(*nominal-induct P avoiding: Q rule: pi.strong-induct, simp-all*)
apply(*insert BC, force*)
apply(*insert IH, auto*)
apply(*blast intro: finite-subset[OF IH']*)
by(*blast intro: finite-subset[OF IH']*)
thus *?thesis*
apply(*rule-tac finite-subset*) **defer** **by**(*blast*) +
qed
moreover **have** *finite* $\{\tau.(\langle \nu y \rangle (P' \parallel (Q'[x::=y]))) \mid y P' x Q'. \exists a. \langle \nu y \rangle a\{y\}.P'$

$\in \text{summands } P \wedge a \langle x \rangle. Q' \in \text{summands } Q \wedge y \# Q\}$

proof –

have *Aux*: $\bigwedge a y P x Q. \llbracket y \# Q; y \neq a \rrbracket \implies \{\tau.(\langle \nu y' \rangle (P' \parallel (Q'[x::=y']))) \mid a' y' P' x Q'. \langle \nu y' \rangle a'\{y'\}.P' = \langle \nu y \rangle a\{y\}.P \wedge a' \langle x \rangle. Q' = a \langle x \rangle. Q \wedge y' \# a \langle x \rangle. Q\} = \{\tau.(\langle \nu y \rangle (P \parallel (Q[x::=y])))\}$

apply(*auto simp add: pi.inject name-abs-eq name-fresh-abs name-calc fresh-fact2 fresh-fact1 forget eqvts fresh-left renaming[symmetric]*)

apply(*subst name-swap, simp add: injPermSubst fresh-fact1 fresh-fact2*) +
by(*simp add: name-swap injPermSubst*) +

have *IH*: $\bigwedge P y a Q Q'. \{\tau.(\langle \nu y' \rangle (P' \parallel (Q''[x::=y']))) \mid a' y' P' x Q''. \langle \nu y' \rangle a'\{y'\}.P' = \langle \nu y \rangle a\{y\}.P \wedge (a' \langle x \rangle. Q'' \in \text{summands } Q \vee a' \langle x \rangle. Q'' \in \text{summands } Q') \wedge y' \# Q \wedge y' \# Q'\} = \{\tau.(\langle \nu y' \rangle (P' \parallel (Q''[x::=y']))) \mid a' y' P' x Q''. \langle \nu y' \rangle a'\{y'\}.P' = \langle \nu y \rangle a\{y\}.P \wedge a' \langle x \rangle. Q'' \in \text{summands } Q \wedge y' \# Q \wedge y' \# Q'\} \cup \{\tau.(\langle \nu y' \rangle (P' \parallel (Q''[x::=y']))) \mid a' y' P' x Q''. \langle \nu y' \rangle a'\{y'\}.P' = \langle \nu y \rangle a\{y\}.P \wedge a' \langle x \rangle. Q'' \in \text{summands } Q' \wedge y' \# Q \wedge y' \# Q'\}$

by *blast*

have *IH'*: $\bigwedge a y P Q Q'. \{\tau.(\langle \nu y' \rangle (P' \parallel (Q''[x::=y']))) \mid a' y' P' x Q''. \langle \nu y' \rangle a'\{y'\}.P' = \langle \nu y \rangle a\{y\}.P \wedge a' \langle x \rangle. Q'' \in \text{summands } Q \wedge y' \# Q \wedge y' \# Q'\} \subseteq \{\tau.(\langle \nu y' \rangle (P' \parallel (Q''[x::=y']))) \mid a' y' P' x Q''. \langle \nu y' \rangle a'\{y'\}.P' = \langle \nu y \rangle a\{y\}.P \wedge a' \langle x \rangle. Q'' \in \text{summands } Q \wedge y' \# Q\}$

by *blast*

have *IH''*: $\bigwedge a y P Q Q'. \{\tau.(\langle \nu y' \rangle (P' \parallel (Q''[x::=y']))) \mid a' y' P' x Q''. \langle \nu y' \rangle a'\{y'\}.P' = \langle \nu y \rangle a\{y\}.P \wedge a' \langle x \rangle. Q'' \in \text{summands } Q' \wedge y' \# Q \wedge y' \# Q'\} \subseteq \{\tau.(\langle \nu y' \rangle (P' \parallel (Q''[x::=y']))) \mid a' y' P' x Q''. \langle \nu y' \rangle a'\{y'\}.P' = \langle \nu y \rangle a\{y\}.P \wedge a' \langle x \rangle. Q'' \in \text{summands } Q' \wedge y' \# Q\}$

by *blast*

have *BC*: $\bigwedge a y P Q. \llbracket y \# Q; y \neq a \rrbracket \implies \text{finite } \{\tau.(\langle \nu y' \rangle (P' \parallel (Q'[x::=y']))) \mid a' y' P' x Q'. \langle \nu y' \rangle a'\{y'\}.P' = \langle \nu y \rangle a\{y\}.P \wedge a' \langle x \rangle. Q' \in \text{summands } Q \wedge y' \# Q\}$

proof –

fix *a y P Q*

assume (*y::name*) $\# (Q::\text{pi})$ **and** $y \neq a$

thus *finite* $\{\tau.(\langle \nu y' \rangle (P' \parallel (Q'[x::=y']))) \mid a' y' P' x Q'. \langle \nu y' \rangle a'\{y'\}.P' = \langle \nu y \rangle a\{y\}.P \wedge a' \langle x \rangle. Q' \in \text{summands } Q \wedge y' \# Q\}$

apply(*nominal-induct Q avoiding: y rule: pi.strong-induct, simp-all*)

apply(*case-tac a=name1*)

apply *auto*

apply(*subgoal-tac ya \# (pi::pi)*)

apply(*insert Aux*)

apply *auto*

apply(*simp add: name-fresh-abs*)

apply(*simp add: pi.inject name-abs-eq name-calc*)

apply(*insert IH*)

apply *auto*

apply(*blast intro: finite-subset[OF IH']*)

by(*blast intro: finite-subset[OF IH']*)

qed

```

have finite { $\tau$ .(< $\nu y$ >(P' || (Q'[x::=y])) | a y P' x Q'. < $\nu y$ >a{y}.P'  $\in$  summands P  $\wedge$  a<x>.Q'  $\in$  summands Q  $\wedge$  y # Q)}

apply(nominal-induct P avoiding: Q rule: pi.strong-induct, simp-all)
apply(simp add: Collect-ex-eq conj-disj-distribL conj-disj-distribR name-fresh-abs
        Collect-disj-eq UN-Un-distrib)
by(auto intro: BC)
thus ?thesis
apply(rule-tac finite-subset) defer by blast+
qed

ultimately show ?thesis
by(simp add: expandSet-def)
qed

lemma expandHnf:
  fixes P :: pi
  and Q :: pi

  shows  $\forall R \in$  (expandSet P Q). hnf R
by(force simp add: expandSet-def)

inductive-set sumComposeSet :: (pi  $\times$  pi set) set
where
  empty: (0, { })  $\in$  sumComposeSet
| insert: [ $Q \in S$ ; (P, S - {Q})  $\in$  sumComposeSet]  $\implies$  (P  $\oplus$  Q, S)  $\in$  sumComposeSet

lemma expandAction:
  fixes P :: pi
  and Q :: pi
  and S :: pi set

  assumes (P, S)  $\in$  sumComposeSet
  and Q  $\in$  S
  and Q  $\mapsto$  Rs

  shows P  $\mapsto$  Rs
using assms
proof(induct arbitrary: Q rule: sumComposeSet.induct)
  case empty
  have Q  $\in$  { } by fact
  hence False by simp
  thus ?case by simp
next
  case(insert Q' S P Q)
  have QTrans: Q  $\mapsto$  Rs by fact
  show ?case
  proof(case-tac Q = Q')

```

```

    assume  $Q = Q'$ 
    with  $QTrans$  show  $P \oplus Q' \mapsto Rs$  by(blast intro: Sum2)
  next
    assume  $Q \neq Q'$ 
    have  $IH: \bigwedge Q. [Q \in S - \{Q'\}; Q \mapsto Rs] \implies P \mapsto Rs$  by fact
    have  $Q \in S$  by fact
    with  $Q \neq Q'$  have  $Q \in S - \{Q'\}$  by simp
    hence  $P \mapsto Rs$  using  $QTrans$  by(rule IH)
    thus ?case by(rule Sum1)
  qed
qed

lemma expandAction':
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $R :: pi$ 

  assumes  $(R, S) \in sumComposeSet$ 
  and  $R \mapsto Rs$ 

  shows  $\exists P \in S. P \mapsto Rs$ 
using assms
proof(induct rule: sumComposeSet.induct)
  case empty
  have  $0 \mapsto Rs$  by fact
  hence False by blast
  thus ?case by simp
next
  case(insert Q S P)
  have  $Q \in S$  by fact
  have  $P \oplus Q \mapsto Rs$  by fact
  thus ?case
proof(induct rule: sumCases)
  case cSum1
  have  $P \mapsto Rs$  by fact
  moreover have  $P \mapsto Rs \implies \exists P \in (S - \{Q\}). P \mapsto Rs$  by fact
  ultimately obtain  $P$  where  $P \in S - \{Q\}$  and  $P \mapsto Rs$ 
by blast
  show ?case
proof(case-tac P = Q)
  assume  $P = Q$ 
  with  $P \mapsto Rs$  show ?case by blast
next
  assume  $P \neq Q$ 
  from  $P \in S$  have  $P \in S - \{Q\}$  by simp
  with  $P \mapsto Rs$  show ?thesis by blast
qed
next
  case cSum2

```

```

    have  $Q \mapsto R$ s by fact
    with  $Q$ inS show ?case by blast
qed
qed

```

lemma expandTrans:

```

fixes  $P :: pi$ 
and  $Q :: pi$ 
and  $R :: pi$ 
and  $a :: name$ 
and  $b :: name$ 
and  $x :: name$ 

```

```

assumes Exp:  $(R, \text{expandSet } P \ Q) \in \text{sumComposeSet}$ 
and Phnf: hnf P
and Qhnf: hnf Q

```

```

shows  $(P \parallel Q \mapsto_{\tau} \prec P') = (R \mapsto_{\tau} \prec P')$ 
and  $(P \parallel Q \mapsto_{a[b]} \prec P') = (R \mapsto_{a[b]} \prec P')$ 
and  $(P \parallel Q \mapsto_{a\langle x \rangle} \prec P') = (R \mapsto_{a\langle x \rangle} \prec P')$ 
and  $(P \parallel Q \mapsto_{a\langle \nu x \rangle} \prec P') = (R \mapsto_{a\langle \nu x \rangle} \prec P')$ 

```

proof -

```

show  $P \parallel Q \mapsto_{\tau} \prec P' = R \mapsto_{\tau} \prec P'$ 

```

proof(rule iffI)

```

  assume  $P \parallel Q \mapsto_{\tau} \prec P'$ 

```

```

  thus  $R \mapsto_{\tau} \prec P'$ 

```

```

  proof(induct rule: parCasesF[of - - - - (P, Q)])

```

```

    case(cPar1 P')

```

```

    have  $P \mapsto_{\tau} \prec P'$  by fact

```

```

    with Phnf have  $\tau.(P') \in \text{summands } P$  by(simp add: summandTransition)

```

```

    hence  $\tau.(P' \parallel Q) \in \text{expandSet } P \ Q$  by(auto simp add: expandSet-def)

```

```

    moreover have  $\tau.(P' \parallel Q) \mapsto_{\tau} \prec (P' \parallel Q)$  by(rule Tau)

```

```

    ultimately show ?case using Exp by(blast intro: expandAction)

```

next

```

    case(cPar2 Q')

```

```

    have  $Q \mapsto_{\tau} \prec Q'$  by fact

```

```

    with Qhnf have  $\tau.(Q') \in \text{summands } Q$  by(simp add: summandTransition)

```

```

    hence  $\tau.(P \parallel Q') \in \text{expandSet } P \ Q$  by(auto simp add: expandSet-def)

```

```

    moreover have  $\tau.(P \parallel Q') \mapsto_{\tau} \prec (P \parallel Q')$  by(rule Tau)

```

```

    ultimately show ?case using Exp by(blast intro: expandAction)

```

next

```

    case(cComm1 P' Q' a b x)

```

```

    have  $P \mapsto_{a\langle x \rangle} \prec P'$  and  $Q \mapsto_{a[b]} \prec Q'$  by fact+

```

```

    with Phnf Qhnf have  $a\langle x \rangle.P' \in \text{summands } P$  and  $a\{b\}.Q' \in \text{summands } Q$ 

```

```

  by(simp add: summandTransition)+

```

```

  hence  $\tau.(P'[x::=b] \parallel Q') \in \text{expandSet } P \ Q$  by(simp add: expandSet-def, blast)

```

```

  moreover have  $\tau.(P'[x::=b] \parallel Q') \mapsto_{\tau} \prec (P'[x::=b] \parallel Q')$  by(rule Tau)

```

```

  ultimately show ?case using Exp by(blast intro: expandAction)

```

next

```

case(cComm2  $P' Q' a b x$ )
have  $P \mapsto a[b] \prec P'$  and  $Q \mapsto a\langle x \rangle \prec Q'$  by fact+
with Phnf Qhnf have  $a\{b\}.P' \in \text{summands } P$  and  $a\langle x \rangle.Q' \in \text{summands } Q$  by(simp add: summandTransition)+
hence  $\tau.(P' \parallel (Q'[x::=b])) \in \text{expandSet } P Q$  by(simp add: expandSet-def, blast)
moreover have  $\tau.(P' \parallel (Q'[x::=b])) \mapsto \tau \prec (P' \parallel (Q'[x::=b]))$  by(rule Tau)
ultimately show ?case using Exp by(blast intro: expandAction)
next
case(cClose1  $P' Q' a x y$ )
have  $y \# (P, Q)$  by fact
hence  $y\text{Fresh}P: y \# P$  by(simp add: fresh-prod)
have  $P\text{Trans}: P \mapsto a\langle x \rangle \prec P'$  by fact
with Phnf have  $P\text{Summ}: a\langle x \rangle.P' \in \text{summands } P$  by(simp add: summandTransition)
have  $Q \mapsto a\langle \nu y \rangle \prec Q'$  by fact
moreover from  $P\text{Trans } y\text{Fresh}P$  have  $y \neq a$  by(force dest: freshBoundDerivative)
ultimately have  $\langle \nu y \rangle a\{y\}.Q' \in \text{summands } Q$  using Qhnf by(simp add: summandTransition)
with  $P\text{Summ } y\text{Fresh}P$  have  $\tau.\langle \nu y \rangle(P'[x::=y] \parallel Q') \in \text{expandSet } P Q$ 
by(auto simp add: expandSet-def)
moreover have  $\tau.\langle \nu y \rangle(P'[x::=y] \parallel Q') \mapsto \tau \prec \langle \nu y \rangle(P'[x::=y] \parallel Q')$ 
by(rule Tau)
ultimately show ?case using Exp by(blast intro: expandAction)
next
case(cClose2  $P' Q' a x y$ )
have  $y \# (P, Q)$  by fact
hence  $y\text{Fresh}Q: y \# Q$  by(simp add: fresh-prod)
have  $Q\text{Trans}: Q \mapsto a\langle x \rangle \prec Q'$  by fact
with Qhnf have  $Q\text{Summ}: a\langle x \rangle.Q' \in \text{summands } Q$  by(simp add: summandTransition)
have  $P \mapsto a\langle \nu y \rangle \prec P'$  by fact
moreover from  $Q\text{Trans } y\text{Fresh}Q$  have  $y \neq a$  by(force dest: freshBoundDerivative)
ultimately have  $\langle \nu y \rangle a\{y\}.P' \in \text{summands } P$  using Phnf by(simp add: summandTransition)
with  $Q\text{Summ } y\text{Fresh}Q$  have  $\tau.\langle \nu y \rangle(P' \parallel (Q'[x::=y])) \in \text{expandSet } P Q$ 
by(simp add: expandSet-def, blast)
moreover have  $\tau.\langle \nu y \rangle(P' \parallel (Q'[x::=y])) \mapsto \tau \prec \langle \nu y \rangle(P' \parallel (Q'[x::=y]))$ 
by(rule Tau)
ultimately show ?case using Exp by(blast intro: expandAction)
qed
next
assume  $R \mapsto \tau \prec P'$ 
with Exp obtain  $R$  where  $R \in \text{expandSet } P Q$  and  $R \mapsto \tau \prec P'$  by(blast dest: expandAction')
thus  $P \parallel Q \mapsto \tau \prec P'$ 
proof(auto simp add: expandSet-def)

```

```

fix P''
assume  $\tau.(P'') \in \text{summands } P$ 
with Phnf have  $P \mapsto_{\tau} \prec P''$  by(simp add: summandTransition)
hence PQTrans:  $P \parallel Q \mapsto_{\tau} \prec P'' \parallel Q$  by(rule Par1F)
assume  $\tau.(P'' \parallel Q) \mapsto_{\tau} \prec P'$ 
hence  $P' = P'' \parallel Q$  by(erule-tac tauCases, auto simp add: pi.inject residual.inject)
with PQTrans show ?thesis by simp
next
fix Q'
assume  $\tau.(Q') \in \text{summands } Q$ 
with Qhnf have  $Q \mapsto_{\tau} \prec Q'$  by(simp add: summandTransition)
hence PQTrans:  $P \parallel Q \mapsto_{\tau} \prec P \parallel Q'$  by(rule Par2F)
assume  $\tau.(P \parallel Q') \mapsto_{\tau} \prec P'$ 
hence  $P' = P \parallel Q'$  by(erule-tac tauCases, auto simp add: pi.inject residual.inject)
with PQTrans show ?thesis by simp
next
fix a x P'' b Q'
assume  $a \langle x \rangle . P'' \in \text{summands } P$  and  $a\{b\}.Q' \in \text{summands } Q$ 
with Phnf Qhnf have  $P \mapsto_{a \langle x \rangle} \prec P''$  and  $Q \mapsto_{a[b]} \prec Q'$  by(simp add: summandTransition)+
hence PQTrans:  $P \parallel Q \mapsto_{\tau} \prec P''[x::=b] \parallel Q'$  by(rule Comm1)
assume  $\tau.(P''[x::=b] \parallel Q') \mapsto_{\tau} \prec P'$ 
hence  $P' = P''[x::=b] \parallel Q'$  by(erule-tac tauCases, auto simp add: pi.inject residual.inject)
with PQTrans show ?thesis by simp
next
fix a b P'' x Q'
assume  $a\{b\}.P'' \in \text{summands } P$  and  $a \langle x \rangle . Q' \in \text{summands } Q$ 
with Phnf Qhnf have  $P \mapsto_{a[b]} \prec P''$  and  $Q \mapsto_{a \langle x \rangle} \prec Q'$  by(simp add: summandTransition)+
hence PQTrans:  $P \parallel Q \mapsto_{\tau} \prec P'' \parallel (Q'[x::=b])$  by(rule Comm2)
assume  $\tau.(P'' \parallel (Q'[x::=b])) \mapsto_{\tau} \prec P'$ 
hence  $P' = P'' \parallel (Q'[x::=b])$  by(erule-tac tauCases, auto simp add: pi.inject residual.inject)
with PQTrans show ?thesis by simp
next
fix a x P'' y Q'
assume yFreshP:  $(y::\text{name}) \# P$ 
assume  $a \langle x \rangle . P'' \in \text{summands } P$ 
with Phnf have PTrans:  $P \mapsto_{a \langle x \rangle} \prec P''$  by(simp add: summandTransition)
assume  $\langle \nu y \rangle a\{y\}.Q' \in \text{summands } Q$ 
moreover from yFreshP PTrans have  $y \neq a$  by(force dest: freshBoundDerivative)
ultimately have  $Q \mapsto_{a \langle \nu y \rangle} \prec Q'$  using Qhnf by(simp add: summandTransition)
with PTrans have PQTrans:  $P \parallel Q \mapsto_{\tau} \prec \langle \nu y \rangle (P''[x::=y] \parallel Q')$  using

```

```

yFreshP by(rule Close1)
  assume  $\tau.(\langle \nu y \rangle (P''[x::=y] \parallel Q')) \mapsto \tau \prec P'$ 
  hence  $P' = \langle \nu y \rangle (P''[x::=y] \parallel Q')$  by(erule-tac tauCases, auto simp add:
pi.inject residual.inject)
  with PQTrans show ?thesis by simp
next
fix a y P'' x Q'
assume yFreshQ: (y::name) # Q
assume a<x>.Q' ∈ summands Q
with Qhnf have QTrans:  $Q \mapsto a\langle x \rangle \prec Q'$  by(simp add: summandTransition)
assume  $\langle \nu y \rangle a\{y\}.P'' \in \text{summands } P$ 
moreover from yFreshQ QTrans have  $y \neq a$  by(force dest: freshBoundDerivative)
ultimately have  $P \mapsto a\langle \nu y \rangle \prec P''$  using Phnf by(simp add: summandTransition)
hence PQTrans:  $P \parallel Q \mapsto \tau \prec \langle \nu y \rangle (P'' \parallel Q'[x::=y])$  using QTrans yFreshQ
by(rule Close2)
assume  $\tau.(\langle \nu y \rangle (P'' \parallel Q'[x::=y])) \mapsto \tau \prec P'$ 
hence  $P' = \langle \nu y \rangle (P'' \parallel Q'[x::=y])$  by(erule-tac tauCases, auto simp add:
pi.inject residual.inject)
with PQTrans show ?thesis by simp
qed
qed
next
show  $P \parallel Q \mapsto a[b] \prec P' = R \mapsto a[b] \prec P'$ 
proof(rule iffI)
assume  $P \parallel Q \mapsto a[b] \prec P'$ 
thus  $R \mapsto a[b] \prec P'$ 
proof(induct rule: parCasesF[where C=()])
case(cPar1 P')
have  $P \mapsto a[b] \prec P'$  by fact
with Phnf have  $a\{b\}.P' \in \text{summands } P$  by(simp add: summandTransition)
hence  $a\{b\}.(P' \parallel Q) \in \text{expandSet } P \ Q$  by(auto simp add: expandSet-def)
moreover have  $a\{b\}.(P' \parallel Q) \mapsto a[b] \prec (P' \parallel Q)$  by(rule Output)
ultimately show ?case using Exp by(blast intro: expandAction)
next
case(cPar2 Q')
have  $Q \mapsto a[b] \prec Q'$  by fact
with Qhnf have  $a\{b\}.Q' \in \text{summands } Q$  by(simp add: summandTransition)
hence  $a\{b\}.(P \parallel Q') \in \text{expandSet } P \ Q$  by(simp add: expandSet-def, blast)
moreover have  $a\{b\}.(P \parallel Q') \mapsto a[b] \prec (P \parallel Q')$  by(rule Output)
ultimately show ?case using Exp by(blast intro: expandAction)
next
case cComm1
thus ?case by auto
next
case cComm2
thus ?case by auto

```

```

next
  case cClose1
  thus ?case by auto
next
  case cClose2
  thus ?case by auto
qed
next
  assume  $R \mapsto a[b] \prec P'$ 
  with Exp obtain R where  $R \in \text{expandSet } P \ Q$  and  $R \mapsto a[b] \prec P'$  by (blast
dest: expandAction')
  thus  $P \parallel Q \mapsto a[b] \prec P'$ 
  proof (auto simp add: expandSet-def)
    fix a' b' P''
    assume  $a'\{b'\}.P'' \in \text{summands } P$ 
    with Phnf have  $P \mapsto a'[b'] \prec P''$  by (simp add: summandTransition)
    hence PQTrans:  $P \parallel Q \mapsto a'[b'] \prec P'' \parallel Q$  by (rule Par1F)
    assume  $a'\{b'\}.(P'' \parallel Q) \mapsto a[b] \prec P'$ 
    hence  $P' = P'' \parallel Q$  and  $a = a'$  and  $b = b'$ 
      by (erule-tac outputCases, auto simp add: pi.inject residual.inject)+
    with PQTrans show ?thesis by simp
  next
    fix a' b' Q'
    assume  $a'\{b'\}.Q' \in \text{summands } Q$ 
    with Qhnf have  $Q \mapsto a'[b'] \prec Q'$  by (simp add: summandTransition)
    hence PQTrans:  $P \parallel Q \mapsto a'[b'] \prec P \parallel Q'$  by (rule Par2F)
    assume  $a'\{b'\}.(P \parallel Q') \mapsto a[b] \prec P'$ 
    hence  $P' = P \parallel Q'$  and  $a = a'$  and  $b = b'$ 
      by (erule-tac outputCases, auto simp add: pi.inject residual.inject)+
    with PQTrans show ?thesis by simp
  qed
qed
next
  show  $P \parallel Q \mapsto a\langle x \rangle \prec P' = R \mapsto a\langle x \rangle \prec P'$ 
  proof (rule iffI)
    {
      fix x P'
      assume  $P \parallel Q \mapsto a\langle x \rangle \prec P'$  and  $x \# P$  and  $x \# Q$ 
      hence  $R \mapsto a\langle x \rangle \prec P'$ 
      proof (induct rule: parCasesB)
        case (cPar1 P')
        have  $P \mapsto a\langle x \rangle \prec P'$  by fact
        with Phnf have  $a\langle x \rangle.P' \in \text{summands } P$  by (simp add: summandTransition)
        moreover have  $x \# Q$  by fact
        ultimately have  $a\langle x \rangle.(P' \parallel Q) \in \text{expandSet } P \ Q$  by (auto simp add:
expandSet-def)
        moreover have  $a\langle x \rangle.(P' \parallel Q) \mapsto a\langle x \rangle \prec (P' \parallel Q)$  by (rule Input)
        ultimately show ?case using Exp by (blast intro: expandAction)
      next

```

```

    case(cPar2 Q')
    have Q  $\mapsto a\langle x \rangle \prec Q'$  by fact
    with Qhnf have  $a\langle x \rangle.Q' \in \text{summands } Q$  by (simp add: summandTransition)
    moreover have  $x \# P$  by fact
    ultimately have  $a\langle x \rangle.(P \parallel Q') \in \text{expandSet } P \ Q$  by (simp add: expand-
Set-def, blast)
    moreover have  $a\langle x \rangle.(P \parallel Q') \mapsto a\langle x \rangle \prec (P \parallel Q')$  by (rule Input)
    ultimately show ?case using Exp by (blast intro: expandAction)
  qed
}
moreover obtain  $y::\text{name}$  where  $y \# P$  and  $y \# Q$  and  $y \# P'$ 
  by (generate-fresh name) auto
assume  $P \parallel Q \mapsto a\langle x \rangle \prec P'$ 
with  $\langle y \# P' \rangle$  have  $P \parallel Q \mapsto a\langle y \rangle \prec ([ (x, y) ] \cdot P')$ 
  by (simp add: alphaBoundResidual)
ultimately have  $R \mapsto a\langle y \rangle \prec ([ (x, y) ] \cdot P')$  using  $\langle y \# P \rangle \langle y \# Q \rangle$ 
  by auto
thus  $R \mapsto a\langle x \rangle \prec P'$  using  $\langle y \# P' \rangle$  by (simp add: alphaBoundResidual)
next
  assume  $R \mapsto a\langle x \rangle \prec P'$ 
  with Exp obtain R where  $R \in \text{expandSet } P \ Q$  and  $R \mapsto a\langle x \rangle \prec P'$  by (blast
dest: expandAction')
  thus  $P \parallel Q \mapsto a\langle x \rangle \prec P'$ 
  proof (auto simp add: expandSet-def)
    fix  $a' y P''$ 
    assume  $a'\langle y \rangle.P'' \in \text{summands } P$ 
    with Phnf have  $P \mapsto a'\langle y \rangle \prec P''$  by (simp add: summandTransition)
    moreover assume  $y \# Q$ 
    ultimately have  $PQTrans: P \parallel Q \mapsto a'\langle y \rangle \prec P'' \parallel Q$  by (rule Par1B)
    assume  $a'\langle y \rangle.(P'' \parallel Q) \mapsto a\langle x \rangle \prec P'$ 
    hence  $a\langle x \rangle \prec P' = a'\langle y \rangle \prec P'' \parallel Q$  and  $a = a'$ 
      by (erule-tac inputCases', auto simp add: pi.inject residual.inject)+
    with PQTrans show ?thesis by simp
  next
    fix  $a' y Q'$ 
    assume  $a'\langle y \rangle.Q' \in \text{summands } Q$ 
    with Qhnf have  $Q \mapsto (a'::\text{name})\langle y \rangle \prec Q'$  by (simp add: summandTransi-
tion)
    moreover assume  $y \# P$ 
    ultimately have  $PQTrans: P \parallel Q \mapsto a'\langle y \rangle \prec P \parallel Q'$  by (rule Par2B)
    assume  $a'\langle y \rangle.(P \parallel Q') \mapsto a\langle x \rangle \prec P'$ 
    hence  $a\langle x \rangle \prec P' = a'\langle y \rangle \prec P \parallel Q'$  and  $a = a'$ 
      by (erule-tac inputCases', auto simp add: pi.inject residual.inject)+
    with PQTrans show ?thesis by simp
  qed
qed
next
  have Goal:  $\bigwedge P \ Q \ a \ x \ P' \ R. \llbracket (R, \text{expandSet } P \ Q) \in \text{sumComposeSet}; \text{hnf } P; \text{hnf } Q; a \neq x \rrbracket \implies P \parallel Q \mapsto a\langle \nu x \rangle \prec P' = R \mapsto a\langle \nu x \rangle \prec P'$ 

```

```

proof –
  fix  $P Q a x P' R$ 
  assume  $aineqx: (a::name) \neq x$ 
  assume  $Exp: (R, expandSet P Q) \in sumComposeSet$ 
  assume  $Phnf: hnf P$ 
  assume  $Qhnf: hnf Q$ 
  show  $P \parallel Q \mapsto a\langle \nu x \rangle \prec P' = R \mapsto a\langle \nu x \rangle \prec P'$ 
  proof(rule iffI)
  {
    fix  $x P'$ 
    assume  $P \parallel Q \mapsto a\langle \nu x \rangle \prec P'$  and  $x \# P$  and  $x \# Q$  and  $a \neq x$ 
    hence  $R \mapsto a\langle \nu x \rangle \prec P'$ 
    proof(induct rule: parCasesB)
      case(cPar1 P')
        have  $P \mapsto a\langle \nu x \rangle \prec P'$  by fact
          with  $Phnf \langle a \neq x \rangle$  have  $\langle \nu x \rangle a\{x\}.P' \in summands P$  by(simp add: summandTransition)
          moreover have  $x \# Q$  by fact
          ultimately have  $\langle \nu x \rangle a\{x\}.(P' \parallel Q) \in expandSet P Q$  by(auto simp add: expandSet-def)
          moreover have  $\langle \nu x \rangle a\{x\}.(P' \parallel Q) \mapsto a\langle \nu x \rangle \prec (P' \parallel Q)$  using  $\langle a \neq x \rangle$ 
          by(blast intro: Open Output)
          ultimately show ?case using  $Exp$  by(blast intro: expandAction)
        next
          case(cPar2 Q')
            have  $Q \mapsto a\langle \nu x \rangle \prec Q'$  by fact
              with  $Qhnf \langle a \neq x \rangle$  have  $\langle \nu x \rangle a\{x\}.Q' \in summands Q$  by(simp add: summandTransition)
              moreover have  $x \# P$  by fact
              ultimately have  $\langle \nu x \rangle a\{x\}.(P \parallel Q') \in expandSet P Q$  by(simp add: expandSet-def, blast)
              moreover have  $\langle \nu x \rangle a\{x\}.(P \parallel Q') \mapsto a\langle \nu x \rangle \prec (P \parallel Q')$  using  $\langle a \neq x \rangle$ 
              by(blast intro: Open Output)
              ultimately show ?case using  $Exp$  by(blast intro: expandAction)
            qed
          }
    moreover obtain  $y::name$  where  $y \# P$  and  $y \# Q$  and  $y \# P'$  and  $y \neq a$ 
      by(generate-fresh name) auto
    assume  $P \parallel Q \mapsto a\langle \nu x \rangle \prec P'$ 
    with  $\langle y \# P' \rangle$  have  $P \parallel Q \mapsto a\langle \nu y \rangle \prec ((x, y) \cdot P')$ 
      by(simp add: alphaBoundResidual)
    ultimately have  $R \mapsto a\langle \nu y \rangle \prec ((x, y) \cdot P')$  using  $\langle y \# P \rangle \langle y \# Q \rangle \langle y \neq a \rangle$ 
    by auto
    thus  $R \mapsto a\langle \nu x \rangle \prec P'$  using  $\langle y \# P' \rangle$  by(simp add: alphaBoundResidual)
  }
  next
  {

```

```

    fix R x P'
    assume R  $\mapsto$  a< $\nu$ x>  $\prec$  P' and R  $\in$  expandSet P Q and x  $\#$  R and x  $\#$  P
and x  $\#$  Q
    hence P  $\parallel$  Q  $\mapsto$  a< $\nu$ x>  $\prec$  P'
    proof(auto simp add: expandSet-def)
      fix a' y P''
      assume < $\nu$ y>a'{y}.P''  $\in$  summands P
      moreover hence a'  $\neq$  y by auto
      ultimately have P  $\mapsto$  a'< $\nu$ y>  $\prec$  P'' using Phnf by(simp add: sum-
mandTransition)
      moreover assume y  $\#$  Q
      ultimately have PQTrans: P  $\parallel$  Q  $\mapsto$  a'< $\nu$ y>  $\prec$  P''  $\parallel$  Q by(rule Par1B)
      assume ResTrans: < $\nu$ y>a'{y}.(P''  $\parallel$  Q)  $\mapsto$  a< $\nu$ x>  $\prec$  P' and x  $\#$ 
[y].a'{y}.(P''  $\parallel$  Q)
      with ResTrans <a'  $\neq$  y> <x  $\#$  P> <x  $\#$  Q> have a< $\nu$ x>  $\prec$  P' = a'< $\nu$ y>  $\prec$ 
P''  $\parallel$  Q
      apply(case-tac x=y)
      defer
      apply(erule-tac resCasesB)
      apply simp
      apply(simp add: abs-fresh)
      apply(auto simp add: residual.inject alpha' calc-atm fresh-left abs-fresh
elim: outputCases)
      apply(ind-cases < $\nu$ y>a'{y}.(P''  $\parallel$  Q)  $\mapsto$  a< $\nu$ y>  $\prec$  P')
      apply(simp add: pi.inject alpha' residual.inject abs-fresh eqvts calc-atm)
      apply(auto elim: outputCases)
      apply(simp add: pi.inject residual.inject alpha' calc-atm)
      apply auto
      apply(ind-cases < $\nu$ y>a'{y}.(P''  $\parallel$  Q)  $\mapsto$  a< $\nu$ y>  $\prec$  P')
      apply(auto simp add: pi.inject alpha' residual.inject abs-fresh eqvts
calc-atm)
      apply(auto elim: outputCases)
      apply(erule-tac outputCases)
      apply(auto simp add: freeRes.inject)
      apply hypsubst-thin
      apply(drule-tac pi=[(b, y)] in pt-bij3)
      by simp
    with PQTrans show ?thesis by simp
  next
    fix a' y Q'
    assume < $\nu$ y>a'{y}.Q'  $\in$  summands Q
    moreover hence a'  $\neq$  y by auto
    ultimately have Q  $\mapsto$  a'< $\nu$ y>  $\prec$  Q' using Qhnf by(simp add: summand-
Transition)
    moreover assume y  $\#$  P
    ultimately have PQTrans: P  $\parallel$  Q  $\mapsto$  a'< $\nu$ y>  $\prec$  P  $\parallel$  Q' by(rule Par2B)
    assume ResTrans: < $\nu$ y>a'{y}.(P  $\parallel$  Q')  $\mapsto$  a< $\nu$ x>  $\prec$  P' and x  $\#$  [y].a'{y}.(P
 $\parallel$  Q')
    with ResTrans <a'  $\neq$  y> have a< $\nu$ x>  $\prec$  P' = a'< $\nu$ y>  $\prec$  P  $\parallel$  Q'

```

```

apply(case-tac x=y)
defer
apply(erule-tac resCasesB)
  apply simp
  apply(simp add: abs-fresh)
  apply(auto simp add: residual.inject alpha' calc-atm fresh-left abs-fresh)
elim: outputCases)
  apply(ind-cases <νy>a'{y}.(P || Q') ↦ a<νy> < P')
  apply(simp add: pi.inject alpha' residual.inject abs-fresh eqvts calc-atm)
  apply(auto elim: outputCases)
  apply(simp add: pi.inject residual.inject alpha' calc-atm)
  apply auto
  apply(ind-cases <νy>a'{y}.(P || Q') ↦ a<νy> < P')
  apply(auto simp add: pi.inject alpha' residual.inject abs-fresh eqvts
calc-atm)
  apply(auto elim: outputCases)
  apply(erule-tac outputCases)
  apply(auto simp add: freeRes.inject)
  apply hypsubst-thin
  apply(drule-tac pi=[(b, y)] in pt-bij3)
  by simp
  with PQTrans show ?thesis by simp
qed
}
moreover assume R ↦ a<νx> < P'
with Exp obtain R where R ∈ expandSet P Q and R ↦ a<νx> < P'
  apply(drule-tac expandAction') by auto
moreover obtain y::name where y # P and y # Q and y # R and y # P'
  by(generate-fresh name) auto
moreover with <y # P'> <R ↦ a<νx> < P'> have R ↦ a<νy> < [(x, y)]
· P') by(simp add: alphaBoundResidual)
  ultimately have P || Q ↦ a<νy> < [(x, y)] · P') by auto
  thus P || Q ↦ a<νx> < P' using <y # P'> by(simp add: alphaBoundResidual)
qed
qed

obtain y where yineqx: a ≠ y and yFreshP': y # P'
  by(force intro: name-exists-fresh[of (a, P')]) simp add: fresh-prod
from Exp Phnf Qhnf yineqx have (P || Q ↦ a<νy> < [(x, y)] · P') = (R
↦ a<νy> < [(x, y)] · P')
  by(rule Goal)
moreover with yFreshP' have x # [(x, y)] · P' by(simp add: name-fresh-left
name-calc)
ultimately show (P || Q ↦ a<νx> < P') = (R ↦ a<νx> < P')
  by(simp add: alphaBoundResidual name-swap)
qed

lemma expandLeft:
  fixes P :: pi

```

```

and   Q  :: pi
and   R  :: pi
and   Rel :: (pi × pi) set

assumes Exp: (R, expandSet P Q) ∈ sumComposeSet
and     Phnf: hnf P
and     Qhnf: hnf Q
and     Id: Id ⊆ Rel

shows P || Q ~>[Rel] R
proof(induct rule: simCases)
  case(Bound a x R')
    have R ↦a«x» < R' by fact
    with Exp Phnf Qhnf have P || Q ↦a«x» < R' by(cases a, auto simp add:
expandTrans)
    moreover from Id have derivative R' R' a x Rel by(cases a, auto simp add:
derivative-def)
    ultimately show ?case by blast
  next
    case(Free α R')
      have R ↦α < R' by fact
      with Exp Phnf Qhnf have P || Q ↦α < R' by(cases α, auto simp add: ex-
pandTrans)
      moreover from Id have (R', R') ∈ Rel by blast
      ultimately show ?case by blast
qed

```

lemma *expandRight*:

```

fixes P  :: pi
and   Q  :: pi
and   R  :: pi
and   Rel :: (pi × pi) set

assumes Exp: (R, expandSet P Q) ∈ sumComposeSet
and     Phnf: hnf P
and     Qhnf: hnf Q
and     Id: Id ⊆ Rel

shows R ~>[Rel] P || Q
proof(induct rule: simCases)
  case(Bound a x R')
    have P || Q ↦a«x» < R' by fact
    with Exp Phnf Qhnf have R ↦a«x» < R' by(cases a, auto simp add: expand-
Trans)
    moreover from Id have derivative R' R' a x Rel by(cases a, auto simp add:
derivative-def)
    ultimately show ?case by blast
  next
    case(Free α R')

```

```

have  $P \parallel Q \mapsto \alpha \prec R'$  by fact
with Exp Phnf Qhnf have  $R \mapsto \alpha \prec R'$  by (cases  $\alpha$ , auto simp add: expandTrans)
moreover from Id have  $(R', R') \in Rel$  by blast
ultimately show ?case by blast
qed

```

lemma *expandSC*:

```

fixes  $P :: pi$ 
and  $Q :: pi$ 
and  $R :: pi$ 

```

```

assumes  $(R, expandSet P Q) \in sumComposeSet$ 
and  $hnf P$ 
and  $hnf Q$ 

```

```

shows  $P \parallel Q \sim R$ 

```

proof –

```

let  $?X = \{(P \parallel Q, R) \mid P Q R. (R, expandSet P Q) \in sumComposeSet \wedge hnf P \wedge hnf Q\} \cup \{(R, P \parallel Q) \mid P Q R. (R, expandSet P Q) \in sumComposeSet \wedge hnf P \wedge hnf Q\}$ 

```

```

from assms have  $(P \parallel Q, R) \in ?X$  by auto

```

```

thus ?thesis

```

```

proof (coinduct rule: bisimCoinduct)

```

```

  case (cSim  $P Q$ )

```

```

    thus ?case

```

```

      by (blast intro: reflexive expandLeft expandRight)

```

```

  next

```

```

    case (cSym  $P Q$ )

```

```

      thus ?case by auto

```

```

  qed

```

```

qed

```

end

theory *Strong-Late-Axiomatisation*

```

  imports Strong-Late-Expansion-Law

```

begin

lemma *inputSuppPres*:

```

fixes  $P :: pi$ 
and  $Q :: pi$ 
and  $x :: name$ 
and  $Rel :: (pi \times pi) set$ 

```

```

assumes  $PRelQ: \bigwedge y. y \in supp(P, Q, x) \implies (P[x::=y], Q[x::=y]) \in Rel$ 

```

```

and  $Eqvt: eqvt Rel$ 

```

```

shows  $a \langle x \rangle . P \rightsquigarrow [Rel] a \langle x \rangle . Q$ 

```

proof –

```

from Eqvt show ?thesis
proof(induct rule: simCasesCont[where  $C=(x, a, Q, P)$ ])
  case(Bound b y Q')
  have  $x \in \text{supp}(P, Q, x)$  by(simp add: supp-prod supp-atm)
  with PRelQ have  $(P, Q) \in \text{Rel}$  by fastforce
  have  $Q\text{Trans}: a\langle x \rangle.Q \mapsto b\langle y \rangle \prec Q'$  by fact
  have  $y \# (x, a, Q, P)$  by fact
  hence  $y \neq a$  and  $y \text{ineq}x: y \neq x$  and  $y \# Q$  and  $y \# P$  by(simp add: fresh-prod)+
  with QTrans show ?case
  proof(induct rule: inputCases)
    have  $a\langle y \rangle.([(x, y)] \cdot P) \mapsto a\langle y \rangle \prec ([x, y]) \cdot P$  by(rule Input)
    hence  $a\langle x \rangle.P \mapsto a\langle y \rangle \prec ([x, y]) \cdot P$  using  $\langle y \# P \rangle$  by(simp add:
alphaInput)
    moreover have  $\text{derivative } ([x, y]) \cdot P \ ([x, y]) \cdot Q \ (InputS\ a)\ y\ \text{Rel}$ 
    proof(auto simp add: derivative-def)
      fix  $u$ 
      have  $x \in \text{supp}(P, Q, x)$  by(simp add: supp-prod supp-atm)
      have  $(P[x::=u], Q[x::=u]) \in \text{Rel}$ 
      proof(cases u \in \text{supp}(P, Q, x))
        case True
          with PRelQ show ?thesis by auto
        next
          case False
            hence  $u \# P$  and  $u \# Q$  by(auto simp add: fresh-def supp-prod)
            moreover from  $\langle \text{eqvt Rel} \rangle \langle (P, Q) \in \text{Rel} \rangle$  have  $([x, u]) \cdot P, [(x, u)] \cdot$ 
 $Q) \in \text{Rel}$ 
            by(rule eqvtRelI)
            ultimately show ?thesis by(simp only: injPermSubst)
          qed
          with  $\langle y \# P \rangle \langle y \# Q \rangle$  show  $([x, y]) \cdot P[y::=u], ([x, y]) \cdot Q[y::=u] \in$ 
 $\text{Rel}$ 
            by(simp add: renaming)
          qed
            ultimately show  $\exists P'. a\langle x \rangle.P \mapsto a\langle y \rangle \prec P' \wedge \text{derivative } P' \ ([x, y]) \cdot$ 
 $Q) \ (InputS\ a)\ y\ \text{Rel}$ 
            by blast
          qed
        next
          case(Free  $\alpha$  Q')
          have  $a\langle x \rangle.Q \mapsto \alpha \prec Q'$  by fact
          hence False by auto
          thus ?case by blast
        qed
      qed

```

lemma *inputSuppPresBisim*:

```

fixes  $P :: pi$ 
and  $Q :: pi$ 
and  $x :: name$ 

```

```

assumes  $PSimQ: \bigwedge y. y \in \text{supp}(P, Q, x) \implies P[x::=y] \sim Q[x::=y]$ 

shows  $a\langle x \rangle.P \sim a\langle x \rangle.Q$ 
proof –
  let  $?X = \{(a\langle x \rangle.P, a\langle x \rangle.Q) \mid a\ x\ P\ Q.\ \forall y \in \text{supp}(P, Q, x). P[x::=y] \sim Q[x::=y]\}$ 
  have  $\text{eqvt } ?X$ 
    apply (auto simp add: eqvt-def)
    apply (rule-tac x=perm · aa in exI)
    apply (rule-tac x=perm · x in exI)
    apply (rule-tac x=perm · P in exI)
    apply auto
    apply (rule-tac x=perm · Q in exI)
    apply auto
    apply (drule-tac pi=rev perm in pt-set-bij2[OF pt-name-inst, OF at-name-inst])
    apply (simp add: eqvts pt-rev-pi[OF pt-name-inst, OF at-name-inst])
    apply (erule-tac x=rev perm · y in ballE)
    apply auto
    apply (drule-tac p=perm in bisimClosed)
    by (simp add: eqvts pt-pi-rev[OF pt-name-inst, OF at-name-inst])
  from assms have  $(a\langle x \rangle.P, a\langle x \rangle.Q) \in ?X$  by fastforce
  thus  $?thesis$ 
  proof (coinduct rule: bisimCoinduct)
    case (cSim P Q)
    thus  $?case$  using  $\langle \text{eqvt } ?X \rangle$ 
    by (fastforce intro: inputSuppPres)
  next
    case (cSym P Q)
    thus  $?case$  by (fastforce simp add: supp-prod dest: symmetric)
  qed
qed

inductive  $\text{equiv} :: \text{pi} \Rightarrow \text{pi} \Rightarrow \text{bool}$  (infixr  $\langle \equiv_e \rangle$  80)
where
  Refl:  $P \equiv_e P$ 
| Sym:  $P \equiv_e Q \implies Q \equiv_e P$ 
| Trans:  $\llbracket P \equiv_e Q; Q \equiv_e R \rrbracket \implies P \equiv_e R$ 

| Match:  $[a\ \wedge\ a]P \equiv_e P$ 
| Match':  $a \neq b \implies [a\ \wedge\ b]P \equiv_e \mathbf{0}$ 

| Mismatch:  $a \neq b \implies [a\ \neq\ b]P \equiv_e P$ 
| Mismatch':  $[a\ \neq\ a]P \equiv_e \mathbf{0}$ 

| SumSym:  $P \oplus Q \equiv_e Q \oplus P$ 
| SumAssoc:  $(P \oplus Q) \oplus R \equiv_e P \oplus (Q \oplus R)$ 
| SumZero:  $P \oplus \mathbf{0} \equiv_e P$ 
| SumIdemp:  $P \oplus P \equiv_e P$ 

```

| *SumRes*: $\langle \nu x \rangle (P \oplus Q) \equiv_e (\langle \nu x \rangle P) \oplus (\langle \nu x \rangle Q)$

| *ResNil*: $\langle \nu x \rangle \mathbf{0} \equiv_e \mathbf{0}$

| *ResInput*: $\llbracket x \neq a; x \neq y \rrbracket \implies \langle \nu x \rangle a \langle y \rangle . P \equiv_e a \langle y \rangle . (\langle \nu x \rangle P)$

| *ResInput'*: $\langle \nu x \rangle x \langle y \rangle . P \equiv_e \mathbf{0}$

| *ResOutput*: $\llbracket x \neq a; x \neq b \rrbracket \implies \langle \nu x \rangle a \{b\} . P \equiv_e a \{b\} . (\langle \nu x \rangle P)$

| *ResOutput'*: $\langle \nu x \rangle x \{b\} . P \equiv_e \mathbf{0}$

| *ResTau*: $\langle \nu x \rangle \tau . (P) \equiv_e \tau . (\langle \nu x \rangle P)$

| *ResComm*: $\langle \nu x \rangle \langle \nu y \rangle P \equiv_e \langle \nu y \rangle \langle \nu x \rangle P$

| *ResFresh*: $x \# P \implies \langle \nu x \rangle P \equiv_e P$

| *Expand*: $\llbracket (R, \text{expandSet } P \ Q) \in \text{sumComposeSet}; \text{hnf } P; \text{hnf } Q \rrbracket \implies P \parallel Q \equiv_e R$

| *SumPres*: $P \equiv_e Q \implies P \oplus R \equiv_e Q \oplus R$

| *ParPres*: $\llbracket P \equiv_e P'; Q \equiv_e Q' \rrbracket \implies P \parallel Q \equiv_e P' \parallel Q'$

| *ResPres*: $P \equiv_e Q \implies \langle \nu x \rangle P \equiv_e \langle \nu x \rangle Q$

| *TauPres*: $P \equiv_e Q \implies \tau . (P) \equiv_e \tau . (Q)$

| *OutputPres*: $P \equiv_e Q \implies a \{b\} . P \equiv_e a \{b\} . Q$

| *InputPres*: $\llbracket \forall y \in \text{supp}(P, Q, x). P[x::=y] \equiv_e Q[x::=y] \rrbracket \implies a \langle x \rangle . P \equiv_e a \langle x \rangle . Q$

lemma *SumIdemp'*:

fixes $P :: pi$
and $P' :: pi$

assumes $P \equiv_e P'$

shows $P \oplus P' \equiv_e P$

proof –

have $P \equiv_e P \oplus P$ **by** (*blast intro: Sym SumIdemp*)

moreover from *assms* **have** $P \oplus P \equiv_e P' \oplus P$ **by** (*rule SumPres*)

moreover have $P' \oplus P \equiv_e P \oplus P'$ **by** (*rule SumSym*)

ultimately have $P \equiv_e P \oplus P'$ **by** (*blast intro: Trans*)

thus *?thesis* **by** (*rule Sym*)

qed

lemma *SumPres'*:

fixes $P :: pi$
and $P' :: pi$
and $Q :: pi$
and $Q' :: pi$

assumes *PeqP'*: $P \equiv_e P'$

and *QeqQ'*: $Q \equiv_e Q'$

shows $P \oplus Q \equiv_e P' \oplus Q'$

proof –

from *PeqP'* **have** $P \oplus Q \equiv_e P' \oplus Q$ **by** (*rule SumPres*)

```

moreover have  $P' \oplus Q \equiv_e Q \oplus P'$  by(rule SumSym)
moreover from  $Q \text{ eq } Q'$  have  $Q \oplus P' \equiv_e Q' \oplus P'$  by(rule SumPres)
moreover have  $Q' \oplus P' \equiv_e P' \oplus Q'$  by(rule SumSym)
ultimately show ?thesis by(blast intro: Trans)
qed

```

```

lemma sound:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 

  assumes  $P \equiv_e Q$ 

  shows  $P \sim Q$ 
using assms
proof(induct)
  case(Refl  $P$ )
    show ?case by(rule reflexive)
  next
    case(Sym  $P$   $Q$ )
    have  $P \sim Q$  by fact
    thus ?case by(rule symmetric)
  next
    case(Trans  $P$   $Q$   $R$ )
    have  $P \sim Q$  and  $Q \sim R$  by fact+
    thus ?case by(rule transitive)
  next
    case(Match  $a$   $P$ )
    show ?case by(rule matchId)
  next
    case(Match'  $a$   $b$   $P$ )
    have  $a \neq b$  by fact
    thus ?case by(rule matchNil)
  next
    case(Mismatch  $a$   $b$   $P$ )
    have  $a \neq b$  by fact
    thus ?case by(rule mismatchId)
  next
    case(Mismatch'  $a$   $P$ )
    show ?case by(rule mismatchNil)
  next
    case(SumSym  $P$   $Q$ )
    show ?case by(rule sumSym)
  next
    case(SumAssoc  $P$   $Q$   $R$ )
    show ?case by(rule sumAssoc)
  next
    case(SumZero  $P$ )
    show ?case by(rule sumZero)
  next

```

```

    case(SumIdemp P)
    show ?case by(rule sumIdemp)
next
    case(SumRes x P Q)
    show ?case by(rule sumRes)
next
    case(ResNil x)
    show ?case by(rule nilRes)
next
    case(ResInput x a y P)
    have  $x \neq a$  and  $x \neq y$  by fact+
    thus ?case by(rule resInput)
next
    case(ResInput' x y P)
    show ?case by(rule resNil)
next
    case(ResOutput x a b P)
    have  $x \neq a$  and  $x \neq b$  by fact+
    thus ?case by(rule resOutput)
next
    case(ResOutput' x b P)
    show ?case by(rule resNil)
next
    case(ResTau x P)
    show ?case by(rule resTau)
next
    case(ResComm x P)
    show ?case by(rule resComm)
next
    case(ResFresh x P)
    have  $x \nmid P$  by fact
    thus ?case by(rule scopeFresh)
next
    case(Expand R P Q)
    have  $(R, \text{expandSet } P \ Q) \in \text{sumComposeSet}$  and hnf P and hnf Q by fact+
    thus ?case by(rule expandSC)
next
    case(SumPres P Q R)
    from  $\langle P \sim Q \rangle$  show ?case by(rule sumPres)
next
    case(ParPres P P' Q Q')
    have  $P \sim P'$  and  $Q \sim Q'$  by fact+
    thus ?case by(metis transitive symmetric parPres parSym)
next
    case(ResPres P Q x)
    from  $\langle P \sim Q \rangle$  show ?case by(rule resPres)
next
    case(TauPres P Q)
    from  $\langle P \sim Q \rangle$  show ?case by(rule tauPres)

```

```

next
  case(OutputPres P Q a b)
  from  $\langle P \sim Q \rangle$  show ?case by(rule outputPres)
next
  case(InputPres P Q x a)
  have  $\forall y \in \text{supp}(P, Q, x). P[x::=y] \equiv_e Q[x::=y] \wedge P[x::=y] \sim Q[x::=y]$  by fact
  hence  $\forall y \in \text{supp}(P, Q, x). P[x::=y] \sim Q[x::=y]$  by blast
  thus ?case by(rule-tac inputSuppPresBisim) auto
qed

```

```

lemma zeroDest[dest]:
  fixes a :: name
  and b :: name
  and x :: name
  and P :: pi

  shows  $(a\{b\}.P) \equiv_e \mathbf{0} \implies \text{False}$ 
  and  $(a\langle x \rangle.P) \equiv_e \mathbf{0} \implies \text{False}$ 
  and  $(\tau.(P)) \equiv_e \mathbf{0} \implies \text{False}$ 

  and  $\mathbf{0} \equiv_e a\{b\}.P \implies \text{False}$ 
  and  $\mathbf{0} \equiv_e a\langle x \rangle.P \implies \text{False}$ 
  and  $\mathbf{0} \equiv_e \tau.(P) \implies \text{False}$ 
by(auto dest: sound)

```

```

lemma eq-eqv:
  fixes pi::name prm
  and x::'a::pt-name
  shows  $pi.(x=y) = ((pi.x)=(pi.y))$ 
by(simp add: perm-bool perm-bij)

```

```

nominal-primrec depth :: pi  $\Rightarrow$  nat where
  depth  $\mathbf{0} = 0$ 
| depth  $(\tau.(P)) = 1 + (\text{depth } P)$ 
|  $a \# x \implies \text{depth } (a\langle x \rangle.P) = 1 + (\text{depth } P)$ 
| depth  $(a\{b\}.P) = 1 + (\text{depth } P)$ 
| depth  $([a \frown b]P) = (\text{depth } P)$ 
| depth  $([a \neq b]P) = (\text{depth } P)$ 
| depth  $(P \oplus Q) = \max (\text{depth } P) (\text{depth } Q)$ 
| depth  $(P \parallel Q) = ((\text{depth } P) + (\text{depth } Q))$ 
| depth  $(\langle \nu x \rangle P) = (\text{depth } P)$ 
| depth  $(!P) = (\text{depth } P)$ 
apply(auto simp add: fresh-nat)
apply(finite-guess)
by(fresh-guess)

```

```

lemma depthEqvt[simp]:
  fixes P :: pi
  and p :: name prm

```

shows $\text{depth}(p \cdot P) = \text{depth } P$
by(*nominal-induct* P *rule: pi.strong-induct, auto simp add: name-bij*)

lemma *depthInput*[*simp*]:

fixes $a :: \text{name}$
and $x :: \text{name}$
and $P :: \text{pi}$

shows $\text{depth } (a \langle x \rangle . P) = 1 + (\text{depth } P)$

proof –

obtain y **where** *yineqa*: $y \neq a$ **and** *yFreshP*: $y \# P$
by(*force intro: name-exists-fresh*[*of* (a, P)] *simp add: fresh-prod*)
from *yFreshP* **have** $a \langle x \rangle . P = a \langle y \rangle . ([x, y] \cdot P)$ **by**(*simp add: alphaInput*)
with *yineqa* **show** *?thesis* **by** *simp*

qed

nominal-primrec *valid* :: $\text{pi} \Rightarrow \text{bool}$ **where**

valid $\mathbf{0} = \text{True}$
| *valid* $(\tau.(P)) = \text{valid } P$
| $x \# a \implies \text{valid } (a \langle x \rangle . P) = \text{valid } P$
| *valid* $(a\{b\}.P) = \text{valid } P$
| *valid* $([a \frown b]P) = \text{valid } P$
| *valid* $([a \neq b]P) = \text{valid } P$
| *valid* $(P \oplus Q) = ((\text{valid } P) \wedge (\text{valid } Q))$
| *valid* $(P \parallel Q) = ((\text{valid } P) \wedge (\text{valid } Q))$
| *valid* $(\langle \nu x \rangle P) = \text{valid } P$
| *valid* $(!P) = \text{False}$

apply(*auto simp add: fresh-bool*)

apply(*finite-guess*)+

by(*fresh-guess*)+

lemma *validEqvt*[*simp*]:

fixes $P :: \text{pi}$
and $p :: \text{name prm}$

shows $\text{valid}(p \cdot P) = \text{valid } P$

by(*nominal-induct* P *rule: pi.strong-induct, auto simp add: name-bij*)

lemma *validInput*[*simp*]:

fixes $a :: \text{name}$
and $x :: \text{name}$
and $P :: \text{pi}$

shows $\text{valid } (a \langle x \rangle . P) = \text{valid } P$

proof –

obtain y **where** *yineqa*: $y \neq a$ **and** *yFreshP*: $y \# P$
by(*force intro: name-exists-fresh*[*of* (a, P)] *simp add: fresh-prod*)
from *yFreshP* **have** $a \langle x \rangle . P = a \langle y \rangle . ([x, y] \cdot P)$ **by**(*simp add: alphaInput*)

with *yineqa* **show** *?thesis* **by** *simp*
qed

lemma *depthMin*[*intro*]:
fixes *P*

shows $0 \leq \text{depth } P$
by(*induct* *P* *rule*: *pi.induct*, *auto*)

lemma *hnfTransition*:
fixes *P* :: *pi*

assumes *hnf* *P*
and $P \neq \mathbf{0}$

shows $\exists R_s. P \mapsto R_s$
using *assms*
by(*induct* *rule*: *pi.induct*, *auto* *intro*: *Output Tau Input Sum1 Open*)

definition *uhnf* :: *pi* \Rightarrow *bool* **where**

$\text{uhnf } P \equiv \text{hnf } P \wedge (\forall R \in \text{summands } P. \forall R' \in \text{summands } P. R \neq R' \longrightarrow \neg(R \equiv_e R'))$

lemma *summandsIdemp*:
fixes *P* :: *pi*
and *Q* :: *pi*

assumes $Q \in \text{summands } P$
and $Q \equiv_e Q'$

shows $P \oplus Q' \equiv_e P$
using *assms*
proof(*nominal-induct* *P* *arbitrary*: *Q* *rule*: *pi.strong-inducts*)

case(*PiNil* *Q*)
have $Q \in \text{summands } \mathbf{0}$ **by** *fact*
hence *False* **by** *simp*
thus *?case* **by** *simp*

next

case(*Output* *a* *b* *P* *Q*)
have $Q \equiv_e Q'$ **by** *fact*
hence $a\{b\}.P \oplus Q' \equiv_e a\{b\}.P \oplus Q$ **by**(*blast* *intro*: *SumPres' Refl Sym*)
moreover **have** $Q = a\{b\}.P$

proof –
have $Q \in \text{summands } (a\{b\}.P)$ **by** *fact*
thus *?thesis* **by** *simp*

qed

ultimately show *?case* **by**(*blast* *intro*: *SumIdemp Trans*)

next

case(*Tau* *P* *Q*)

```

have  $Q \equiv_e Q'$  by fact
hence  $\tau.(P) \oplus Q' \equiv_e \tau.(P) \oplus Q$  by(blast intro: SumPres' Refl Sym)
moreover have  $Q = \tau.(P)$ 
proof -
  have  $Q \in \text{summands } (\tau.(P))$  by fact
  thus ?thesis by simp
qed
ultimately show ?case by(blast intro: SumIdemp Trans)
next
case(Input a x P Q)
have  $Q \equiv_e Q'$  by fact
hence  $a \langle x \rangle . P \oplus Q' \equiv_e a \langle x \rangle . P \oplus Q$  by(blast intro: SumPres' Refl Sym)
moreover have  $Q = a \langle x \rangle . P$ 
proof -
  have  $Q \in \text{summands } (a \langle x \rangle . P)$  by fact
  thus ?thesis by simp
qed
ultimately show ?case by(blast intro: SumIdemp Trans)
next
case(Match a b P Q)
have  $Q \in \text{summands } ([a \frown b]P)$  by fact
hence False by simp
thus ?case by simp
next
case(Mismatch a b P Q)
have  $Q \in \text{summands } ([a \neq b]P)$  by fact
hence False by simp
thus ?case by simp
next
case(Sum P Q R)
have IHP:  $\bigwedge P'. \llbracket P' \in \text{summands } P; P' \equiv_e Q' \rrbracket \implies P \oplus Q' \equiv_e P$  by fact
have IHQ:  $\bigwedge Q''. \llbracket Q'' \in \text{summands } Q; Q'' \equiv_e Q' \rrbracket \implies Q \oplus Q' \equiv_e Q$  by fact
have ReqQ':  $R \equiv_e Q'$  by fact
have  $R \in \text{summands } (P \oplus Q)$  by fact
hence  $R \in \text{summands } P \vee R \in \text{summands } Q$  by simp
thus ?case
proof(rule disjE)
  assume  $R \in \text{summands } P$ 
  hence  $PQ'eqP: P \oplus Q' \equiv_e P$  using ReqQ' by(rule IHP)

  have  $(P \oplus Q) \oplus Q' \equiv_e P \oplus (Q \oplus Q')$  by(rule SumAssoc)
  moreover have  $P \oplus (Q \oplus Q') \equiv_e P \oplus (Q' \oplus Q)$  by(blast intro: Refl SumSym SumPres')
  moreover have  $P \oplus (Q' \oplus Q) \equiv_e (P \oplus Q') \oplus Q$  by(blast intro: SumAssoc Sym)
  moreover from  $PQ'eqP$  have  $(P \oplus Q') \oplus Q \equiv_e P \oplus Q$  by(blast intro: SumPres' Refl)
  ultimately show ?case by(blast intro: Trans)
next

```

```

assume  $R \in \text{summands } Q$ 
hence  $QQ'eqQ: Q \oplus Q' \equiv_e Q$  using  $ReqQ'$  by(rule  $IHQ$ )

have  $(P \oplus Q) \oplus Q' \equiv_e P \oplus (Q \oplus Q')$  by(rule  $SumAssoc$ )
moreover from  $QQ'eqQ$  have  $P \oplus (Q \oplus Q') \equiv_e P \oplus Q$  by(blast intro:  $Refl$ 
 $SumPres'$ )
ultimately show  $?case$  by(rule  $Trans$ )
qed
next
case( $Par P Q R$ )
have  $R \in \text{summands } (P \parallel Q)$  by fact
hence  $False$  by simp
thus  $?case$  by simp
next
case( $Res x P Q$ )
have  $Q \equiv_e Q'$  by fact
hence  $\langle \nu x \rangle P \oplus Q' \equiv_e \langle \nu x \rangle P \oplus Q$  by(blast intro:  $SumPres' Refl Sym$ )
moreover have  $Q = \langle \nu x \rangle P$ 
proof -
have  $Q \in \text{summands } \langle \nu x \rangle P$  by fact
thus  $?thesis$  by(auto simp add:  $if-split$ )
qed
ultimately show  $?case$  by(blast intro:  $SumIdemp Trans$ )
next
case( $Bang P Q$ )
have  $Q \in \text{summands}(!P)$  by fact
hence  $False$  by simp
thus  $?case$  by simp
qed

lemma  $uhnfsSum$ :
fixes  $P :: pi$ 
and  $Q :: pi$ 

assumes  $Phnf: uhnf P$ 
and  $Qhnf: uhnf Q$ 
and  $validP: valid P$ 
and  $validQ: valid Q$ 

shows  $\exists R. uhnf R \wedge valid R \wedge P \oplus Q \equiv_e R \wedge (depth R) \leq (depth (P \oplus Q))$ 
using  $assms$ 
proof(nominal-induct  $P$  arbitrary:  $Q$  rule:  $pi.strong-inducts$ )
case( $PiNil Q$ )
have  $uhnf Q$  by fact
moreover have  $valid Q$  by fact
moreover have  $\mathbf{0} \oplus Q \equiv_e Q$  by(blast intro:  $SumZero SumSym Trans$ )
moreover have  $depth Q \leq depth(\mathbf{0} \oplus Q)$  by auto
ultimately show  $?case$  by blast
next

```

```

case(Output a b P Q)
show ?case
proof(case-tac Q = 0)
  assume Q = 0
  moreover have uhnf (a{b}.P) by(simp add: uhnf-def)
  moreover have valid (a{b}.P) by fact
  moreover have a{b}.P  $\oplus$  0  $\equiv_e$  a{b}.P by(rule SumZero)
  moreover have depth(a{b}.P)  $\leq$  depth(a{b}.P  $\oplus$  0) by simp
  ultimately show ?case by blast
next
assume QineqNil: Q  $\neq$  0
have Qhnf: uhnf Q by fact
have validQ: valid Q by fact
have validP: valid(a{b}.P) by fact
show ?case
proof(case-tac  $\exists Q' \in$  summands Q.  $Q' \equiv_e$  a{b}.P)
  assume  $\exists Q' \in$  summands Q.  $Q' \equiv_e$  a{b}.P
  then obtain Q' where Q'  $\in$  summands Q and  $Q' \equiv_e$  a{b}.P by blast
  hence Q  $\oplus$  a{b}.P  $\equiv_e$  Q by(rule summandsIdemp)
  moreover have depth Q  $\leq$  depth(Q  $\oplus$  a{b}.P) by simp
  ultimately show ?case using Qhnf validQ by(force intro: SumSym Trans)
next
assume  $\neg(\exists Q' \in$  summands Q.  $Q' \equiv_e$  a{b}.P)
hence  $\forall Q' \in$  summands Q.  $\neg(Q' \equiv_e$  a{b}.P) by simp
with Qhnf QineqNil have uhnf (a{b}.P  $\oplus$  Q)
  by(force dest: Sym simp add: uhnf-def)
moreover from validQ validP have valid(a{b}.P  $\oplus$  Q) by simp
moreover have a{b}.P  $\oplus$  Q  $\equiv_e$  a{b}.P  $\oplus$  Q by(rule Refl)
moreover have depth(a{b}.P  $\oplus$  Q)  $\leq$  depth(a{b}.P  $\oplus$  Q) by simp
ultimately show ?case by blast
qed
qed
next
case(Tau P Q)
show ?case
proof(case-tac Q = 0)
  assume Q = 0
  moreover have uhnf ( $\tau$ .(P)) by(simp add: uhnf-def)
  moreover have valid ( $\tau$ .(P)) by fact
  moreover have  $\tau$ .(P)  $\oplus$  0  $\equiv_e$   $\tau$ .(P) by(rule SumZero)
  moreover have depth( $\tau$ .(P))  $\leq$  depth( $\tau$ .(P)  $\oplus$  0) by simp
  ultimately show ?case by blast
next
assume QineqNil: Q  $\neq$  0
have Qhnf: uhnf Q by fact
have validP: valid( $\tau$ .(P)) and validQ: valid Q by fact+
show ?case
proof(case-tac  $\exists Q' \in$  summands Q.  $Q' \equiv_e$   $\tau$ .(P))
  assume  $\exists Q' \in$  summands Q.  $Q' \equiv_e$   $\tau$ .(P)

```

then obtain Q' where $Q' \in \text{summands } Q$ and $Q' \equiv_e \tau.(P)$ by *blast*
hence $Q \oplus \tau.(P) \equiv_e Q$ by (rule *summandsIdemp*)
moreover have $\text{depth } Q \leq \text{depth}(Q \oplus \tau.(P))$ by *simp*
ultimately show ?case using *Qhnf validQ* by (force intro: *SumSym Trans*)
next
assume $\neg(\exists Q' \in \text{summands } Q. Q' \equiv_e \tau.(P))$
hence $\forall Q' \in \text{summands } Q. \neg(Q' \equiv_e \tau.(P))$ by *simp*
with *Qhnf QineqNil* have *uhnf* ($\tau.(P) \oplus Q$)
by (force dest: *Sym simp add: uhnf-def*)
moreover from *validP validQ* have *valid*($\tau.(P) \oplus Q$) by *simp*
moreover have $\tau.(P) \oplus Q \equiv_e \tau.(P) \oplus Q$ by (rule *Refl*)
moreover have $\text{depth}(\tau.(P) \oplus Q) \leq \text{depth}(\tau.(P) \oplus Q)$ by *simp*
ultimately show ?case by *blast*
qed
qed
next
case (Input a x P Q)
show ?case
proof (case-tac $Q = \mathbf{0}$)
assume $Q = \mathbf{0}$
moreover have *uhnf* ($a \langle x \rangle . P$) by (*simp add: uhnf-def*)
moreover have *valid* ($a \langle x \rangle . P$) by *fact*
moreover have $a \langle x \rangle . P \oplus \mathbf{0} \equiv_e a \langle x \rangle . P$ by (rule *SumZero*)
moreover have $\text{depth}(a \langle x \rangle . P) \leq \text{depth}(a \langle x \rangle . P \oplus \mathbf{0})$ by *simp*
ultimately show ?case by *blast*
next
assume *QineqNil*: $Q \neq \mathbf{0}$
have *validP*: *valid*($a \langle x \rangle . P$) and *validQ*: *valid* Q by *fact+*
have *Qhnf*: *uhnf* Q by *fact*
show ?case
proof (case-tac $\exists Q' \in \text{summands } Q. Q' \equiv_e a \langle x \rangle . P$)
assume $\exists Q' \in \text{summands } Q. Q' \equiv_e a \langle x \rangle . P$
then obtain Q' where $Q' \in \text{summands } Q$ and $Q' \equiv_e a \langle x \rangle . P$ by *blast*
hence $Q \oplus a \langle x \rangle . P \equiv_e Q$ by (rule *summandsIdemp*)
moreover have $\text{depth } Q \leq \text{depth}(Q \oplus a \langle x \rangle . P)$ by *simp*
ultimately show ?case using *Qhnf validQ* by (force intro: *SumSym Trans*)
next
assume $\neg(\exists Q' \in \text{summands } Q. Q' \equiv_e a \langle x \rangle . P)$
hence $\forall Q' \in \text{summands } Q. \neg(Q' \equiv_e a \langle x \rangle . P)$ by *simp*
with *Qhnf QineqNil* have *uhnf* ($a \langle x \rangle . P \oplus Q$)
by (force dest: *Sym simp add: uhnf-def*)
moreover from *validP validQ* have *valid*($a \langle x \rangle . P \oplus Q$) by *simp*
moreover have $a \langle x \rangle . P \oplus Q \equiv_e a \langle x \rangle . P \oplus Q$ by (rule *Refl*)
moreover have $\text{depth}(a \langle x \rangle . P \oplus Q) \leq \text{depth}(a \langle x \rangle . P \oplus Q)$ by *simp*
ultimately show ?case by *blast*
qed
qed
next
case (Match a b P Q)

have $uhnf$ ($[a \neg b]P$) **by** *fact*
hence *False* **by** (*simp add: uhnf-def*)
thus *?case* **by** *simp*
next
case (*Mismatch a b P Q*)
have $uhnf$ ($[a \neq b]P$) **by** *fact*
hence *False* **by** (*simp add: uhnf-def*)
thus *?case* **by** *simp*
next
case (*Sum P Q R*)
have $Rhnf$: $uhnf$ R **by** *fact*
have $validR$: $valid$ R **by** *fact*
have $PQhnf$: $uhnf$ ($P \oplus Q$) **by** *fact*
have $validPQ$: $valid$ ($P \oplus Q$) **by** *fact*
have $\exists T$. $uhnf$ $T \wedge valid$ $T \wedge Q \oplus R \equiv_e T \wedge depth$ $T \leq depth$ ($Q \oplus R$)
proof –
from $PQhnf$ **have** $uhnf$ Q **by** (*simp add: uhnf-def*)
moreover from $validPQ$ **have** $valid$ Q **by** *simp+*
moreover have $\bigwedge R$. $\llbracket uhnf$ Q ; $uhnf$ R ; $valid$ Q ; $valid$ R $\rrbracket \implies \exists T$. $uhnf$ $T \wedge$
 $valid$ $T \wedge Q \oplus R \equiv_e T \wedge depth$ $T \leq depth$ ($Q \oplus R$) **by** *fact*
ultimately show *?thesis* **using** $Rhnf$ $validR$ **by** *blast*
qed
then obtain T **where** $Thnf$: $uhnf$ T **and** $QReqT$: $Q \oplus R \equiv_e T$ **and** $validT$:
 $valid$ T
and $Tdepth$: $depth$ $T \leq depth$ ($Q \oplus R$) **by** *blast*

have $\exists S$. $uhnf$ $S \wedge valid$ $S \wedge P \oplus T \equiv_e S \wedge depth$ $S \leq depth$ ($P \oplus T$)
proof –
from $PQhnf$ **have** $uhnf$ P **by** (*simp add: uhnf-def*)
moreover from $validPQ$ **have** $valid$ P **by** *simp*
moreover have $\bigwedge T$. $\llbracket uhnf$ P ; $uhnf$ T ; $valid$ P ; $valid$ T $\rrbracket \implies \exists S$. $uhnf$ $S \wedge$
 $valid$ $S \wedge P \oplus T \equiv_e S \wedge depth$ $S \leq depth$ ($P \oplus T$) **by** *fact*
ultimately show *?thesis* **using** $Thnf$ $validT$ **by** *blast*
qed
then obtain S **where** $Shnf$: $uhnf$ S **and** $PTeqS$: $P \oplus T \equiv_e S$ **and** $validS$: $valid$
 S
and $Sdepth$: $depth$ $S \leq depth$ ($P \oplus T$) **by** *blast*

have ($P \oplus Q$) $\oplus R \equiv_e S$
proof –
have ($P \oplus Q$) $\oplus R \equiv_e P \oplus (Q \oplus R)$ **by** (*rule SumAssoc*)
moreover from $QReqT$ **have** $P \oplus (Q \oplus R) \equiv_e P \oplus T$
by (*blast intro: Refl SumPres'*)
ultimately show *?thesis* **using** $PTeqS$ **by** (*blast intro: Trans*)
qed
moreover from $Tdepth$ $Sdepth$ **have** $depth$ $S \leq depth$ ($(P \oplus Q) \oplus R$) **by** *auto*

ultimately show *?case* **using** $Shnf$ $validS$ **by** *blast*
next

```

case(Par P Q R)
have uhnf (P || Q) by fact
hence False by(simp add: uhnf-def)
thus ?case by simp
next
case(Res x P Q)
show ?case
proof(case-tac Q = 0)
  assume Q = 0
  moreover have uhnf (<νx>P) by fact
  moreover have valid (<νx>P) by fact
  moreover have <νx>P ⊕ 0 ≡e <νx>P by(rule SumZero)
  moreover have depth(<νx>P) ≤ depth((<νx>P) ⊕ 0) by simp
  ultimately show ?case by blast
next
assume QineqNil: Q ≠ 0
have Qhnf: uhnf Q by fact
have validP: valid(<νx>P) and validQ: valid Q by fact+
show ?case
proof(case-tac ∃ Q' ∈ summands Q. Q' ≡e <νx>P)
  assume ∃ Q' ∈ summands Q. Q' ≡e <νx>P
  then obtain Q' where Q' ∈ summands Q and Q' ≡e <νx>P by blast
  hence Q ⊕ <νx>P ≡e Q by(rule summandsIdemp)
  moreover have depth Q ≤ depth(Q ⊕ <νx>P) by simp
  ultimately show ?case using Qhnf validQ by(force intro: SumSym Trans)
next
assume ¬(∃ Q' ∈ summands Q. Q' ≡e <νx>P)
hence ∀ Q' ∈ summands Q. ¬(Q' ≡e <νx>P) by simp
moreover have uhnf (<νx>P) by fact
ultimately have uhnf (<νx>P ⊕ Q) using Qhnf QineqNil
  by(force dest: Sym simp add: uhnf-def)
moreover from validP validQ have valid(<νx>P ⊕ Q) by simp
moreover have (<νx>P) ⊕ Q ≡e (<νx>P) ⊕ Q by(rule Refl)
moreover have depth((<νx>P) ⊕ Q) ≤ depth((<νx>P) ⊕ Q) by simp
ultimately show ?case by blast
qed
qed
next
case(Bang P Q)
have uhnf (!P) by fact
hence False by(simp add: uhnf-def)
thus ?case by simp
qed

lemma uhnfRes:
  fixes x :: name
  and P :: pi

  assumes Phnf: uhnf P

```

```

and   validP: valid P

shows  $\exists P'. \text{uhn}f P' \wedge \text{valid} P' \wedge \langle \nu x \rangle P \equiv_e P' \wedge \text{depth} P' \leq \text{depth}(\langle \nu x \rangle P)$ 
using assms
proof(nominal-induct P avoiding: x rule: pi.strong-inducts)
  case(PiNil x)
  have uhnf 0 by(simp add: uhnf-def)
  moreover have valid 0 by simp
  moreover have  $\langle \nu x \rangle 0 \equiv_e 0$  by(rule ResNil)
  moreover have  $\text{depth} 0 \leq \text{depth}(\langle \nu x \rangle 0)$  by simp
  ultimately show ?case by blast
next
  case(Output a b P)
  have valid( $a\{b\}.P$ ) by fact
  hence validP: valid P by simp
  show ?case
  proof(case-tac x=a)
    assume  $x = a$ 
    moreover have uhnf 0 by(simp add: uhnf-def)
    moreover have valid 0 by simp
    moreover have  $0 \equiv_e \langle \nu x \rangle x\{b\}.P$  by(blast intro: ResOutput' Sym)
    moreover have  $\text{depth} 0 \leq \text{depth}(\langle \nu x \rangle x\{b\}.P)$  by simp
    ultimately show ?case by(blast intro: Sym)
  next
    assume xineqa: x  $\neq$  a
    show ?case
    proof(case-tac x=b)
      assume  $x=b$ 
      moreover from xineqa have uhnf( $\langle \nu x \rangle a\{x\}.P$ ) by(force simp add: uhnf-def)
      moreover from validP have valid( $\langle \nu x \rangle a\{x\}.P$ ) by simp
      moreover have  $\langle \nu x \rangle a\{x\}.P \equiv_e \langle \nu x \rangle a\{x\}.P$  by(rule Refl)
      moreover have  $\text{depth}(\langle \nu x \rangle a\{x\}.P) \leq \text{depth}(\langle \nu x \rangle a\{x\}.P)$  by simp
      ultimately show ?case by blast
    next
      assume xineqb: x  $\neq$  b
      have uhnf( $a\{b\}.(\langle \nu x \rangle P)$ ) by(simp add: uhnf-def)
      moreover from validP have valid( $a\{b\}.(\langle \nu x \rangle P)$ ) by simp
      moreover from xineqa xineqb have  $a\{b\}.(\langle \nu x \rangle P) \equiv_e \langle \nu x \rangle a\{b\}.P$  by(blast
intro: ResOutput Sym)
      moreover have  $\text{depth}(a\{b\}.(\langle \nu x \rangle P)) \leq \text{depth}(\langle \nu x \rangle a\{b\}.P)$  by simp
      ultimately show ?case by(blast intro: Sym)
    qed
  qed
next
  case(Tau P)
  have valid( $\tau.(P)$ ) by fact
  hence validP: valid P by simp

  have uhnf( $\tau.(\langle \nu x \rangle P)$ ) by(simp add: uhnf-def)

```

moreover from $validP$ **have** $valid(\tau.<\nu x>P)$ **by** *simp*
moreover have $\tau.<\nu x>P \equiv_e <\nu x>\tau.(P)$ **by**(*blast intro: ResTau Sym*)
moreover have $depth(\tau.<\nu x>P) \leq depth(<\nu x>\tau.(P))$ **by** *simp*
ultimately show $?case$ **by**(*blast intro: Sym*)
next
case(*Input a y P x*)
have $valid(a<y>.P)$ **by** *fact*
hence $validP: valid P$ **by** *simp*
have $y \# x$ **by** *fact* **hence** $yineqx: y \neq x$ **by** *simp*
show $?case$
proof(*case-tac x=a*)
assume $x = a$
moreover have $uhnf \mathbf{0}$ **by**(*simp add: uhnf-def*)
moreover have $valid \mathbf{0}$ **by** *simp*
moreover have $\mathbf{0} \equiv_e <\nu x>x<y>.P$ **by**(*blast intro: ResInput' Sym*)
moreover have $depth \mathbf{0} \leq depth(<\nu x>x<y>.P)$ **by** *simp*
ultimately show $?case$ **by**(*blast intro: Sym*)
next
assume $xineqa: x \neq a$
have $uhnf(a<y>.<\nu x>P)$ **by**(*simp add: uhnf-def*)
moreover from $validP$ **have** $valid(a<y>.<\nu x>P)$ **by** *simp*
moreover from $xineqa yineqx$ **have** $a<y>.<\nu x>P \equiv_e <\nu x>a<y>.P$ **by**(*blast intro: ResInput Sym*)
moreover have $depth(a<y>.<\nu x>P) \leq depth(<\nu x>a<y>.P)$ **by** *simp*
ultimately show $?case$ **by**(*blast intro: Sym*)
qed
next
case(*Match a b P x*)
have $uhnf([a \sim b]P)$ **by** *fact*
hence $False$ **by**(*simp add: uhnf-def*)
thus $?case$ **by** *simp*
next
case(*Mismatch a b P x*)
have $uhnf([a \neq b]P)$ **by** *fact*
hence $False$ **by**(*simp add: uhnf-def*)
thus $?case$ **by** *simp*
next
case(*Sum P Q x*)
have $valid(P \oplus Q)$ **by** *fact*
hence $validP: valid P$ **and** $validQ: valid Q$ **by** *simp+*
have $uhnf(P \oplus Q)$ **by** *fact*
hence $Phnf: uhnf P$ **and** $Qhnf: uhnf Q$ **by**(*auto simp add: uhnf-def*)

have $\exists P'. uhnf P' \wedge valid P' \wedge P' \equiv_e <\nu x>P \wedge (depth P') \leq (depth(<\nu x>P))$
proof –
have $\llbracket uhnf P; valid P \rrbracket \implies \exists P'. uhnf P' \wedge valid P' \wedge <\nu x>P \equiv_e P' \wedge (depth P') \leq (depth(<\nu x>P))$ **by** *fact*
with $validP Phnf$ **show** $?thesis$ **by**(*blast intro: Sym*)
qed

then obtain P' **where** $P'hnf: uhnf P'$ **and** $P'eqP: P' \equiv_e \langle \nu x \rangle P$ **and** $validP':$
 $valid P'$

and $P'depth: (depth P') \leq (depth(\langle \nu x \rangle P))$ **by** *blast*

have $\exists Q'. uhnf Q' \wedge valid Q' \wedge Q' \equiv_e \langle \nu x \rangle Q \wedge (depth Q') \leq (depth(\langle \nu x \rangle Q))$
proof –

have $\llbracket uhnf Q; valid Q \rrbracket \implies \exists Q'. uhnf Q' \wedge valid Q' \wedge \langle \nu x \rangle Q \equiv_e Q' \wedge (depth$
 $Q') \leq (depth(\langle \nu x \rangle Q))$ **by** *fact*

with $validQ Qhnf$ **show** *?thesis* **by**(*blast intro: Sym*)

qed

then obtain Q' **where** $Q'hnf: uhnf Q'$ **and** $Q'eqQ: Q' \equiv_e \langle \nu x \rangle Q$ **and** $validQ':$
 $valid Q'$

and $Q'depth: (depth Q') \leq (depth(\langle \nu x \rangle Q))$ **by** *blast*

from $P'hnf Q'hnf validP' validQ'$ **obtain** R **where** $Rhnf: uhnf R$ **and** $validR:$
 $valid R$

and $P'Q'eqR: P' \oplus Q' \equiv_e R$

and $Rdepth: depth R \leq depth(P' \oplus Q')$

apply(*drule-tac uhnfSum*) **apply** *assumption+* **by** *blast*

from $P'eqP Q'eqQ P'Q'eqR$ **have** $\langle \nu x \rangle (P \oplus Q) \equiv_e R$ **by**(*blast intro: Sym*
 $SumPres' SumRes Trans$)

moreover from $Rdepth P'depth Q'depth$ **have** $depth R \leq depth(\langle \nu x \rangle (P \oplus Q))$
by *auto*

ultimately show *?case* **using** $validR Rhnf$ **by**(*blast intro: Sym*)

next

case(*Par P Q*)

have $uhnf(P \parallel Q)$ **by** *fact*

hence *False* **by**(*simp add: uhnf-def*)

thus *?case* **by** *simp*

next

case(*Res y P x*)

have $valid(\langle \nu y \rangle P)$ **by** *fact* **hence** $validP: valid P$ **by** *simp*

have $uhnf(\langle \nu y \rangle P)$ **by** *fact*

then obtain $a P'$ **where** $aineqy: a \neq y$ **and** $PeqP': P = a\{y\}.P'$

by(*force simp add: uhnf-def*)

show *?case*

proof(*case-tac x=y*)

assume $x=y$

moreover from $aineqy$ **have** $uhnf(\langle \nu y \rangle a\{y\}.P')$ **by**(*force simp add: uhnf-def*)

moreover from $validP PeqP'$ **have** $valid(\langle \nu y \rangle a\{y\}.P')$ **by** *simp*

moreover have $\langle \nu y \rangle \langle \nu y \rangle a\{y\}.P' \equiv_e \langle \nu y \rangle a\{y\}.P'$

proof –

have $y \# \langle \nu y \rangle a\{y\}.P'$ **by**(*simp add: name-fresh-abs*)

hence $\langle \nu y \rangle \langle \nu y \rangle a\{y\}.P' \equiv_e \langle \nu y \rangle a\{y\}.P'$ **by**(*rule ResFresh*)

thus *?thesis* **by**(*blast intro: Trans*)

qed

```

moreover have  $\text{depth}(\langle \nu y \rangle a\{y\}.P') \leq \text{depth}(\langle \nu y \rangle \langle \nu y \rangle a\{y\}.P')$  by simp
ultimately show ?case using PeqP' by blast
next
assume xineqy: x ≠ y
show ?case
proof(case-tac x=a)
  assume  $x=a$ 
  moreover have uhnf 0 by(simp add: uhnf-def)
  moreover have valid 0 by simp
  moreover have  $\langle \nu a \rangle \langle \nu y \rangle a\{y\}.P' \equiv_e \mathbf{0}$ 
  proof –
    have  $\langle \nu a \rangle \langle \nu y \rangle a\{y\}.P' \equiv_e \langle \nu y \rangle \langle \nu a \rangle a\{y\}.P'$  by(rule ResComm)
    moreover have  $\langle \nu y \rangle \langle \nu a \rangle a\{y\}.P' \equiv_e \mathbf{0}$ 
      by(blast intro: ResOutput' ResNil ResPres Trans)
    ultimately show ?thesis by(blast intro: Trans)
  qed
  moreover have  $\text{depth } \mathbf{0} \leq \text{depth}(\langle \nu a \rangle \langle \nu y \rangle a\{y\}.P')$  by simp
  ultimately show ?case using PeqP' by blast
next
assume xineqa: x ≠ a
from aineqy have uhnf( $\langle \nu y \rangle a\{y\}.(\langle \nu x \rangle P')$ ) by(force simp add: uhnf-def)
moreover from validP PeqP' have valid( $\langle \nu y \rangle a\{y\}.(\langle \nu x \rangle P')$ ) by simp
moreover have  $\langle \nu x \rangle \langle \nu y \rangle a\{y\}.P' \equiv_e \langle \nu y \rangle a\{y\}.(\langle \nu x \rangle P')$ 
proof –
  have  $\langle \nu x \rangle \langle \nu y \rangle a\{y\}.P' \equiv_e \langle \nu y \rangle \langle \nu x \rangle a\{y\}.P'$  by(rule ResComm)
moreover from xineqa xineqy have  $\langle \nu y \rangle \langle \nu x \rangle a\{y\}.P' \equiv_e \langle \nu y \rangle a\{y\}.(\langle \nu x \rangle P')$ 
  by(blast intro: ResOutput ResPres Trans)
  ultimately show ?thesis by(blast intro: Trans)
qed
moreover have  $\text{depth}(\langle \nu y \rangle a\{y\}.(\langle \nu x \rangle P')) \leq \text{depth}(\langle \nu x \rangle \langle \nu y \rangle a\{y\}.P')$ 
  by simp
ultimately show ?case using PeqP' by blast
qed
qed
next
case(Bang P x)
have valid(!P) by fact
hence False by simp
thus ?case by simp
qed

lemma expandHnf:
fixes  $P :: \pi$ 
and  $S :: \pi \text{ set}$ 

assumes  $(P, S) \in \text{sumComposeSet}$ 
and  $\forall P \in S. \text{uhnf } P \wedge \text{valid } P$ 

shows  $\exists P'. \text{uhnf } P' \wedge \text{valid } P' \wedge P \equiv_e P' \wedge \text{depth } P' \leq \text{depth } P$ 

```

```

using assms
proof(induct rule: sumComposeSet.induct)
  case empty
    have uhnf 0 by(simp add: uhnf-def)
    moreover have valid 0 by simp
    moreover have  $0 \equiv_e 0$  by(rule Refl)
    moreover have depth 0 ≤ depth 0 by simp
    ultimately show ?case by blast
next
  case(insert Q S P)
    have Shnf: ∀ P ∈ S. uhnf P ∧ valid P by fact
    hence  $\forall P \in (S - \{(Q)\})$ . uhnf P ∧ valid P by simp
    moreover have  $\forall P \in (S - \{(Q)\})$ . uhnf P ∧ valid P  $\implies \exists P'. \text{uhnf } P' \wedge \text{valid } P' \wedge P \equiv_e P' \wedge \text{depth } P' \leq \text{depth } P$  by fact
    ultimately obtain P' where P'hnf: uhnf P' and validP': valid P'
      and PeqP': P ≡e P' and PP'depth: depth P' ≤ depth P
      by blast

    have  $Q \in S$  by fact
    with Shnf have uhnf Q and valid Q by simp+
    with P'hnf validP' obtain R where Rhnf: uhnf R and validR: valid R
      and P'QeqR: P' ⊕ Q ≡e R and P'QRdepth: depth R ≤
depth (P' ⊕ Q)
      by(auto dest: uhnfSum)

    from PeqP' P'QeqR have  $P \oplus Q \equiv_e R$  by(blast intro: SumPres Trans)
    moreover from PP'depth P'QRdepth have depth R ≤ depth (P ⊕ Q) by simp
    ultimately show ?case using Rhnf validR by blast
qed

lemma hnfSummandsRemove:
  fixes P :: pi
  and Q :: pi

  assumes  $P \in \text{summands } Q$ 
  and uhnf Q

  shows  $(\text{summands } Q) - \{P' \mid P'. P' \in \text{summands } Q \wedge P' \equiv_e P\} = (\text{summands } Q) - \{P\}$ 
using assms
by(auto intro: Refl simp add: uhnf-def)

lemma pullSummand:
  fixes P :: pi
  and Q :: pi

  assumes PsummQ: P ∈ summands Q
  and Qhnf: uhnf Q

```

shows $\exists Q'. P \oplus Q' \equiv_e Q \wedge (\text{summands } Q') = ((\text{summands } Q) - \{x. \exists P'. x = P' \wedge P' \in (\text{summands } Q) \wedge P' \equiv_e P\}) \wedge \text{uhnf } Q'$

proof –

have *SumGoal*: $\bigwedge P Q R. \llbracket P \in \text{summands } Q; \text{uhnf}(Q \oplus R);$
 $\bigwedge P. \llbracket P \in \text{summands } Q \rrbracket \implies \exists Q'. P \oplus Q' \equiv_e Q \wedge$
 $(\text{summands } Q') = ((\text{summands } Q) - \{P' \mid P'. P' \in$
 $\text{summands } Q \wedge P' \equiv_e P\}) \wedge \text{uhnf } Q';$
 $\bigwedge P. \llbracket P \in \text{summands } R \rrbracket \implies \exists R'. P \oplus R' \equiv_e R \wedge$
 $(\text{summands } R') = ((\text{summands } R) - \{P' \mid P'. P' \in$
 $\text{summands } R \wedge P' \equiv_e P\}) \wedge \text{uhnf } R'\rrbracket$
 $\implies \exists Q'. P \oplus Q' \equiv_e Q \oplus R \wedge$
 $\text{summands } Q' = \text{summands } (\text{pi.Sum } Q R) - \{P' \mid P'. P' \in \text{summands}$
 $(Q \oplus R) \wedge P' \equiv_e P\} \wedge \text{uhnf } Q'$

proof –

fix $P Q R$

assume *IHR*: $\bigwedge P. P \in \text{summands } R \implies \exists R'. P \oplus R' \equiv_e R \wedge$
 $(\text{summands } R') = ((\text{summands } R) - \{P' \mid P'. P' \in$
 $\text{summands } R \wedge P' \equiv_e P\}) \wedge \text{uhnf } R'$

assume *PsummQ*: $P \in \text{summands } Q$

moreover assume $\bigwedge P. P \in \text{summands } Q \implies \exists Q'. P \oplus Q' \equiv_e Q \wedge$
 $(\text{summands } Q') = ((\text{summands } Q) - \{P' \mid P'. P' \in$
 $\text{summands } Q \wedge P' \equiv_e P\}) \wedge \text{uhnf } Q'$

ultimately obtain *Q'* where *PQ'eqQ*: $P \oplus Q' \equiv_e Q$

and *Q'summQ*: $(\text{summands } Q') = ((\text{summands } Q) - \{P'$
 $\mid P'. P' \in \text{summands } Q \wedge P' \equiv_e P\})$

and *Q'hnf*: $\text{uhnf } Q'$

by *blast*

assume *QRhnf*: $\text{uhnf}(Q \oplus R)$

show $\exists Q'. P \oplus Q' \equiv_e Q \oplus R \wedge$

$\text{summands } Q' = \text{summands } (\text{pi.Sum } Q R) - \{P' \mid P'. P' \in \text{summands}$
 $(Q \oplus R) \wedge P' \equiv_e P\} \wedge \text{uhnf } Q'$

proof(*cases* $\exists P' \in \text{summands } R. P' \equiv_e P$)

assume $\exists P' \in \text{summands } R. P' \equiv_e P$

then obtain *P'* where *P'summR*: $P' \in \text{summands } R$ **and** *P'eqP*: $P' \equiv_e P$
by *blast*

with *IHR* **obtain** *R'* where *PR'eqR*: $P' \oplus R' \equiv_e R$

and *R'summR*: $(\text{summands } R') = ((\text{summands } R) - \{P'' \mid P''. P'' \in$
 $\text{summands } R \wedge P'' \equiv_e P'\})$

and *R'hnf*: $\text{uhnf } R'$

by *blast*

have *L1*: $P \oplus (Q' \oplus R') \equiv_e Q \oplus R$

proof –

from *P'eqP* **have** $P \oplus (Q' \oplus R') \equiv_e (P \oplus P') \oplus (Q' \oplus R')$

by(*blast intro: SumIdemp' SumPres Sym*)

moreover have $(P \oplus P') \oplus (Q' \oplus R') \equiv_e P \oplus (P' \oplus (Q' \oplus R'))$ **by**(*rule*
SumAssoc)

```

moreover have  $P \oplus (P' \oplus (Q' \oplus R')) \equiv_e P \oplus (P' \oplus (R' \oplus Q'))$ 
  by(blast intro: Refl SumPres' SumSym)
moreover have  $P \oplus (P' \oplus (R' \oplus Q')) \equiv_e P \oplus (P' \oplus R') \oplus Q'$ 
  by(blast intro: Refl SumPres' Sym SumAssoc)
moreover have  $P \oplus (P' \oplus R') \oplus Q' \equiv_e (P \oplus Q') \oplus (P' \oplus R')$ 
proof –
  have  $P \oplus (P' \oplus R') \oplus Q' \equiv_e P \oplus Q' \oplus (P' \oplus R')$ 
    by(blast intro: Refl SumPres' SumSym)
  thus ?thesis by(blast intro: Sym SumAssoc Trans)
qed
moreover from PQ'eqQ PR'eqR have  $(P \oplus Q') \oplus (P' \oplus R') \equiv_e Q \oplus R$ 
by(rule SumPres')
  ultimately show ?thesis by(blast intro!: Trans)
qed

show ?thesis
proof(cases Q' = 0)
  assume Q'eqNil: Q' = 0
  have  $P \oplus R' \equiv_e Q \oplus R$ 
  proof –
    have  $P \oplus R' \equiv_e P \oplus (R' \oplus \mathbf{0})$  by(blast intro: SumZero Refl Trans
SumPres' Sym)
    moreover have  $P \oplus (R' \oplus \mathbf{0}) \equiv_e P \oplus (\mathbf{0} \oplus R')$ 
      by(blast intro: SumSym Trans SumPres' Refl)
    ultimately show ?thesis using L1 Q'eqNil by(blast intro: Trans)
  qed
  moreover from R'summR Q'summQ P'eqP Q'eqNil have summands (R')
  = (summands (Q ⊕ R) - {P' | P'. P' ∈ summands(Q ⊕ R) ∧ P' ≡e P})
    by(auto intro: Sym Trans)
  ultimately show ?thesis using R'hnf by blast
next
  assume Q'ineqNil: Q' ≠ 0
  show ?thesis
  proof(case-tac R' = 0)
    assume R'eqNil: R' = 0
    have  $P \oplus Q' \equiv_e Q \oplus R$ 
    proof –
      have  $P \oplus Q' \equiv_e P \oplus (Q' \oplus \mathbf{0})$  by(blast intro: SumZero Refl Trans
SumPres' Sym)
      with L1 R'eqNil show ?thesis by(blast intro: Trans)
    qed
    moreover from R'summR Q'summQ P'eqP R'eqNil have summands (Q')
  = (summands (Q ⊕ R) - {P' | P'. P' ∈ summands(Q ⊕ R) ∧ P' ≡e P})
    by(auto intro: Sym Trans)
    ultimately show ?thesis using Q'hnf by blast
  next
    assume R'ineqNil: R' ≠ 0
from R'summR Q'summQ P'eqP have summands (Q' ⊕ R') = (summands

```

$(Q \oplus R) - \{P' \mid P'. P' \in \text{summands}(Q \oplus R) \wedge P' \equiv_e P\}$
by(*auto intro: Sym Trans*)
moreover from $QRhnf\ Q'hnf\ R'hnf\ R'summR\ Q'summQ\ Q'ineqNil$
 $R'ineqNil$ **have** $uhnf(Q' \oplus R')$
by(*auto simp add: uhnf-def*)

ultimately show *?thesis using L1 by blast*
qed
qed
next

assume $\neg(\exists P' \in \text{summands } R. P' \equiv_e P)$
hence *Case: $\forall P' \in \text{summands } R. \neg(P' \equiv_e P)$* **by** *simp*
show *?thesis*
proof(*case-tac Q' = 0*)
assume $Q'eqNil: Q' = \mathbf{0}$
have $P \oplus R \equiv_e Q \oplus R$
proof –

have $P \oplus R \equiv_e (P \oplus \mathbf{0}) \oplus R$ **by**(*blast intro: SumZero Sym Trans SumPres*)
moreover from $PQ'eqQ$ **have** $P \oplus (Q' \oplus R) \equiv_e Q \oplus R$
by(*blast intro: SumAssoc Trans Sym SumPres*)
ultimately show *?thesis using Q'eqNil by(blast intro: SumAssoc Trans)*
qed

moreover from $Q'summQ\ Q'eqNil\ Case$ **have** $\text{summands } (R) = (\text{summands } (Q \oplus R) - \{P' \mid P'. P' \in \text{summands}(Q \oplus R) \wedge P' \equiv_e P\})$
by *auto*
moreover from $QRhnf$ **have** $uhnf\ R$ **by**(*simp add: uhnf-def*)

ultimately show *?thesis by blast*
next

assume $Q'ineqNil: Q' \neq \mathbf{0}$
from $PQ'eqQ$ **have** $P \oplus (Q' \oplus R) \equiv_e Q \oplus R$
by(*blast intro: SumAssoc Trans Sym SumPres*)

moreover from $Q'summQ\ Case$ **have** $\text{summands } (Q' \oplus R) = (\text{summands } (Q \oplus R) - \{P' \mid P'. P' \in \text{summands}(Q \oplus R) \wedge P' \equiv_e P\})$
by *auto*
moreover from $QRhnf\ Q'hnf\ Q'summQ\ Q'ineqNil$ **have** $uhnf(Q' \oplus R)$
by(*auto simp add: uhnf-def*)
ultimately show *?thesis by blast*
qed
qed
qed

from *assms show ?thesis*
proof(*nominal-induct Q arbitrary: P rule: pi.strong-inducts*)
case $PiNil$
have $P \in \text{summands } \mathbf{0}$ **by** *fact*

hence *False* by *auto*
 thus *?case* by *simp*
next
 case(*Output a b Q*)
 have $P \in \text{summands } (a\{b\}.Q)$ by *fact*
 hence *PeqQ*: $P = a\{b\}.Q$ by *simp*
 have $P \oplus \mathbf{0} \equiv_e a\{b\}.Q$
proof –
 have $P \oplus \mathbf{0} \equiv_e P$ by(*rule SumZero*)
 with *PeqQ* show *?thesis* by *simp*
qed
 moreover have $(\text{summands } \mathbf{0}) = (\text{summands } (a\{b\}.Q)) - \{P' \mid P'. P' \in \text{summands } (a\{b\}.Q) \wedge P' \equiv_e P\}$
proof –
 have $a\{b\}.Q \equiv_e a\{b\}.Q$ by(*rule Refl*)
 with *PeqQ* show *?thesis* by *simp*
qed
 moreover have *uhnf 0* by(*simp add: uhnf-def*)
 ultimately show *?case* by *blast*
next
 case(*Tau Q*)
 have $P \in \text{summands } (\tau.(Q))$ by *fact*
 hence *PeqQ*: $P = \tau.(Q)$ by *simp*
 have $P \oplus \mathbf{0} \equiv_e \tau.(Q)$
proof –
 have $P \oplus \mathbf{0} \equiv_e P$ by(*rule SumZero*)
 with *PeqQ* show *?thesis* by *simp*
qed
 moreover have $(\text{summands } \mathbf{0}) = (\text{summands } (\tau.(Q))) - \{P' \mid P'. P' \in \text{summands } (\tau.(Q)) \wedge P' \equiv_e P\}$
proof –
 have $\tau.(Q) \equiv_e \tau.(Q)$ by(*rule Refl*)
 with *PeqQ* show *?thesis* by *simp*
qed
 moreover have *uhnf 0* by(*simp add: uhnf-def*)
 ultimately show *?case* by *blast*
next
 case(*Input a x Q*)
 have $P \in \text{summands } (a\langle x \rangle.Q)$ by *fact*
 hence *PeqQ*: $P = a\langle x \rangle.Q$ by *simp*
 have $P \oplus \mathbf{0} \equiv_e a\langle x \rangle.Q$
proof –
 have $P \oplus \mathbf{0} \equiv_e P$ by(*rule SumZero*)
 with *PeqQ* show *?thesis* by *simp*
qed
 moreover have $(\text{summands } \mathbf{0}) = (\text{summands } (a\langle x \rangle.Q)) - \{P' \mid P'. P' \in \text{summands } (a\langle x \rangle.Q) \wedge P' \equiv_e P\}$
proof –
 have $a\langle x \rangle.Q \equiv_e a\langle x \rangle.Q$ by(*rule Refl*)

```

    with PeqQ show ?thesis by simp
  qed
  moreover have uhnf 0 by(simp add: uhnf-def)
  ultimately show ?case by blast
next
  case(Match a b Q)
  have  $P \in \text{summands } ([a \sim b] Q)$  by fact
  hence False by simp
  thus ?case by simp
next
  case(Mismatch a b Q)
  have  $P \in \text{summands } ([a \neq b] Q)$  by fact
  hence False by simp
  thus ?case by simp
next
  case(Sum Q R)
  have QRhnf: uhnf ( $Q \oplus R$ ) by fact
  hence Qhnf: uhnf Q and Rhnf: uhnf R by(simp add: uhnf-def)+
  have  $\bigwedge P. [P \in \text{summands } Q; \text{uhnf } Q] \implies \exists Q'. P \oplus Q' \equiv_e Q \wedge$ 
     $(\text{summands } Q') = ((\text{summands } Q) - \{P' \mid P'. P' \in$ 
 $\in \text{summands } Q \wedge P' \equiv_e P\}) \wedge \text{uhnf } Q'$ 
    by fact
    with Qhnf have IHQ:  $\bigwedge P. P \in \text{summands } Q \implies \exists Q'. P \oplus Q' \equiv_e Q \wedge$ 
       $(\text{summands } Q') = ((\text{summands } Q) - \{P' \mid P'. P' \in$ 
 $\text{summands } Q \wedge P' \equiv_e P\}) \wedge \text{uhnf } Q'$ 
      by simp
      have  $\bigwedge P. [P \in \text{summands } R; \text{uhnf } R] \implies \exists R'. P \oplus R' \equiv_e R \wedge$ 
         $(\text{summands } R') = ((\text{summands } R) - \{P' \mid P'. P' \in$ 
 $\in \text{summands } R \wedge P' \equiv_e P\}) \wedge \text{uhnf } R'$ 
        by fact
        with Rhnf have IHR:  $\bigwedge P. P \in \text{summands } R \implies \exists R'. P \oplus R' \equiv_e R \wedge$ 
           $(\text{summands } R') = ((\text{summands } R) - \{P' \mid P'. P' \in$ 
 $\text{summands } R \wedge P' \equiv_e P\}) \wedge \text{uhnf } R'$ 
          by simp
          have  $P \in \text{summands } (Q \oplus R)$  by fact
          hence  $P \in \text{summands } Q \vee P \in \text{summands } R$  by simp
          thus ?case
        proof(rule disjE)
          assume  $P \in \text{summands } Q$ 
          thus ?case using QRhnf IHQ IHR by(rule SumGoal)
        next
          assume  $P \in \text{summands } R$ 
          moreover from QRhnf have uhnf ( $R \oplus Q$ ) by(auto simp add: uhnf-def)
          ultimately have  $\exists Q'. (\text{pi.Sum } P \ Q') \equiv_e (\text{pi.Sum } R \ Q) \wedge$ 
             $\text{summands } Q' = \text{summands } (\text{pi.Sum } R \ Q) - \{P' \mid P'. P' \in \text{summands}$ 
 $(\text{pi.Sum } R \ Q) \wedge P' \equiv_e P\} \wedge \text{uhnf } Q'$  using IHR IHQ
            by(rule SumGoal)
          thus ?case
            by(force intro: SumSym Trans)

```

```

qed
next
case(Par Q R P)
have P ∈ summands (Q || R) by fact
hence False by simp
thus ?case by simp
next
case(Res x Q P)
have P ∈ summands (<νx>Q) by fact
hence P eqQ: P = <νx>Q by (simp add: if-split)
have P ⊕ 0 ≡e <νx>Q
proof -
  have P ⊕ 0 ≡e P by (rule SumZero)
  with P eqQ show ?thesis by simp
qed
moreover have (summands 0) = (summands (<νx>Q)) - {P' | P'. P' ∈
summands (<νx>Q) ∧ P' ≡e P}
proof -
  have <νx>Q ≡e <νx>Q by (rule Refl)
  with P eqQ show ?thesis by simp
qed
moreover have hnf 0 by (simp add: hnf-def)
ultimately show ?case by blast
next
case(Bang Q P)
have P ∈ summands (!Q) by fact
hence False by simp
thus ?case by simp
qed
qed

```

lemma *nSym*:

```

fixes P :: pi
and Q :: pi

```

assumes $\neg(P \equiv_e Q)$

shows $\neg(Q \equiv_e P)$

using *assms*

by (*blast dest: Sym*)

lemma *summandsZero*:

```

fixes P :: pi

```

assumes $\text{summands } P = \{\}$

and $\text{hnf } P$

shows $P = 0$

using *assms*

by(*nominal-induct P rule: pi.strong-inducts, auto intro: Refl SumIdemp SumPres' Trans*)

lemma *summandsZero'*:

fixes $P :: pi$

assumes $summP: summands P = \{\}$

and $PuhnP: uhnf P$

shows $P = 0$

proof –

from $PuhnP$ **have** $hnf P$ **by**(*simp add: uhnf-def*)

with $summP$ **show** *?thesis* **by**(*rule summandsZero*)

qed

lemma *summandEquiv*:

fixes $P :: pi$

and $Q :: pi$

assumes $PhnP: uhnf P$

and $QhnP: uhnf Q$

and $PinQ: \forall P' \in summands P. \exists Q' \in summands Q. P' \equiv_e Q'$

and $QinP: \forall Q' \in summands Q. \exists P' \in summands P. Q' \equiv_e P'$

shows $P \equiv_e Q$

proof –

from *finiteSummands assms* **show** *?thesis*

proof(*induct F==summands P arbitrary: P Q rule: finite-induct*)

case(*empty P Q*)

have $PEmpty: \{\} = summands P$ **by** *fact*

moreover **have** $\forall Q' \in summands Q. \exists P' \in summands P. Q' \equiv_e P'$ **by** *fact*

ultimately **have** $QEmpty: summands Q = \{\}$ **by** *simp*

have $P = 0$

proof –

have $uhnP: uhnf P$ **by** *fact*

with $PEmpty$ **show** *?thesis* **by**(*blast intro: summandsZero'*)

qed

moreover **have** $Q = 0$

proof –

have $uhnQ: uhnf Q$ **by** *fact*

with $QEmpty$ **show** *?thesis* **by**(*blast intro: summandsZero'*)

qed

ultimately **show** *?case* **by**(*blast intro: Refl*)

next

case(*insert P' F P Q*)

have $PhnP: uhnf P$ **by** *fact*

have $QhnP: uhnf Q$ **by** *fact*

have $IH: \bigwedge P Q. \llbracket F = \text{summands } P; \text{uhnf } P; \text{uhnf } Q; \forall P' \in \text{summands } P. \exists Q' \in \text{summands } Q. P' \equiv_e Q' \rrbracket$
 $\forall Q' \in \text{summands } Q. \exists P' \in \text{summands } P. Q' \equiv_e P' \rrbracket \implies P \equiv_e Q$
by fact
have $PeqQ: \forall P' \in \text{summands } P. \exists Q' \in \text{summands } Q. P' \equiv_e Q'$ **by fact**
have $QeqP: \forall Q' \in \text{summands } Q. \exists P' \in \text{summands } P. Q' \equiv_e P'$ **by fact**

have $P\text{Summ}: \text{insert } P' F = \text{summands } P$ **by fact**
hence $P'\text{Summ}P: P' \in \text{summands } P$ **by auto**

with $Phnf$ **obtain** P'' **where** $P'P''eqP: P' \oplus P'' \equiv_e P$
and $P''\text{Summ}: \text{summands } P'' = \text{summands } P - \{P'' \mid P''\}$
 $P'' \in \text{summands } P \wedge P'' \equiv_e P'$
and $P''hnf: \text{uhnf } P''$
by(blast dest: pullSummand)

from $PeqQ P'\text{Summ}P$ **obtain** Q' **where** $Q'\text{Summ}Q: Q' \in \text{summands } Q$ **and**
 $P'eqQ': P' \equiv_e Q'$ **by blast**

from $Q'\text{Summ}Q Qhnf$ **obtain** Q'' **where** $Q'Q''eqQ: Q' \oplus Q'' \equiv_e Q$
and $Q''\text{Summ}: \text{summands } Q'' = \text{summands } Q - \{Q'' \mid Q''\}$
 $Q'' \in \text{summands } Q \wedge Q'' \equiv_e Q'$
and $Q''hnf: \text{uhnf } Q''$
by(blast dest: pullSummand)

have $FeqP'': F = \text{summands } P''$
proof –
have $P' \notin F$ **by fact**
with $P''\text{Summ } P\text{Summ } hnf\text{SummandsRemove}[OF P'\text{Summ}P Phnf]$ **show**
 $?thesis$ **by blast**
qed

moreover have $\forall P' \in \text{summands } P''. \exists Q' \in \text{summands } Q''. P' \equiv_e Q'$
proof(rule ballI)
fix P'''
assume $P'''\text{Summ}: P''' \in \text{summands } P''$
with $P''\text{Summ}$ **have** $P''' \in \text{summands } P$ **by simp**
with $PeqQ$ **obtain** Q''' **where** $Q'''\text{Summ}: Q''' \in \text{summands } Q$ **and** $P'''\text{eq}Q''':$
 $P''' \equiv_e Q'''$ **by blast**
have $Q''' \in \text{summands } Q''$
proof –
from $P'''\text{Summ } P''\text{Summ}$ **have** $\neg(P''' \equiv_e P')$ **by simp**
with $P'eqQ' P'''\text{eq}Q'''$ **have** $\neg(Q''' \equiv_e Q')$ **by**(blast intro: Trans Sym)
with $Q''\text{Summ } Q'''\text{Summ}$ **show** $?thesis$ **by simp**
qed

with $P'''\text{eq}Q'''$ **show** $\exists Q' \in \text{summands } Q''. P''' \equiv_e Q'$ **by blast**
qed

moreover have $\forall Q' \in \text{summands } Q''. \exists P' \in \text{summands } P''. Q' \equiv_e P'$
proof(*rule ballI*)
fix Q'''
assume $Q''' \text{Summ}: Q''' \in \text{summands } Q''$
with $Q'' \text{Summ}$ **have** $Q''' \in \text{summands } Q''$ **by** *simp*
with $Q \text{eq} P$ **obtain** P''' **where** $P''' \text{Summ}: P''' \in \text{summands } P$
and $Q''' \text{eq} P''': Q''' \equiv_e P'''$ **by** *blast*
have $P''' \in \text{summands } P''$
proof –
from $Q''' \text{Summ } Q'' \text{Summ}$ **have** $\neg(Q''' \equiv_e Q')$ **by** *simp*
with $P' \text{eq} Q' \ Q''' \text{eq} P'''$ **have** $\neg(P''' \equiv_e P')$ **by**(*blast intro: Trans*)
with $P'' \text{Summ } P''' \text{Summ}$ **show** *?thesis* **by** *simp*
qed
with $Q''' \text{eq} P'''$ **show** $\exists P' \in \text{summands } P''. Q''' \equiv_e P'$ **by** *blast*
qed

ultimately have $P'' \text{eq} Q'': P'' \equiv_e Q''$ **using** $P'' \text{hnf } Q'' \text{hnf}$ **by**(*rule-tac IH*)

from $P' P'' \text{eq} P$ **have** $P \equiv_e P' \oplus P''$ **by**(*rule Sym*)
moreover from $P' \text{eq} Q' \ P'' \text{eq} Q''$ **have** $P' \oplus P'' \equiv_e Q' \oplus Q''$ **by**(*rule SumPres'*)
ultimately show *?case* **using** $Q' Q'' \text{eq} Q$ **by**(*blast intro: Trans*)
qed
qed

lemma *validSubst[*simp*]*:

fixes $P :: pi$
and $a :: name$
and $b :: name$
and $p :: pi$

shows $\text{valid}(P[a := b]) = \text{valid } P$
by(*nominal-induct P avoiding: a b rule: pi.strong-inducts, auto*)

lemma *validOutputTransition*:

fixes $P :: pi$
and $a :: name$
and $b :: name$
and $P' :: pi$

assumes $P \mapsto_a[b] \prec P'$
and $\text{valid } P$

shows $\text{valid } P'$
using *assms*
by(*nominal-induct rule: outputInduct, auto*)

lemma *validInputTransition*:

fixes $P :: pi$

```

and a :: name
and x :: name
and P' :: pi

assumes PTrans: P  $\mapsto$  a<x>  $\prec$  P'
and validP: valid P

shows valid P'
proof –
have Goal:  $\bigwedge$  P a x P'.  $\llbracket$  P  $\mapsto$  a<x>  $\prec$  P'; x  $\#$  P; valid P  $\rrbracket \implies$  valid P'
proof –
  fix P a x P'
  assume P  $\mapsto$  a<x>  $\prec$  P' and x  $\#$  P and valid P
  thus valid P'
  by(nominal-induct rule: inputInduct, auto)
qed
obtain y::name where yFreshP: y  $\#$  P and yFreshP': y  $\#$  P'
  by(rule-tac name-exists-fresh[of (P, P')], auto simp add: fresh-prod)
  from yFreshP' PTrans have P  $\mapsto$  a<y>  $\prec$  [(x, y)]  $\cdot$  P' by(simp add: algebraBoundResidual)
  hence valid [(x, y)]  $\cdot$  P' using yFreshP validP by(rule Goal)
  thus valid P' by simp
qed

lemma validBoundOutputTransition:
  fixes P :: pi
  and a :: name
  and x :: name
  and P' :: pi

  assumes PTrans: P  $\mapsto$  a< $\nu$ x>  $\prec$  P'
  and validP: valid P

  shows valid P'
  proof –
  have Goal:  $\bigwedge$  P a x P'.  $\llbracket$  P  $\mapsto$  a< $\nu$ x>  $\prec$  P'; x  $\#$  P; valid P  $\rrbracket \implies$  valid P'
  proof –
    fix P a x P'
    assume P  $\mapsto$  a< $\nu$ x>  $\prec$  P' and x  $\#$  P and valid P
    thus valid P'
    apply(nominal-induct rule: boundOutputInduct, auto)
    proof –
      fix P a x P'
      assume P  $\mapsto$  (a::name)[x]  $\prec$  P' and valid P
      thus valid P'
      by(nominal-induct rule: outputInduct, auto)
    qed
  qed
obtain y::name where yFreshP: y  $\#$  P and yFreshP': y  $\#$  P'

```

```

  by(rule-tac name-exists-fresh[of (P, P')], auto simp add: fresh-prod)
  from yFreshP' PTrans have P  $\mapsto$  a< $\nu$ y> <(x, y) . P' by(simp add: alphaBoundResidual)
  hence valid [(x, y) . P'] using yFreshP validP by(rule Goal)
  thus valid P' by simp
qed

```

lemma *validTauTransition*:

```

fixes P :: pi
and P' :: pi

```

```

assumes PTrans: P  $\mapsto$   $\tau$  < P'
and validP: valid P

```

```

shows valid P'

```

```

using assms

```

```

by(nominal-induct rule: tauInduct, auto dest: validOutputTransition validInputTransition validBoundOutputTransition)

```

lemmas *validTransition = validInputTransition validOutputTransition validTauTransition validBoundOutputTransition*

lemma *validSummand*:

```

fixes P :: pi
and P' :: pi
and a :: name
and b :: name
and x :: name

```

```

assumes valid P
and hnf P

```

```

shows  $\tau.(P') \in$  summands P  $\implies$  valid P'
and a{b}.P'  $\in$  summands P  $\implies$  valid P'
and a<x>.P'  $\in$  summands P  $\implies$  valid P'
and  $\llbracket a \neq x; <\nu x>a\{x\}.P' \in$  summands P  $\rrbracket \implies$  valid P'

```

```

proof -

```

```

  assume  $\tau.(P') \in$  summands P

```

```

  with assms show valid P' by(force intro: validTauTransition simp add: summandTransition)

```

```

next

```

```

  assume a{b}.P'  $\in$  summands P

```

```

  with assms show valid P' by(force intro: validOutputTransition simp add: summandTransition)

```

```

next

```

```

  assume a<x>.P'  $\in$  summands P

```

```

  with assms show valid P' by(force intro: validInputTransition simp add: summandTransition)

```

```

next

```

```

assume  $\langle \nu x \rangle a\{x\}.P' \in \text{summands } P$  and  $a \neq x$ 
with assms show valid P'
  by(force intro: validBoundOutputTransition simp add: summandTransition[THEN
sym])
qed

```

lemma *validExpand*:

```

fixes  $P :: pi$ 
and  $Q :: pi$ 

```

```

assumes valid P
and valid Q
and uhnf P
and uhnf Q

```

shows $\forall R \in (\text{expandSet } P \ Q). \text{uhnf } R \wedge \text{valid } R$

proof –

```

from assms have hnf P and hnf Q by(simp add: uhnf-def)+
with assms show ?thesis

```

```

  apply(auto simp add: expandSet-def)
  apply(force dest: validSummand simp add: uhnf-def)
  apply(force dest: validSummand)
  apply(force dest: validSummand simp add: uhnf-def)
  apply(force dest: validSummand)
  apply(force dest: validSummand simp add: uhnf-def)
  apply(force dest: validSummand)
  apply(force dest: validSummand simp add: uhnf-def)
  apply(force dest: validSummand)
  apply(force dest: validSummand simp add: uhnf-def)
  apply(force dest: validSummand)
  apply(force dest: validSummand simp add: uhnf-def)
  apply(force dest: validSummand)
  apply(subgoal-tac a≠x)
  apply(force dest: validSummand simp add: uhnf-def)
  apply blast
  apply(subgoal-tac a≠x)
  apply(drule-tac validSummand(4)) apply assumption+
  apply blast
  apply(subgoal-tac a≠x)
  apply(drule-tac validSummand(4) [where  $P=Q$ ]) apply assumption+
  apply(force dest: validSummand simp add: uhnf-def)
  apply blast
  apply(subgoal-tac a≠x)
  apply(drule-tac validSummand(4) [where  $P=Q$ ]) apply assumption+
  apply blast
  apply(force dest: validSummand simp add: uhnf-def)
  apply(force dest: validSummand)
  apply(force dest: validSummand simp add: uhnf-def)
  apply(force simp add: uhnf-def)

```

```

apply(force dest: validSummand)
apply(force dest: validSummand)
apply(force simp add: uhnf-def)
apply(force dest: validSummand)
apply(subgoal-tac a≠y)
apply(drule-tac validSummand(4)[where P=Q]) apply assumption+
apply blast
apply(force dest: validSummand simp add: uhnf-def)
apply(subgoal-tac a≠y)
apply(drule-tac validSummand(4)) apply assumption+
apply blast
by(force dest: validSummand)
qed

```

```

lemma expandComplete:
  fixes F :: pi set

  assumes finite F

  shows  $\exists P. (P, F) \in \text{sumComposeSet}$ 
using assms
proof(induct F rule: finite-induct)
  case empty
  have  $(\mathbf{0}, \{\}) \in \text{sumComposeSet}$  by(rule sumComposeSet.empty)
  thus ?case by blast
next
  case(insert Q F)
  have  $\exists P. (P, F) \in \text{sumComposeSet}$  by fact
  then obtain P where  $(P, F) \in \text{sumComposeSet}$  by blast
  moreover have  $Q \in \text{insert } Q F$  by simp
  moreover have  $Q \notin F$  by fact
  ultimately have  $(P \oplus Q, \text{insert } Q F) \in \text{sumComposeSet}$ 
    by(force intro: sumComposeSet.insert)
  thus ?case by blast
qed

```

```

lemma expandDepth:
  fixes F :: pi set
  and P :: pi
  and Q :: pi

  assumes  $(P, F) \in \text{sumComposeSet}$ 
  and  $F \neq \{\}$ 

  shows  $\exists Q \in F. \text{depth } P \leq \text{depth } Q \wedge (\forall R \in F. \text{depth } R \leq \text{depth } Q)$ 
using assms
proof(induct arbitrary: Q rule: sumComposeSet.induct)
  case empty
  have  $(\{\}::\text{pi set}) \neq \{\}$  by fact

```

```

hence False by simp
thus ?case by simp
next
case(insert Q S P)
have QinS:  $Q \in S$  by fact
show ?case
proof(case-tac ( $S - \{Q\} = \{\}$ )
  assume ( $S - \{Q\} = \{\}$ )
  with QinS have SeqQ:  $S = \{Q\}$  by auto
  have ( $P, S - \{Q\}$ )  $\in$  sumComposeSet by fact
  with SeqQ have ( $P, \{\}$ )  $\in$  sumComposeSet by simp
  hence  $P = \mathbf{0}$  apply - by(ind-cases ( $P, \{\}$ )  $\in$  sumComposeSet, auto)
  with QinS SeqQ show ?case by simp
next
  assume ( $S - \{Q\} \neq \{\}$ )
  moreover have ( $S - \{Q\} \neq \{\}$ )  $\implies \exists Q' \in (S - \{Q\}). \text{depth } P \leq \text{depth } Q'$ 
 $\wedge (\forall R \in (S - \{Q\}). \text{depth } R \leq \text{depth } Q')$  by fact
  ultimately obtain Q' where Q'inS:  $Q' \in S - \{Q\}$  and PQ'depth:  $\text{depth } P \leq \text{depth } Q'$  and All:  $\forall R \in (S - \{Q\}). \text{depth } R \leq \text{depth } Q'$  by auto
  show ?case
  proof(case-tac  $Q = Q'$ )
    assume  $Q = Q'$ 
    with PQ'depth All QinS show ?case by auto
  next
    assume QineqQ':  $Q \neq Q'$ 
    show ?case
    proof(case-tac  $\text{depth } Q \leq \text{depth } Q'$ )
      assume  $\text{depth } Q \leq \text{depth } Q'$ 
      with QineqQ' PQ'depth All Q'inS show ?thesis by force
    next
      assume  $\neg \text{depth } Q \leq \text{depth } Q'$ 
      with QineqQ' PQ'depth All Q'inS QinS show ?thesis apply auto
      apply(rule-tac  $x=Q$  in bexI)
      apply auto
      apply(case-tac  $R=Q$ )
      apply auto
      apply(erule-tac  $x=R$  in ballE)
      by auto
    qed
  qed
qed
qed
qed

```

```

lemma depthSubst[simp]:
  fixes  $P :: pi$ 
  and  $a :: name$ 
  and  $b :: name$ 

  shows  $\text{depth}(P[a::=b]) = \text{depth } P$ 

```

by(*nominal-induct P avoiding: a b rule: pi.strong-inducts, auto*)

lemma *depthTransition*:

fixes $P :: pi$
and $a :: name$
and $b :: name$
and $P' :: pi$

assumes $Phnf: hnf P$

shows $P \mapsto a[b] \prec P' \implies depth P' < depth P$
and $P \mapsto a\langle x \rangle \prec P' \implies depth P' < depth P$
and $P \mapsto \tau \prec P' \implies depth P' < depth P$
and $P \mapsto a\langle \nu x \rangle \prec P' \implies depth P' < depth P$

proof –

assume $P \mapsto a[b] \prec P'$
thus $depth P' < depth P$ **using** *assms*
by(*nominal-induct rule: outputInduct, auto*)

next

assume $Trans: P \mapsto a\langle x \rangle \prec P'$
have $Goal: \bigwedge P a x P'. \llbracket P \mapsto a\langle x \rangle \prec P'; x \# P; hnf P \rrbracket \implies depth P' < depth P$

proof –

fix $P a x P'$
assume $P \mapsto a\langle x \rangle \prec P'$ **and** $x \# P$ **and** $hnf P$
thus $depth P' < depth P$
by(*nominal-induct rule: inputInduct, auto*)

qed

obtain $y :: name$ **where** $yFreshP: y \# P$ **and** $yFreshP': y \# P'$

by(*rule-tac name-exists-fresh[of (P, P')], auto simp add: fresh-prod*)

from $yFreshP'$ $Trans$ **have** $P \mapsto a\langle y \rangle \prec [(x, y)] \cdot P'$ **by**(*simp add: alphaBound-Residual*)

hence $depth([(x, y)] \cdot P') < depth P$ **using** $yFreshP$ $Phnf$ **by**(*rule Goal*)

thus $depth P' < depth P$ **by** *simp*

next

assume $P \mapsto \tau \prec P'$
thus $depth P' < depth P$ **using** *assms*
by(*nominal-induct rule: tauInduct, auto simp add: uhnf-def*)

next

assume $Trans: P \mapsto a\langle \nu x \rangle \prec P'$
have $Goal: \bigwedge P a x P'. \llbracket P \mapsto a\langle \nu x \rangle \prec P'; x \# P; hnf P \rrbracket \implies depth P' < depth P$

proof –

fix $P a x P'$
assume $P \mapsto a\langle \nu x \rangle \prec P'$ **and** $x \# P$ **and** $hnf P$
thus $depth P' < depth P$
by(*nominal-induct rule: boundOutputInduct,*
auto elim: outputCases simp add: residual.inject)

qed

```

obtain  $y::name$  where  $yFreshP: y \# P$  and  $yFreshP': y \# P'$ 
  by(rule-tac name-exists-fresh[of  $(P, P')$ ], auto simp add: fresh-prod)
  from  $yFreshP'$  Trans have  $P \mapsto a \langle \nu y \rangle \prec [(x, y)] \cdot P'$  by(simp add: alphaBoundResidual)
  hence  $depth\ [(x, y)] \cdot P' < depth\ P$  using  $yFreshP\ Phnf$  by(rule Goal)
  thus  $depth\ P' < depth\ P$  by simp
qed

```

lemma *maxExpandDepth*:

```

fixes  $P :: pi$ 
and  $Q :: pi$ 
and  $R :: pi$ 

```

```

assumes  $R \in expandSet\ P\ Q$ 
and  $hnf\ P$ 
and  $hnf\ Q$ 

```

```

shows  $depth\ R \leq depth(P \parallel Q)$ 
using assms
apply(auto simp add: expandSet-def summandTransition[THEN sym] dest: depthTransition)
apply(subgoal-tac a  $\neq$  x)
apply(simp add: summandTransition[THEN sym])
apply(force dest: depthTransition)
apply blast
apply(subgoal-tac a  $\neq$  x)
apply(simp add: summandTransition[THEN sym])
apply(force dest: depthTransition)
apply blast
apply(force dest: depthTransition)
apply(force dest: depthTransition)
apply(subgoal-tac a  $\neq$  y)
apply(simp add: summandTransition[THEN sym])
apply(force dest: depthTransition)
apply blast
apply(subgoal-tac a  $\neq$  y)
apply(simp add: summandTransition[THEN sym])
apply(force dest: depthTransition)
by blast

```

lemma *expandDepth'*:

```

fixes  $P :: pi$ 
and  $Q :: pi$ 

```

```

assumes  $Phnf: hnf\ P$ 
and  $Qhnf: hnf\ Q$ 

```

```

shows  $\exists R. (R, expandSet\ P\ Q) \in sumComposeSet \wedge depth\ R \leq depth(P \parallel Q)$ 
proof(case-tac expandSet P Q = {})

```

```

assume expandSet  $P \ Q = \{\}$ 
with Phnf Qhnf show ?thesis by(auto intro: sumComposeSet.empty)
next
assume expandSet  $P \ Q \neq \{\}$ 

moreover from Phnf Qhnf finiteExpand obtain  $R$  where  $TSC: (R, \text{expandSet } P \ Q) \in \text{sumComposeSet}$ 
by(blast dest: expandComplete)
ultimately obtain  $T$  where  $T \in \text{expandSet } P \ Q$ 
and  $\text{depth } R \leq \text{depth } T$ 
by(blast dest: expandDepth)
with Phnf Qhnf have  $\text{depth } R \leq \text{depth}(P \parallel Q)$ 
by(force dest: maxExpandDepth)
with  $TSC$  show ?thesis by blast
qed

lemma validToHnf:
fixes  $P :: pi$ 

assumes valid P

shows  $\exists Q. \text{uhnf } Q \wedge \text{valid } Q \wedge Q \equiv_e P \wedge (\text{depth } Q) \leq (\text{depth } P)$ 
proof –
have MatchGoal:  $\bigwedge a \ b \ P \ Q. \llbracket \text{uhnf } Q; \text{valid } Q; Q \equiv_e P; \text{depth } Q \leq \text{depth } P \rrbracket \implies$ 
 $\exists Q. \text{uhnf } Q \wedge \text{valid } Q \wedge Q \equiv_e [a \frown b]P \wedge \text{depth } Q \leq \text{depth } ([a \frown b]P)$ 
proof –
fix  $a \ b \ P \ Q$ 
assume Qhnf: uhnf Q and validQ: valid Q and QeqP:  $Q \equiv_e P$ 
and QPdepth:  $\text{depth } Q \leq \text{depth } P$ 
show  $\exists Q. \text{uhnf } Q \wedge \text{valid } Q \wedge Q \equiv_e [a \frown b]P \wedge \text{depth } Q \leq \text{depth } ([a \frown b]P)$ 
proof(case-tac a = b)
assume  $a = b$ 
with QeqP have  $Q \equiv_e [a \frown b]P$  by(blast intro: Sym Trans equiv.Match)
with Qhnf validQ QPdepth show ?thesis by force
next
assume  $a \neq b$ 
hence  $0 \equiv_e [a \frown b]P$  by(blast intro: Sym Match')
moreover have uhnf 0 by(simp add: uhnf-def)
ultimately show ?thesis by force
qed
qed

from assms show ?thesis
proof(nominal-induct P rule: pi.strong-inducts)
case PiNil
have uhnf 0 by(simp add: uhnf-def)
moreover have valid 0 by simp
moreover have  $0 \equiv_e 0$  by(rule Refl)

```

```

moreover have (depth 0) ≤ (depth 0) by simp
ultimately show ?case by blast
next
  case(Output a b P)
  have uhnf (a{b}.P) by(simp add: uhnf-def)
  moreover have valid(a{b}.P) by fact
  moreover have a{b}.P ≡e a{b}.P by(rule Refl)
  moreover have (depth (a{b}.P)) ≤ (depth (a{b}.P)) by simp
  ultimately show ?case by blast
next
  case(Tau P)
  have uhnf (τ.(P)) by(simp add: uhnf-def)
  moreover have valid (τ.(P)) by fact
  moreover have τ.(P) ≡e τ.(P) by(rule Refl)
  moreover have (depth (τ.(P))) ≤ (depth (τ.(P))) by simp
  ultimately show ?case by blast
next
  case(Input a x P)
  have uhnf (a<x>.P) by(simp add: uhnf-def)
  moreover have valid (a<x>.P) by fact
  moreover have a<x>.P ≡e a<x>.P by(rule Refl)
  moreover have (depth (a<x>.P)) ≤ (depth (a<x>.P)) by simp
  ultimately show ?case by blast
next
  case(Match a b P)
  have valid ([a¬b]P) by fact
  hence valid P by simp
  moreover have valid P ⇒ ∃ Q. uhnf Q ∧ valid Q ∧ Q ≡e P ∧ depth Q ≤
depth P by fact
  ultimately obtain Q where Qhnf: uhnf Q and validQ: valid Q and QeqP:
  Q ≡e P
  and QPdepth: depth Q ≤ depth P by blast
  thus ?case by(rule MatchGoal)
next
  case(Mismatch a b P)
  have valid ([a≠b]P) by fact
  hence valid P by simp
  moreover have valid P ⇒ ∃ Q. uhnf Q ∧ valid Q ∧ Q ≡e P ∧ depth Q ≤
depth P by fact
  ultimately obtain Q where Qhnf: uhnf Q and validQ: valid Q and QeqP:
  Q ≡e P
  and QPdepth: depth Q ≤ depth P by blast
show ?case
proof(case-tac a = b)
  assume a = b
  hence 0 ≡e [a≠b]P by(blast intro: Sym Mismatch')
  moreover have uhnf 0 by(simp add: uhnf-def)
  ultimately show ?case by force
next

```

```

    assume  $a \neq b$ 
    with  $QeqP$  have  $Q \equiv_e [a \neq b]P$  by(blast intro: Sym Trans equiv.Mismatch)
    with  $Qhnf\ validQ\ QPdepth$  show ?case by force
  qed
next
case(Sum P Q)
have  $valid(P \oplus Q)$  by fact
hence  $validP: valid\ P$  and  $validQ: valid\ Q$  by simp+

have  $\exists P'. uhnf\ P' \wedge valid\ P' \wedge P' \equiv_e P \wedge (depth\ P') \leq (depth\ P)$ 
proof -
  have  $valid\ P \implies \exists P'. uhnf\ P' \wedge valid\ P' \wedge P' \equiv_e P \wedge (depth\ P') \leq (depth\ P)$ 
  by fact
  with  $validP$  show ?thesis by simp
qed
then obtain  $P'$  where  $P'hnf: uhnf\ P'$  and  $P'eqP: P' \equiv_e P$  and  $validP': valid\ P'$ 
and  $P'depth: (depth\ P') \leq (depth\ P)$  by blast

have  $\exists Q'. uhnf\ Q' \wedge valid\ Q' \wedge Q' \equiv_e Q \wedge (depth\ Q') \leq (depth\ Q)$ 
proof -
  have  $valid\ Q \implies \exists Q'. uhnf\ Q' \wedge valid\ Q' \wedge Q' \equiv_e Q \wedge (depth\ Q') \leq (depth\ Q)$ 
  by fact
  with  $validQ$  show ?thesis by simp
qed

then obtain  $Q'$  where  $Q'hnf: uhnf\ Q'$  and  $Q'eqQ: Q' \equiv_e Q$  and  $validQ': valid\ Q'$ 
and  $Q'depth: (depth\ Q') \leq (depth\ Q)$  by blast

from  $P'hnf\ Q'hnf\ validP'\ validQ'$  obtain  $R$  where  $Rhnf: uhnf\ R$  and  $validR: valid\ R$ 
and  $P'Q'eqR: P' \oplus Q' \equiv_e R$ 
and  $Rdepth: depth\ R \leq depth(P' \oplus Q')$ 
apply(drule-tac uhnfSum) apply assumption+ by blast

from  $validP'\ validQ'$  have  $valid(P' \oplus Q')$  by simp
from  $P'eqP\ Q'eqQ\ P'Q'eqR$  have  $P \oplus Q \equiv_e R$  by(blast intro: Sym SumPres' Trans)
moreover from  $Rdepth\ P'depth\ Q'depth$  have  $depth\ R \leq depth(P \oplus Q)$  by auto
ultimately show ?case using  $validR\ Rhnf$  by(blast intro: Sym)
next
case(Par P Q)
have  $valid(P \parallel Q)$  by fact

hence  $validP: valid\ P$  and  $validQ: valid\ Q$  by simp+
have  $\exists P'. uhnf\ P' \wedge valid\ P' \wedge P' \equiv_e P \wedge (depth\ P') \leq (depth\ P)$ 
proof -

```

have $\text{valid } P \implies \exists P'. \text{uhnfnf } P' \wedge \text{valid } P' \wedge P' \equiv_e P \wedge (\text{depth } P') \leq (\text{depth } P)$ **by fact**
with $\text{valid}P$ **show** *?thesis* **by simp**
qed
then obtain P' **where** $P'\text{hnfnf}: \text{uhnfnf } P'$ **and** $P'\text{eq}P: P' \equiv_e P$ **and** $\text{valid}P': \text{valid } P'$
and $P'\text{depth}: (\text{depth } P') \leq (\text{depth } P)$ **by blast**

have $\exists Q'. \text{uhnfnf } Q' \wedge \text{valid } Q' \wedge Q' \equiv_e Q \wedge (\text{depth } Q') \leq (\text{depth } Q)$
proof –
have $\text{valid } Q \implies \exists Q'. \text{uhnfnf } Q' \wedge \text{valid } Q' \wedge Q' \equiv_e Q \wedge (\text{depth } Q') \leq (\text{depth } Q)$ **by fact**
with $\text{valid}Q$ **show** *?thesis* **by simp**
qed

then obtain Q' **where** $Q'\text{hnfnf}: \text{uhnfnf } Q'$ **and** $Q'\text{eq}Q: Q' \equiv_e Q$ **and** $\text{valid}Q': \text{valid } Q'$
and $Q'\text{depth}: (\text{depth } Q') \leq (\text{depth } Q)$ **by blast**

from $P'\text{hnfnf } Q'\text{hnfnf}$ **obtain** R **where** $\text{Exp}: (R, \text{expandSet } P' Q') \in \text{sumCompos-eSet}$ **and** $R\text{depth}: \text{depth } R \leq \text{depth}(P' \parallel Q')$
by(*force dest: expandDepth' simp add: uhnfnf-def*)

from $\text{Exp } P'\text{hnfnf } Q'\text{hnfnf}$ **have** $P'Q'\text{eq}R: P' \parallel Q' \equiv_e R$ **by**(*force intro: Expand simp add: uhnfnf-def*)
from $P'\text{hnfnf } Q'\text{hnfnf } \text{valid}P' \text{valid}Q'$ **have** $\forall P \in (\text{expandSet } P' Q'). \text{uhnfnf } P \wedge \text{valid } P$ **by**(*blast dest: validExpand*)
with Exp **obtain** R' **where** $R'\text{hnfnf}: \text{uhnfnf } R'$ **and** $\text{valid}R': \text{valid } R'$
and $\text{Req}R': R \equiv_e R'$
and $R'\text{depth}: \text{depth } R' \leq \text{depth } R$
by(*blast dest: expandHnfnf*)
from $P'\text{eq}P \ Q'\text{eq}Q \ P'Q'\text{eq}R \ \text{Req}R'$ **have** $P \parallel Q \equiv_e R'$ **by**(*blast intro: Sym ParPres Trans*)
moreover from $R\text{depth } P'\text{depth } Q'\text{depth } R'\text{depth}$ **have** $\text{depth } R' \leq \text{depth}(P \parallel Q)$ **by auto**
ultimately show *?case* **using** $\text{valid}R' \ R'\text{hnfnf}$ **by**(*blast dest: Sym*)
next
case(*Res x P*)
have $\text{valid } (<\nu x>P)$ **by fact**
hence $\text{valid}P: \text{valid } P$ **by simp**
moreover have $\text{valid } P \implies \exists Q. \text{uhnfnf } Q \wedge \text{valid } Q \wedge Q \equiv_e P \wedge \text{depth } Q \leq \text{depth } P$ **by fact**
ultimately obtain Q **where** $Q\text{hnfnf}: \text{uhnfnf } Q$ **and** $\text{valid}Q: \text{valid } Q$ **and** $Q\text{eq}P: Q \equiv_e P$
and $QP\text{Depth}: \text{depth } Q \leq \text{depth } P$ **by blast**

from $\text{valid}P$ **show** *?case*
proof(*nominal-induct P avoiding: x rule: pi.strong-inducts*)
case $PiNil$

```

have uhnf 0 by(simp add: uhnf-def)
moreover have valid 0 by simp
moreover have 0  $\equiv_e$   $\langle \nu x \rangle 0$ 
proof -
  have  $x \# 0$  by simp
  thus ?thesis by(blast intro: Sym ResFresh)
qed
moreover have depth 0  $\leq$  depth  $\langle \nu x \rangle 0$  by simp
ultimately show ?case by blast
next
case(Output a b P)
have valid(a{b}.P) by fact
hence validP: valid P by simp
show ?case
proof(case-tac x=a)
  assume x = a
  moreover have uhnf 0 by(simp add: uhnf-def)
  moreover have valid 0 by simp
  moreover have 0  $\equiv_e$   $\langle \nu x \rangle x\{b\}.P$  by(blast intro: ResOutput' Sym)
  moreover have depth 0  $\leq$  depth $\langle \nu x \rangle x\{b\}.P$  by simp
  ultimately show ?case by blast
next
assume xineqa:  $x \neq a$ 
show ?case
proof(case-tac x=b)
  assume x=b
  moreover from xineqa have uhnf $\langle \nu x \rangle a\{x\}.P$  by(force simp add:
uhnf-def)
  moreover from validP have valid $\langle \nu x \rangle a\{x\}.P$  by simp
  moreover have  $\langle \nu x \rangle a\{x\}.P \equiv_e \langle \nu x \rangle a\{x\}.P$  by(rule Refl)
  moreover have depth $\langle \nu x \rangle a\{x\}.P \leq$  depth $\langle \nu x \rangle a\{x\}.P$  by simp
  ultimately show ?case by blast
next
assume xineqb:  $x \neq b$ 
have uhnf(a{b}. $\langle \nu x \rangle P$ ) by(simp add: uhnf-def)
moreover from validP have valid(a{b}. $\langle \nu x \rangle P$ ) by simp
moreover from xineqa xineqb have a{b}. $\langle \nu x \rangle P \equiv_e \langle \nu x \rangle a\{b\}.P$ 
by(blast intro: ResOutput Sym)
moreover have depth(a{b}. $\langle \nu x \rangle P$ )  $\leq$  depth $\langle \nu x \rangle a\{b\}.P$  by simp
ultimately show ?case by blast
qed
qed
next
case(Tau P)
have valid( $\tau.(P)$ ) by fact
hence validP: valid P by simp

have uhnf( $\tau.\langle \nu x \rangle P$ ) by(simp add: uhnf-def)
moreover from validP have valid( $\tau.\langle \nu x \rangle P$ ) by simp

```

moreover have $\tau.\langle \nu x \rangle P \equiv_e \langle \nu x \rangle \tau.(P)$ **by** (*blast intro: ResTau Sym*)
moreover have $\text{depth}(\tau.\langle \nu x \rangle P) \leq \text{depth}(\langle \nu x \rangle \tau.(P))$ **by** *simp*
ultimately show *?case* **by** *blast*

next
case (*Input a y P*)
have $\text{valid}(a \langle y \rangle . P)$ **by** *fact*
hence $\text{valid}P$: $\text{valid } P$ **by** *simp*
have $y \# x$ **by** *fact* **hence** $y \text{ineq}x$: $y \neq x$ **by** *simp*
show *?case*
proof (*case-tac x=a*)
assume $x = a$
moreover have $\text{uhn}f \ 0$ **by** (*simp add: uhnf-def*)
moreover have $\text{valid } 0$ **by** *simp*
moreover have $0 \equiv_e \langle \nu x \rangle x \langle y \rangle . P$ **by** (*blast intro: ResInput' Sym*)
moreover have $\text{depth } 0 \leq \text{depth}(\langle \nu x \rangle x \langle y \rangle . P)$ **by** *simp*
ultimately show *?case* **by** *blast*

next
assume $x \text{ineq}a$: $x \neq a$
have $\text{uhn}f(a \langle y \rangle . \langle \nu x \rangle P)$ **by** (*simp add: uhnf-def*)
moreover from $\text{valid}P$ **have** $\text{valid}(a \langle y \rangle . \langle \nu x \rangle P)$ **by** *simp*
moreover from $x \text{ineq}a$ $y \text{ineq}x$ **have** $a \langle y \rangle . \langle \nu x \rangle P \equiv_e \langle \nu x \rangle a \langle y \rangle . P$
by (*blast intro: ResInput Sym*)
moreover have $\text{depth}(a \langle y \rangle . \langle \nu x \rangle P) \leq \text{depth}(\langle \nu x \rangle a \langle y \rangle . P)$ **by** *simp*
ultimately show *?case* **by** *blast*

qed

next
case (*Match a b P x*)
have $\text{valid}([a \frown b]P)$ **by** *fact* **hence** $\text{valid } P$ **by** *simp*
moreover have $\bigwedge x. \text{valid } P \implies \exists Q. \text{uhn}f \ Q \wedge \text{valid } Q \wedge Q \equiv_e \langle \nu x \rangle P \wedge$
 $\text{depth } Q \leq \text{depth}(\langle \nu x \rangle P)$
by *fact*
ultimately obtain Q **where** $Q \text{hn}f$: $\text{uhn}f \ Q$ **and** $\text{valid}Q$: $\text{valid } Q$
and $Q \text{eq}P$: $Q \equiv_e \langle \nu x \rangle P$
and $QP \text{depth}$: $\text{depth } Q \leq \text{depth}(\langle \nu x \rangle P)$
by *blast*
show *?case*
proof (*case-tac a = b*)
assume $a = b$
moreover have $Q \equiv_e \langle \nu x \rangle [a \frown a]P$
proof –
have $P \equiv_e [a \frown a]P$ **by** (*blast intro: equiv.Match Sym*)
hence $\langle \nu x \rangle P \equiv_e \langle \nu x \rangle [a \frown a]P$ **by** (*rule ResPres*)
with $Q \text{eq}P$ **show** *?thesis* **by** (*blast intro: Trans*)
qed
moreover from $QP \text{depth}$ **have** $\text{depth } Q \leq \text{depth}(\langle \nu x \rangle [a \frown a]P)$ **by** *simp*
ultimately show *?case* **using** $Q \text{hn}f$ $\text{valid}Q$ **by** *blast*

next
assume $a \text{ineq}b$: $a \neq b$
have $\text{uhn}f \ 0$ **by** (*simp add: uhnf-def*)

```

moreover have valid 0 by simp
moreover have  $0 \equiv_e \langle \nu x \rangle [a \frown b] P$ 
proof –
  from aineqb have  $0 \equiv_e [a \frown b] P$  by(blast intro: Match' Sym)
  hence  $\langle \nu x \rangle 0 \equiv_e \langle \nu x \rangle [a \frown b] P$  by(rule ResPres)
  thus ?thesis by(blast intro: ResNil Trans Sym)
qed
moreover have depth 0  $\leq$  depth( $\langle \nu x \rangle [a \frown b] P$ ) by simp
ultimately show ?case by blast
qed
next
case(Mismatch a b P x)
have valid( $[a \neq b] P$ ) by fact hence valid P by simp
moreover have  $\bigwedge x. \text{valid } P \implies \exists Q. \text{uhnf } Q \wedge \text{valid } Q \wedge Q \equiv_e \langle \nu x \rangle P \wedge$ 
   $\text{depth } Q \leq \text{depth}(\langle \nu x \rangle P)$ 
  by fact
ultimately obtain Q where Qhnf: uhnf Q and validQ: valid Q
  and QeqP:  $Q \equiv_e \langle \nu x \rangle P$ 
  and QPdepth:  $\text{depth } Q \leq \text{depth}(\langle \nu x \rangle P)$ 
  by blast
show ?case
proof(case-tac a = b)
  assume  $a = b$ 
  moreover have uhnf 0 by(simp add: uhnf-def)
  moreover have valid 0 by simp
  moreover have  $0 \equiv_e \langle \nu x \rangle [a \neq a] P$ 
  proof –
    have  $0 \equiv_e [a \neq a] P$  by(blast intro: Mismatch' Sym)
    hence  $\langle \nu x \rangle 0 \equiv_e \langle \nu x \rangle [a \neq a] P$  by(rule ResPres)
    thus ?thesis by(blast intro: ResNil Trans Sym)
  qed
  moreover have depth 0  $\leq$  depth( $\langle \nu x \rangle [a \neq a] P$ ) by simp
  ultimately show ?case by blast
next
assume aineqb:  $a \neq b$ 
have  $Q \equiv_e \langle \nu x \rangle [a \neq b] P$ 
proof –
  from aineqb have  $P \equiv_e [a \neq b] P$  by(blast intro: equiv.Mismatch Sym)
  hence  $\langle \nu x \rangle P \equiv_e \langle \nu x \rangle [a \neq b] P$  by(rule ResPres)
  with QeqP show ?thesis by(blast intro: Trans)
qed
moreover from QPdepth have depth Q  $\leq$  depth( $\langle \nu x \rangle [a \neq b] P$ ) by simp
ultimately show ?case using Qhnf validQ by blast
qed
next
case(Sum P Q x)
have valid( $P \oplus Q$ ) by fact
hence validP: valid P and validQ: valid Q by simp+

```

have $\exists P'. \text{uhn}f P' \wedge \text{valid} P' \wedge P' \equiv_e \langle \nu x \rangle P \wedge (\text{depth} P') \leq (\text{depth}(\langle \nu x \rangle P))$
proof –
have $\text{valid} P \implies \exists P'. \text{uhn}f P' \wedge \text{valid} P' \wedge P' \equiv_e \langle \nu x \rangle P \wedge (\text{depth} P') \leq$
 $(\text{depth}(\langle \nu x \rangle P))$ **by fact**
with $\text{valid}P$ **show** $?thesis$ **by simp**
qed
then obtain P' **where** $P'hnf: \text{uhn}f P'$ **and** $P'eqP: P' \equiv_e \langle \nu x \rangle P$ **and**
 $\text{valid}P': \text{valid} P'$
and $P'depth: (\text{depth} P') \leq (\text{depth}(\langle \nu x \rangle P))$ **by blast**

have $\exists Q'. \text{uhn}f Q' \wedge \text{valid} Q' \wedge Q' \equiv_e \langle \nu x \rangle Q \wedge (\text{depth} Q') \leq (\text{depth}(\langle \nu x \rangle Q))$
proof –
have $\text{valid} Q \implies \exists Q'. \text{uhn}f Q' \wedge \text{valid} Q' \wedge Q' \equiv_e \langle \nu x \rangle Q \wedge (\text{depth} Q')$
 $\leq (\text{depth}(\langle \nu x \rangle Q))$ **by fact**
with $\text{valid}Q$ **show** $?thesis$ **by simp**
qed

then obtain Q' **where** $Q'hnf: \text{uhn}f Q'$ **and** $Q'eqQ: Q' \equiv_e \langle \nu x \rangle Q$ **and**
 $\text{valid}Q': \text{valid} Q'$
and $Q'depth: (\text{depth} Q') \leq (\text{depth}(\langle \nu x \rangle Q))$ **by blast**

from $P'hnf$ $Q'hnf$ $\text{valid}P'$ $\text{valid}Q'$ **obtain** R **where** $Rhnf: \text{uhn}f R$ **and** $\text{valid}R:$
 $\text{valid} R$
and $P'Q'eqR: P' \oplus Q' \equiv_e R$
and $Rdepth: \text{depth} R \leq \text{depth}(P' \oplus Q')$
apply(drule-tac uhnfSum) **apply** assumption+ **by blast**

from $P'eqP$ $Q'eqQ$ $P'Q'eqR$ **have** $\langle \nu x \rangle (P \oplus Q) \equiv_e R$ **by**(blast intro: Sym
 $\text{SumPres' SumRes Trans}$)
moreover from $Rdepth$ $P'depth$ $Q'depth$ **have** $\text{depth} R \leq \text{depth}(\langle \nu x \rangle (P \oplus$
 $Q))$ **by auto**
ultimately show $?case$ **using** $\text{valid}R$ $Rhnf$ **by**(blast intro: Sym)
next
case($\text{Par } P \ Q \ x$)
have $\text{valid}(P \parallel Q)$ **by fact**

hence $\text{valid}P: \text{valid} P$ **and** $\text{valid}Q: \text{valid} Q$ **by simp+**
have $\exists P'. \text{uhn}f P' \wedge \text{valid} P' \wedge P' \equiv_e P \wedge (\text{depth} P') \leq (\text{depth} P)$
proof –
obtain $x::\text{name}$ **where** $xFreshP: x \# P$ **by**($\text{rule name-exists-fresh}$)
moreover have $\bigwedge x. \text{valid} P \implies \exists P'. \text{uhn}f P' \wedge \text{valid} P' \wedge P' \equiv_e (\langle \nu x \rangle P)$
 $\wedge (\text{depth} P') \leq (\text{depth}(\langle \nu x \rangle P))$ **by fact**
with $\text{valid}P$ **obtain** P' **where** $\text{uhn}f P'$ **and** $\text{valid} P'$ **and** $P'eqP: P' \equiv_e$
 $(\langle \nu x \rangle P)$ **and** $P'depth: (\text{depth} P') \leq (\text{depth}(\langle \nu x \rangle P))$ **by blast**
moreover from $xFreshP$ $P'eqP$ **have** $P' \equiv_e P$ **by**($\text{blast intro: Trans}$
 ResFresh)
moreover with $P'depth$ **have** $\text{depth} P' \leq \text{depth} P$ **by simp**
ultimately show $?thesis$ **by blast**
qed

then obtain P' **where** $P'hnf: uhnf\ P'$ **and** $P'eqP: P' \equiv_e P$ **and** $validP':$
 $valid\ P'$

and $P'depth: (depth\ P') \leq (depth\ P)$ **by** *blast*

have $\exists Q'. uhnf\ Q' \wedge valid\ Q' \wedge Q' \equiv_e Q \wedge (depth\ Q') \leq (depth\ Q)$

proof –

obtain $x::name$ **where** $xFreshQ: x \# Q$ **by**(*rule name-exists-fresh*)

moreover have $\bigwedge x. valid\ Q \implies \exists Q'. uhnf\ Q' \wedge valid\ Q' \wedge Q' \equiv_e \langle \nu x \rangle Q$
 $\wedge (depth\ Q') \leq (depth\ \langle \nu x \rangle Q)$ **by** *fact*

with $validQ$ **obtain** Q' **where** $uhnf\ Q'$ **and** $valid\ Q'$ **and** $Q'eqQ: Q' \equiv_e$
 $\langle \nu x \rangle Q$ **and** $Q'depth: (depth\ Q') \leq (depth\ \langle \nu x \rangle Q)$ **by** *blast*

moreover from $xFreshQ\ Q'eqQ$ **have** $Q' \equiv_e Q$ **by**(*blast intro: Trans*
ResFresh)

moreover with $Q'depth$ **have** $depth\ Q' \leq depth\ Q$ **by** *simp*

ultimately show *?thesis* **by** *blast*

qed

then obtain Q' **where** $Q'hnf: uhnf\ Q'$ **and** $Q'eqQ: Q' \equiv_e Q$ **and** $validQ':$
 $valid\ Q'$

and $Q'depth: (depth\ Q') \leq (depth\ Q)$ **by** *blast*

from $P'hnf\ Q'hnf$ **obtain** R **where** $Exp: (R, expandSet\ P'\ Q') \in sumCom-$
 $poseSet$ **and** $Rdepth: depth\ R \leq depth(P' \parallel Q')$

by(*force dest: expandDepth' simp add: uhnf-def*)

from $Exp\ P'hnf\ Q'hnf$ **have** $P'Q'eqR: P' \parallel Q' \equiv_e R$ **by**(*force intro: Expand*
simp add: uhnf-def)

from $P'hnf\ Q'hnf\ validP'\ validQ'$ **have** $\forall P \in (expandSet\ P'\ Q'). uhnf\ P \wedge$
 $valid\ P$ **by**(*blast dest: validExpand*)

with Exp **obtain** R' **where** $R'hnf: uhnf\ R'$ **and** $validR': valid\ R'$

and $ReqR': R \equiv_e R'$

and $R'depth: depth\ R' \leq depth\ R$

by(*blast dest: expandHnf*)

from $P'eqP\ Q'eqQ\ P'Q'eqR\ ReqR'$ **have** $P \parallel Q \equiv_e R'$ **by**(*blast intro: Sym*
ParPres Trans)

hence $ResTrans: \langle \nu x \rangle (P \parallel Q) \equiv_e \langle \nu x \rangle R'$ **by**(*rule ResPres*)

from $validR'\ R'hnf$ **obtain** R'' **where** $R''hnf: uhnf\ R''$ **and** $validR'': valid$
 R'' **and** $R'eqR'': \langle \nu x \rangle R' \equiv_e R''$ **and** $R''depth: depth\ R'' \leq depth\ \langle \nu x \rangle R'$

by(*force dest: uhnfRes*)

from $ResTrans\ R'eqR''$ **have** $\langle \nu x \rangle (P \parallel Q) \equiv_e R''$ **by**(*rule Trans*)

moreover from $Rdepth\ P'depth\ Q'depth\ R'depth\ R''depth$ **have** $depth\ R'' \leq$
 $depth\ \langle \nu x \rangle (P \parallel Q)$ **by** *auto*

ultimately show *?case* **using** $validR''\ R''hnf$ **by**(*blast dest: Sym*)

next

case($Res\ y\ P\ x$)

have $valid\ \langle \nu y \rangle P$ **by** *fact* **hence** $valid\ P$ **by** *simp*

moreover have $\bigwedge x. valid\ P \implies \exists Q. uhnf\ Q \wedge valid\ Q \wedge Q \equiv_e \langle \nu x \rangle P \wedge$

```

depth Q ≤ depth(<νx>P)
  by fact
  ultimately obtain Q where Qhnf: uhnf Q and validQ: valid Q and QeqP:
Q ≡e <νy>P
      and QPdepth: depth Q ≤ depth(<νy>P) by blast

  from Qhnf validQ obtain Q' where Q'hnf: uhnf Q' and validQ': valid Q'
and QeqQ': <νx>Q ≡e Q'
      and Q'Qdepth: depth Q' ≤ depth(<νx>Q)
  by(force dest: uhnfRes)

  from QeqP have <νx>Q ≡e <νx><νy>P by(rule ResPres)
  with QeqQ' have Q' ≡e <νx><νy>P by(blast intro: Trans Sym)
  moreover from Q'Qdepth QPdepth have depth Q' ≤ depth(<νx><νy>P)
by simp
  ultimately show ?case using Q'hnf validQ' by blast
next
  case(Bang P x)
  have valid(!P) by fact
  hence False by simp
  thus ?case by simp
qed
next
  case(Bang P)
  have valid(!P) by fact
  hence False by simp
  thus ?case by simp
qed
qed

```

lemma *depthZero*:

```

fixes P :: pi

assumes depth P = 0
and      uhnf P

shows P = 0
using assms
apply(nominal-induct P rule: pi.strong-inducts, auto simp add: uhnf-def max-def
if-split)
apply(case-tac depth pi1 ≤ depth pi2)
by auto

```

lemma *completeAux*:

```

fixes n :: nat
and   P :: pi
and   Q :: pi

assumes depth P + depth Q ≤ n

```

```

and   valid P
and   valid Q
and   uhnf P
and   uhnf Q
and   P ~ Q

shows P ≡e Q
using assms
proof(induct n arbitrary: P Q rule: nat.induct)
  case(zero P Q)
    have depth P + depth Q ≤ 0 by fact
    hence Pdepth: depth P = 0 and Qdepth: depth Q = 0 by auto
    moreover have uhnf P and uhnf Q by fact+
    ultimately have P = 0 and Q = 0 by(blast intro: depthZero)+
    thus ?case by(blast intro: Refl)
next
  case(Suc n P Q)
    have validP: valid P and validQ: valid Q by fact+
    have Phnf: uhnf P and Qhnf: uhnf Q by fact+
    have PBisimQ: P ~ Q by fact
    have IH: ∧P Q. [[depth P + depth Q ≤ n; valid P; valid Q; uhnf P; uhnf Q; P
    ~ Q]] ⇒ P ≡e Q
    by fact
    have PQdepth: depth P + depth Q ≤ Suc n by fact

    have Goal: ∧P Q Q'. [[depth P + depth Q ≤ Suc n; valid P; valid Q; uhnf P;
    uhnf Q;
    
$$P \rightsquigarrow[\text{bisim}] Q; Q' \in \text{summands } Q]] \Rightarrow \exists P' \in \text{summands } P.$$

    Q' ≡e P'
    proof –
      fix P Q Q'
      assume PQdepth: depth P + depth Q ≤ Suc n
      assume validP: valid P and validQ: valid Q
      assume Phnf: uhnf P and Qhnf: uhnf Q
      assume PSimQ: P ~>[bisim] Q
      assume Q'inQ: Q' ∈ summands Q

      thus  $\exists P' \in \text{summands } P. Q' \equiv_e P'$  using PSimQ Phnf validP PQdepth
      proof(nominal-induct Q' avoiding: P rule: pi.strong-inducts)
        case PiNil
          have  $0 \in \text{summands } Q$  by fact
          hence False by(nominal-induct Q rule: pi.strong-inducts, auto simp add:
if-split)
          thus ?case by simp
        next
          case(Output a b Q' P)
            have validP: valid P and Phnf: uhnf P and PSimQ: P ~>[bisim] Q by fact+
            have PQdepth: depth P + depth Q ≤ Suc n by fact
            have a{b}.Q' ∈ summands Q by fact

```

with $Qhnf$ **have** $QTrans: Q \mapsto_a[b] \prec Q'$ **by**(*simp add: summandTransition uhnf-def*)
with $PSimQ$ **obtain** P' **where** $PTrans: P \mapsto_a[b] \prec P'$ **and** $P'BisimQ': P' \sim Q'$
by(*blast dest: simE*)

from $Phnf PTrans$ **have** $a\{b\}.P' \in \text{summands } P$ **by**(*simp add: summand-Transition uhnf-def*)
moreover have $P' \equiv_e Q'$
proof –
from $validP PTrans$ **have** $validP': valid P'$ **by**(*blast intro: validTransition*)
from $validQ QTrans$ **have** $validQ': valid Q'$ **by**(*blast intro: validTransition*)

from $validP'$ **obtain** P'' **where** $P''hnf: uhnf P''$ **and** $validP'': valid P''$
and $P''eqP': P'' \equiv_e P'$ **and** $P''depth: depth P'' \leq$
depth P'
by(*blast dest: validToHnf*)

from $validQ'$ **obtain** Q'' **where** $Q''hnf: uhnf Q''$ **and** $validQ'': valid Q''$
and $Q''eqQ': Q'' \equiv_e Q'$ **and** $Q''depth: depth Q'' \leq$
depth Q'
by(*blast dest: validToHnf*)

have $depth P'' + depth Q'' \leq n$
proof –
from $Phnf PTrans$ **have** $depth P' < depth P$
by(*force intro: depthTransition simp add: uhnf-def*)
moreover from $Qhnf QTrans$ **have** $depth Q' < depth Q$
by(*force intro: depthTransition simp add: uhnf-def*)
ultimately show *?thesis* **using** $PQdepth P''depth Q''depth$ **by** *simp*
qed

moreover have $P'' \sim Q''$
proof –
from $P''eqP'$ **have** $P'' \sim P'$ **by**(*rule sound*)
moreover from $Q''eqQ'$ **have** $Q'' \sim Q'$ **by**(*rule sound*)
ultimately show *?thesis* **using** $P'BisimQ'$ **by**(*blast dest: transitive symmetric*)
qed

ultimately have $P'' \equiv_e Q''$ **using** $validP'' validQ'' P''hnf Q''hnf$ **by**(*rule-tac IH*)
with $P''eqP' Q''eqQ'$ **show** *?thesis* **by**(*blast intro: Sym Trans*)
qed
ultimately show *?case* **by**(*blast intro: Sym equiv.OutputPres*)

next
case($\tau Q' P$)
have $validP: valid P$ **and** $Phnf: uhnf P$ **and** $PSimQ: P \rightsquigarrow[bisim] Q$ **by** *fact+*
have $PQdepth: depth P + depth Q \leq Suc n$ **by** *fact*
have $\tau.(Q') \in \text{summands } Q$ **by** *fact*

with $Qhnf$ **have** $QTrans: Q \mapsto_{\tau} \prec Q'$ **by**(*simp add: summandTransition uhnf-def*)
with $PSimQ$ **obtain** P' **where** $PTrans: P \mapsto_{\tau} \prec P'$ **and** $P'BisimQ': P' \sim Q'$
by(*blast dest: simE*)

from $Phnf PTrans$ **have** $\tau.(P') \in \text{summands } P$ **by**(*simp add: summandTransition uhnf-def*)
moreover **have** $P' \equiv_e Q'$
proof –
from $validP PTrans$ **have** $validP': valid P'$ **by**(*blast intro: validTransition*)
from $validQ QTrans$ **have** $validQ': valid Q'$ **by**(*blast intro: validTransition*)

from $validP'$ **obtain** P'' **where** $P''hnf: uhnf P''$ **and** $validP'': valid P''$
and $P''eqP': P'' \equiv_e P'$ **and** $P''depth: depth P'' \leq$
depth P'
by(*blast dest: validToHnf*)

from $validQ'$ **obtain** Q'' **where** $Q''hnf: uhnf Q''$ **and** $validQ'': valid Q''$
and $Q''eqQ': Q'' \equiv_e Q'$ **and** $Q''depth: depth Q'' \leq$
depth Q'
by(*blast dest: validToHnf*)

have $depth P'' + depth Q'' \leq n$
proof –
from $Phnf PTrans$ **have** $depth P' < depth P$
by(*force intro: depthTransition simp add: uhnf-def*)
moreover **from** $Qhnf QTrans$ **have** $depth Q' < depth Q$
by(*force intro: depthTransition simp add: uhnf-def*)
ultimately show *?thesis* **using** $PQdepth P''depth Q''depth$ **by** *simp*
qed

moreover **have** $P'' \sim Q''$
proof –
from $P''eqP'$ **have** $P'' \sim P'$ **by**(*rule sound*)
moreover **from** $Q''eqQ'$ **have** $Q'' \sim Q'$ **by**(*rule sound*)
ultimately show *?thesis* **using** $P'BisimQ'$ **by**(*blast dest: transitive symmetric*)
qed

ultimately have $P'' \equiv_e Q''$ **using** $validP'' validQ'' P''hnf Q''hnf$ **by**(*rule-tac IH*)

with $P''eqP' Q''eqQ'$ **show** *?thesis* **by**(*blast intro: Sym Trans*)
qed
ultimately show *?case* **by**(*blast intro: Sym equiv.TauPres*)

next
case(*Input a x Q' P*)
have $validP: valid P$ **and** $Phnf: uhnf P$ **and** $PSimQ: P \rightsquigarrow[bisim] Q$ **and**
 $xFreshP: x \# P$ **by** *fact+*
have $PQdepth: depth P + depth Q \leq Suc n$ **by** *fact*

```

have  $a\langle x \rangle.Q' \in \text{summands } Q$  by fact
with  $Q\text{hnf}$  have  $Q\text{Trans}: Q \mapsto a\langle x \rangle \prec Q'$  by(simp add: summandTransition
uhnf-def)
with  $PSimQ$   $x\text{Fresh}P$  obtain  $P'$  where  $P\text{Trans}: P \mapsto a\langle x \rangle \prec P'$ 
and  $P'\text{der}Q'$ : derivative  $P' Q' (\text{InputS } a) x$  bisim
by(blast dest: simE)

from  $Phnf$   $P\text{Trans}$  have  $a\langle x \rangle.P' \in \text{summands } P$  by(simp add: summand-
Transition uhnf-def)
moreover have  $\forall y \in \text{supp}(P', Q', x). P'[x::=y] \equiv_e Q'[x::=y]$ 
proof(rule ballI)
  fix  $y::\text{name}$ 
  assume  $y\text{supp}: y \in \text{supp}(P', Q', x)$ 
  have  $\text{valid}P'$ : valid( $P'[x::=y]$ )
  proof –
    from  $\text{valid}P$   $P\text{Trans}$  have  $\text{valid}P'$ : valid  $P'$  by(blast intro: validTransition)
    thus ?thesis by simp
  qed
  have  $\text{valid}Q'$ : valid( $Q'[x::=y]$ )
  proof –
    from  $\text{valid}Q$   $Q\text{Trans}$  have  $\text{valid}Q'$ : valid  $Q'$  by(blast intro: validTransition)
    thus ?thesis by simp
  qed

from  $\text{valid}P'$  obtain  $P''$  where  $P''\text{hnf}: \text{uhnf } P''$  and  $\text{valid}P''$ : valid  $P''$ 
and  $P''\text{eq}P'$ :  $P'' \equiv_e P'[x::=y]$  and  $P''\text{depth}: \text{depth } P''$ 
 $\leq \text{depth}(P'[x::=y])$ 
by(blast dest: validToHnf)

from  $\text{valid}Q'$  obtain  $Q''$  where  $Q''\text{hnf}: \text{uhnf } Q''$  and  $\text{valid}Q''$ : valid  $Q''$ 
and  $Q''\text{eq}Q'$ :  $Q'' \equiv_e Q'[x::=y]$  and  $Q''\text{depth}: \text{depth } Q''$ 
 $\leq \text{depth}(Q'[x::=y])$ 
by(blast dest: validToHnf)

have  $\text{depth } P'' + \text{depth } Q'' \leq n$ 
proof –
  from  $Phnf$   $P\text{Trans}$  have  $\text{depth } P' < \text{depth } P$ 
  by(force intro: depthTransition simp add: uhnf-def)
  moreover from  $Q\text{hnf}$   $Q\text{Trans}$  have  $\text{depth } Q' < \text{depth } Q$ 
  by(force intro: depthTransition simp add: uhnf-def)
  ultimately show ?thesis using  $PQ\text{depth } P''\text{depth } Q''\text{depth}$  by simp
qed

moreover have  $P'' \sim Q''$ 
proof –
  from  $P'\text{der}Q'$  have  $P'\text{Bisim}Q'$ :  $P'[x::=y] \sim Q'[x::=y]$ 
  by(auto simp add: derivative-def)
  from  $P''\text{eq}P'$  have  $P'' \sim P'[x::=y]$  by(rule sound)
  moreover from  $Q''\text{eq}Q'$  have  $Q'' \sim Q'[x::=y]$  by(rule sound)

```

```

      ultimately show ?thesis using P'BisimQ' by(blast dest: transitive
symmetric)
    qed
    ultimately have P'' ≡e Q'' using validP'' validQ'' P''hnf Q''hnf by(rule-tac
IH)
      with P''eqP' Q''eqQ' show P'[x::=y] ≡e Q'[x::=y] by(blast intro: Sym
Trans)
    qed

    ultimately show ?case
      apply -
      apply(rule-tac x=a<x>.P' in bexI)
      apply(rule equiv.InputPres)
      apply(rule ballI)
      apply(erule-tac x=y in ballE)
      apply(blast dest: Sym)
      by(auto simp add: supp-prod)
    next
      case(Match a b P' P)
      have [a↔b]P' ∈ summands Q by fact
      hence False by(nominal-induct Q rule: pi.strong-inducts, auto simp add:
if-split)
      thus ?case by simp
    next
      case(Mismatch a b P' P)
      have [a≠b]P' ∈ summands Q by fact
      hence False by(nominal-induct Q rule: pi.strong-inducts, auto simp add:
if-split)
      thus ?case by simp
    next
      case(Sum P' Q' P)
      have P' ⊕ Q' ∈ summands Q by fact
      hence False by(nominal-induct Q rule: pi.strong-inducts, auto simp add:
if-split)
      thus ?case by simp
    next
      case(Par P' Q' P)
      have P' ∥ Q' ∈ summands Q by fact
      hence False by(nominal-induct Q rule: pi.strong-inducts, auto simp add:
if-split)
      thus ?case by simp
    next
      case(Res x Q'' P)
      have xFreshP: x # P by fact
      have validP: valid P and Phnf: uhnf P and PSimQ: P ~>[bisim] Q by fact+
      have PQdepth: depth P + depth Q ≤ Suc n by fact
      have Q''summQ: <νx>Q'' ∈ summands Q by fact
      hence ∃ a Q'. a ≠ x ∧ Q'' = a{x}.Q'
      by(nominal-induct Q rule: pi.strong-inducts, auto simp add: if-split pi.inject

```

name-abs-eq name-calc)
then obtain a Q' **where** $a \text{ineq}x: a \neq x$ **and** $Q' \text{eq} Q'': Q'' = a\{x\}.Q'$
by *blast*
with $Q \text{hnf} Q' \text{summ} Q$ **have** $Q \text{Trans}: Q \mapsto a\langle \nu x \rangle \prec Q'$ **by** (*simp add: summandTransition uhnf-def*)
with $P \text{Sim} Q \text{xFresh} P$ **obtain** P' **where** $P \text{Trans}: P \mapsto a\langle \nu x \rangle \prec P'$ **and** $P' \text{Bisim} Q': P' \sim Q'$
by (*force dest: simE simp add: derivative-def*)

from $P \text{hnf} P \text{Trans} a \text{ineq}x$ **have** $(\langle \nu x \rangle a\{x\}.P') \in \text{summands } P$ **by** (*simp add: summandTransition uhnf-def*)
moreover have $a\{x\}.P' \equiv_e a\{x\}.Q'$
proof –
have $P' \equiv_e Q'$
proof –
from $\text{valid} P \text{Trans}$ **have** $\text{valid} P': \text{valid } P'$ **by** (*blast intro: validTransition*)
from $\text{valid} Q \text{Trans}$ **have** $\text{valid} Q': \text{valid } Q'$ **by** (*blast intro: validTransition*)

from $\text{valid} P'$ **obtain** P'' **where** $P'' \text{hnf}: \text{uhnf } P''$ **and** $\text{valid} P'': \text{valid } P''$
and $P'' \text{eq} P': P'' \equiv_e P'$ **and** $P'' \text{depth}: \text{depth } P'' \leq$
depth P'
by (*blast dest: validToHnf*)

from $\text{valid} Q'$ **obtain** Q'' **where** $Q'' \text{hnf}: \text{uhnf } Q''$ **and** $\text{valid} Q'': \text{valid } Q''$
and $Q'' \text{eq} Q': Q'' \equiv_e Q'$ **and** $Q'' \text{depth}: \text{depth } Q'' \leq$
depth Q'
by (*blast dest: validToHnf*)

have $\text{depth } P'' + \text{depth } Q'' \leq n$
proof –
from $P \text{hnf} P \text{Trans}$ **have** $\text{depth } P' < \text{depth } P$
by (*force intro: depthTransition simp add: uhnf-def*)
moreover from $Q \text{hnf} Q \text{Trans}$ **have** $\text{depth } Q' < \text{depth } Q$
by (*force intro: depthTransition simp add: uhnf-def*)
ultimately show *?thesis* **using** $PQ \text{depth } P'' \text{depth } Q'' \text{depth}$ **by** *simp qed*

moreover have $P'' \sim Q''$
proof –
from $P'' \text{eq} P'$ **have** $P'' \sim P'$ **by** (*rule sound*)
moreover from $Q'' \text{eq} Q'$ **have** $Q'' \sim Q'$ **by** (*rule sound*)
ultimately show *?thesis* **using** $P' \text{Bisim} Q'$ **by** (*blast dest: transitive symmetric*)
qed
ultimately have $P'' \equiv_e Q''$ **using** $\text{valid} P'' \text{ valid} Q'' P'' \text{hnf} Q'' \text{hnf}$
by (*rule-tac IH*)
with $P'' \text{eq} P' Q'' \text{eq} Q'$ **show** *?thesis* **by** (*blast intro: Sym Trans*)
qed
thus *?thesis* **by** (*rule OutputPres*)

```

qed
ultimately show ?case using Q'eqQ'' by(blast intro: Sym equiv.ResPres)
next
case(Bang P' P)
have !P' ∈ summands Q by fact
hence False by(nominal-induct Q rule: pi.strong-inducts, auto simp add:
if-split)
thus ?case by simp
qed
qed

from Phnf Qhnf PQdepth validP validQ PBisimQ show ?case
apply(rule-tac summandEquiv, auto)
apply(rule Goal)
apply auto
apply(blast dest: bisimE symmetric)
by(blast intro: Goal dest: bisimE)
qed

lemma complete:
fixes P :: pi
and Q :: pi

assumes validP: valid P
and validQ: valid Q
and PBisimQ: P ~ Q

shows P ≡e Q
proof -
from validP obtain P' where validP': valid P' and P'hnf: uhnf P' and P'eqP:
P' ≡e P
by(blast dest: validToHnf)
from validQ obtain Q' where validQ': valid Q' and Q'hnf: uhnf Q' and Q'eqQ:
Q' ≡e Q
by(blast dest: validToHnf)

have ∃n. depth P' + depth Q' ≤ n by auto
then obtain n where depth P' + depth Q' ≤ n by blast
moreover have P' ~ Q'
proof -
from P'eqP have P' ~ P by(rule sound)
moreover from Q'eqQ have Q' ~ Q by(rule sound)
ultimately show ?thesis using PBisimQ by(blast intro: symmetric transitive)
qed
ultimately have P' ≡e Q' using validP' validQ' P'hnf Q'hnf by(rule-tac completeAux)
with P'eqP Q'eqQ show ?thesis by(blast intro: Sym Trans)
qed

```

end

References

- [1] J. Bengtson. *Formalising process calculi*, volume 94. Uppsala Dissertations from the Faculty of Science and Technology, 2010.
- [2] J. Bengtson and J. Parrow. A completeness proof for bisimulation in the pi-calculus using isabelle. *Electr. Notes Theor. Comput. Sci.*, 192(1):61–75, 2007.
- [3] J. Bengtson and J. Parrow. Formalising the pi-calculus using nominal logic. *Logical Methods in Computer Science*, 5(2), 2009.