

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

<proof>

lemma *pt-swap*:

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

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

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

<proof>

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* ($\langle 0 \rangle$)

| *Output name name pi* ($\langle \{-\} \cdot \rightarrow [120, 120, 110] 110$)

| *Tau pi* ($\langle \tau \cdot \rightarrow [120] 110$)

| *Input name «name» pi* ($\langle \langle - \rangle \cdot \rightarrow [120, 120, 110] 110$)

| *Match name name pi* ($\langle \langle \sim \rangle \cdot \rightarrow [120, 120, 110] 110$)

| *Mismatch name name pi* ($\langle \langle \neq \rangle \cdot \rightarrow [120, 120, 110] 110$)

<i>Sum pi pi</i>	(infixr $\langle \oplus \rangle$ 90)
<i>Par pi pi</i>	(infixr $\langle \rangle$ 85)
<i>Res «name» pi</i>	($\langle \nu \rightarrow \rangle$ [100, 100] 100)
<i>Bang pi</i>	($\langle ! \rightarrow \rangle$ [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 $a \langle x \rangle . P = a \langle c \rangle . ([x, c] \cdot P)$

<proof>

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>

definition *subst-name* :: *name* \Rightarrow *name* \Rightarrow *name* \Rightarrow *name* ($\langle [- ::= -] \rangle$ [110, 110, 110] 110)

where

$a[b ::= c] \equiv$ if ($a = b$) then *c* else *a*

declare *subst-name-def*[simp]

lemma *subst-name-eqv*[eqvt]:

fixes *p* :: *name prm*
and *a* :: *name*
and *b* :: *name*
and *c* :: *name*

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

<proof>

nominal-primrec (*freshness-context*: (*c*::*name*, *d*::*name*))

subs :: *pi* \Rightarrow *name* \Rightarrow *name* \Rightarrow *pi* ($\langle [- ::= -] \rangle$ [100,100,100] 100)

where

$\mathbf{0}[c::=d] = \mathbf{0}$
| $\tau.(P)[c::=d] = \tau.(P[c::=d])$
| $a\{b\}.P[c::=d] = (a[c::=d])\{(b[c::=d])\}.(P[c::=d])$
| $\llbracket x \neq a; x \neq c; x \neq d \rrbracket \implies (a\langle x \rangle.P)[c::=d] = (a[c::=d])\langle x \rangle.(P[c::=d])$
| $\llbracket a \frown b \rrbracket P[c::=d] = \llbracket (a[c::=d]) \frown (b[c::=d]) \rrbracket (P[c::=d])$
| $\llbracket a \neq b \rrbracket P[c::=d] = \llbracket (a[c::=d]) \neq (b[c::=d]) \rrbracket (P[c::=d])$
| $(P \oplus Q)[c::=d] = (P[c::=d]) \oplus (Q[c::=d])$
| $(P \parallel Q)[c::=d] = (P[c::=d]) \parallel (Q[c::=d])$
| $\llbracket x \neq c; x \neq d \rrbracket \implies (\langle \nu x \rangle P)[c::=d] = \langle \nu x \rangle (P[c::=d])$
| $\llbracket !P[c::=d] \rrbracket = \llbracket !(P[c::=d]) \rrbracket$
<proof>

lemma *forget*:

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

assumes $a \# P$

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

<proof>

lemma *fresh-fact2*[*rule-format*]:

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

assumes $a \neq b$

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

<proof>

lemma *subst-identity*[*simp*]:

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

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

<proof>

lemma *renaming*:

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

assumes $c \# P$

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

$\langle proof \rangle$

lemma *fresh-fact1*:

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

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

shows $a \# P[b::=c]$

$\langle proof \rangle$

lemma *eqvt-subs*[*eqvt*]:

fixes $p :: name$ prm
and $P :: pi$
and $a :: name$
and $b :: name$

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

$\langle proof \rangle$

lemma *substInput*[*simp*]:

fixes $x :: name$
and $b :: name$
and $c :: name$
and $a :: name$
and $P :: pi$

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

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

$\langle proof \rangle$

lemma *injPermSubst*:

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

assumes $b \# P$

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

$\langle proof \rangle$

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>

lemma *freshRes*:

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

shows $a \# \langle \nu a \rangle P$
<proof>

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>

lemma *suppSubst*:

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

shows $\text{supp}(P[a::=b]) \subseteq \text{insert } b ((\text{supp } P) - \{a\})$
<proof>

primrec *seqSubs* :: $pi \Rightarrow (name \times name) list \Rightarrow pi$ ($\langle \cdot \rangle$ [100,100] 100)

where

$P[\langle [] \rangle] = P$
 $| P[\langle (x\#\sigma) \rangle] = (P[(fst\ x)::=(snd\ x)])[\langle \sigma \rangle]$

primrec *seq-subst-name* :: $name \Rightarrow (name \times name) list \Rightarrow name$ **where**

$seq\text{-subst-name } a [] = a$
 $| seq\text{-subst-name } a (x\#\sigma) = seq\text{-subst-name } (a[(fst\ x)::=(snd\ x)]) \sigma$

lemma *freshSeqSubstName*:

fixes $x :: name$
and $a :: name$

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

assumes $x \neq a$

and $x \# s$

shows $x \neq \text{seq-subst-name } a \ s$
<proof>

lemma *seqSubstZero[simp]*:

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

shows $\mathbf{0}[\langle \sigma \rangle] = \mathbf{0}$
<proof>

lemma *seqSubstTau[simp]*:

fixes $P :: \text{pi}$

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

shows $(\tau.(P))[\langle \sigma \rangle] = \tau.(P[\langle \sigma \rangle])$
<proof>

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 $(a\{b\}.P)[\langle \sigma \rangle] = (\text{seq-subst-name } a \ \sigma)\{(\text{seq-subst-name } b \ \sigma)\}.(P[\langle \sigma \rangle])$
<proof>

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 $(a\langle x \rangle.P)[\langle \sigma \rangle] = (\text{seq-subst-name } a \ \sigma)\langle x \rangle.(P[\langle \sigma \rangle])$
<proof>

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 $([a \frown b]P)[\langle \sigma \rangle] = [(\text{seq-subst-name } a \ \sigma) \frown (\text{seq-subst-name } b \ \sigma)](P[\langle \sigma \rangle])$

<proof>

lemma *seqSubstMismatch*[*simp*]:

fixes $a :: \textit{name}$
and $b :: \textit{name}$
and $P :: \textit{pi}$
and $\sigma :: (\textit{name} \times \textit{name}) \textit{list}$

shows $([a \neq b]P)[\langle \sigma \rangle] = [(seq\text{-}subst\text{-}name\ a\ \sigma) \neq (seq\text{-}subst\text{-}name\ b\ \sigma)](P[\langle \sigma \rangle])$
<proof>

lemma *seqSubstSum*[*simp*]:

fixes $P :: \textit{pi}$
and $Q :: \textit{pi}$
and $\sigma :: (\textit{name} \times \textit{name}) \textit{list}$

shows $(P \oplus Q)[\langle \sigma \rangle] = (P[\langle \sigma \rangle]) \oplus (Q[\langle \sigma \rangle])$
<proof>

lemma *seqSubstPar*[*simp*]:

fixes $P :: \textit{pi}$
and $Q :: \textit{pi}$
and $\sigma :: (\textit{name} \times \textit{name}) \textit{list}$

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

lemma *seqSubstRes*[*simp*]:

fixes $x :: \textit{name}$
and $P :: \textit{pi}$
and $\sigma :: (\textit{name} \times \textit{name}) \textit{list}$

assumes $x \# \sigma$

shows $(\langle \nu x \rangle P)[\langle \sigma \rangle] = \langle \nu x \rangle (P[\langle \sigma \rangle])$
<proof>

lemma *seqSubstBang*[*simp*]:

fixes $P :: \textit{pi}$
and $s :: (\textit{name} \times \textit{name}) \textit{list}$

shows $(!P)[\langle \sigma \rangle] = !(P[\langle \sigma \rangle])$
<proof>

lemma *seqSubstEqvt*[*eqvt*, *simp*]:

fixes $P :: \textit{pi}$
and $\sigma :: (\textit{name} \times \textit{name}) \textit{list}$
and $p :: \textit{name\ prm}$

shows $p \cdot (P[\langle\sigma\rangle]) = (p \cdot P)[\langle(p \cdot \sigma)\rangle]$
 $\langle proof \rangle$

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

shows $P[\langle(\sigma @ \sigma')\rangle] = (P[\langle\sigma\rangle])[\langle\sigma'\rangle]$
 $\langle proof \rangle$

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

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

shows $a \# P[\langle\sigma\rangle]$
 $\langle proof \rangle$

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 $a \langle x \rangle \prec P' \equiv ((InputS a) \langle x \rangle \prec P')$

abbreviation *Transitions-BoundOutputjudge* :: $name \Rightarrow name \Rightarrow pi \Rightarrow residual$
($\langle \cdot \rangle$ [80, 80, 80] 80)

where $a \langle \nu x \rangle \prec P' \equiv (BoundR (BoundOutputS a) x P')$

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

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: $\llbracket P \mapsto Rs \rrbracket \implies [b \frown b]P \mapsto Rs$
Mismatch: $\llbracket P \mapsto Rs; a \neq b \rrbracket \implies [a \neq b]P \mapsto Rs$

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: $\llbracket P \mapsto Rs \rrbracket \implies (P \oplus Q) \mapsto Rs$
Sum2: $\llbracket Q \mapsto Rs \rrbracket \implies (P \oplus Q) \mapsto Rs$

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

\langle proof \rangle

lemma *alphaBoundResidual:*

fixes $a :: subject$

and $x :: name$

and $P :: pi$

and $x' :: name$

assumes $A1: x' \# P$

shows $a\langle x \rangle \prec P = a\langle x' \rangle \prec ((x, x') \cdot P)$
<proof>

lemma *freshResidual*:
fixes $P :: pi$
and $Rs :: residual$
and $x :: name$

assumes $P \mapsto Rs$
and $x \# P$

shows $x \# Rs$
<proof>

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

shows $y \# a$
and $y \neq x \implies y \# P'$
<proof>

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 $y \# P'$
<proof>

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

assumes $b \# P$

shows $(P[a::=b])[b::=c] = P[a::=c]$
<proof>

lemma *Input*:
fixes $a :: name$
and $x :: name$

```

and P :: pi

shows a<x>.P  $\mapsto$  a<x> < P
<proof>

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«x» < P'
  and x # Q

  shows P || Q  $\mapsto$  a«x» < P' || Q
  <proof>

lemma Par2B:
  fixes Q :: pi
  and a :: subject
  and x :: name
  and Q' :: pi
  and P :: pi

  assumes QTrans: Q  $\mapsto$  a«x» < Q'
  and x # P

  shows P || Q  $\mapsto$  a«x» < P || Q'
  <proof>

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<x> < P'
  and QTrans: Q  $\mapsto$  a[b] < Q'

  shows P || Q  $\mapsto$   $\tau$  < P'[x::=b] || Q'
  <proof>

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

lemma *Close1*:
fixes $P :: pi$
and $a :: name$
and $x :: name$
and $P' :: pi$
and $Q :: pi$
and $y :: name$
and $Q' :: pi$

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

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

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

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 $\langle \nu y \rangle P \mapsto a\langle x \rangle \prec \langle \nu y \rangle P'$
 $\langle proof \rangle$

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 $\bigwedge P\ a\ b\ P'\ c\ C. \llbracket P \mapsto OutputR\ a\ b \prec P'; \bigwedge C. F\ C\ P\ a\ b\ P' \rrbracket \Longrightarrow F\ C$
 $([c \sim c]P)\ a\ b\ P'$
and $\bigwedge P\ a\ b\ P'\ c\ d\ C. \llbracket P \mapsto OutputR\ a\ b \prec P'; \bigwedge C. F\ C\ P\ a\ b\ P'; c \neq d \rrbracket$
 $\Longrightarrow F\ C\ ([c \neq d]P)\ a\ b\ P'$
and $\bigwedge P\ a\ b\ P'\ Q\ C. \llbracket P \mapsto OutputR\ a\ b \prec P'; \bigwedge C. F\ C\ P\ a\ b\ P' \rrbracket \Longrightarrow F\ C$
 $(P \oplus Q)\ a\ b\ P'$
and $\bigwedge Q\ a\ b\ Q'\ P\ C. \llbracket Q \mapsto OutputR\ a\ b \prec Q'; \bigwedge C. F\ C\ Q\ a\ b\ Q' \rrbracket \Longrightarrow F\ C$
 $(P \oplus Q)\ a\ b\ Q'$
and $\bigwedge P\ a\ b\ P'\ Q\ C. \llbracket P \mapsto OutputR\ a\ b \prec P'; \bigwedge C. F\ C\ P\ a\ b\ P' \rrbracket \Longrightarrow F\ C$
 $(P \parallel Q)\ a\ b\ (P' \parallel Q)$
and $\bigwedge Q\ a\ b\ Q'\ P\ C. \llbracket Q \mapsto OutputR\ a\ b \prec Q'; \bigwedge C. F\ C\ Q\ a\ b\ Q' \rrbracket \Longrightarrow F\ C$
 $C\ (P \parallel Q)\ a\ b\ (P \parallel Q')$
and $\bigwedge P\ a\ b\ P'\ x\ C. \llbracket P \mapsto OutputR\ a\ b \prec P'; x \neq a; x \neq b; x \# C; \bigwedge C. F\ C\ P\ a\ b\ P' \rrbracket \Longrightarrow$
 $F\ C\ (\langle \nu x \rangle P)\ a\ b\ (\langle \nu x \rangle P')$
and $\bigwedge P\ a\ b\ P'\ C. \llbracket P \parallel !P \mapsto OutputR\ a\ b \prec P'; \bigwedge C. F\ C\ (P \parallel !P)\ a\ b\ P' \rrbracket$
 $\Longrightarrow F\ C\ (!P)\ a\ b\ P'$

shows $F\ C\ P\ a\ b\ P'$
 $\langle proof \rangle$

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 $F :: ('a::fs\text{-name}) \Rightarrow pi \Rightarrow name \Rightarrow name \Rightarrow pi \Rightarrow bool$
and $C :: 'a::fs\text{-name}$

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 $cMatch: \bigwedge P a x P' b C. \llbracket P \mapsto a\langle x \rangle \prec P'; \bigwedge C. F C P a x P' \rrbracket \Rightarrow F C ([b \sim b]P) a x P'$
and $cMismatch: \bigwedge P a x P' b c C. \llbracket P \mapsto a\langle x \rangle \prec P'; \bigwedge C. F C P a x P'; b \neq c \rrbracket \Rightarrow F C ([b \neq c]P) a x P'$
and $cSum1: \bigwedge P Q a x P' C. \llbracket P \mapsto a\langle x \rangle \prec P'; \bigwedge C. F C P a x P' \rrbracket \Rightarrow F C (P \oplus Q) a x P'$
and $cSum2: \bigwedge P Q a x Q' C. \llbracket Q \mapsto a\langle x \rangle \prec Q'; \bigwedge C. F C Q a x Q' \rrbracket \Rightarrow F C (P \oplus Q) a x Q'$
and $cPar1B: \bigwedge P P' Q a x C. \llbracket P \mapsto a\langle x \rangle \prec P'; x \# P; x \# Q; x \neq a; \bigwedge C. F C P a x P' \rrbracket \Rightarrow F C (P \parallel Q) a x (P' \parallel Q)$
and $cPar2B: \bigwedge P Q Q' a x C. \llbracket Q \mapsto a\langle x \rangle \prec Q'; x \# P; x \# Q; x \neq a; \bigwedge C. F C Q a x Q' \rrbracket \Rightarrow F C (P \parallel Q) a x (P \parallel Q')$
and $cResB: \bigwedge P P' a x y C. \llbracket P \mapsto a\langle x \rangle \prec P'; y \neq a; y \neq x; y \# C; \bigwedge C. F C P a x P' \rrbracket \Rightarrow F C (\nu y \rangle P) a x (\nu y \rangle P')$
and $cBang: \bigwedge P a x P' C. \llbracket P \parallel !P \mapsto a\langle x \rangle \prec P'; \bigwedge C. F C (P \parallel !P) a x P' \rrbracket \Rightarrow F C (!P) a x P'$

shows $F C P a x P'$
 $\langle proof \rangle$

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 $F :: ('a::fs\text{-name}) \Rightarrow pi \Rightarrow name \Rightarrow name \Rightarrow pi \Rightarrow bool$
and $C :: 'a::fs\text{-name}$

assumes $a: P \mapsto a\langle \nu x \rangle \prec P'$
and $x \# P$
and $cMatch: \bigwedge P a x P' b C. \llbracket P \mapsto a\langle \nu x \rangle \prec P'; \bigwedge C. F C P a x P' \rrbracket \Rightarrow F C ([b \sim b]P) a x P'$
and $cMismatch: \bigwedge P a x P' b c C. \llbracket P \mapsto a\langle \nu x \rangle \prec P'; \bigwedge C. F C P a x P'; b \neq c \rrbracket \Rightarrow F C ([b \neq c]P) a x P'$
and $cOpen: \bigwedge P a x P' C. \llbracket P \mapsto (OutputR a x) \prec P'; a \neq x \rrbracket \Rightarrow F C (\langle \nu x \rangle P) a x P'$
and $cSum1: \bigwedge P Q a x P' C. \llbracket P \mapsto a\langle \nu x \rangle \prec P'; \bigwedge C. F C P a x P' \rrbracket \Rightarrow F C (P \oplus Q) a x P'$
and $cSum2: \bigwedge P Q a x Q' C. \llbracket Q \mapsto a\langle \nu x \rangle \prec Q'; \bigwedge C. F C Q a x Q' \rrbracket \Rightarrow F C (P \oplus Q) a x Q'$

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

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

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

$$F C (!P) a x P'$$
shows $F C P a x P'$
 $\langle proof \rangle$

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

assumes $Trans$: $P \mapsto \tau \prec P'$
and $\bigwedge P C. F C (\tau.(P)) P$
and $\bigwedge P P' c C. \llbracket P \mapsto \tau \prec P'; \bigwedge C. F C P P \rrbracket \Rightarrow F C ([c \sim c]P) P'$
and $\bigwedge P P' c d C. \llbracket P \mapsto \tau \prec P'; \bigwedge C. F C P P'; c \neq d \rrbracket \Rightarrow F C ([c \neq d]P) P'$

and $\bigwedge P P' Q C. \llbracket P \mapsto \tau \prec P'; \bigwedge C. F C P P \rrbracket \Rightarrow F C (P \oplus Q) P'$
and $\bigwedge Q Q' P C. \llbracket Q \mapsto \tau \prec Q'; \bigwedge C. F C Q Q \rrbracket \Rightarrow F C (P \oplus Q) Q'$
and $\bigwedge P P' Q C. \llbracket P \mapsto \tau \prec P'; \bigwedge C. F C P P \rrbracket \Rightarrow F C (P \parallel Q) (P' \parallel Q)$
and $\bigwedge Q Q' P C. \llbracket Q \mapsto \tau \prec Q'; \bigwedge C. F C Q Q \rrbracket \Rightarrow F C (P \parallel Q) (P \parallel Q')$
and $\bigwedge P a x P' Q b Q' C. \llbracket P \mapsto (BoundR (InputS a) x P'); Q \mapsto (OutputR a b \prec Q'; x \# P; x \# Q; x \# C) \rrbracket \Rightarrow F C (P \parallel Q) (P'[x::=b] \parallel Q')$

and $\bigwedge P a b P' Q x Q' C. \llbracket P \mapsto (OutputR a b \prec P'; Q \mapsto (BoundR (InputS a) x Q'); x \# P; x \# Q; x \# C) \rrbracket \Rightarrow F C (P \parallel Q) (P' \parallel Q'[x::=b])$
and $\bigwedge P a x P' Q y Q' C. \llbracket P \mapsto (BoundR (InputS a) x P'); Q \mapsto a \langle \nu y \rangle \prec Q'; x \# P; x \# Q; x \# C; y \# P; y \# Q; y \# C; x \neq y \rrbracket \Rightarrow F C (P \parallel Q) (\langle \nu y \rangle (P'[x::=y] \parallel Q'))$

and $\bigwedge P a y P' Q x Q' C. \llbracket P \mapsto a \langle \nu y \rangle \prec P'; Q \mapsto (BoundR (InputS a) x Q'); x \# P; x \# Q; x \# C; y \# P; y \# Q; y \# C; x \neq y \rrbracket \Rightarrow F C (P \parallel Q) (\langle \nu y \rangle (P' \parallel Q'[x::=y]))$

and $\bigwedge P P' x C. \llbracket P \mapsto \tau \prec P'; x \# C; \bigwedge C. F C P P \rrbracket \Rightarrow$

$$F C (\langle \nu x \rangle P) (\langle \nu x \rangle P')$$

and $\bigwedge P P' C. \llbracket P \parallel !P \mapsto \tau \prec P'; \bigwedge C. F C (P \parallel !P) P \rrbracket \Rightarrow F C (!P) P'$

shows $F C P P'$
 $\langle proof \rangle$

inductive \tauInduct $:: pi \Rightarrow pi \Rightarrow bool$

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 $Prop \alpha P'$

<proof>

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 $Prop \alpha P'$

<proof>

lemma $zeroTrans$ [*dest*]:

fixes $Rs :: residual$

```

assumes  $\mathbf{0} \mapsto Rs$ 

shows False
⟨proof⟩

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

  assumes  $\langle \nu x \rangle \mathbf{0} \mapsto R_s$ 

  shows False
⟨proof⟩

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

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

  shows False
⟨proof⟩

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

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

  shows False
⟨proof⟩

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

```

shows $Prop\ b\ y\ yP'$
<proof>

lemma $\tauBoundTrans[dest]$:

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

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

shows $False$
<proof>

lemma $\tauOutputTrans[dest]$:

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

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

shows $False$
<proof>

lemma $inputFreeTrans[dest]$:

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

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

shows $False$
<proof>

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 $False$
<proof>

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*

<proof>

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*

<proof>

lemma *outputIneqTrans*[*dest*]:

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

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

and $a \neq c \vee b \neq d$

shows *False*

<proof>

lemma *outputFreshTrans*[*dest*]:

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

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

and $a \# \alpha \vee b \# \alpha$

shows *False*
<proof>

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*
<proof>

lemma *resTauBoundTrans[dest]*:

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

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

shows *False*
<proof>

lemma *resTauOutputTrans[dest]*:

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

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

shows *False*
<proof>

lemma *resInputFreeTrans[dest]*:

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

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

shows *False*

\langle proof \rangle

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*

\langle proof \rangle

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*

\langle proof \rangle

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*

\langle proof \rangle

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  $False$ 
 $\langle proof \rangle$ 

lemma  $resTrans[dest]$ :
fixes  $x :: name$ 
and  $b :: name$ 
and  $Rs :: residual$ 
and  $y :: name$ 

shows  $\langle \nu x \rangle x\{b\}.P \mapsto Rs \implies False$ 
and  $\langle \nu x \rangle x\langle y \rangle.P \mapsto Rs \implies False$ 
 $\langle proof \rangle$ 

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

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

shows  $F a b$ 
 $\langle proof \rangle$ 

lemma  $mismatchCases[consumes 1, case-names cMismatch]$ :
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  $F a b$ 
 $\langle proof \rangle$ 

lemma  $sumCases[consumes 1, case-names cSum1 cSum2]$ :
fixes  $P :: pi$ 
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*
 $\langle proof \rangle$

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

assumes $b \# P$

shows $[a].P = [b].([a, b] \cdot P)$
 $\langle proof \rangle$

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'
 $\langle proof \rangle$

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*: $\bigwedge P'. \llbracket P \mapsto \alpha \prec P \rrbracket \implies F \alpha (P' \parallel Q)$
and *icPar2F*: $\bigwedge Q'. \llbracket Q \mapsto \alpha \prec Q \rrbracket \implies F \alpha (P \parallel Q')$
and *icComm1*: $\bigwedge P' Q' a b x. \llbracket P \mapsto a \langle x \rangle \prec P'; Q \mapsto a[b] \prec Q'; x \# P;$

$x \# Q; x \neq a; x \neq b; x \# Q'; x \# C; \alpha = \tau \implies F(\tau) (P'[x::=b] \parallel Q')$
and $icComm2: \bigwedge P' Q' a b x. \llbracket P \mapsto a[b] \prec P'; Q \mapsto a\langle x \rangle \prec Q'; x \# P; x \# Q; x \neq a; x \neq b; x \# P'; x \# C; \alpha = \tau \rrbracket \implies F(\tau) (P' \parallel Q'[x::=b])$
and $icClose1: \bigwedge P' Q' a x y. \llbracket P \mapsto a\langle x \rangle \prec P'; Q \mapsto a\langle \nu y \rangle \prec Q'; x \# P; x \# Q; x \neq a; x \neq y; x \# Q'; y \# P; y \# Q; y \neq a; y \# P'; x \# C; y \# C; \alpha = \tau \rrbracket$
 \implies
 $F(\tau) (\langle \nu y \rangle (P'[x::=y] \parallel Q'))$
and $icClose2: \bigwedge P' Q' a x y. \llbracket P \mapsto a\langle \nu y \rangle \prec P'; Q \mapsto a\langle x \rangle \prec Q'; x \# P; x \# Q; x \neq a; x \neq y; x \# P'; y \# P; y \# Q; y \neq a; y \# Q'; x \# C; y \# C; \alpha = \tau \rrbracket$
 \implies
 $F(\tau) (\langle \nu y \rangle (P' \parallel Q'[x::=y]))$

shows $F \alpha PQ'$
 $\langle proof \rangle$

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 $F xP'$
 $\langle proof \rangle$

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

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

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:  $\bigwedge a x P' C. \llbracket P \mapsto a\langle x \rangle \prec P'; x \# P; x \# C \rrbracket \implies F C (P \parallel !P) (a\langle x \rangle \prec P' \parallel !P)$ 
and cPar1F:  $\bigwedge \alpha P' C. \llbracket P \mapsto \alpha \prec P' \rrbracket \implies F C (P \parallel !P) (\alpha \prec P' \parallel !P)$ 
and cPar2B:  $\bigwedge a x P' C. \llbracket !P \mapsto a\langle x \rangle \prec P'; x \# P; x \# C; \bigwedge C. F C (!P) (a\langle x \rangle \prec P') \rrbracket \implies$ 

$$F C (P \parallel !P) (a\langle x \rangle \prec P \parallel P')$$

and cPar2F:  $\bigwedge \alpha P' C. \llbracket !P \mapsto \alpha \prec P'; \bigwedge C. F C (!P) (\alpha \prec P') \rrbracket \implies F C (P \parallel !P) (\alpha \prec P \parallel P')$ 
and cComm1:  $\bigwedge a x P' b P'' C. \llbracket P \mapsto a\langle x \rangle \prec P'; !P \mapsto (OutputR a b) \prec P''; x \# C; \bigwedge C. F C (!P) ((OutputR a b) \prec P'') \rrbracket \implies$ 

$$F C (P \parallel !P) (\tau \prec (P'[x::=b]) \parallel P'')$$

and cComm2:  $\bigwedge a b P' x P'' C. \llbracket P \mapsto (OutputR a b) \prec P'; !P \mapsto a\langle x \rangle \prec P''; x \# C; \bigwedge C. F C (!P) (a\langle x \rangle \prec P'') \rrbracket \implies$ 

$$F C (P \parallel !P) (\tau \prec P' \parallel (P''[x::=b]))$$

and cClose1:  $\bigwedge a x P' y P'' C. \llbracket P \mapsto a\langle x \rangle \prec P'; !P \mapsto a\langle \nu y \rangle \prec P''; y \# P; x \# C; y \# C; \bigwedge C. F C (!P) (a\langle \nu y \rangle \prec P'') \rrbracket \implies$ 

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

and cClose2:  $\bigwedge a y P' x P'' C. \llbracket P \mapsto a\langle \nu y \rangle \prec P'; !P \mapsto a\langle x \rangle \prec P''; y \# P; x \# C; y \# C; \bigwedge C. F C (!P) (a\langle x \rangle \prec P'') \rrbracket \implies$ 

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

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>

end

theory Late-Semantics1
imports Late-Semantics
begin

free-constructors case-subject for
  InputS
| BoundOutputS
<proof>

free-constructors case-freeRes for
  OutputR
| TauR

```

<proof>

end

theory *Rel*
 imports *Agent*
begin

definition *eqvt* :: $((\text{'a}::\text{pt-name}) \times (\text{'a}::\text{pt-name})) \text{ set} \Rightarrow \text{bool}$
 where $\text{eqvt } Rel \equiv (\forall x (\text{perm}::\text{name } prm). x \in Rel \longrightarrow \text{perm} \cdot x \in Rel)$

lemma *eqvtRelI*:
 fixes $Rel :: (\text{'a}::\text{pt-name} \times \text{'a}) \text{ set}$
 and $P :: \text{'a}$
 and $Q :: \text{'a}$
 and $\text{perm} :: \text{name } prm$

assumes *eqvt Rel*
 and $(P, Q) \in Rel$

shows $(\text{perm} \cdot P, \text{perm} \cdot Q) \in Rel$
<proof>

lemma *eqvtRelE*:
 fixes $Rel :: (\text{'a}::\text{pt-name} \times \text{'a}) \text{ set}$
 and $P :: \text{'a}$
 and $Q :: \text{'a}$
 and $\text{perm} :: \text{name } prm$

assumes *eqvt Rel*
 and $(\text{perm} \cdot P, \text{perm} \cdot Q) \in Rel$

shows $(P, Q) \in Rel$
<proof>

lemma *eqvtTrans[intro]*:
 fixes $Rel :: (\text{'a}::\text{pt-name} \times \text{'a}) \text{ set}$
 and $Rel' :: (\text{'a} \times \text{'a}) \text{ set}$

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

shows *eqvt (Rel O Rel')*
<proof>

lemma *eqvtUnion[intro]*:
 fixes $Rel :: (\text{'a}::\text{pt-name} \times \text{'a}) \text{ set}$
 and $Rel' :: (\text{'a} \times \text{'a}) \text{ set}$

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

shows *eqvt (Rel \cup Rel')*
<proof>

definition *substClosed* :: (*pi* \times *pi*) *set* \Rightarrow (*pi* \times *pi*) *set* **where**
substClosed Rel \equiv $\{(P, Q) \mid P \ Q. \forall \sigma. (P[\langle \sigma \rangle], Q[\langle \sigma \rangle]) \in Rel\}$

lemma *eqvtSubstClosed*:
fixes *Rel* :: (*pi* \times *pi*) *set*

assumes *eqvtRel*: *eqvt Rel*

shows *eqvt (substClosed Rel)*
<proof>

lemma *substClosedSubset*:
fixes *Rel* :: (*pi* \times *pi*) *set*

shows *substClosed Rel* \subseteq *Rel*
<proof>

lemma *partUnfold*:
fixes *P* :: *pi*
and *Q* :: *pi*
and σ :: (*name* \times *name*) *list*
and *Rel* :: (*pi* \times *pi*) *set*

assumes $(P, Q) \in$ *substClosed Rel*

shows $(P[\langle \sigma \rangle], Q[\langle \sigma \rangle]) \in$ *substClosed Rel*
<proof>

inductive-set *bangRel* :: (*pi* \times *pi*) *set* \Rightarrow (*pi* \times *pi*) *set*
for *Rel* :: (*pi* \times *pi*) *set*
where

BRBang: $(P, Q) \in Rel \Longrightarrow (!P, !Q) \in$ *bangRel Rel*
| *BRPar*: $(R, T) \in Rel \Longrightarrow (P, Q) \in (bangRel Rel) \Longrightarrow (R \parallel P, T \parallel Q) \in (bangRel Rel)$
| *BRRes*: $(P, Q) \in bangRel Rel \Longrightarrow (\langle \nu a \rangle P, \langle \nu a \rangle Q) \in bangRel Rel$

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⟩

lemma BRBangCases[consumes 1, case-names BRBang]:
  fixes P :: pi
  and Q :: pi
  and Rel :: (pi × pi) set
  and F :: pi ⇒ bool

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

  shows F P
⟨proof⟩

lemma BRParCases[consumes 1, case-names BRPar]:
  fixes P :: pi
  and Q :: pi
  and Rel :: (pi × pi) set
  and F :: pi ⇒ bool

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

  shows F P
⟨proof⟩

lemma bangRelSubset:
  fixes Rel :: (pi × pi) set
  and Rel' :: (pi × pi) set

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

  shows (P, Q) ∈ bangRel Rel'
⟨proof⟩

lemma bangRelSymetric:
  fixes P :: pi
  and Q :: pi
  and Rel :: (pi × pi) set

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

  shows (Q, P) ∈ bangRel Rel
⟨proof⟩

```

```

primrec resChain :: name list  $\Rightarrow$  pi  $\Rightarrow$  pi where
  base: resChain [] P = P
  | step: resChain (x#xs) P = <math>\nu x</math>(resChain xs P)

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

  shows perm  $\cdot$  (resChain lst P) = resChain (perm  $\cdot$  lst) (perm  $\cdot$  P)
  <math>\langle proof \rangle</math>

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

  assumes a  $\#$  (lst, P)

  shows a  $\#$  (resChain lst P)
  <math>\langle proof \rangle</math>

end

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

definition derivative :: pi  $\Rightarrow$  pi  $\Rightarrow$  subject  $\Rightarrow$  name  $\Rightarrow$  (pi  $\times$  pi) set  $\Rightarrow$  bool where
  derivative P Q a x Rel  $\equiv$  case a of InputS b  $\Rightarrow$  ( $\forall$  u. (P[x::=u], Q[x::=u])  $\in$  Rel)
  | BoundOutputS b  $\Rightarrow$  (P, Q)  $\in$  Rel

definition simulation :: pi  $\Rightarrow$  (pi  $\times$  pi) set  $\Rightarrow$  pi  $\Rightarrow$  bool ( $\prec$   $\rightsquigarrow$  [-]  $\rightarrow$  [80, 80, 80]
80) where
  P  $\rightsquigarrow$ [Rel] Q  $\equiv$  ( $\forall$  a x Q'. Q  $\mapsto$ a«x»  $\prec$  Q'  $\wedge$  x  $\#$  P  $\longrightarrow$  ( $\exists$  P'. P  $\mapsto$ a«x»  $\prec$  P'
 $\wedge$  derivative P' Q' a x Rel))  $\wedge$ 
  ( $\forall$   $\alpha$  Q'. Q  $\mapsto$  $\alpha$   $\prec$  Q'  $\longrightarrow$  ( $\exists$  P'. P  $\mapsto$  $\alpha$   $\prec$  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  $\rightsquigarrow$ [A] P'
  and A  $\subseteq$  B

  shows P  $\rightsquigarrow$ [B] P'

```

<proof>

lemma *derivativeMonotonic*:

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

assumes *derivative P Q a x A*
and $A \subseteq B$

shows *derivative P Q a x B*

<proof>

lemma *derivativeEqvtI*:

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

<proof>

lemma *derivativeEqvtI2*:

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

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

shows *derivative (perm · P) (perm · Q) a (perm · x) Rel*

<proof>

lemma *freshUnit[simp]*:

fixes $y :: name$

shows $y \# ()$

<proof>

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*: $\bigwedge a x Q'. \llbracket Q \mapsto a\langle x \rangle \prec Q'; x \# P; x \# Q; x \# a; x \# C \rrbracket \implies \exists P'. P \mapsto a\langle x \rangle \prec P' \wedge \text{derivative } P' Q' a x \text{ Rel}$

and *Free*: $\bigwedge \alpha Q'. Q \mapsto \alpha \prec Q' \implies \exists P'. P \mapsto \alpha \prec P' \wedge (P', Q') \in Rel$

shows $P \rightsquigarrow[Rel] Q$

<proof>

lemma *simCases*[*case-names Bound Free*]:

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

assumes *Bound*: $\bigwedge a y Q'. \llbracket Q \mapsto a\langle y \rangle \prec Q'; y \# P \rrbracket \implies \exists P'. P \mapsto a\langle y \rangle \prec P' \wedge \text{derivative } P' Q' a y \text{ Rel}$

and *Free*: $\bigwedge \alpha Q'. Q \mapsto \alpha \prec Q' \implies \exists P'. P \mapsto \alpha \prec P' \wedge (P', Q') \in Rel$

shows $P \rightsquigarrow[Rel] Q$

<proof>

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

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

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

<proof>

lemma *simE*:

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

and $a :: \text{subject}$
and $x :: \text{name}$
and $Q' :: \text{pi}$

assumes $P \rightsquigarrow[\text{Rel}] Q$

shows $Q \mapsto a\langle x \rangle \prec Q' \implies x \# P \implies \exists P'. P \mapsto a\langle x \rangle \prec P' \wedge (\text{derivative } P' Q' a x \text{ Rel})$
and $Q \mapsto \alpha \prec Q' \implies \exists P'. P \mapsto \alpha \prec P' \wedge (P', Q') \in \text{Rel}$
 $\langle \text{proof} \rangle$

lemma *eqvtI*:
fixes $P :: \text{pi}$
and $Q :: \text{pi}$
and $\text{Rel} :: (\text{pi} \times \text{pi}) \text{ set}$
and $\text{perm} :: \text{name prm}$

assumes $\text{Sim}: P \rightsquigarrow[\text{Rel}] Q$
and $\text{RelRel}': \text{Rel} \subseteq \text{Rel}'$
and $\text{EqvtRel}': \text{eqvt Rel}'$

shows $(\text{perm} \cdot P) \rightsquigarrow[\text{Rel}'] (\text{perm} \cdot Q)$
 $\langle \text{proof} \rangle$

lemma *derivativeReflexive*:
fixes $P :: \text{pi}$
and $a :: \text{subject}$
and $x :: \text{name}$
and $\text{Rel} :: (\text{pi} \times \text{pi}) \text{ set}$

assumes $\text{Id} \subseteq \text{Rel}$

shows $\text{derivative } P P a x \text{ Rel}$
 $\langle \text{proof} \rangle$

lemma *reflexive*:
fixes $P :: \text{pi}$
and $\text{Rel} :: (\text{pi} \times \text{pi}) \text{ set}$

assumes $\text{Id} \subseteq \text{Rel}$

shows $P \rightsquigarrow[\text{Rel}] P$
 $\langle \text{proof} \rangle$

lemma *transitive*:
fixes $P :: \text{pi}$
and $Q :: \text{pi}$

```

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

shows  $P \rightsquigarrow[Rel''] R$ 
<proof>

end

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

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

coinductive-set bisim ::  $(pi \times pi) \text{ set}$ 
where
  step:  $\llbracket P \rightsquigarrow[bisim] Q; (Q, P) \in bisim \rrbracket \implies (P, Q) \in bisim$ 
monos monoAux

abbreviation
  strongBisimJudge (infixr  $\rightsquigarrow$  65) where  $P \sim Q \equiv (P, Q) \in bisim$ 

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

lemma monotonic:  $\text{mono}(\lambda p \ x1 \ x2. \exists P \ Q. x1 = P \wedge x2 = Q \wedge P \rightsquigarrow[\{(xa, x). p \ xa \ x\}] Q \wedge Q \rightsquigarrow[\{(xa, x). p \ xa \ x\}] P)$ 
<proof>

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>

lemma bisimE:
fixes  $P :: pi$ 

```

and $Q :: pi$
assumes $P \sim Q$
shows $P \rightsquigarrow[bisim] Q$
 $\langle proof \rangle$

lemma *bisimI*:
fixes $P :: pi$
and $Q :: pi$
assumes $P \rightsquigarrow[bisim] Q$
and $Q \sim P$
shows $P \sim Q$
 $\langle proof \rangle$

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

lemma *oldBisimBisimEq*:
shows $(\bigcup \{Rel. (old-bisim\ Rel)\}) = bisim$ (**is** ?LHS = ?RHS)
 $\langle proof \rangle$

lemma *reflexive*:
fixes $P :: pi$
shows $P \sim P$
 $\langle proof \rangle$

lemma *symmetric*:
fixes $P :: pi$
and $Q :: pi$
assumes $P \sim Q$
shows $Q \sim P$
 $\langle proof \rangle$

lemma *bisimClosed*:
fixes $P :: pi$
and $Q :: pi$
and $p :: name\ prm$
assumes $P \sim Q$
shows $(p \cdot P) \sim (p \cdot Q)$
 $\langle proof \rangle$

lemma *bisimEqvt*[*simp*]:
 shows *eqvt bisim*
 ⟨*proof*⟩

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

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

shows $P \sim R$
 ⟨*proof*⟩

lemma *bisimTransitiveCoinduct*[*case-names cSim cSym, case-conclusion bisim step, consumes 2*]:

assumes $(P, Q) \in X$
 and *eqvt* 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*⟩

end

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

abbreviation

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

lemma *congBisim*:

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

assumes $P \sim^s Q$

shows $P \sim Q$
 ⟨*proof*⟩

lemma *eqvt*:

shows *eqvt* (*substClosed bisim*)
 ⟨*proof*⟩

lemma *eqClosed*:

```

fixes  $P :: pi$ 
and  $Q :: pi$ 
and  $perm :: name prm$ 

assumes  $P \sim^s Q$ 

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

lemma reflexive:
fixes  $P :: pi$ 

shows  $P \sim^s P$ 
 $\langle proof \rangle$ 

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

assumes  $P \sim^s Q$ 

shows  $Q \sim^s P$ 
 $\langle proof \rangle$ 

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

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

shows  $P \sim^s R$ 
 $\langle proof \rangle$ 

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⟩

lemma *inputPres*:

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

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

shows $a\langle x \rangle.P \rightsquigarrow_{[Rel]} a\langle x \rangle.Q$
 ⟨proof⟩

lemma *outputPres*:

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

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

shows $a\{b\}.P \rightsquigarrow_{[Rel]} a\{b\}.Q$
 ⟨proof⟩

lemma *matchPres*:

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

assumes $PSimQ: P \rightsquigarrow_{[Rel]} Q$
and $Rel \subseteq Rel'$

shows $[a \frown b]P \rightsquigarrow_{[Rel']} [a \frown b]Q$
 ⟨proof⟩

lemma *mismatchPres*:

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

and $Rel' :: (pi \times pi) \text{ set}$
assumes $PSimQ: P \rightsquigarrow[Rel] Q$
and $Rel \subseteq Rel'$
shows $[a \neq b]P \rightsquigarrow[Rel'] [a \neq b]Q$
 $\langle proof \rangle$

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 $P \oplus R \rightsquigarrow[Rel'] Q \oplus R$
 $\langle proof \rangle$

lemma $parCompose$:
fixes $P :: pi$
and $Q :: pi$
and $R :: pi$
and $T :: pi$
and $Rel :: (pi \times pi) \text{ set}$
and $Rel' :: (pi \times pi) \text{ set}$
and $Rel'' :: (pi \times pi) \text{ set}$
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 $Par: \bigwedge P Q R T. [(P, Q) \in Rel; (R, T) \in Rel'] \implies (P \parallel R, Q \parallel T) \in Rel''$
and $Res: \bigwedge P Q a. (P, Q) \in Rel'' \implies (\langle \nu a \rangle P, \langle \nu a \rangle Q) \in Rel''$
and $EqtRel: eqvt Rel$
and $EqtRel': eqvt Rel'$
and $EqtRel'': eqvt Rel''$
shows $P \parallel R \rightsquigarrow[Rel''] Q \parallel T$
 $\langle proof \rangle$

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 $P \parallel R \rightsquigarrow[Rel'] Q \parallel R$
 $\langle proof \rangle$

lemma $resDerivative:$

fixes $P :: pi$
and $Q :: pi$
and $a :: subject$
and $x :: name$
and $y :: name$
and $Rel :: (pi \times pi)$ set
and $Rel' :: (pi \times pi)$ 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 $derivative (\langle \nu y \rangle P) (\langle \nu y \rangle Q) a x Rel'$
 $\langle proof \rangle$

lemma $resPres:$

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

assumes $PSimQ: P \rightsquigarrow[Rel] Q$
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 $\langle \nu x \rangle P \rightsquigarrow[Rel'] \langle \nu x \rangle Q$
 $\langle proof \rangle$

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>

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>

end

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

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

assumes  $P \sim Q$ 

shows  $\tau.(P) \sim \tau.(Q)$ 
<proof>

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>

```

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>

lemma *matchPres*:

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

assumes $P \sim Q$

shows $[a\curvearrowright b]P \sim [a\curvearrowright b]Q$
<proof>

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>

lemma *sumPres*:

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

assumes $P \sim Q$

shows $P \oplus R \sim Q \oplus R$
<proof>

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>

lemma *parPres*:

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

assumes $P \sim Q$

shows $P \parallel R \sim Q \parallel R$
<proof>

lemma *bangPres*:

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

assumes *PBiSimQ*: $P \sim Q$

shows $!P \sim !Q$
<proof>

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 $\tau.(P) \sim^s \tau.(Q)$
<proof>

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>

lemma *outputPres*:
fixes $P :: pi$
and $Q :: pi$

assumes $P \sim^s Q$

shows $a\{b\}.P \sim^s a\{b\}.Q$
 $\langle proof \rangle$

lemma *matchPres*:
fixes $P :: pi$
and $Q :: pi$
and $a :: name$
and $b :: name$

assumes $P \sim^s Q$

shows $[a \frown b]P \sim^s [a \frown b]Q$
 $\langle proof \rangle$

lemma *mismatchPres*:
fixes $P :: pi$
and $Q :: pi$
and $a :: name$
and $b :: name$

assumes $P \sim^s Q$

shows $[a \neq b]P \sim^s [a \neq b]Q$
 $\langle proof \rangle$

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

assumes $P \sim^s Q$

shows $P \oplus R \sim^s Q \oplus R$
 $\langle proof \rangle$

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

assumes $P \sim^s Q$

shows $P \parallel R \sim^s Q \parallel R$
 $\langle proof \rangle$

lemma *resPres*:

fixes $P :: pi$
and $Q :: pi$
and $x :: name$

assumes $PeqQ: P \sim^s Q$

shows $\langle \nu x \rangle P \sim^s \langle \nu x \rangle Q$
 $\langle proof \rangle$

lemma *bangPres*:

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

assumes $P \sim^s Q$

shows $!P \sim^s !Q$
 $\langle proof \rangle$

end

theory *Late-Tau-Chain*

imports *Late-Semantics1*

begin

abbreviation *tauChain-judge* $:: pi \Rightarrow pi \Rightarrow bool$ ($\langle \cdot \Longrightarrow_{\tau} \cdot \rangle [80, 80]$ 80)
where $P \Longrightarrow_{\tau} P' \equiv (P, P') \in \{(P, P') \mid P P'. P \mapsto_{\tau} \prec P'\}^*$

lemma *singleTauChain*:

fixes $P :: pi$
and $P' :: pi$

assumes $P \mapsto_{\tau} \prec P'$

shows $P \Longrightarrow_{\tau} P'$
 $\langle proof \rangle$

lemma *tauChainAddTau[dest]*:

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 $P \mapsto_{\tau} \prec P' \Longrightarrow P' \Longrightarrow_{\tau} P'' \Longrightarrow P \Longrightarrow_{\tau} P''$
 $\langle proof \rangle$

lemma *tauChainInduct*[*consumes 1, case-names id ih*]:
fixes $P :: pi$
and $P' :: pi$

assumes $P \Longrightarrow_{\tau} P'$
and $F P$
and $\bigwedge P' P''. \llbracket P \Longrightarrow_{\tau} P'; P' \mapsto_{\tau} < P''; F P \rrbracket \Longrightarrow F P''$

shows $F P'$
 $\langle proof \rangle$

lemma *eqvtChainI*[*eqvt*]:
fixes $P :: pi$
and $P' :: pi$
and $perm :: name prm$

assumes $P \Longrightarrow_{\tau} P'$

shows $(perm \cdot P) \Longrightarrow_{\tau} (perm \cdot P')$
 $\langle proof \rangle$

lemma *eqvtChainE*:
fixes $perm :: name prm$
and $P :: pi$
and $P' :: pi$

assumes *Trans*: $(perm \cdot P) \Longrightarrow_{\tau} (perm \cdot P')$

shows $P \Longrightarrow_{\tau} P'$
 $\langle proof \rangle$

lemma *eqvtChainEq*:
fixes $P :: pi$
and $P' :: pi$
and $perm :: name prm$

shows $P \Longrightarrow_{\tau} P' = (perm \cdot P) \Longrightarrow_{\tau} (perm \cdot P')$
 $\langle proof \rangle$

lemma *freshChain*:
fixes $P :: pi$
and $P' :: pi$
and $x :: name$

assumes $P \Longrightarrow_{\tau} P'$
and $x \# P$

shows $x \# P'$

<proof>

lemma *matchChain*:

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

assumes $P \implies_{\tau} P'$
and $P \neq P'$

shows $[b \frown b]P \implies_{\tau} P'$
<proof>

lemma *mismatchChain*:

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

assumes $PChain: P \implies_{\tau} P'$
and $a \neq b$
and $P \neq P'$

shows $[a \neq b]P \implies_{\tau} P'$
<proof>

lemma *sum1Chain*[*rule-format*]:

fixes $P :: pi$
and $P' :: pi$
and $Q :: pi$

assumes $P \implies_{\tau} P'$
and $P \neq P'$

shows $P \oplus Q \implies_{\tau} P'$
<proof>

lemma *sum2Chain*[*rule-format*]:

fixes $P :: pi$
and $Q :: pi$
and $Q' :: pi$

assumes $Q \implies_{\tau} Q'$
and $Q \neq Q'$

shows $P \oplus Q \implies_{\tau} Q'$
<proof>

lemma *Par1Chain*:
fixes $P :: pi$
and $P' :: pi$
and $Q :: pi$

assumes $P \Longrightarrow_{\tau} P'$

shows $P \parallel Q \Longrightarrow_{\tau} P' \parallel Q$
 $\langle proof \rangle$

lemma *Par2Chain*:
fixes $P :: pi$
and $Q :: pi$
and $Q' :: pi$

assumes $Q \Longrightarrow_{\tau} Q'$

shows $P \parallel Q \Longrightarrow_{\tau} P \parallel Q'$
 $\langle proof \rangle$

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 $P \parallel Q \Longrightarrow_{\tau} P' \parallel Q'$
 $\langle proof \rangle$

lemma *ResChain*:
fixes $P :: pi$
and $P' :: pi$
and $a :: name$

assumes $P \Longrightarrow_{\tau} P'$

shows $\langle \nu a \rangle P \Longrightarrow_{\tau} \langle \nu a \rangle P'$
 $\langle proof \rangle$

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⟩

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 $!P \Longrightarrow_{\tau} P'$
 ⟨proof⟩

end

theory *Weak-Late-Step-Semantics*

imports *Late-Tau-Chain*

begin

definition *inputTransition* :: $pi \Rightarrow name \Rightarrow pi \Rightarrow name \Rightarrow name \Rightarrow pi \Rightarrow bool$ ($\langle - \Rightarrow_l - in \rightarrow \langle - \rangle \prec \rightarrow [80, 80, 80, 80, 80] 80$)
where $P \Rightarrow_l u in P'' \rightarrow a \langle x \rangle \prec P' \equiv \exists P''' . P \Longrightarrow_{\tau} P''' \wedge P''' \mapsto a \langle x \rangle \prec P'' \wedge P''[x::=u] \Longrightarrow_{\tau} P'$

definition *transition* :: $(pi \times LateSemantics.residual)$ set **where**
 $transition \equiv \{x. \exists P P' \alpha P'' P''' . P \Longrightarrow_{\tau} P' \wedge P' \mapsto \alpha \prec P'' \wedge P'' \Longrightarrow_{\tau} P''' \wedge x = (P, \alpha \prec P''')\} \cup$
 $\{x. \exists P P' a y P'' P''' . P \Longrightarrow_{\tau} P' \wedge (P' \mapsto (a \langle \nu y \rangle \prec P'')) \wedge P'' \Longrightarrow_{\tau} P''' \wedge x = (P, (a \langle \nu y \rangle \prec P'''))\}$

abbreviation *weakTransition-judge* :: $pi \Rightarrow LateSemantics.residual \Rightarrow bool$ ($\langle - \Rightarrow_l - \rightarrow [80, 80] 80$)

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*
 ⟨proof⟩

lemma *transitionI*:

fixes $P :: pi$
and $P''' :: pi$

and $\alpha :: \text{freeRes}$
and $P'' :: \text{pi}$
and $P' :: \text{pi}$
and $a :: \text{name}$
and $x :: \text{name}$
and $u :: \text{name}$

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 $\llbracket P \Rightarrow_{\tau} P''; P''' \mapsto a \langle x \rangle \prec P''; P''[x::=u] \Rightarrow_{\tau} P' \rrbracket \Rightarrow P \Rightarrow_l u \text{ in } P'' \rightarrow a \langle x \rangle \prec P'$
 $\langle \text{proof} \rangle$

lemma *transitionE*:

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

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 - \Rightarrow ?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 $\llbracket P \Rightarrow_l u \text{ in } P'' \rightarrow a \langle x \rangle \prec P' \rrbracket \Rightarrow \exists P'''. P \Rightarrow_{\tau} P''' \wedge P''' \mapsto a \langle x \rangle \prec P'' \wedge P''[x::=u] \Rightarrow_{\tau} P'$
 $\langle \text{proof} \rangle$

lemma *alphaInput*:

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

assumes *PTrans*: $P \Rightarrow_l u \text{ in } P'' \rightarrow a \langle x \rangle \prec P'$
and *yFreshP*: $y \# P$

shows $P \Rightarrow_l u \text{ in } ([x, y] \cdot P'') \rightarrow a \langle y \rangle \prec P'$
 $\langle \text{proof} \rangle$

lemma *tauActionChain*:

fixes $P :: \text{pi}$
and $P' :: \text{pi}$

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>

lemma *singleActionChain*:

fixes $P :: pi$
and $a :: name$
and $x :: name$
and $\alpha :: freeRes$
and $u :: name$

shows $P \mapsto_a \langle \nu x \rangle \prec P' \Longrightarrow P \Longrightarrow_l a \langle \nu x \rangle \prec P'$
and $\llbracket P \mapsto_a \langle x \rangle \prec P' \rrbracket \Longrightarrow P \Longrightarrow_l u \text{ in } P' \mapsto_a \langle x \rangle \prec P'[x ::= u]$
and $P \mapsto_\alpha \prec P' \Longrightarrow P \Longrightarrow_l \alpha \prec P'$
 <proof>

lemma *Tau*:

fixes $P :: pi$

shows $\tau.(P) \Longrightarrow_l \tau \prec P$
 <proof>

lemma *Input*:

fixes $a :: name$
and $x :: name$
and $u :: name$
and $P :: pi$

shows $a \langle x \rangle . P \Longrightarrow_l u \text{ in } P \mapsto_a \langle x \rangle \prec P[x ::= u]$
 <proof>

lemma *Output*:

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

shows $a\{b\}.P \Longrightarrow_l a[b] \prec P$
 <proof>

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 $P \Rightarrow_l u \text{ in } P'' \rightarrow b \langle x \rangle \prec P' \Rightarrow [a \frown a]P \Rightarrow_l u \text{ in } P'' \rightarrow b \langle x \rangle \prec P'$
 ⟨proof⟩

lemma *Mismatch*:

fixes $P :: pi$
and $R_s :: residual$
and $a :: name$
and $c :: name$
and $u :: name$
and $b :: name$
and $x :: name$
and $P' :: pi$

shows $\llbracket P \Rightarrow_l Rs; a \neq c \rrbracket \Rightarrow [a \neq c]P \Rightarrow_l Rs$
and $\llbracket P \Rightarrow_l u \text{ in } P'' \rightarrow b \langle x \rangle \prec P'; a \neq c \rrbracket \Rightarrow [a \neq c]P \Rightarrow_l u \text{ in } P'' \rightarrow b \langle x \rangle \prec P'$
 ⟨proof⟩

lemma *Open*:

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

assumes *Trans*: $P \Rightarrow_l a[b] \prec P'$
and *aInEqb*: $a \neq b$

shows $\langle \nu b \rangle P \Rightarrow_l a \langle \nu b \rangle \prec P'$
 ⟨proof⟩

lemma *Sum1*:

fixes $P :: pi$
and $R_s :: residual$
and $Q :: pi$
and $u :: name$
and $P'' :: pi$
and $a :: name$
and $x :: name$
and $P' :: pi$

shows $P \Rightarrow_l Rs \Rightarrow P \oplus Q \Rightarrow_l Rs$
and $P \Rightarrow_l u \text{ in } P'' \rightarrow a \langle x \rangle \prec P' \Rightarrow P \oplus Q \Rightarrow_l u \text{ in } P'' \rightarrow a \langle x \rangle \prec P'$
 ⟨proof⟩

lemma *Sum2*:

fixes $Q :: pi$
and $R_s :: residual$
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 $Q \Longrightarrow_{lu} in Q'' \rightarrow a \langle x \rangle \prec Q' \Longrightarrow P \oplus Q \Longrightarrow_{lu} in Q'' \rightarrow a \langle x \rangle \prec Q'$
<proof>

lemma *Par1B*:

fixes $P :: pi$
and $a :: name$
and $x :: name$
and $P' :: pi$
and $u :: name$
and $P'' :: pi$

shows $\llbracket P \Longrightarrow_{la} \langle \nu x \rangle \prec P'; x \# Q \rrbracket \Longrightarrow P \parallel Q \Longrightarrow_{la} \langle \nu x \rangle \prec (P' \parallel Q)$
and $\llbracket P \Longrightarrow_{lu} in P'' \rightarrow a \langle x \rangle \prec P'; x \# Q \rrbracket \Longrightarrow P \parallel Q \Longrightarrow_{lu} in (P'' \parallel Q) \rightarrow a \langle x \rangle \prec P' \parallel Q$
<proof>

lemma *Par1F*:

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

assumes *PTrans*: $P \Longrightarrow_l \alpha \prec P'$

shows $P \parallel Q \Longrightarrow_l \alpha \prec (P' \parallel Q)$
<proof>

lemma *Par2B*:

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

shows $Q \Longrightarrow_{la} \langle \nu x \rangle \prec Q' \Longrightarrow x \# P \Longrightarrow P \parallel Q \Longrightarrow_{la} \langle \nu x \rangle \prec (P \parallel Q')$
and $Q \Longrightarrow_{lu} in Q'' \rightarrow a \langle x \rangle \prec Q' \Longrightarrow x \# P \Longrightarrow P \parallel Q \Longrightarrow_{lu} in (P \parallel Q'') \rightarrow a \langle x \rangle \prec P \parallel Q'$
<proof>

lemma *Par2F*:

fixes $Q :: pi$

```

and  $\alpha :: \text{freeRes}$ 
and  $Q' :: \text{pi}$ 

assumes  $Q\text{Trans}: Q \Longrightarrow_l \alpha \prec Q'$ 

shows  $P \parallel Q \Longrightarrow_l \alpha \prec (P \parallel Q')$ 
 $\langle \text{proof} \rangle$ 

lemma Comm1:
fixes  $P :: \text{pi}$ 
and  $b :: \text{name}$ 
and  $P'' :: \text{pi}$ 
and  $a :: \text{name}$ 
and  $x :: \text{name}$ 
and  $P' :: \text{pi}$ 
and  $Q :: \text{pi}$ 
and  $Q' :: \text{pi}$ 

assumes  $P\text{Trans}: P \Longrightarrow_l b \text{ in } P'' \rightarrow a \langle x \rangle \prec P'$ 
and  $Q\text{Trans}: Q \Longrightarrow_l a[b] \prec Q'$ 

shows  $P \parallel Q \Longrightarrow_l \tau \prec P' \parallel Q'$ 
 $\langle \text{proof} \rangle$ 

lemma Comm2:
fixes  $P :: \text{pi}$ 
and  $a :: \text{name}$ 
and  $b :: \text{name}$ 
and  $P' :: \text{pi}$ 
and  $Q :: \text{pi}$ 
and  $x :: \text{name}$ 
and  $Q'' :: \text{pi}$ 
and  $Q' :: \text{pi}$ 

assumes  $P\text{Trans}: P \Longrightarrow_l a[b] \prec P'$ 
and  $Q\text{Trans}: Q \Longrightarrow_l b \text{ in } Q'' \rightarrow a \langle x \rangle \prec Q'$ 

shows  $P \parallel Q \Longrightarrow_l \tau \prec P' \parallel Q'$ 
 $\langle \text{proof} \rangle$ 

lemma Close1:
fixes  $P :: \text{pi}$ 
and  $y :: \text{name}$ 
and  $P'' :: \text{pi}$ 
and  $a :: \text{name}$ 
and  $x :: \text{name}$ 
and  $P' :: \text{pi}$ 
and  $Q :: \text{pi}$ 
and  $Q' :: \text{pi}$ 

```

assumes $PTrans: P \Rightarrow_{\iota} y \text{ in } P'' \rightarrow a \langle x \rangle \prec P'$
and $QTrans: Q \Rightarrow_{\iota} a \langle \nu y \rangle \prec Q'$
and $yFreshP: y \# P$
and $yFreshQ: y \# Q$

shows $P \parallel Q \Rightarrow_{\iota} \tau \prec \langle \nu y \rangle (P' \parallel Q')$
 $\langle proof \rangle$

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 $QTrans: Q \Rightarrow_{\iota} y \text{ in } Q'' \rightarrow a \langle x \rangle \prec Q'$
and $yFreshP: y \# P$
and $yFreshQ: y \# Q$

shows $P \parallel Q \Rightarrow_{\iota} \tau \prec \langle \nu y \rangle (P' \parallel Q')$
 $\langle proof \rangle$

lemma *ResF*:

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

assumes $PTrans: P \Rightarrow_{\iota} \alpha \prec P'$
and $xFreshAlpha: x \# \alpha$

shows $\langle \nu x \rangle P \Rightarrow_{\iota} \alpha \prec \langle \nu x \rangle P'$
 $\langle proof \rangle$

lemma *ResB*:

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

shows $\llbracket P \Rightarrow_{\iota} a \langle \nu x \rangle \prec P'; y \neq a; y \neq x; x \# P \rrbracket \Rightarrow \langle \nu y \rangle P \Rightarrow_{\iota} a \langle \nu x \rangle \prec$

$(\langle \nu y \rangle P')$
and $\llbracket P \Longrightarrow_l u \text{ in } P'' \rightarrow a \langle x \rangle \prec P'; y \neq a; y \neq x; y \neq u \rrbracket \Longrightarrow \langle \nu y \rangle P \Longrightarrow_l u \text{ in } (\langle \nu y \rangle P'') \rightarrow a \langle x \rangle \prec (\langle \nu y \rangle P')$
 $\langle \text{proof} \rangle$

lemma *Bang*:

fixes $P :: pi$
and $Rs :: residual$
and $u :: name$
and $P'' :: pi$
and $a :: name$
and $x :: name$
and $P' :: pi$

shows $P \parallel !P \Longrightarrow_l Rs \Longrightarrow !P \Longrightarrow_l Rs$
and $P \parallel !P \Longrightarrow_l u \text{ in } P'' \rightarrow a \langle x \rangle \prec P' \Longrightarrow !P \Longrightarrow_l u \text{ in } P'' \rightarrow a \langle x \rangle \prec P'$
 $\langle \text{proof} \rangle$

lemma *tauTransitionChain*:

fixes $P :: pi$
and $P' :: pi$

assumes $P \Longrightarrow_l \tau \prec P'$

shows $P \Longrightarrow_\tau P'$
 $\langle \text{proof} \rangle$

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 $P \Longrightarrow_\tau P' \Longrightarrow P' \Longrightarrow_l Rs \Longrightarrow P \Longrightarrow_l Rs$
and $P \Longrightarrow_\tau P'' \Longrightarrow P'' \Longrightarrow_l u \text{ in } P''' \rightarrow a \langle x \rangle \prec P' \Longrightarrow P \Longrightarrow_l u \text{ in } P''' \rightarrow a \langle x \rangle \prec P'$
and $P \Longrightarrow_l a \langle \nu x \rangle \prec P'' \Longrightarrow P'' \Longrightarrow_\tau P' \Longrightarrow x \# P \Longrightarrow P \Longrightarrow_l a \langle \nu x \rangle \prec P'$
and $P \Longrightarrow_l u \text{ in } P''' \rightarrow a \langle x \rangle \prec P'' \Longrightarrow P'' \Longrightarrow_\tau P' \Longrightarrow P \Longrightarrow_l u \text{ in } P''' \rightarrow a \langle x \rangle \prec P'$
and $P \Longrightarrow_l \alpha \prec P'' \Longrightarrow P'' \Longrightarrow_\tau P' \Longrightarrow P \Longrightarrow_l \alpha \prec P'$
 $\langle \text{proof} \rangle$

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 $c \# P'$
 $\langle proof \rangle$

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 $c \# P'$
 $\langle proof \rangle$

lemma *freshTauTransition:*

fixes $P :: pi$
and $c :: name$

assumes $PTrans: P \Longrightarrow_l \tau \prec P'$
and $cFreshP: c \# P$

shows $c \# P'$
 $\langle proof \rangle$

lemma *freshOutputTransition:*

fixes $P :: pi$
and $a :: name$
and $b :: name$
and $P' :: pi$
and $c :: name$

assumes $PTrans: P \Longrightarrow_l a[b] \prec P'$
and $cFreshP: c \# P$

shows $c \# P'$

<proof>

lemma *eqvtI*:

fixes $P :: pi$
and $Rs :: residual$
and $perm :: name\ prm$
and $u :: name$
and $P'' :: pi$
and $a :: name$
and $x :: name$
and $P' :: pi$

shows $P \Longrightarrow_l Rs \Longrightarrow (perm \cdot P) \Longrightarrow_l (perm \cdot Rs)$
and $P \Longrightarrow_l u\ in\ P'' \rightarrow a \langle x \rangle \prec P' \Longrightarrow (perm \cdot P) \Longrightarrow_l (perm \cdot u)\ in\ (perm \cdot P'') \rightarrow (perm \cdot a) \langle (perm \cdot x) \rangle \prec (perm \cdot P')$
<proof>

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* :: $pi \Rightarrow residual \Rightarrow bool$ ($\prec \Longrightarrow_l \hat{\ } \rightarrow [80, 80]$ 80)

where $P \Longrightarrow_l \hat{\ } Rs \equiv (P, Rs) \in weakTransition$

lemma *transitionI*:

fixes $P :: pi$
and $Rs :: residual$
and $P' :: pi$

shows $P \Longrightarrow_l Rs \Longrightarrow P \Longrightarrow_l \hat{\ } Rs$
and $P \Longrightarrow_l \hat{\ } \tau \prec P$
<proof>

lemma *transitionCases*[*consumes 1, case-names Step Stay*]:

fixes $P :: pi$
and $Rs :: residual$
and $P' :: pi$

assumes $P \Longrightarrow_i \hat{R}s$
and $P \Longrightarrow_l R_s \Longrightarrow F R_s$
and $R_s = \tau \prec P \Longrightarrow F (\tau \prec P)$

shows $F R_s$
 $\langle proof \rangle$

lemma *singleActionChain*:

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

assumes $P \mapsto \alpha \prec P'$

shows $P \Longrightarrow_i \hat{(\alpha \prec P')}$
 $\langle proof \rangle$

lemma *Tau*:

fixes $P :: pi$

shows $\tau.(P) \Longrightarrow_i \hat{\tau \prec P}$
 $\langle proof \rangle$

lemma *Output*:

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

shows $a\{b\}.P \Longrightarrow_i \hat{a[b] \prec P}$
 $\langle proof \rangle$

lemma *Match*:

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

shows $P \Longrightarrow_i \hat{b\langle \nu x \rangle \prec P'} \Longrightarrow [a \frown a]P \Longrightarrow_i \hat{b\langle \nu x \rangle \prec P'}$
and $P \Longrightarrow_i \hat{\alpha \prec P'} \Longrightarrow P \neq P' \Longrightarrow [a \frown a]P \Longrightarrow_i \hat{\alpha \prec P'}$
 $\langle proof \rangle$

lemma *Mismatch*:

fixes $a :: name$
and $c :: name$
and $P :: pi$
and $b :: name$
and $x :: name$

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

shows $\llbracket P \Rightarrow_l \hat{b} \langle \nu x \rangle \prec P'; a \neq c \rrbracket \Longrightarrow [a \neq c] P \Rightarrow_l \hat{b} \langle \nu x \rangle \prec P'$
and $P \Rightarrow_l \hat{\alpha} \prec P' \Longrightarrow P \neq P' \Longrightarrow a \neq c \Longrightarrow [a \neq c] P \Rightarrow_l \hat{\alpha} \prec P'$
 $\langle proof \rangle$

lemma *Open*:

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

assumes *Trans*: $P \Rightarrow_l \hat{a}[b] \prec P'$
and *aInEqb*: $a \neq b$

shows $\langle \nu b \rangle P \Rightarrow_l \hat{a} \langle \nu b \rangle \prec P'$
 $\langle proof \rangle$

lemma *Par1B*:

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

assumes *PTrans*: $P \Rightarrow_l \hat{a} \langle \nu x \rangle \prec P'$
and *xFreshQ*: $x \# Q$

shows $P \parallel Q \Rightarrow_l \hat{a} \langle \nu x \rangle \prec (P' \parallel Q)$
 $\langle proof \rangle$

lemma *Par1F*:

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

assumes *PTrans*: $P \Rightarrow_l \hat{\alpha} \prec P'$

shows $P \parallel Q \Rightarrow_l \hat{\alpha} \prec (P' \parallel Q)$
 $\langle proof \rangle$

lemma *Par2B*:

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

assumes *QTrans*: $Q \Rightarrow_l \hat{a} \langle \nu x \rangle \prec Q'$
and *xFreshP*: $x \# P$

shows $P \parallel Q \Longrightarrow_l \hat{a} \langle \nu x \rangle \prec (P \parallel Q')$
<proof>

lemma *Par2F*:

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

assumes *QTrans*: $Q \Longrightarrow_l \hat{\alpha} \prec Q'$

shows $P \parallel Q \Longrightarrow_l \hat{\alpha} \prec (P \parallel Q')$
<proof>

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 $P \parallel Q \Longrightarrow_l \hat{\tau} \prec P' \parallel Q'$
<proof>

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 $P \parallel Q \Longrightarrow_l \hat{\tau} \prec P' \parallel Q'$
<proof>

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 $P \parallel Q \Longrightarrow_l \hat{\tau} \prec \langle \nu y \rangle (P' \parallel Q')$
 $\langle proof \rangle$

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 $P \parallel Q \Longrightarrow_l \hat{\tau} \prec \langle \nu y \rangle (P' \parallel Q')$
 $\langle proof \rangle$

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 $\langle \nu x \rangle P \Longrightarrow_l \hat{\alpha} \prec \langle \nu x \rangle P'$
 $\langle proof \rangle$

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 $\langle \nu y \rangle P \Longrightarrow_l \hat{a} \langle \nu x \rangle \prec (\langle \nu y \rangle P')$
 $\langle proof \rangle$

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 $!P \Longrightarrow_l \hat{Rs}$
 $\langle proof \rangle$

lemma *tauTransitionChain*:

fixes $P :: pi$
and $P' :: pi$

assumes $P \Longrightarrow_l \hat{\tau} \prec P'$

shows $P \Longrightarrow_\tau P'$
 $\langle proof \rangle$

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 $P \Longrightarrow_l \hat{\alpha} \prec P'' \Longrightarrow P'' \Longrightarrow_\tau P' \Longrightarrow P \Longrightarrow_l \hat{\alpha} \prec P'$
 $\langle proof \rangle$

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 $P \Longrightarrow_l \alpha \prec P''$
 $\langle proof \rangle$

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 $c \# P'$
 $\langle proof \rangle$

lemma *freshTauTransition*:

fixes $P :: pi$
and $c :: name$

assumes $PTrans: P \Longrightarrow_l \hat{\tau} \prec P'$
and $cFreshP: c \# P$

shows $c \# P'$
 $\langle proof \rangle$

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 $c \# P'$
 $\langle proof \rangle$

lemma *eqvtI*:

fixes $P :: pi$
and $Rs :: residual$
and $perm :: name prm$

assumes $P \Longrightarrow_l \hat{Rs}$

shows $(perm \cdot P) \Longrightarrow_l \hat{\ } (perm \cdot Rs)$
 $\langle proof \rangle$

lemma *freshInputTransition*:

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 $cFreshP: c \# P$
and $cineqb: c \neq b$

shows $c \# P'$
 $\langle proof \rangle$

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

$weakSimAct P Rs C Rel \equiv (\forall Q' a x. Rs = a < \nu x > \prec Q' \longrightarrow x \# C \longrightarrow (\exists P'. P \Longrightarrow_l \hat{\ } a < \nu x > \prec P' \wedge (P', Q') \in Rel)) \wedge$
 $(\forall Q' a x. Rs = a < x > \prec Q' \longrightarrow x \# C \longrightarrow (\exists P''. \forall u. \exists P'. P \Longrightarrow_l u \text{ in } P'' \rightarrow a < x > \prec P' \wedge (P', Q'[x::=u]) \in Rel)) \wedge$
 $(\forall Q' \alpha. Rs = \alpha \prec Q' \longrightarrow (\exists P'. P \Longrightarrow_l \hat{\ } \alpha \prec P' \wedge (P', Q') \in Rel))$

definition *weakSimAux* :: $pi \Rightarrow (pi \times pi) set \Rightarrow pi \Rightarrow bool$ **where**

$weakSimAux P Rel Q \equiv (\forall Q' a x. (Q \mapsto a < \nu x > \prec Q' \wedge x \# P) \longrightarrow (\exists P'. P \Longrightarrow_l \hat{\ } a < \nu x > \prec P' \wedge (P', Q') \in Rel)) \wedge$
 $(\forall Q' a x. (Q \mapsto a < x > \prec Q' \wedge x \# P) \longrightarrow (\exists P''. \forall u. \exists P'. P \Longrightarrow_l u \text{ in } P'' \rightarrow a < x > \prec P' \wedge (P', Q'[x::=u]) \in Rel)) \wedge$
 $(\forall Q' \alpha. Q \mapsto \alpha \prec Q' \longrightarrow (\exists P'. P \Longrightarrow_l \hat{\ } \alpha \prec P' \wedge (P', Q') \in Rel))$

definition *weakSimulation* :: $pi \Rightarrow (pi \times pi) set \Rightarrow pi \Rightarrow bool$ ($\prec \rightsquigarrow \hat{\ } \prec \rightarrow$) [80, 80, 80] **where**

$P \rightsquigarrow \hat{\ } \prec Rel \rangle Q \equiv (\forall Rs. Q \mapsto Rs \longrightarrow weakSimAct P Rs P Rel)$

lemmas *simDef = weakSimAct-def weakSimulation-def*

lemma *weakSimAux* $P \text{ Rel } Q = \text{weakSimulation } P \text{ Rel } Q$

<proof>

lemma *monotonic*:

fixes $A :: (pi \times pi) \text{ set}$
and $B :: (pi \times pi) \text{ set}$
and $P :: pi$
and $P' :: pi$

assumes $P \rightsquigarrow^{\wedge} \langle A \rangle P'$
and $A \subseteq B$

shows $P \rightsquigarrow^{\wedge} \langle B \rangle P'$

<proof>

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

fixes $P :: pi$
and $Q :: pi$
and $Rel :: (pi \times pi) \text{ set}$
and $C :: 'a::fs\text{-name}$

assumes *Eqvt*: *eqvt* Rel

and *Bound*: $\bigwedge Q' a x. \llbracket x \# C; Q \mapsto a \langle \nu x \rangle \prec Q' \rrbracket \implies \exists P'. P \implies_l^{\wedge} a \langle \nu x \rangle \prec P' \wedge (P', Q') \in Rel$

and *Input*: $\bigwedge Q' a x. \llbracket x \# C; Q \mapsto a \langle x \rangle \prec Q' \rrbracket \implies \exists P''. \forall u. \exists P'. P \implies_l u \text{ in } P'' \rightarrow a \langle x \rangle \prec P' \wedge (P', Q'[x::=u]) \in Rel$

and *Free*: $\bigwedge Q' \alpha. Q \mapsto \alpha \prec Q' \implies (\exists P'. P \implies_l^{\wedge} \alpha \prec P' \wedge (P', Q') \in Rel)$

shows $P \rightsquigarrow^{\wedge} \langle Rel \rangle Q$

<proof>

lemma *simCases*[*case-names Bound Input Free*]:

fixes $P :: pi$
and $Q :: pi$
and $Rel :: (pi \times pi) \text{ set}$
and $C :: 'a::fs\text{-name}$

assumes *Bound*: $\bigwedge Q' a x. \llbracket Q \mapsto a \langle \nu x \rangle \prec Q'; x \# P \rrbracket \implies \exists P'. P \implies_l^{\wedge} a \langle \nu x \rangle \prec P' \wedge (P', Q') \in Rel$

and *Input*: $\bigwedge Q' a x. \llbracket Q \mapsto a \langle x \rangle \prec Q'; x \# P \rrbracket \implies \exists P''. \forall u. \exists P'. P \implies_l u \text{ in } P'' \rightarrow a \langle x \rangle \prec P' \wedge (P', Q'[x::=u]) \in Rel$

and *Free*: $\bigwedge Q' \alpha. Q \mapsto \alpha \prec Q' \implies (\exists P'. P \implies_l^{\wedge} \alpha \prec P' \wedge (P', Q') \in Rel)$

shows $P \rightsquigarrow^{\wedge} \langle Rel \rangle Q$

<proof>

lemma *simActBoundCases*[consumes 1, case-names *Input BoundOutput*]:

fixes $P :: pi$
and $a :: subject$
and $x :: name$
and $Q' :: pi$
and $C :: 'a::fs-name$
and $Rel :: (pi \times pi) set$

assumes *EqvtRel*: *eqvt Rel*

and *DerInput*: $\bigwedge b. a = InputS\ b \implies (\exists P''. \forall u. \exists P'. (P \implies_l u\ in\ P'' \rightarrow b \langle x \rangle \prec P') \wedge (P', Q'[x::=u]) \in Rel)$
and *DerBoundOutput*: $\bigwedge b. a = BoundOutputS\ b \implies (\exists P'. (P \implies_l \hat{b} \langle \nu x \rangle \prec P') \wedge (P', Q') \in Rel)$

shows *weakSimAct* $P (a \langle x \rangle \prec Q') P Rel$
 $\langle proof \rangle$

lemma *simActFreeCases*[consumes 0, case-names *Der*]:

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

assumes $\exists P'. (P \implies_l \hat{\alpha} \prec P') \wedge (P', Q') \in Rel$

shows *weakSimAct* $P (\alpha \prec Q') P Rel$
 $\langle proof \rangle$

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 $Q \mapsto a \langle \nu x \rangle \prec Q' \implies x \# P \implies \exists P'. P \implies_l \hat{a} \langle \nu x \rangle \prec P' \wedge (P', Q') \in Rel$

and $Q \mapsto a \langle x \rangle \prec Q' \implies x \# P \implies \exists P''. \forall u. \exists P'. P \implies_l u\ in\ P'' \rightarrow a \langle x \rangle \prec P' \wedge (P', Q'[x::=u]) \in Rel$

and $Q \mapsto \alpha \prec Q' \implies (\exists P'. P \implies_l \hat{\alpha} \prec P' \wedge (P', Q') \in Rel)$
 $\langle proof \rangle$

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 $Sim: \bigwedge P Q. (P, Q) \in Rel \Longrightarrow P \rightsquigarrow^{\wedge} \langle Rel \rangle Q$

shows $\exists P'. P \Longrightarrow_{\tau} P' \wedge (P', Q') \in Rel$
<proof>

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 $PSimQ: P \rightsquigarrow^{\wedge} \langle Rel \rangle Q$
and $Sim: \bigwedge P Q. (P, Q) \in Rel \Longrightarrow P \rightsquigarrow^{\wedge} \langle Rel \rangle Q$
and $Eqvt: eqvt Rel$
and $PRelQ: (P, Q) \in Rel$

shows $Q \Longrightarrow_l^{\wedge} a \langle \nu x \rangle \prec Q' \Longrightarrow x \# P \Longrightarrow \exists P'. P \Longrightarrow_l^{\wedge} a \langle \nu x \rangle \prec P' \wedge (P', Q') \in Rel$
and $Q \Longrightarrow_l^{\wedge} \alpha \prec Q' \Longrightarrow \exists P'. P \Longrightarrow_l^{\wedge} \alpha \prec P' \wedge (P', Q') \in Rel$
<proof>

lemma *eqvtI*:
fixes $P :: pi$
and $Q :: pi$
and $Rel :: (pi \times pi) \text{ set}$
and $perm :: name \text{ prm}$

assumes $Sim: P \rightsquigarrow^{\wedge} \langle Rel \rangle Q$
and $RelRel': Rel \subseteq Rel'$
and $EqvtRel': eqvt Rel'$

shows $(perm \cdot P) \rightsquigarrow^{\wedge} \langle Rel' \rangle (perm \cdot Q)$
<proof>

lemma *reflexive*:
fixes $P :: pi$
and $Rel :: (pi \times pi) \text{ set}$

assumes $Id \subseteq Rel$

shows $P \rightsquigarrow^{\wedge} \langle Rel \rangle P$
 ⟨proof⟩

lemma *transitive*:

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

assumes $QSimR: Q \rightsquigarrow^{\wedge} \langle Rel' \rangle R$
 and $Eqvt: eqvt Rel$
 and $Eqvt': eqvt Rel''$
 and $Trans: Rel \circ Rel' \subseteq Rel''$
 and $Sim: \bigwedge P Q. (P, Q) \in Rel \implies P \rightsquigarrow^{\wedge} \langle Rel \rangle Q$
 and $PRelQ: (P, Q) \in Rel$

shows $P \rightsquigarrow^{\wedge} \langle Rel'' \rangle R$
 ⟨proof⟩

lemma *strongSimWeakSim*:

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

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

shows $P \rightsquigarrow^{\wedge} \langle Rel \rangle Q$
 ⟨proof⟩

lemma *strongAppend*:

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

assumes $PSimQ: P \rightsquigarrow^{\wedge} \langle Rel \rangle Q$
 and $QSimR: Q \rightsquigarrow [Rel'] R$
 and $Eqvt'': eqvt Rel''$
 and $Trans: Rel \circ Rel' \subseteq Rel''$

shows $P \rightsquigarrow^{\wedge} \langle Rel'' \rangle R$
 ⟨proof⟩

end

```

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

lemma monoAux:  $A \subseteq B \implies P \rightsquigarrow^{\hat{A}} Q \longrightarrow P \rightsquigarrow^{\hat{B}} Q$ 
  <proof>

coinductive-set weakBisim ::  $(pi \times pi)$  set
where
  step:  $\llbracket P \rightsquigarrow^{\hat{weakBisim}} Q; (Q, P) \in weakBisim \rrbracket \implies (P, Q) \in weakBisim$ 
monos monoAux

abbreviation
  weakBisimJudge (infixr  $\langle \approx \rangle$  65) where  $P \approx Q \equiv (P, Q) \in weakBisim$ 

lemma weakBisimCoinductAux[case-names weakBisim, case-conclusion weakBisim
step, consumes 1]:
  assumes  $p: (P, Q) \in X$ 
  and step:  $\bigwedge P Q. (P, Q) \in X \implies P \rightsquigarrow^{\hat{\langle X \cup weakBisim \rangle}} Q \wedge ((Q, P) \in X \vee Q \approx P)$ 

  shows  $P \approx Q$ 
  <proof>

lemma weakBisimCoinduct[consumes 1, case-names cSim cSym]:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 

  assumes  $(P, Q) \in X$ 
  and  $\bigwedge P Q. (P, Q) \in X \implies P \rightsquigarrow^{\hat{\langle X \cup weakBisim \rangle}} Q$ 
  and  $\bigwedge P Q. (P, Q) \in X \implies (Q, P) \in X$ 

  shows  $P \approx Q$ 
  <proof>

lemma weak-coinduct[case-names weakBisim, case-conclusion weakBisim step, consumes 1]:
  assumes  $p: (P, Q) \in X$ 
  and step:  $\bigwedge P Q. (P, Q) \in X \implies P \rightsquigarrow^{\hat{\langle X \rangle}} Q \wedge (Q, P) \in X$ 

  shows  $P \approx Q$ 
  <proof>

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 $P \approx Q$

$\langle proof \rangle$

lemma *monotonic*: $mono(\lambda p x1 x2. \exists P Q. x1 = P \wedge x2 = Q \wedge P \rightsquigarrow^{\hat{}} \langle \{(xa, x). p \ x a \ x \} \rangle Q \wedge Q \rightsquigarrow^{\hat{}} \langle \{(xa, x). p \ x a \ x \} \rangle P)$

$\langle proof \rangle$

lemma *unfoldE*:

fixes $P :: pi$

and $Q :: pi$

assumes $P \approx Q$

shows $P \rightsquigarrow^{\hat{}} \langle weakBisim \rangle Q$

and $Q \approx P$

$\langle proof \rangle$

lemma *unfoldI*:

fixes $P :: pi$

and $Q :: pi$

assumes $P \rightsquigarrow^{\hat{}} \langle weakBisim \rangle Q$

and $Q \approx P$

shows $P \approx Q$

$\langle proof \rangle$

lemma *eqvt*:

shows $eqvt \ weakBisim$

$\langle proof \rangle$

lemma *eqvtI*:

fixes $P :: pi$

and $Q :: pi$

and $perm :: name \ prm$

assumes $P \approx Q$

shows $(perm \cdot P) \approx (perm \cdot Q)$

$\langle proof \rangle$

lemma *weakBisimEqvt[simp]*:

shows $eqvt \ weakBisim$

$\langle proof \rangle$

lemma *strongBisimWeakBisim*:

fixes $P :: pi$

and $Q :: pi$

assumes $PSimQ: P \sim Q$

shows $P \approx Q$
 $\langle proof \rangle$

lemma *reflexive*:
fixes $P :: pi$

shows $P \approx P$
 $\langle proof \rangle$

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

assumes $P \approx Q$

shows $Q \approx P$
 $\langle proof \rangle$

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

assumes $PBiSimQ: P \approx Q$
and $QBiSimR: Q \approx R$

shows $P \approx R$
 $\langle proof \rangle$

lemma *transitive-coinduct-weak*[*case-names WeakBisimEarly, case-conclusion WeakBisimEarly step, consumes 2*]:
assumes $p: (P, Q) \in X$
and $Eqvt: eqvt X$
and $step: \bigwedge P Q. (P, Q) \in X \implies P \rightsquigarrow^{\langle bisim \ O \ X \ O \ bisim \rangle} Q \wedge (Q, P) \in X$

shows $P \approx Q$
 $\langle proof \rangle$

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 $P \approx Q$
 ⟨proof⟩

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**

$weakStepSimAct P Rs C Rel \equiv (\forall Q' a x. Rs = a \langle \nu x \rangle \prec Q' \longrightarrow x \# C \longrightarrow$
 $(\exists P'. P \Longrightarrow_{l a \langle \nu x \rangle} \prec P' \wedge (P', Q') \in Rel)) \wedge$
 $(\forall Q' a x. Rs = a \langle x \rangle \prec Q' \longrightarrow x \# C \longrightarrow (\exists P''. \forall u. \exists P'.$
 $P \Longrightarrow_{l u} in P'' \rightarrow a \langle x \rangle \prec P' \wedge (P', Q'[x ::= u]) \in Rel)) \wedge$
 $(\forall Q' \alpha. Rs = \alpha \prec Q' \longrightarrow (\exists P'. P \Longrightarrow_{l \alpha} \prec P' \wedge (P', Q') \in$
 $Rel))$

definition *weakStepSimAux* :: $pi \Rightarrow (pi \times pi) set \Rightarrow pi \Rightarrow bool$ **where**

$weakStepSimAux P Rel Q \equiv (\forall Q' a x. (Q \mapsto a \langle \nu x \rangle \prec Q' \wedge x \# P) \longrightarrow (\exists P'.$
 $P \Longrightarrow_{l a \langle \nu x \rangle} \prec P' \wedge (P', Q') \in Rel)) \wedge$
 $(\forall Q' a x. (Q \mapsto a \langle x \rangle \prec Q' \wedge x \# P) \longrightarrow (\exists P''. \forall u. \exists P'.$
 $P \Longrightarrow_{l u} in P'' \rightarrow a \langle x \rangle \prec P' \wedge (P', Q'[x ::= u]) \in Rel)) \wedge$
 $(\forall Q' \alpha. Q \mapsto \alpha \prec Q' \longrightarrow (\exists P'. P \Longrightarrow_{l \alpha} \prec P' \wedge (P', Q') \in$
 $Rel))$

definition *weakStepSim* :: $pi \Rightarrow (pi \times pi) set \Rightarrow pi \Rightarrow bool$ ($\prec \rightsquigarrow \langle - \rangle \prec - \rightarrow [80, 80, 80] 80$) **where**

$P \rightsquigarrow \langle Rel \rangle Q \equiv (\forall Rs. Q \mapsto Rs \longrightarrow weakStepSimAct P Rs P Rel)$

lemmas *weakStepSimDef* = *weakStepSimAct-def weakStepSim-def*

lemma *weakStepSimAux* $P Rel Q = weakStepSim P Rel Q$

⟨proof⟩

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 $P \rightsquigarrow \langle B \rangle P'$

⟨proof⟩

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

fixes $P :: pi$

and $Q :: pi$

and $Rel :: (pi \times pi)$ set
and $C :: 'a::fs\text{-name}$

assumes $Eqvt$: $eqvt\ Rel$
and $Bound$: $\bigwedge Q' a x. \llbracket x \# C; Q \mapsto a \langle \nu x \rangle \prec Q' \rrbracket \implies \exists P'. P \implies_l a \langle \nu x \rangle \prec P' \wedge (P', Q') \in Rel$
and $Input$: $\bigwedge Q' a x. \llbracket x \# C; Q \mapsto a \langle x \rangle \prec Q' \rrbracket \implies \exists P''. \forall u. \exists P'. P \implies_l u$
in $P'' \mapsto a \langle x \rangle \prec P' \wedge (P', Q'[x::=u]) \in Rel$
and $Free$: $\bigwedge Q' \alpha. Q \mapsto \alpha \prec Q' \implies (\exists P'. P \implies_l \alpha \prec P' \wedge (P', Q') \in Rel)$

shows $P \rightsquigarrow \langle Rel \rangle Q$
 $\langle proof \rangle$

lemma $simCases[consumes\ 0, case\text{-names}\ Bound\ Input\ Free]$:

fixes $P :: pi$
and $Q :: pi$
and $Rel :: (pi \times pi)$ set
and $C :: 'a::fs\text{-name}$

assumes $Bound$: $\bigwedge Q' a x. \llbracket Q \mapsto a \langle \nu x \rangle \prec Q'; x \# P \rrbracket \implies \exists P'. P \implies_l a \langle \nu x \rangle \prec P' \wedge (P', Q') \in Rel$
and $Input$: $\bigwedge Q' a x. \llbracket Q \mapsto a \langle x \rangle \prec Q'; x \# P \rrbracket \implies \exists P''. \forall u. \exists P'. P \implies_l u$
in $P'' \mapsto a \langle x \rangle \prec P' \wedge (P', Q'[x::=u]) \in Rel$
and $Free$: $\bigwedge Q' \alpha. Q \mapsto \alpha \prec Q' \implies (\exists P'. P \implies_l \alpha \prec P' \wedge (P', Q') \in Rel)$

shows $P \rightsquigarrow \langle Rel \rangle Q$
 $\langle proof \rangle$

lemma $simActBoundCases[consumes\ 1, case\text{-names}\ Input\ BoundOutput]$:

fixes $P :: pi$
and $a :: subject$
and $x :: name$
and $Q' :: pi$
and $C :: 'a::fs\text{-name}$
and $Rel :: (pi \times pi)$ set

assumes $EqvtRel$: $eqvt\ Rel$
and $DerInput$: $\bigwedge b. a = InputS\ b \implies (\exists P''. \forall u. \exists P'. (P \implies_l u \text{ in } P'' \mapsto b \langle x \rangle \prec P') \wedge (P', Q'[x::=u]) \in Rel)$
and $DerBoundOutput$: $\bigwedge b. a = BoundOutputS\ b \implies (\exists P'. (P \implies_l b \langle \nu x \rangle \prec P') \wedge (P', Q') \in Rel)$

shows $weakStepSimAct\ P\ (a \langle x \rangle \prec Q')\ P\ Rel$
 $\langle proof \rangle$

lemma $simActFreeCases[consumes\ 0, case\text{-names}\ Free]$:

fixes $P :: pi$
and $\alpha :: freeRes$
and $C :: 'a::fs\text{-name}$

and $Rel :: (pi \times pi) \text{ set}$
assumes $Der: \exists P'. (P \Longrightarrow_l \alpha \prec P') \wedge (P', Q') \in Rel$
shows $weakStepSimAct P (\alpha \prec Q') P Rel$
 $\langle proof \rangle$

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

assumes $P \rightsquigarrow \langle Rel \rangle Q$

shows $Q \mapsto a \langle \nu x \rangle \prec Q' \Longrightarrow x \# P \Longrightarrow \exists P'. P \Longrightarrow_l a \langle \nu x \rangle \prec P' \wedge (P', Q') \in Rel$
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 $Q \mapsto \alpha \prec Q' \Longrightarrow (\exists P'. P \Longrightarrow_l \alpha \prec P' \wedge (P', Q') \in Rel)$
 $\langle proof \rangle$

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

assumes $QChain: Q \Longrightarrow_\tau Q'$
and $PRelQ: (P, Q) \in Rel$
and $Sim: \bigwedge P Q. (P, Q) \in Rel \Longrightarrow P \rightsquigarrow \langle Rel \rangle Q$

shows $\exists P'. P \Longrightarrow_\tau P' \wedge (P', Q') \in Rel$
 $\langle proof \rangle$

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

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

shows $P \rightsquigarrow \langle Rel \rangle Q$
 $\langle proof \rangle$

lemma $weakSimWeakEqSim$:

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

assumes $P \rightsquigarrow \langle Rel \rangle Q$

shows $P \rightsquigarrow^{\wedge} \langle Rel \rangle Q$
 $\langle proof \rangle$

lemma *eqvtI*:

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

assumes $Sim: P \rightsquigarrow \langle Rel \rangle Q$

and $RelRel': Rel \subseteq Rel'$

and $EqvtRel': eqvt Rel'$

shows $(perm \cdot P) \rightsquigarrow \langle Rel' \rangle (perm \cdot Q)$
 $\langle proof \rangle$

lemma *simE2*:

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

assumes $PSimQ: P \rightsquigarrow \langle Rel \rangle Q$

and $Sim: \bigwedge P Q. (P, Q) \in Rel \implies P \rightsquigarrow^{\wedge} \langle Rel \rangle Q$

and $Eqvt: eqvt Rel$

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

shows $Q \implies_{\lrcorner} a \langle \nu x \rangle \prec Q' \implies x \sharp P \implies \exists P'. P \implies_{\lrcorner} a \langle \nu x \rangle \prec P' \wedge (P', Q') \in Rel$

and $Q \implies_{\lrcorner} \alpha \prec Q' \implies \exists P'. P \implies_{\lrcorner} \alpha \prec P' \wedge (P', Q') \in Rel$
 $\langle proof \rangle$

lemma *reflexive*:

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

assumes $Id \subseteq Rel$

shows $P \rightsquigarrow \langle Rel \rangle P$

<proof>

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 $P \rightsquigarrow \langle Rel'' \rangle R$

<proof>

end

theory *Weak-Late-Cong*

imports *Weak-Late-Bisim Weak-Late-Step-Sim Strong-Late-Bisim*
begin

definition *congruence* :: $(pi \times pi)$ set **where**

$congruence \equiv \{(P, Q) \mid P\ Q. P \rightsquigarrow \langle weakBisim \rangle Q \wedge Q \rightsquigarrow \langle weakBisim \rangle P\}$

abbreviation *congruenceJudge* (**infixr** $\langle \simeq \rangle$ 65) **where** $P \simeq Q \equiv (P, Q) \in congruence$

lemma *unfoldE*:

fixes P :: pi
and Q :: pi
and s :: $(name \times name)$ list

assumes $P \simeq Q$

shows $P \rightsquigarrow \langle weakBisim \rangle Q$
and $Q \rightsquigarrow \langle weakBisim \rangle P$

<proof>

lemma *unfoldI*:

fixes P :: pi
and Q :: pi

assumes $P \rightsquigarrow \langle weakBisim \rangle Q$
and $Q \rightsquigarrow \langle weakBisim \rangle P$

shows $P \simeq Q$
<proof>

lemma *eqvt*:
shows *eqvt congruence*
<proof>

lemma *eqvtI*:
fixes $P :: pi$
and $Q :: pi$
and $perm :: name prm$

assumes $P \simeq Q$

shows $(perm \cdot P) \simeq (perm \cdot Q)$
<proof>

lemma *strongBisimWeakEq*:
fixes $P :: pi$
and $Q :: pi$

assumes $P \sim Q$

shows $P \simeq Q$
<proof>

lemma *congruenceWeakBisim*:
fixes $P :: pi$
and $Q :: pi$

assumes $P \simeq Q$

shows $P \approx Q$
<proof>

lemma *congruenceSubsetWeakBisim*:
shows $congruence \subseteq weakBisim$
<proof>

lemma *reflexive*:
fixes $P :: pi$

shows $P \simeq P$
<proof>

lemma *symetric*:
fixes $P :: pi$

```

and   Q :: pi

assumes P ≈ Q

shows Q ≈ P
⟨proof⟩

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

  assumes P ≈ Q
  and     Q ≈ R

  shows P ≈ R
⟨proof⟩

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⟩

lemma strongBisimWeakBisim:
  fixes P :: pi
  and   Q :: pi

  assumes P ≈s Q

  shows P ≈s Q
⟨proof⟩

lemma eqvt:
  shows eqvt (substClosed weakBisim)

```

<proof>

lemma *eqvtI*:

fixes $P :: pi$

and $Q :: pi$

and $perm :: name\ prm$

assumes $P \approx^s Q$

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

<proof>

lemma *reflexive*:

fixes $P :: pi$

shows $P \approx^s P$

<proof>

lemma *symetric*:

fixes $P :: pi$

and $Q :: pi$

assumes $P \approx^s Q$

shows $Q \approx^s P$

<proof>

lemma *transitive*:

fixes $P :: pi$

and $Q :: pi$

and $R :: pi$

assumes $P \approx^s Q$

and $Q \approx^s R$

shows $P \approx^s R$

<proof>

lemma *partUnfold*:

fixes $P :: pi$

and $Q :: pi$

and $s :: (name \times name)\ list$

assumes $P \approx^s Q$

shows $P[<s>] \approx^s Q[<s>]$

<proof>

```

end

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

definition congruenceSubst :: pi ⇒ pi ⇒ bool (infixr <≃s> 65) where
  P ≃s Q ≡ (P, Q) ∈ (substClosed congruence)

lemmas congruenceSubstDef = congruenceSubst-def congruence-def substClosed-def

lemma unfoldE:
  fixes P :: pi
  and Q :: pi
  and s :: (name × name) list

  assumes P ≃s Q

  shows P[<s>] ∼<weakBisim> Q[<s>]
  and Q[<s>] ∼<weakBisim> P[<s>]
  <proof>

lemma unfoldI:
  fixes P :: pi
  and Q :: pi

  assumes ∀ s. P[<s>] ∼<weakBisim> Q[<s>] ∧ Q[<s>] ∼<weakBisim> P[<s>]

  shows P ≃s Q
  <proof>

lemma weakEqSubset:
  shows substClosed congruence ⊆ weakBisim
  <proof>

lemma weakCongWeakEq:
  fixes P :: pi
  and Q :: pi

  assumes P ≃s Q

  shows P ≃ Q
  <proof>

lemma eqvt:
  shows eqvt (substClosed congruence)
  <proof>

```

lemma *eqvtI*:
fixes $P :: pi$
and $Q :: pi$
and $perm :: name\ prm$

assumes $P \simeq^s Q$

shows $(perm \cdot P) \simeq^s (perm \cdot Q)$
 $\langle proof \rangle$

lemma *strongEqWeakCong*:
fixes $P :: pi$
and $Q :: pi$

assumes $P \sim^s Q$

shows $P \simeq^s Q$
 $\langle proof \rangle$

lemma *congSubstBisimSubst*:
fixes $P :: pi$
and $Q :: pi$

assumes $P \simeq^s Q$

shows $P \approx^s Q$
 $\langle proof \rangle$

lemma *reflexive*:
fixes $P :: pi$

shows $P \simeq^s P$
 $\langle proof \rangle$

lemma *symetric*:
fixes $P :: pi$
and $Q :: pi$

assumes $P \simeq^s Q$

shows $Q \simeq^s P$
 $\langle proof \rangle$

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

```

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

shows  $P \simeq^s R$ 
⟨proof⟩

lemma partUnfold:
fixes  $P :: pi$ 
and  $Q :: pi$ 
and  $s :: (name \times name) list$ 

assumes  $P \simeq^s Q$ 

shows  $P[<s>] \simeq^s Q[<s>]$ 
⟨proof⟩

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  $\mathbf{0} \rightsquigarrow[Rel] \tau.(P) \implies False$ 
and  $\mathbf{0} \rightsquigarrow[Rel] a<x>.P \implies False$ 
and  $\mathbf{0} \rightsquigarrow[Rel] a\{b\}.P \implies False$ 
⟨proof⟩

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

shows  $P \rightsquigarrow[Rel] \mathbf{0}$ 
⟨proof⟩

lemma matchIdLeft:
fixes  $a :: name$ 
and  $P :: pi$ 
and  $Rel :: (pi \times pi) set$ 

```

```

assumes  $Id \subseteq Rel$ 

shows  $[a \frown a]P \rightsquigarrow[Rel] P$ 
 $\langle proof \rangle$ 

lemma matchIdRight:
fixes  $P :: pi$ 
and  $a :: name$ 
and  $Rel :: (pi \times pi) set$ 

assumes IdRel:  $Id \subseteq Rel$ 

shows  $P \rightsquigarrow[Rel] [a \frown a]P$ 
 $\langle proof \rangle$ 

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

assumes  $a \neq b$ 

shows  $0 \rightsquigarrow[Rel] [a \frown b]P$ 
 $\langle proof \rangle$ 

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

assumes  $Id \subseteq Rel$ 
and  $a \neq b$ 

shows  $[a \neq b]P \rightsquigarrow[Rel] P$ 
 $\langle proof \rangle$ 

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 $P \rightsquigarrow[Rel] [a \neq b] P$
<proof>

lemma *mismatchNilLeft*:
fixes $a :: name$
and $P :: pi$

shows $0 \rightsquigarrow[Rel] [a \neq a] P$
<proof>

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

assumes $Id: Id \subseteq Rel$

shows $P \oplus Q \rightsquigarrow[Rel] Q \oplus P$
<proof>

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

assumes $Id \subseteq Rel$

shows $P \rightsquigarrow[Rel] P \oplus P$
<proof>

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

assumes $I: Id \subseteq Rel$

shows $P \oplus P \rightsquigarrow[Rel] P$
<proof>

lemma *sumAssocLeft*:
fixes $P :: pi$
and $Q :: pi$
and $R :: pi$
and $Rel :: (pi \times pi) set$

assumes $Id: Id \subseteq Rel$

shows $(P \oplus Q) \oplus R \rightsquigarrow[Rel] P \oplus (Q \oplus R)$

<proof>

lemma *sumAssocRight*:

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

assumes $Id: Id \subseteq Rel$

shows $P \oplus (Q \oplus R) \rightsquigarrow[Rel] (P \oplus Q) \oplus R$
<proof>

lemma *sumZeroLeft*:

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

assumes $Id: Id \subseteq Rel$

shows $P \oplus \mathbf{0} \rightsquigarrow[Rel] P$
<proof>

lemma *sumZeroRight*:

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

assumes $Id: Id \subseteq Rel$

shows $P \rightsquigarrow[Rel] P \oplus \mathbf{0}$
<proof>

lemma *sumResLeft*:

fixes $x :: name$
and $P :: pi$
and $Q :: pi$

assumes $Id: Id \subseteq Rel$
and $Eqvt: eqvt Rel$

shows $(\langle \nu x \rangle P) \oplus (\langle \nu x \rangle Q) \rightsquigarrow[Rel] \langle \nu x \rangle (P \oplus Q)$
<proof>

lemma *sumResRight*:

fixes $x :: name$
and $P :: pi$
and $Q :: pi$

assumes $Id: Id \subseteq Rel$
and $Eqvt: eqvt Rel$

shows $\langle \nu x \rangle (P \oplus Q) \rightsquigarrow [Rel] (\langle \nu x \rangle P) \oplus (\langle \nu x \rangle Q)$
 $\langle proof \rangle$

lemma *parZeroLeft*:

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

assumes *ParZero*: $\bigwedge Q. (Q \parallel \mathbf{0}, Q) \in Rel$

shows $P \parallel \mathbf{0} \rightsquigarrow [Rel] P$
 $\langle proof \rangle$

lemma *parZeroRight*:

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

assumes *ParZero*: $\bigwedge Q. (Q, Q \parallel \mathbf{0}) \in Rel$

shows $P \rightsquigarrow [Rel] P \parallel \mathbf{0}$
 $\langle proof \rangle$

lemma *parSym*:

fixes $P :: pi$
and $Q :: pi$
and $Rel :: (pi \times pi)$ 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 $P \parallel Q \rightsquigarrow [Rel] Q \parallel P$
 $\langle proof \rangle$

lemma *parAssocLeft*:

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

assumes *Ass*: $\bigwedge S T U. ((S \parallel T) \parallel U, S \parallel (T \parallel U)) \in Rel$
and *Res*: $\bigwedge S T x. (S, T) \in Rel \implies (\langle \nu x \rangle S, \langle \nu x \rangle T) \in Rel$
and *FreshExt*: $\bigwedge S T U x. x \# S \implies (\langle \nu x \rangle ((S \parallel T) \parallel U), S \parallel \langle \nu x \rangle (T \parallel U)) \in Rel$
and *FreshExt'*: $\bigwedge S T U x. x \# U \implies ((\langle \nu x \rangle (S \parallel T)) \parallel U, \langle \nu x \rangle (S \parallel (T \parallel U))) \in Rel$

shows $(P \parallel Q) \parallel R \rightsquigarrow [Rel] P \parallel (Q \parallel R)$

$\langle proof \rangle$

lemma *substRes3*:

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

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

$\langle proof \rangle$

lemma *scopeExtParLeft*:

fixes $P :: pi$
and $Q :: pi$
and $a :: name$
and $lst :: name\ list$
and $Rel :: (pi \times pi)\ set$

assumes $x \# P$

and $Id: Id \subseteq Rel$

and $EqvtRel: eqvt\ Rel$

and $Res: \bigwedge R\ S\ y. y \# R \implies (\langle \nu y \rangle (R \parallel S), R \parallel \langle \nu y \rangle S) \in Rel$

and $ScopeExt: \bigwedge R\ S\ y\ z. y \# R \implies (\langle \nu y \rangle \langle \nu z \rangle (R \parallel S), \langle \nu z \rangle (R \parallel \langle \nu y \rangle S)) \in Rel$

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

$\langle proof \rangle$

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 $P \parallel \langle \nu x \rangle Q \rightsquigarrow [Rel] \langle \nu x \rangle (P \parallel Q)$

$\langle proof \rangle$

lemma *resNilRight*:

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

shows $0 \rightsquigarrow [Rel] \langle \nu x \rangle 0$

$\langle proof \rangle$

lemma *resComm*:
fixes $a :: name$
and $b :: name$
and $P :: pi$
and $Rel :: (pi \times pi) set$

assumes *ResComm*: $\bigwedge c d Q. (\nu c <\nu d> Q, <\nu d> <\nu c> Q) \in Rel$
and *Id*: $Id \subseteq Rel$
and *EqvtRel*: *eqvt* Rel

shows $<\nu a> <\nu b> P \rightsquigarrow[Rel] <\nu b> <\nu a> P$
<proof>

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

assumes $Id \subseteq Rel$

shows $!P \rightsquigarrow[Rel] P \parallel !P$
<proof>

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

assumes *IdRel*: $Id \subseteq Rel$

shows $P \parallel !P \rightsquigarrow[Rel] !P$
<proof>

lemma *resNilLeft*:
fixes $x :: name$
and $y :: name$
and $P :: pi$
and $Rel :: (pi \times pi) set$
and $b :: name$

shows $0 \rightsquigarrow[Rel] <\nu x>(x <y>.P)$
and $0 \rightsquigarrow[Rel] <\nu x>(x \{b\}.P)$
<proof>

lemma *resInputLeft*:
fixes $x :: name$
and $a :: name$
and $y :: name$

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

assumes $xineqa: x \neq a$
and $xineqy: x \neq y$
and $Eqvt: eqvt Rel$
and $Id: Id \subseteq Rel$

shows $\langle \nu x \rangle a \langle y \rangle . P \rightsquigarrow [Rel] a \langle y \rangle . (\langle \nu x \rangle P)$
 ⟨proof⟩

lemma *resInputRight*:

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

assumes $xineqa: x \neq a$
and $xineqy: x \neq y$
and $Eqvt: eqvt Rel$
and $Id: Id \subseteq Rel$

shows $a \langle y \rangle . (\langle \nu x \rangle P) \rightsquigarrow [Rel] \langle \nu x \rangle a \langle y \rangle . P$
 ⟨proof⟩

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 $\langle \nu x \rangle a \{b\} . P \rightsquigarrow [Rel] a \{b\} . (\langle \nu x \rangle P)$
 ⟨proof⟩

lemma *resOutputRight*:

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:  $Id \subseteq Rel$ 
and   Eqvt: eqvt Rel

shows  $a\{b\}.\langle \nu x \rangle P \rightsquigarrow[Rel] \langle \nu x \rangle a\{b\}.P$ 
 $\langle proof \rangle$ 

lemma resTauLeft:
  fixes x :: name
  and   P :: pi
  and   Rel :: (pi × pi) set

  assumes Id:  $Id \subseteq Rel$ 

  shows  $\langle \nu x \rangle (\tau.(P)) \rightsquigarrow[Rel] \tau.\langle \nu x \rangle P$ 
   $\langle proof \rangle$ 

lemma resTauRight:
  fixes x :: name
  and   P :: pi
  and   Rel :: (pi × pi) set

  assumes Id:  $Id \subseteq Rel$ 

  shows  $\tau.\langle \nu x \rangle P \rightsquigarrow[Rel] \langle \nu x \rangle (\tau.(P))$ 
   $\langle proof \rangle$ 

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    $\mathbf{0} \sim a\{b\}.P \implies False$ 
   $\langle proof \rangle$ 

lemma matchId:

```

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

shows $[a \frown a]P \sim P$
 $\langle proof \rangle$

lemma *matchNil*:
fixes $a :: name$
and $b :: name$

assumes $a \neq b$

shows $[a \frown b]P \sim \mathbf{0}$
 $\langle proof \rangle$

lemma *mismatchId*:
fixes $a :: name$
and $b :: name$
and $P :: pi$

assumes $a \neq b$

shows $[a \neq b]P \sim P$
 $\langle proof \rangle$

lemma *mismatchNil*:
fixes $a :: name$
and $P :: pi$

shows $[a \neq a]P \sim \mathbf{0}$
 $\langle proof \rangle$

lemma *nilRes*:
fixes $x :: name$

shows $\langle \nu x \rangle \mathbf{0} \sim \mathbf{0}$
 $\langle proof \rangle$

lemma *resComm*:
fixes $x :: name$
and $y :: name$
and $P :: pi$

shows $\langle \nu x \rangle \langle \nu y \rangle P \sim \langle \nu y \rangle \langle \nu x \rangle P$
 $\langle proof \rangle$

lemma *sumSym*:

fixes $P :: pi$

and $Q :: pi$

shows $P \oplus Q \sim Q \oplus P$

<proof>

lemma *sumIdemp*:

fixes $P :: pi$

shows $P \oplus P \sim P$

<proof>

lemma *sumAssoc*:

fixes $P :: pi$

and $Q :: pi$

and $R :: pi$

shows $(P \oplus Q) \oplus R \sim P \oplus (Q \oplus R)$

<proof>

lemma *sumZero*:

fixes $P :: pi$

shows $P \oplus \mathbf{0} \sim P$

<proof>

lemma *parZero*:

fixes $P :: pi$

shows $P \parallel \mathbf{0} \sim P$

<proof>

lemma *parSym*:

fixes $P :: pi$

and $Q :: pi$

shows $P \parallel Q \sim Q \parallel P$

<proof>

lemma *scopeExtPar*:

fixes $P :: pi$

and $Q :: pi$

and $x :: name$

assumes $x \# P$

shows $\langle \nu x \rangle (P \parallel Q) \sim P \parallel \langle \nu x \rangle Q$
<proof>

lemma *scopeExtPar'*:

fixes $P :: pi$
and $Q :: pi$
and $x :: name$

assumes $xFreshQ: x \# Q$

shows $\langle \nu x \rangle (P \parallel Q) \sim (\langle \nu x \rangle P) \parallel Q$
<proof>

lemma *parAssoc*:

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

shows $(P \parallel Q) \parallel R \sim P \parallel (Q \parallel R)$
<proof>

lemma *scopeFresh*:

fixes $x :: name$
and $P :: pi$

assumes $x \# P$

shows $\langle \nu x \rangle P \sim P$
<proof>

lemma *sumRes*:

fixes $x :: name$
and $P :: pi$
and $Q :: pi$

shows $\langle \nu x \rangle (P \oplus Q) \sim (\langle \nu x \rangle P) \oplus (\langle \nu x \rangle Q)$
<proof>

lemma *scopeExtSum*:

fixes $P :: pi$
and $Q :: pi$
and $x :: name$

assumes $x \# P$

shows $\langle \nu x \rangle (P \oplus Q) \sim P \oplus \langle \nu x \rangle Q$
<proof>

lemma *bangSC*:

fixes $P :: pi$

shows $!P \sim P \parallel !P$

<proof>

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>

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>

lemma *resOutput*:

fixes $x :: name$

and $a :: name$

and $b :: name$

and $P :: pi$

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>

lemma *resTau*:

fixes $x :: name$

and $P :: pi$

shows $\langle \nu x \rangle \tau . (P) \sim \tau . (\langle \nu x \rangle P)$

<proof>

inductive *structCong* :: $pi \Rightarrow pi \Rightarrow bool$ ($\langle - \equiv_s - \rangle [70, 70] 70$)

where

$RefI: P \equiv_s P$
 $| Sym: P \equiv_s Q \implies Q \equiv_s P$
 $| Trans: \llbracket P \equiv_s Q; Q \equiv_s R \rrbracket \implies P \equiv_s R$

$| SumComm: P \oplus Q \equiv_s Q \oplus P$
 $| SumAssoc: (P \oplus Q) \oplus R \equiv_s P \oplus (Q \oplus R)$
 $| SumId: P \oplus \mathbf{0} \equiv_s P$

$| ParComm: P \parallel Q \equiv_s Q \parallel P$
 $| ParAssoc: (P \parallel Q) \parallel R \equiv_s P \parallel (Q \parallel R)$
 $| ParId: P \parallel \mathbf{0} \equiv_s P$

$| MatchId: [a \frown a]P \equiv_s P$

$| ResNil: \langle \nu x \rangle \mathbf{0} \equiv_s \mathbf{0}$
 $| ResComm: \langle \nu x \rangle \langle \nu y \rangle P \equiv_s \langle \nu y \rangle \langle \nu x \rangle P$
 $| ResSum: \langle \nu x \rangle (P \oplus Q) \equiv_s \langle \nu x \rangle P \oplus \langle \nu x \rangle Q$
 $| ScopeExtPar: x \# P \implies \langle \nu x \rangle (P \parallel Q) \equiv_s P \parallel \langle \nu x \rangle Q$
 $| InputRes: \llbracket x \neq a; x \neq y \rrbracket \implies \langle \nu x \rangle a \langle y \rangle . P \equiv_s a \langle y \rangle . (\langle \nu x \rangle P)$
 $| OutputRes: \llbracket x \neq a; x \neq b \rrbracket \implies \langle \nu x \rangle a \{ b \} . P \equiv_s a \{ b \} . (\langle \nu x \rangle P)$
 $| TauRes: \langle \nu x \rangle \tau . (P) \equiv_s \tau . (\langle \nu x \rangle P)$

$| BangUnfold: !P \equiv_s P \parallel !P$

lemma *structCongBisim*:

fixes $P :: pi$

and $Q :: pi$

assumes $P \equiv_s Q$

shows $P \sim Q$

<proof>

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 $[a \frown a]P \sim^s P$

<proof>

lemma *mismatchNil*:

fixes $a :: name$

and $P :: pi$

shows $[a \neq a]P \sim^s \mathbf{0}$
<proof>

lemma *scopeFresh*:
fixes $P :: pi$
and $x :: name$

assumes $xFreshP: x \# P$

shows $\langle \nu x \rangle P \sim^s P$
<proof>

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>

lemma *sumZero*:
fixes $P :: pi$

shows $P \oplus \mathbf{0} \sim^s P$
<proof>

lemma *sumSym*:
fixes $P :: pi$
and $Q :: pi$

shows $P \oplus Q \sim^s Q \oplus P$
<proof>

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

shows $(P \oplus Q) \oplus R \sim^s P \oplus (Q \oplus R)$
<proof>

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>

lemma *scopeExtSum*:
fixes $P :: pi$
and $Q :: pi$
and $x :: name$

assumes $xFreshP: x \# P$

shows $\langle \nu x \rangle (P \oplus Q) \sim^s P \oplus \langle \nu x \rangle Q$
 $\langle proof \rangle$

lemma *parZero*:
fixes $P :: pi$

shows $P \parallel \mathbf{0} \sim^s P$
 $\langle proof \rangle$

lemma *parSym*:
fixes $P :: pi$
and $Q :: pi$

shows $P \parallel Q \sim^s Q \parallel P$
 $\langle proof \rangle$

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

shows $(P \parallel Q) \parallel R \sim^s P \parallel (Q \parallel R)$
 $\langle proof \rangle$

lemma *scopeExtPar*:
fixes $P :: pi$
and $Q :: pi$
and $x :: name$

assumes $xFreshP: x \# P$

shows $\langle \nu x \rangle (P \parallel Q) \sim^s P \parallel \langle \nu x \rangle Q$
 $\langle proof \rangle$

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>

lemma *bangSC*:
fixes $P :: pi$

shows $!P \sim^s P \parallel !P$
<proof>

lemma *nilRes*:
fixes $x :: name$

shows $\langle \nu x \rangle \mathbf{0} \sim^s \mathbf{0}$
<proof>

lemma *resTau*:
fixes $x :: name$
and $P :: pi$

shows $\langle \nu x \rangle (\tau.(P)) \sim^s \tau.(\langle \nu x \rangle P)$
<proof>

lemma *resOutput*:
fixes $x :: name$
and $a :: name$
and $b :: name$
and $P :: pi$

assumes $x \neq a$
and $x \neq b$

shows $\langle \nu x \rangle (a\{b\}.(P)) \sim^s a\{b\}.(\langle \nu x \rangle P)$
<proof>

lemma *resInput*:
fixes $x :: name$
and $a :: name$
and $b :: name$
and $P :: pi$

assumes $x \neq a$
and $x \neq y$

shows $\langle \nu x \rangle (a\langle y \rangle.(P)) \sim^s a\langle y \rangle.(\langle \nu x \rangle P)$
<proof>

lemma *bisimSubstStructCong*:
fixes $P :: pi$
and $Q :: pi$

```

assumes  $P \equiv_s Q$ 
shows  $P \sim^s Q$ 

<proof>

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>

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

shows  $[a \frown a]P \simeq^s P$ 
<proof>

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

shows  $[a \neq a]P \simeq^s \mathbf{0}$ 
<proof>

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

shows  $P \oplus Q \simeq^s Q \oplus P$ 
<proof>

```

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

shows $(P \oplus Q) \oplus R \simeq^s P \oplus (Q \oplus R)$
 $\langle proof \rangle$

lemma *sumZero*:
fixes $P :: pi$

shows $P \oplus \mathbf{0} \simeq^s P$
 $\langle proof \rangle$

lemma *parZero*:
fixes $P :: pi$

shows $P \parallel \mathbf{0} \simeq^s P$
 $\langle proof \rangle$

lemma *parSym*:
fixes $P :: pi$
and $Q :: pi$

shows $P \parallel Q \simeq^s Q \parallel P$
 $\langle proof \rangle$

lemma *scopeExtPar*:
fixes $P :: pi$
and $Q :: pi$
and $x :: name$

assumes $x \# P$

shows $\langle \nu x \rangle (P \parallel Q) \simeq^s P \parallel \langle \nu x \rangle Q$
 $\langle proof \rangle$

lemma *scopeExtPar'*:
fixes $P :: pi$
and $Q :: pi$
and $x :: name$

assumes $xFreshQ: x \# Q$

shows $\langle \nu x \rangle (P \parallel Q) \simeq^s (\langle \nu x \rangle P) \parallel Q$
 $\langle proof \rangle$

```

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

  shows  $(P \parallel Q) \parallel R \simeq^s P \parallel (Q \parallel R)$ 
   $\langle proof \rangle$ 

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

  assumes  $aFreshP: a \# P$ 

  shows  $\langle \nu a \rangle P \simeq^s P$ 
   $\langle proof \rangle$ 

lemma scopeExtSum:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $x :: name$ 

  assumes  $x \# P$ 

  shows  $\langle \nu x \rangle (P \oplus Q) \simeq^s P \oplus \langle \nu x \rangle Q$ 
   $\langle proof \rangle$ 

lemma bangSC:
  fixes  $P$ 

  shows  $!P \simeq^s P \parallel !P$ 
   $\langle proof \rangle$ 

end

theory Weak-Late-Step-Sim-Pres
  imports Weak-Late-Step-Sim
  begin

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

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

  shows  $\tau.(P) \rightsquigarrow \langle Rel \rangle \tau.(Q)$ 

```

<proof>

lemma *inputPres*:

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

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

shows $a\langle x \rangle.P \rightsquigarrow\langle Rel \rangle a\langle x \rangle.Q$

<proof>

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>

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 $[a\curvearrowright b]P \rightsquigarrow\langle Rel' \rangle [a\curvearrowright b]Q$

<proof>

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 $[a \neq b]P \rightsquigarrow \langle Rel' \rangle [a \neq b]Q$
 $\langle proof \rangle$

lemma *sumCompose*:

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

assumes $PSimQ: P \rightsquigarrow \langle Rel \rangle Q$
and $RSimT: R \rightsquigarrow \langle Rel \rangle T$
and $RelRel': Rel \subseteq Rel'$

shows $P \oplus R \rightsquigarrow \langle Rel' \rangle Q \oplus T$
 $\langle proof \rangle$

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 $P \oplus R \rightsquigarrow \langle Rel' \rangle Q \oplus R$
 $\langle proof \rangle$

lemma *parPres*:

fixes $P :: pi$
and $Q :: pi$
and $R :: pi$
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 $Par: \bigwedge P Q R. (P, Q) \in Rel \implies (P \parallel R, Q \parallel R) \in Rel'$
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 $P \parallel R \rightsquigarrow \langle Rel' \rangle Q \parallel R$
 $\langle proof \rangle$

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

assumes $PSimQ: P \rightsquigarrow_{<Rel>} Q$
and $ResRel: \bigwedge (P::pi) (Q::pi) (x::name). (P, Q) \in Rel \implies (<\nu x>P, <\nu x>Q) \in Rel'$
and $RelRel': Rel \subseteq Rel'$
and $EqvtRel: eqvt\ Rel$
and $EqvtRel': eqvt\ Rel'$

shows $<\nu x>P \rightsquigarrow_{<Rel'>} <\nu x>Q$
 $\langle proof \rangle$

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

assumes $PSimQ: P \rightsquigarrow_{<Rel'>} Q$
and $PRelQ: (P, Q) \in Rel$
and $Sim: \bigwedge P Q. (P, Q) \in Rel \implies P \rightsquigarrow_{<Rel'>} Q$
and $RelRel': \bigwedge P Q. (P, Q) \in Rel \implies (P, Q) \in Rel'$
and $eqvtRel': eqvt\ Rel'$

shows $!P \rightsquigarrow_{<bangRel\ Rel'>} !Q$
 $\langle proof \rangle$

end

theory *Weak-Late-Bisim-SC*
imports *Weak-Late-Bisim Strong-Late-Bisim-SC*
begin

lemma *resComm*:
fixes $P :: pi$

shows $<\nu a><\nu b>P \approx <\nu b><\nu a>P$
 $\langle proof \rangle$

lemma *matchId*:
fixes $a :: \textit{name}$
and $P :: \textit{pi}$

shows $[a \curvearrowright a]P \approx P$
 $\langle \textit{proof} \rangle$

lemma *mismatchId*:
fixes $a :: \textit{name}$
and $b :: \textit{name}$
and $P :: \textit{pi}$

assumes $a \neq b$

shows $[a \neq b]P \approx P$
 $\langle \textit{proof} \rangle$

lemma *mismatchZero*:
fixes $a :: \textit{name}$
and $P :: \textit{pi}$

shows $[a \neq a]P \approx \mathbf{0}$
 $\langle \textit{proof} \rangle$

lemma *sumSym*:
fixes $P :: \textit{pi}$
and $Q :: \textit{pi}$

shows $P \oplus Q \approx Q \oplus P$
 $\langle \textit{proof} \rangle$

lemma *sumAssoc*:
fixes $P :: \textit{pi}$
and $Q :: \textit{pi}$
and $R :: \textit{pi}$

shows $(P \oplus Q) \oplus R \approx P \oplus (Q \oplus R)$
 $\langle \textit{proof} \rangle$

lemma *sumZero*:
fixes $P :: \textit{pi}$

shows $P \oplus \mathbf{0} \approx P$
 $\langle \textit{proof} \rangle$

lemma *parZero*:

fixes $P :: pi$

shows $P \parallel \mathbf{0} \approx P$

<proof>

lemma *parSym*:

fixes $P :: pi$

and $Q :: pi$

shows $P \parallel Q \approx Q \parallel P$

<proof>

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>

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>

lemma *parAssoc*:

fixes $P :: pi$

and $Q :: pi$

and $R :: pi$

shows $(P \parallel Q) \parallel R \approx P \parallel (Q \parallel R)$

<proof>

lemma *freshRes*:

fixes $P :: pi$

and $a :: name$

assumes $aFreshP: a \# P$

shows $\langle \nu a \rangle P \approx P$
 $\langle proof \rangle$

lemma *scopeExtSum*:

fixes $P :: pi$
and $Q :: pi$
and $x :: name$

assumes $x \# P$

shows $\langle \nu x \rangle (P \oplus Q) \approx P \oplus \langle \nu x \rangle Q$
 $\langle proof \rangle$

lemma *bangSC*:

fixes P

shows $!P \approx P \parallel !P$
 $\langle proof \rangle$

end

theory *Weak-Late-Sim-Pres*

imports *Weak-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 $\tau.(P) \rightsquigarrow^{\wedge} \langle Rel \rangle \tau.(Q)$
 $\langle proof \rangle$

lemma *inputPres*:

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

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

shows $a \langle x \rangle . P \rightsquigarrow^{\wedge} \langle Rel \rangle a \langle x \rangle . Q$
 $\langle proof \rangle$

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 $a\{b\}.P \rightsquigarrow^{\langle Rel \rangle} a\{b\}.Q$
 $\langle proof \rangle$

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 $RelStay$: $\bigwedge P Q a. (P, Q) \in Rel \implies ([a \frown a]P, Q) \in Rel$
and $RelRel'$: $Rel \subseteq Rel'$

shows $[a \frown b]P \rightsquigarrow^{\langle Rel' \rangle} [a \frown b]Q$
 $\langle proof \rangle$

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 $RelStay$: $\bigwedge P Q a b. [(P, Q) \in Rel; a \neq b] \implies ([a \neq b]P, Q) \in Rel$
and $RelRel'$: $Rel \subseteq Rel'$

shows $[a \neq b]P \rightsquigarrow^{\langle Rel' \rangle} [a \neq b]Q$
 $\langle proof \rangle$

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 $PSimQ$: $P \rightsquigarrow^{\wedge} \langle Rel \rangle Q$
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 $P \parallel R \rightsquigarrow^{\wedge} \langle Rel'' \rangle Q \parallel T$
 $\langle proof \rangle$

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 $PSimQ$: $P \rightsquigarrow^{\wedge} \langle Rel \rangle Q$
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 $P \parallel R \rightsquigarrow^{\wedge} \langle Rel' \rangle Q \parallel R$
 $\langle proof \rangle$

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

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

shows $\langle \nu x \rangle P \rightsquigarrow^{\hat{\langle Rel' \rangle}} \langle \nu x \rangle Q$
 $\langle proof \rangle$

lemma *resChainI*:

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

assumes *eqvtRel*: $eqvt \ Rel$

and *Res*: $\bigwedge P \ Q \ a. (P, Q) \in Rel \implies (\langle \nu a \rangle P, \langle \nu a \rangle Q) \in Rel$
and *PRelQ*: $P \rightsquigarrow^{\hat{\langle Rel \rangle}} Q$

shows $(resChain \ lst) \ P \rightsquigarrow^{\hat{\langle Rel \rangle}} (resChain \ lst) \ Q$
 $\langle proof \rangle$

lemma *bangPres*:

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

assumes *PSimQ*: $P \rightsquigarrow^{\hat{\langle Rel \rangle}} Q$

and *PRelQ*: $(P, Q) \in Rel$

and *Sim*: $\bigwedge P \ Q. (P, Q) \in Rel \implies P \rightsquigarrow^{\hat{\langle Rel \rangle}} Q$

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 $!P \rightsquigarrow^{\hat{\langle Rel' \rangle}} !Q$
 $\langle proof \rangle$

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>

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>

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>

lemma *resPres*:

fixes $P :: pi$
and $Q :: pi$
and $x :: name$

assumes *PBiSimQ*: $P \approx Q$

shows $\langle \nu x \rangle P \approx \langle \nu x \rangle Q$
<proof>

lemma *matchPres*:

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

assumes $P \approx Q$

shows $[a\curvearrowright b]P \approx [a\curvearrowright b]Q$
<proof>

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>

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

assumes  $P \approx Q$ 

shows  $P \parallel R \approx Q \parallel R$ 
<proof>

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

assumes PBisimQ:  $P \approx Q$ 

shows  $!P \approx !Q$ 
<proof>

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  $P \simeq Q$ 

shows  $\tau.(P) \simeq \tau.(Q)$ 
<proof>

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

assumes  $P \simeq Q$ 

shows  $a\{b\}.P \simeq a\{b\}.Q$ 

```

<proof>

lemma *inputPres*:

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

assumes $PSimQ: \forall y. P[x::=y] \simeq Q[x::=y]$

shows $a \langle x \rangle . P \simeq a \langle x \rangle . Q$

<proof>

lemma *matchPres*:

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

assumes $P \simeq Q$

shows $[a \frown b] P \simeq [a \frown b] Q$

<proof>

lemma *mismatchPres*:

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

assumes $P \simeq Q$

shows $[a \neq b] P \simeq [a \neq b] Q$

<proof>

lemma *sumPres*:

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

assumes $P \simeq Q$

shows $P \oplus R \simeq Q \oplus R$

<proof>

lemma *parPres*:

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

```

assumes  $P \simeq Q$ 

shows  $P \parallel R \simeq Q \parallel R$ 
⟨proof⟩

lemma resPres:
fixes  $P :: pi$ 
and  $Q :: pi$ 
and  $x :: name$ 

assumes PeqQ:  $P \simeq Q$ 

shows  $\langle \nu x \rangle P \simeq \langle \nu x \rangle Q$ 
⟨proof⟩

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

assumes  $P \simeq Q$ 

shows  $!P \simeq !Q$ 
⟨proof⟩

end

theory Early-Semantics
imports Agent
begin

declare name-fresh[simp del]

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

nominal-datatype residual = BoundOutputR name «name» pi ( $\langle - \langle \nu - \rangle - \rangle$ 
[110, 110, 110] 110)
| FreeR freeRes pi

lemma alphaBoundOutput:
fixes  $a :: name$ 
and  $x :: name$ 
and  $P :: pi$ 
and  $x' :: name$ 

assumes A1:  $x' \# P$ 

```

shows $a\langle\nu x\rangle \prec P = a\langle\nu x'\rangle \prec ((x, x') \cdot P)$
 ⟨proof⟩

declare *name-fresh*[simp]

abbreviation *Transitions-Freejudge* ($\langle - \prec - \rangle$ [80, 80] 80) where $\alpha \prec P' \equiv (\text{FreeR } \alpha P')$

inductive *TransitionsEarly* :: *pi* \Rightarrow *residual* \Rightarrow *bool* ($\langle - \mapsto - \rangle$ [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*: $\llbracket P \mapsto V \rrbracket \Longrightarrow [b \frown b]P \mapsto V$
 | *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*

⟨proof⟩

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** $c \# P'$
 $\langle proof \rangle$

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 $c \neq a$
 $\langle proof \rangle$

lemma *freshBoundOutputAction*:
fixes $P :: pi$
and $a :: name$
and $x :: name$
and $P' :: pi$
and $c :: name$

assumes $P \mapsto a \langle \nu x \rangle \prec P'$
and $c \# P$

shows $c \neq a$
 $\langle proof \rangle$

lemmas *freshAction* = *freshOutputAction* *freshInputAction* *freshBoundOutputAction*

lemma *freshInputTransition*:
fixes $P :: pi$
and $a :: name$
and $u :: name$
and $P' :: pi$
and $c :: name$

assumes $P \mapsto a\langle u \rangle \prec P'$
and $c \# P$
and $c \neq u$

shows $c \# P'$
 $\langle proof \rangle$

lemma *freshBoundOutputTransition*:

fixes $P :: pi$
and $a :: name$
and $x :: name$
and $P' :: pi$
and $c :: name$

assumes $P \mapsto a\langle \nu x \rangle \prec P'$
and $c \# P$
and $c \neq x$

shows $c \# P'$
 $\langle proof \rangle$

lemma *freshTauTransition*:

fixes $P :: pi$
and $P' :: pi$
and $c :: name$

assumes $P \mapsto \tau \prec P'$
and $c \# P$

shows $c \# P'$
 $\langle proof \rangle$

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 $c \# P'$
 $\langle proof \rangle$

lemmas *freshTransition = freshInputTransition freshOutputAction freshFreeTransition*

freshBoundOutputTransition freshTauTransition

lemma *substTrans[simp]*: $b \# P \implies ((P::pi)[a::=b])[b::=c] = P[a::=c]$
 ⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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 $\bigwedge a x P u C. \llbracket x \# C; x \neq u; x \neq a \rrbracket \implies F C (a \langle x \rangle . P) a u (P[x::=u])$
and $\bigwedge P a u P' b C. \llbracket P \mapsto a \langle u \rangle \prec P'; \bigwedge C. F C P a u P' \rrbracket \implies F C (\llbracket b \dot{\sim} b \rrbracket P)$
 $a u P'$
and $\bigwedge P a u P' b c C. \llbracket P \mapsto a \langle u \rangle \prec P'; \bigwedge C. F C P a u P'; b \neq c \rrbracket \implies F$
 $C (\llbracket b \neq c \rrbracket P) a u P'$
and $\bigwedge P a u P' Q C. \llbracket P \mapsto a \langle u \rangle \prec P'; \bigwedge C. F C P a u P' \rrbracket \implies F C (P \oplus$
 $Q) a u P'$
and $\bigwedge Q a u Q' P C. \llbracket Q \mapsto a \langle u \rangle \prec Q'; \bigwedge C. F C Q a u Q' \rrbracket \implies F C (P$
 $\oplus Q) a u Q'$
and $\bigwedge P a u P' Q C. \llbracket P \mapsto a \langle u \rangle \prec P'; \bigwedge C. F C P a u P' \rrbracket \implies F C (P \parallel$
 $Q) a u (P' \parallel Q)$
and $\bigwedge Q a u Q' P C. \llbracket Q \mapsto a \langle u \rangle \prec Q'; \bigwedge C. F C Q a u Q' \rrbracket \implies F C (P \parallel$
 $Q) a u (P' \parallel Q')$
and $\bigwedge P a u P' x C. \llbracket P \mapsto a \langle u \rangle \prec P'; x \neq a; x \neq u; x \# C; \bigwedge C. F C P a$
 $u P' \rrbracket \implies F C (\langle \nu x \rangle P) a u (\langle \nu x \rangle P')$
and $\bigwedge P a u P' C. \llbracket P \parallel !P \mapsto a \langle u \rangle \prec P'; \bigwedge C. F C (P \parallel !P) a u P' \rrbracket \implies$
 $F C (!P) a u P'$

shows $F C P a u P'$
 $\langle proof \rangle$

lemma *inputAlpha*:

assumes $P \mapsto a \langle u \rangle \prec P'$
and $u \# P$
and $r \# P'$

shows $P \mapsto a \langle r \rangle \prec ((u, r) \cdot P')$
 $\langle proof \rangle$

lemma *Close1*:

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

assumes $P \mapsto a \langle x \rangle \prec P'$
and $Q \mapsto a \langle \nu x \rangle \prec Q'$
and $x \# P$

shows $P \parallel Q \mapsto \tau \prec \langle \nu x \rangle (P' \parallel Q')$
 $\langle proof \rangle$

lemma *Close2*:

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

and $Q :: pi$
and $Q' :: pi$

assumes $P \mapsto a \langle \nu x \rangle \prec P'$
and $Q \mapsto a \langle x \rangle \prec Q'$
and $x \# Q$

shows $P \parallel Q \mapsto \tau \prec \langle \nu x \rangle (P' \parallel Q')$
 $\langle proof \rangle$

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 $\langle \nu y \rangle P \mapsto a \langle \nu x \rangle \prec (\langle \nu y \rangle P')$
 $\langle proof \rangle$

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 $\bigwedge P a b P' c C. \llbracket P \mapsto OutputR a b \prec P'; \bigwedge C. F C P a b P \rrbracket \Longrightarrow F C$
 $([c \frown c]P) a b P'$
and $\bigwedge P a b P' c d C. \llbracket P \mapsto OutputR a b \prec P'; \bigwedge C. F C P a b P'; c \neq d \rrbracket$
 $\Longrightarrow F C ([c \neq d]P) a b P'$
and $\bigwedge P a b P' Q C. \llbracket P \mapsto OutputR a b \prec P'; \bigwedge C. F C P a b P \rrbracket \Longrightarrow F C$
 $(P \oplus Q) a b P'$
and $\bigwedge Q a b Q' P C. \llbracket Q \mapsto OutputR a b \prec Q'; \bigwedge C. F C Q a b Q \rrbracket \Longrightarrow F$
 $C (P \oplus Q) a b Q'$
and $\bigwedge P a b P' Q C. \llbracket P \mapsto OutputR a b \prec P'; \bigwedge C. F C P a b P \rrbracket \Longrightarrow F C$
 $(P \parallel Q) a b (P' \parallel Q)$
and $\bigwedge Q a b Q' P C. \llbracket Q \mapsto OutputR a b \prec Q'; \bigwedge C. F C Q a b Q \rrbracket \Longrightarrow F$
 $C (P \parallel Q) a b (P' \parallel Q')$
and $\bigwedge P a b P' x C. \llbracket P \mapsto OutputR a b \prec P'; x \neq a; x \neq b; x \# C; \bigwedge C. F$
 $C P a b P \rrbracket \Longrightarrow$

$F C (\langle \nu x \rangle P) a b (\langle \nu x \rangle P')$

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 $F C P a b P'$
 $\langle \text{proof} \rangle$

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 $x \text{Fresh} P: x \# P$
and $c \text{Match}: \bigwedge P a x P' b C. \llbracket P \mapsto a \langle \nu x \rangle \prec P'; \bigwedge C. F C P a x P' \rrbracket \implies$
 $F C ([b \sim b] P) a x P'$
and $c \text{Mismatch}: \bigwedge P a x P' b c C. \llbracket P \mapsto a \langle \nu x \rangle \prec P'; \bigwedge C. F C P a x P';$
 $b \neq c \rrbracket \implies F C ([b \neq c] P) a x P'$
and $c \text{Open}: \bigwedge P a x P' C. \llbracket P \mapsto (\text{OutputR } a x) \prec P'; a \neq x \rrbracket \implies F C$
 $(\langle \nu x \rangle P) a x P'$
and $c \text{Sum1}: \bigwedge P Q a x P' C. \llbracket P \mapsto a \langle \nu x \rangle \prec P'; \bigwedge C. F C P a x P' \rrbracket \implies$
 $F C (P \oplus Q) a x P'$
and $c \text{Sum2}: \bigwedge P Q a x Q' C. \llbracket Q \mapsto a \langle \nu x \rangle \prec Q'; \bigwedge C. F C Q a x Q' \rrbracket$
 $\implies F C (P \oplus Q) a x Q'$
and $c \text{Par1B}: \bigwedge P P' Q a x C. \llbracket P \mapsto a \langle \nu x \rangle \prec P'; x \# Q; \bigwedge C. F C P a x$
 $P' \rrbracket \implies$
 $F C (P \parallel Q) a x (P' \parallel Q)$
and $c \text{Par2B}: \bigwedge P Q Q' a x C. \llbracket Q \mapsto a \langle \nu x \rangle \prec Q'; x \# P; \bigwedge C. F C Q a x$
 $Q' \rrbracket \implies$
 $F C (P \parallel Q) a x (P \parallel Q')$
and $c \text{ResB}: \bigwedge P P' a x y C. \llbracket P \mapsto a \langle \nu x \rangle \prec P'; y \neq a; y \neq x; y \# C;$
 $\bigwedge C. F C P a x P' \rrbracket \implies F C (\langle \nu y \rangle P) a x (\langle \nu y \rangle P')$
and $c \text{Bang}: \bigwedge P a x P' C. \llbracket P \parallel !P \mapsto a \langle \nu x \rangle \prec P'; \bigwedge C. F C (P \parallel !P) a$
 $x P' \rrbracket \implies$
 $F C (!P) a x P'$

shows $F C P a x P'$
 $\langle \text{proof} \rangle$

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

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

shows $F C P P'$
<proof>

inductive *bangPred* :: $pi \Rightarrow pi \Rightarrow bool$
where
aux1: $bangPred P (!P)$
aux2: $bangPred P (P \parallel !P)$

inductive-cases *tauCases*'[*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*]: $[a \sim b]P \mapsto Rs$
inductive-cases *mismatchCases*'[*simplified pi.inject residual.inject*]: $[a \neq b]P \mapsto Rs$
inductive-cases *sumCases*'[*simplified pi.inject residual.inject*]: $P \oplus Q \mapsto Rs$
inductive-cases *parCasesB*'[*simplified pi.distinct residual.distinct*]: $A \parallel B \mapsto b < \nu y > \prec A'$
inductive-cases *parCasesF*'[*simplified pi.distinct residual.distinct*]: $P \parallel Q \mapsto \alpha \prec P'$
inductive-cases *resCasesB*'[*simplified pi.distinct residual.distinct*]: $< \nu x > A \mapsto a < \nu y > \prec A'$
inductive-cases *resCasesF*'[*simplified pi.distinct residual.distinct*]: $< \nu x > A \mapsto \alpha \prec A'$

lemma *tauCases*:
fixes $P :: pi$
and $\alpha :: freeRes$
and $P' :: pi$

```

assumes  $\tau.(P) \mapsto \alpha \prec P'$ 
and  $Prop(\tau) P$ 

shows  $Prop \alpha P'$ 
<proof>

lemma inputCases[consumes 1, case-names cInput]:
fixes  $a :: name$ 
and  $x :: name$ 
and  $P :: pi$ 
and  $P' :: pi$ 

assumes Input:  $a \langle x \rangle . P \mapsto \alpha \prec P'$ 
and  $A: \bigwedge u. Prop(a \langle u \rangle) (P[x ::= u])$ 

shows  $Prop \alpha P'$ 
<proof>

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

assumes  $a\{b\}.P \mapsto \alpha \prec P'$ 
and  $Prop(OutputR\ a\ b)\ P$ 

shows  $Prop \alpha P'$ 
<proof>

lemma zeroTrans[dest]:
fixes  $Rs :: residual$ 

assumes  $0 \mapsto e\ Rs$ 

shows False
<proof>

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

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

shows False
<proof>

lemma matchCases[consumes 1, case-names Match]:

```

```

fixes  $a :: name$ 
and  $b :: name$ 
and  $P :: pi$ 
and  $Rs :: residual$ 
and  $F :: name \Rightarrow name \Rightarrow bool$ 

assumes  $Trans: [a \sim b]P \mapsto Rs$ 
and  $cMatch: P \mapsto Rs \implies F a a$ 

shows  $F a b$ 
<proof>

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  $F a b$ 
<proof>

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  $F$ 
<proof>

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>

lemma *parCasesOutput*[*consumes 1, case-names Par1 Par2*]:

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

assumes $P \parallel Q \mapsto a[b] \prec PQ'$
and $\bigwedge P'. \llbracket P \mapsto a[b] \prec P' \rrbracket \implies F (P' \parallel Q)$
and $\bigwedge Q'. \llbracket Q \mapsto a[b] \prec Q' \rrbracket \implies F (P \parallel Q')$

shows $F PQ'$

<proof>

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*: $P \parallel Q \mapsto a\langle b \rangle \prec PQ'$
and *icPar1F*: $\bigwedge P'. \llbracket P \mapsto a\langle b \rangle \prec P' \rrbracket \implies F (P' \parallel Q)$
and *icPar2F*: $\bigwedge Q'. \llbracket Q \mapsto a\langle b \rangle \prec Q' \rrbracket \implies F (P \parallel Q')$

shows $F PQ'$

<proof>

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

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 $F \alpha PQ'$
 $\langle proof \rangle$

lemma *resCasesF*[*consumes 2, case-names Res*]:

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

assumes *Trans*: $\langle \nu x \rangle P \mapsto \alpha \prec RP'$
and *xFreshAlpha*: $x \# \alpha$
and *rcResF*: $\bigwedge P'. P \mapsto \alpha \prec P' \implies F (\langle \nu x \rangle P')$

shows $F RP'$
 $\langle proof \rangle$

lemma *resCasesB*[*consumes 2, case-names Open Res*]:

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

assumes *Trans*: $\langle \nu y \rangle P \mapsto a \langle \nu x \rangle \prec RP'$
and *xineqy*: $x \neq y$
and *rcOpen*: $\bigwedge P'. \llbracket P \mapsto (OutputR\ a\ y) \prec P'; a \neq y \rrbracket \implies F ([x, y] \cdot P')$
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 $F RP'$
 $\langle proof \rangle$

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*: $\bigwedge a\ x\ P'\ C. \llbracket P \mapsto a \langle \nu x \rangle \prec P'; x \# P; x \# C \rrbracket \implies F\ C\ (P \parallel !P)\ (a \langle \nu x \rangle \prec (P' \parallel !P))$
and *cPar1F*: $\bigwedge (\alpha::freeRes)\ (P'::pi)\ C. \llbracket P \mapsto \alpha \prec P' \rrbracket \implies F\ C\ (P \parallel !P)\ (\alpha \prec P' \parallel !P)$
and *cPar2B*: $\bigwedge a\ x\ P'\ C. \llbracket !P \mapsto a \langle \nu x \rangle \prec P'; x \# P; x \# C; \bigwedge C. F\ C\ (!P)\ (a \langle \nu x \rangle \prec P') \rrbracket \implies F\ C\ (P \parallel !P)\ (a \langle \nu x \rangle \prec (P \parallel P'))$
and *cPar2F*: $\bigwedge \alpha\ P'\ C. \llbracket !P \mapsto \alpha \prec P'; \bigwedge C. F\ C\ (!P)\ (\alpha \prec P') \rrbracket \implies F\ C\ (P \parallel !P)\ (\alpha \prec P \parallel P')$
and *cComm1*: $\bigwedge a\ P'\ b\ P''\ C. \llbracket P \mapsto a \langle b \rangle \prec P'; !P \mapsto (OutputR\ a\ b) \prec$

$P''; \bigwedge C. F C (!P) ((OutputR\ a\ b) \prec P'') \implies$
 $F C (P \parallel !P) (\tau \prec P' \parallel P'')$
and $cComm2: \bigwedge a\ b\ P'\ P''\ C. \llbracket P \mapsto (OutputR\ a\ b) \prec P'; !P \mapsto a \prec b \prec P''; \bigwedge C. F C (!P) (a \prec b \prec P'') \rrbracket \implies$
 $F C (P \parallel !P) (\tau \prec P' \parallel P'')$
and $cClose1: \bigwedge a\ x\ P'\ P''\ C. \llbracket P \mapsto a \prec x \prec P'; !P \mapsto a \prec \nu x \prec P''; x \# P; x \# C; \bigwedge C. F C (!P) (a \prec \nu x \prec P'') \rrbracket \implies$
 $F C (P \parallel !P) (\tau \prec \langle \nu x \rangle (P' \parallel P''))$
and $cClose2: \bigwedge a\ x\ P'\ P''\ C. \llbracket P \mapsto a \prec \nu x \prec P'; !P \mapsto a \prec x \prec P''; x \# P; x \# C; \bigwedge C. F C (!P) (a \prec x \prec P'') \rrbracket \implies$
 $F C (P \parallel !P) (\tau \prec \langle \nu x \rangle (P' \parallel P''))$
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 $F C (!P)\ Rs$
 $\langle proof \rangle$

end

theory *Strong-Early-Sim*
imports *Early-Semantics Rel*
begin

definition *strongSimEarly* :: $pi \Rightarrow (pi \times pi)\ set \Rightarrow pi \Rightarrow bool$ ($\prec \rightsquigarrow [-] \rightarrow [80, 80, 80] 80$) **where**
 $P \rightsquigarrow [Rel]\ Q \equiv (\forall a\ y\ Q'. Q \mapsto a \prec \nu y \prec Q' \longrightarrow y \# P \longrightarrow (\exists P'. P \mapsto a \prec \nu y \prec P' \wedge (P', Q') \in Rel)) \wedge$
 $(\forall \alpha\ Q'. Q \mapsto \alpha \prec Q' \longrightarrow (\exists P'. P \mapsto \alpha \prec P' \wedge (P', Q') \in Rel))$

lemma *monotonic*:

fixes $A :: (pi \times pi)\ set$
and $B :: (pi \times pi)\ set$
and $P :: pi$
and $P' :: pi$

assumes $P \rightsquigarrow [A]\ P'$
and $A \subseteq B$

shows $P \rightsquigarrow [B]\ P'$
 $\langle proof \rangle$

lemma *freshUnit[simp]*:
fixes $y :: name$

shows $y \# ()$
 $\langle proof \rangle$

lemma *simCasesCont[consumes 1, case-names Bound Free]*:
fixes $P :: pi$

and $Q :: pi$
and $Rel :: (pi \times pi) \text{ set}$
and $C :: 'a::fs\text{-name}$

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

shows $P \rightsquigarrow[Rel] Q$
 $\langle proof \rangle$

lemma $simCases[consumes \ 0, \ \text{case-names } Bound \ Free]:$

fixes $P :: pi$
and $Q :: pi$
and $Rel :: (pi \times pi) \text{ set}$
and $C :: 'a::fs\text{-name}$

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

shows $P \rightsquigarrow[Rel] Q$
 $\langle proof \rangle$

lemma $elim:$

fixes $P :: pi$
and $Rel :: (pi \times pi) \text{ set}$
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 $Q \mapsto \alpha \prec Q' \implies \exists P'. P \mapsto \alpha \prec P' \wedge (P', Q') \in Rel$
 $\langle proof \rangle$

lemma $eqvtI:$

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

assumes $Sim: P \rightsquigarrow[Rel] Q$
and $RelRel': Rel \subseteq Rel'$
and $EqvtRel': eqvt \ Rel'$

shows $(perm \cdot P) \rightsquigarrow_{[Rel']} (perm \cdot Q)$
 $\langle proof \rangle$

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

assumes $Id \subseteq Rel$

shows $P \rightsquigarrow_{[Rel]} P$
 $\langle proof \rangle$

lemmas *fresh-prod*[*simp*]

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 $Trans: Rel\ O\ Rel' \subseteq Rel''$

shows $P \rightsquigarrow_{[Rel'']} R$
 $\langle proof \rangle$

end

theory *Strong-Early-Bisim*
imports *Strong-Early-Sim*
begin

lemma *monoAux*: $A \subseteq B \implies P \rightsquigarrow_{[A]} Q \longrightarrow P \rightsquigarrow_{[B]} Q$
 $\langle proof \rangle$

coinductive-set *bisim* :: $(pi \times pi)$ 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 *bisimCoinductAux*[*case-names bisim, case-conclusion StrongBisim step, consumes 1*]:

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

and step: $\bigwedge P Q. (P, Q) \in X \implies P \rightsquigarrow[(X \cup \text{bisim})] Q \wedge (Q, P) \in \text{bisim} \cup X$

shows $P \sim Q$

<proof>

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

fixes $P :: pi$

and $Q :: pi$

assumes $(P, Q) \in X$

and $\bigwedge R S. (R, S) \in X \implies R \rightsquigarrow[(X \cup \text{bisim})] S$

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

shows $P \sim Q$

<proof>

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 $P \sim Q$

<proof>

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 $P \sim Q$

<proof>

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

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

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

<proof>

lemma *bisimE*:

fixes $P :: pi$

and $Q :: pi$

```

assumes  $P \sim Q$ 

shows  $P \rightsquigarrow[bisim] Q$ 
and  $Q \sim P$ 
⟨proof⟩

lemma bisimClosed[eqvt]:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $p :: name prm$ 

  assumes  $P \sim Q$ 

  shows  $(p \cdot P) \sim (p \cdot Q)$ 
  ⟨proof⟩

lemma eqvt[simp]:
  shows eqvt bisim
  ⟨proof⟩

lemma reflexive:
  fixes  $P :: pi$ 

  shows  $P \sim P$ 
  ⟨proof⟩

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

  assumes PBiSimQ:  $P \sim Q$ 
  and QBiSimR:  $Q \sim R$ 

  shows  $P \sim R$ 
  ⟨proof⟩

end

theory Strong-Early-Bisim-Subst
  imports Strong-Early-Bisim
begin

abbreviation StrongCongEarlyJudge (infixr  $\langle \sim^s \rangle$  65) where  $P \sim^s Q \equiv (P, Q) \in (substClosed\ bisim)$ 

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

```

```

assumes  $P \sim^s Q$ 

shows  $P \sim Q$ 
<proof>

lemma eqvt:
  shows eqvt (substClosed bisim)
<proof>

lemma eqvtI:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $perm :: name prm$ 

  assumes  $P \sim^s Q$ 

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

lemma reflexive:
  fixes  $P :: pi$ 

  shows  $P \sim^s P$ 
<proof>

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

  assumes  $P \sim^s Q$ 

  shows  $Q \sim^s P$ 
<proof>

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

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

  shows  $P \sim^s R$ 
<proof>

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

```

```

and s :: (name × name) list

assumes P ~s Q

shows P[<s>] ~s Q[<s>]
⟨proof⟩

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⟩

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
⟨proof⟩

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

assumes PRelQ: (P, Q) ∈ Rel

shows a{b}.P ~→[Rel] a{b}.Q
⟨proof⟩

```

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 $[a \frown b]P \rightsquigarrow[Rel'] [a \frown b]Q$
 $\langle proof \rangle$

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 $[a \neq b]P \rightsquigarrow[Rel'] [a \neq b]Q$
 $\langle proof \rangle$

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

assumes $P \rightsquigarrow[Rel] Q$
and $C1$: $Id \subseteq Rel'$
and $Rel \subseteq Rel'$

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

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 $Rel'' :: (pi \times pi) \text{ set}$
assumes $PSimQ: P \rightsquigarrow[Rel] Q$
and $RSimT: R \rightsquigarrow[Rel'] S$
and $PRelQ: (P, Q) \in Rel$
and $RRel'T: (R, S) \in Rel'$
and $Par: \bigwedge P' Q' R' S'. [(P', Q') \in Rel; (R', S') \in Rel'] \implies (P' \parallel R', Q' \parallel S') \in Rel''$
and $Res: \bigwedge S T x. (S, T) \in Rel'' \implies (\langle \nu x \rangle S, \langle \nu x \rangle T) \in Rel''$
shows $P \parallel R \rightsquigarrow[Rel''] Q \parallel S$
 $\langle proof \rangle$

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

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 $P \parallel R \rightsquigarrow[Rel'] Q \parallel R$
 $\langle proof \rangle$

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 $PSimQ: P \rightsquigarrow[Rel] Q$
and $ResSet: \bigwedge (R::pi) (S::pi) (y::name). (R, S) \in Rel \implies (\langle \nu y \rangle R, \langle \nu y \rangle S) \in Rel'$
and $RelRel': Rel \subseteq Rel'$
and $EqvtRel: eqvt Rel$
and $EqvtRel': eqvt Rel'$

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

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  $(resChain\ lst)\ P \rightsquigarrow[Rel] (resChain\ lst)\ Q$ 
 $\langle proof \rangle$ 

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  $!P \rightsquigarrow[bangRel\ Rel] !Q$ 
 $\langle proof \rangle$ 

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  $\tau.(P) \sim \tau.(Q)$ 
 $\langle proof \rangle$ 

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>

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>

lemma *matchPres*:

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

assumes $P \sim Q$

shows $[a\curvearrowright b]P \sim [a\curvearrowright b]Q$
<proof>

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>

lemma *sumPres*:

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

assumes $P \sim Q$

shows $P \oplus R \sim Q \oplus R$
<proof>

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>

lemma parPres:
fixes  $P :: pi$ 
and  $Q :: pi$ 
and  $R :: pi$ 
and  $T :: pi$ 

assumes  $P \sim Q$ 

shows  $P \parallel R \sim Q \parallel R$ 
<proof>

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

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

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

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

assumes PBiSimQ:  $P \sim Q$ 

shows  $!P \sim !Q$ 
<proof>

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 $\tau.(P) \sim^s \tau.(Q)$
<proof>

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>

lemma *outputPres*:
fixes $P :: pi$
and $Q :: pi$

assumes $P \sim^s Q$

shows $a\{b\}.P \sim^s a\{b\}.Q$
<proof>

lemma *matchPres*:
fixes $P :: pi$
and $Q :: pi$
and $a :: name$
and $b :: name$

assumes $P \sim^s Q$

shows $[a \frown b]P \sim^s [a \frown b]Q$
<proof>

lemma *mismatchPres*:
fixes $P :: pi$
and $Q :: pi$
and $a :: name$
and $b :: name$

assumes $P \sim^s Q$

shows $[a \neq b]P \sim^s [a \neq b]Q$
<proof>

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

```

assumes  $P \sim^s Q$ 

shows  $P \oplus R \sim^s Q \oplus R$ 
<proof>

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

assumes  $P \sim^s Q$ 

shows  $P \parallel R \sim^s Q \parallel R$ 
<proof>

lemma resPres:
fixes  $P :: pi$ 
and  $Q :: pi$ 
and  $x :: name$ 

assumes  $P \text{ eq } Q: P \sim^s Q$ 

shows  $\langle \nu x \rangle P \sim^s \langle \nu x \rangle Q$ 
<proof>

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

assumes  $P \sim^s Q$ 

shows  $!P \sim^s !Q$ 
<proof>

end

theory Early-Tau-Chain
imports Early-Semantics
begin

abbreviation tauChain ::  $pi \Rightarrow pi \Rightarrow bool$  ( $\langle \cdot \rangle \Longrightarrow_{\tau} \cdot \rightarrow [80, 80] 80$ )
where  $P \Longrightarrow_{\tau} P' \equiv (P, P') \in \{(P, P') \mid P \text{ P}'. P \mapsto_{\tau} \prec P'\}^*$ 

lemma tauActTauChain:
fixes  $P :: pi$ 
and  $P' :: pi$ 

assumes  $P \mapsto_{\tau} \prec P'$ 

```

shows $P \Longrightarrow_{\tau} P'$
 $\langle proof \rangle$

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 $P \mapsto_{\tau} \prec P' \Longrightarrow P' \Longrightarrow_{\tau} P'' \Longrightarrow P \Longrightarrow_{\tau} P''$
 $\langle proof \rangle$

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 $F P'$
 $\langle proof \rangle$

lemma *eqvtChainI*:

fixes $P :: pi$
and $P' :: pi$
and $perm :: name prm$

assumes $P \Longrightarrow_{\tau} P'$

shows $(perm \cdot P) \Longrightarrow_{\tau} (perm \cdot P')$
 $\langle proof \rangle$

lemma *eqvtChainE*:

fixes $perm :: name prm$
and $P :: pi$
and $P' :: pi$

assumes *Trans*: $(perm \cdot P) \Longrightarrow_{\tau} (perm \cdot P')$

shows $P \Longrightarrow_{\tau} P'$
 $\langle proof \rangle$

lemma *eqvtChainEq*:

fixes $P :: pi$
and $P' :: pi$
and $perm :: name prm$

shows $P \Longrightarrow_{\tau} P' = (\text{perm} \cdot P) \Longrightarrow_{\tau} (\text{perm} \cdot P')$
<proof>

lemma *freshChain*:

fixes $P :: pi$
and $P' :: pi$
and $x :: name$

assumes $P \Longrightarrow_{\tau} P'$
and $x \# P$

shows $x \# P'$
<proof>

lemma *matchChain*:

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

assumes $P \Longrightarrow_{\tau} P'$
and $P \neq P'$

shows $[b \frown b]P \Longrightarrow_{\tau} P'$
<proof>

lemma *mismatchChain*:

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

assumes $PChain: P \Longrightarrow_{\tau} P'$
and $a \text{ineq} b: a \neq b$
and $P \text{ineq} P': P \neq P'$

shows $[a \neq b]P \Longrightarrow_{\tau} P'$
<proof>

lemma *sum1Chain*:

fixes $P :: pi$
and $P' :: pi$
and $Q :: pi$

assumes $P \Longrightarrow_{\tau} P'$
and $P \neq P'$

shows $P \oplus Q \Longrightarrow_{\tau} P'$
<proof>

lemma *sum2Chain*:

fixes $P :: pi$
and $Q :: pi$
and $Q' :: pi$

assumes $Q \Longrightarrow_{\tau} Q'$
and $Q \neq Q'$

shows $P \oplus Q \Longrightarrow_{\tau} Q'$
<proof>

lemma *Par1Chain*:

fixes $P :: pi$
and $P' :: pi$
and $Q :: pi$

assumes $P \Longrightarrow_{\tau} P'$

shows $P \parallel Q \Longrightarrow_{\tau} P' \parallel Q$
<proof>

lemma *Par2Chain*:

fixes $P :: pi$
and $Q :: pi$
and $Q' :: pi$

assumes $Q \Longrightarrow_{\tau} Q'$

shows $P \parallel Q \Longrightarrow_{\tau} P \parallel Q'$
<proof>

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>

lemma *ResChain*:

fixes $P :: pi$
and $P' :: pi$
and $a :: name$

assumes $P \Longrightarrow_{\tau} P'$

shows $\langle \nu a \rangle P \Longrightarrow_{\tau} \langle \nu a \rangle P'$
 $\langle proof \rangle$

lemma *substChain*:

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

assumes $PTrans: P[x::=b] \Longrightarrow_{\tau} P'$

shows $P[x::=b] \Longrightarrow_{\tau} P'[x::=b]$
 $\langle proof \rangle$

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 $!P \Longrightarrow_{\tau} P'$
 $\langle proof \rangle$

end

theory *Weak-Early-Step-Semantics*

imports *Early-Tau-Chain*

begin

lemma *inputSupportDerivative*:

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

shows $(supp P') - \{x\} \subseteq supp P$
 $\langle proof \rangle$

lemma *outputSupportDerivative*:

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

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

shows $(supp P') \subseteq ((supp P)::name set)$
 $\langle proof \rangle$

lemma *boundOutputSupportDerivative*:

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

shows $(\text{supp } P') - \{x\} \subseteq \text{supp } P$
 $\langle \text{proof} \rangle$

lemma *tauSupportDerivative*:

assumes $P \mapsto \tau \prec P'$

shows $((\text{supp } P') :: \text{name set}) \subseteq \text{supp } P$
 $\langle \text{proof} \rangle$

lemma *tauChainSupportDerivative*:

fixes $P :: pi$
and $P' :: pi$

assumes $P \Longrightarrow_{\tau} P'$

shows $((\text{supp } P') :: \text{name set}) \subseteq (\text{supp } P)$
 $\langle \text{proof} \rangle$

definition *outputTransition* :: $pi \Rightarrow \text{name} \Rightarrow \text{name} \Rightarrow pi \Rightarrow \text{bool}$ ($\prec - \Longrightarrow - \langle \nu - \rangle \prec$
 $\rightarrow [80, 80, 80, 80] 80$)

where $P \Longrightarrow a \langle \nu x \rangle \prec P' \equiv \exists P''' P''. P \Longrightarrow_{\tau} P''' \wedge P''' \mapsto a \langle \nu x \rangle \prec P'' \wedge P'' \Longrightarrow_{\tau} P'$

definition *freeTransition* :: $pi \Rightarrow \text{freeRes} \Rightarrow pi \Rightarrow \text{bool}$ ($\prec - \Longrightarrow - \prec \rightarrow [80, 80, 80]$
 80)

where $P \Longrightarrow_{\alpha} \prec P' \equiv \exists P''' P''. P \Longrightarrow_{\tau} P''' \wedge P''' \mapsto_{\alpha} \prec P'' \wedge P'' \Longrightarrow_{\tau} P'$

lemma *transitionI*:

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

shows $\llbracket P \Longrightarrow_{\tau} P'''; P''' \mapsto_{\alpha} \prec P''; P'' \Longrightarrow_{\tau} P' \rrbracket \Longrightarrow P \Longrightarrow_{\alpha} \prec P'$
and $\llbracket P \Longrightarrow_{\tau} P'''; P''' \mapsto a \langle \nu x \rangle \prec P''; P'' \Longrightarrow_{\tau} P' \rrbracket \Longrightarrow P \Longrightarrow a \langle \nu x \rangle \prec P'$
 $\langle \text{proof} \rangle$

lemma *transitionE*:

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

and $x :: name$
shows $P \Longrightarrow \alpha \prec P' \Longrightarrow (\exists P'' P'''. P \Longrightarrow_{\tau} P'' \wedge P'' \mapsto \alpha \prec P''' \wedge P''' \Longrightarrow_{\tau} P')$
and $P \Longrightarrow a \langle \nu x \rangle \prec P' \Longrightarrow \exists P'' P'''. P \Longrightarrow_{\tau} P'' \wedge P'' \mapsto a \langle \nu x \rangle \prec P'' \wedge P'' \Longrightarrow_{\tau} P'$
 $\langle proof \rangle$

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 $P \Longrightarrow a \langle \nu y \rangle \prec ([x, y] \cdot P')$
 $\langle proof \rangle$

lemma *singleActionChain*:

fixes $P :: pi$
and $R_s :: residual$

shows $P \mapsto a \langle \nu x \rangle \prec P' \Longrightarrow P \Longrightarrow a \langle \nu x \rangle \prec P'$
and $P \mapsto \alpha \prec P' \Longrightarrow P \Longrightarrow \alpha \prec P'$
 $\langle proof \rangle$

lemma *Tau*:

fixes $P :: pi$

shows $\tau.(P) \Longrightarrow \tau \prec P$
 $\langle proof \rangle$

lemma *Input*:

fixes $a :: name$
and $x :: name$
and $u :: name$
and $P :: pi$

shows $a \langle x \rangle . P \Longrightarrow a \langle u \rangle \prec P[x ::= u]$
 $\langle proof \rangle$

lemma *Output*:

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

shows $a\{b\}.P \Longrightarrow a[b] \prec P$
 ⟨proof⟩

lemma *Match*:

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

shows $P \Longrightarrow b\langle\nu x\rangle \prec P' \Longrightarrow [a\dot{\wedge}a]P \Longrightarrow b\langle\nu x\rangle \prec P'$
and $P \Longrightarrow \alpha \prec P' \Longrightarrow [a\dot{\wedge}a]P \Longrightarrow \alpha \prec P'$
 ⟨proof⟩

lemma *Mismatch*:

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

shows $\llbracket P \Longrightarrow c\langle\nu x\rangle \prec P'; a \neq b \rrbracket \Longrightarrow [a\neq b]P \Longrightarrow c\langle\nu x\rangle \prec P'$
and $\llbracket P \Longrightarrow \alpha \prec P'; a \neq b \rrbracket \Longrightarrow [a\neq b]P \Longrightarrow \alpha \prec P'$
 ⟨proof⟩

lemma *Open*:

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

assumes $PTrans$: $P \Longrightarrow a[b] \prec P'$
and $a \neq b$

shows $\langle\nu b\rangle P \Longrightarrow a\langle\nu b\rangle \prec P'$
 ⟨proof⟩

lemma *Sum1*:

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

shows $P \Longrightarrow a\langle\nu x\rangle \prec P' \Longrightarrow P \oplus Q \Longrightarrow a\langle\nu x\rangle \prec P'$

and $P \Longrightarrow \alpha \prec P' \Longrightarrow P \oplus Q \Longrightarrow \alpha \prec P'$
 ⟨proof⟩

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⟩

lemma *Par1B*:

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

assumes *PTrans*: $P \Longrightarrow a \langle \nu x \rangle \prec P'$
and $x \# Q$

shows $P \parallel Q \Longrightarrow a \langle \nu x \rangle \prec (P' \parallel Q)$
 ⟨proof⟩

lemma *Par1F*:

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

assumes *PTrans*: $P \Longrightarrow \alpha \prec P'$

shows $P \parallel Q \Longrightarrow \alpha \prec (P' \parallel Q)$
 ⟨proof⟩

lemma *Par2B*:

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

assumes *QTrans*: $Q \Longrightarrow a \langle \nu x \rangle \prec Q'$
and $x \# P$

shows $P \parallel Q \Longrightarrow a \langle \nu x \rangle \prec (P \parallel Q')$
<proof>

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>

lemma *Comm1*:

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

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

shows $P \parallel Q \Longrightarrow \tau \prec P' \parallel Q'$
<proof>

lemma *Comm2*:

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

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

shows $P \parallel Q \Longrightarrow \tau \prec P' \parallel Q'$
<proof>

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 $P \parallel Q \Longrightarrow \tau \prec \langle \nu x \rangle (P' \parallel Q')$
 $\langle proof \rangle$

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 $P \parallel Q \Longrightarrow \tau \prec \langle \nu x \rangle (P' \parallel Q')$
 $\langle proof \rangle$

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 $\langle \nu x \rangle P \Longrightarrow \alpha \prec \langle \nu x \rangle P'$
 $\langle proof \rangle$

lemma *ResB*:

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 $y \neq a$
and $y \neq x$

shows $\langle \nu y \rangle P \Longrightarrow a \langle \nu x \rangle \prec (\langle \nu y \rangle P')$
 $\langle proof \rangle$

lemma *Bang*:
fixes $P :: pi$
and $Rs :: residual$

shows $P \parallel !P \Rightarrow a \langle \nu x \rangle \prec P' \Rightarrow !P \Rightarrow a \langle \nu x \rangle \prec P'$
and $P \parallel !P \Rightarrow \alpha \prec P' \Rightarrow !P \Rightarrow \alpha \prec P'$
 $\langle proof \rangle$

lemma *tauTransitionChain*:
fixes $P :: pi$
and $P' :: pi$

assumes $P \Rightarrow_{\tau} \prec P'$

shows $P \Rightarrow_{\tau} P'$
 $\langle proof \rangle$

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 $P \Rightarrow_{\tau} P'' \Rightarrow P'' \Rightarrow \alpha \prec P' \Rightarrow P \Rightarrow \alpha \prec P'$
 $\langle proof \rangle$

lemma *freshBoundOutputTransition*:

fixes $P :: pi$
and $a :: name$
and $x :: name$
and $P' :: pi$
and $c :: name$

assumes $PTrans: P \Rightarrow a \langle \nu x \rangle \prec P'$
and $c \# P$
and $c \neq x$

shows $c \# P'$
 $\langle proof \rangle$

lemma *freshTauTransition*:

fixes $P :: pi$

and $c :: name$

assumes $PTrans: P \Longrightarrow \tau \prec P'$
and $c \# P$

shows $c \# P'$
 $\langle proof \rangle$

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 $c \# P'$
 $\langle proof \rangle$

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 $P \Longrightarrow \alpha \prec P' \Longrightarrow (p \cdot P) \Longrightarrow (p \cdot \alpha) \prec (p \cdot P')$
 $\langle proof \rangle$

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 $c \# P'$
 $\langle proof \rangle$

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* ($\langle \cdot \Longrightarrow^{\wedge} \cdot \prec \cdot \rangle$ [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 α :: *freeRes*
and *P'* :: *pi*

shows $P \Longrightarrow \alpha \prec P' \Longrightarrow P \Longrightarrow^{\wedge} \alpha \prec P'$
and $P \Longrightarrow^{\wedge} \tau \prec P$

\langle *proof* \rangle

lemma *transitionCases*[*consumes 1, case-names Step Stay*]:

fixes *P* :: *pi*
and α :: *freeRes*
and *P'* :: *pi*

assumes $P \Longrightarrow^{\wedge} \alpha \prec P'$
and $P \Longrightarrow \alpha \prec P' \Longrightarrow F \alpha P'$
and $F (\tau) P$

shows $F \alpha P'$

\langle *proof* \rangle

lemma *singleActionChain*:

fixes *P* :: *pi*
and α :: *freeRes*
and *P'* :: *pi*

assumes $P \mapsto \alpha \prec P'$

shows $P \Longrightarrow^{\wedge} \alpha \prec P'$

\langle *proof* \rangle

lemma *Tau*:

fixes *P* :: *pi*

shows $\tau.(P) \Longrightarrow^{\wedge} \tau \prec P$

\langle *proof* \rangle

lemma *Input*:

fixes $a :: name$
and $x :: name$
and $u :: name$
and $P :: pi$

shows $a \langle x \rangle . P \Longrightarrow \hat{a} \langle u \rangle \prec P[x ::= u]$
 $\langle proof \rangle$

lemma *Output*:
fixes $a :: name$
and $b :: name$
and $P :: pi$

shows $a\{b\}.P \Longrightarrow \hat{a}[b] \prec P$
 $\langle proof \rangle$

lemma *Par1F*:
fixes $P :: pi$
and $\alpha :: freeRes$
and $P' :: pi$
and $Q :: pi$

assumes $P \Longrightarrow \hat{\alpha} \prec P'$

shows $P \parallel Q \Longrightarrow \hat{\alpha} \prec (P' \parallel Q)$
 $\langle proof \rangle$

lemma *Par2F*:
fixes $Q :: pi$
and $\alpha :: freeRes$
and $Q' :: pi$
and $P :: pi$

assumes $QTrans: Q \Longrightarrow \hat{\alpha} \prec Q'$

shows $P \parallel Q \Longrightarrow \hat{\alpha} \prec (P \parallel Q')$
 $\langle proof \rangle$

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 $\langle \nu x \rangle P \Longrightarrow \hat{\alpha} \prec \langle \nu x \rangle P'$

$\langle proof \rangle$

lemma *Bang*:

fixes $P :: pi$

and $Rs :: residual$

assumes $P \parallel !P \Longrightarrow \hat{\alpha} \prec P'$

and $P' \neq P \parallel !P$

shows $!P \Longrightarrow \hat{\alpha} \prec P'$

$\langle proof \rangle$

lemma *tauTransitionChain[simp]*:

fixes $P :: pi$

and $P' :: pi$

shows $P \Longrightarrow \hat{\tau} \prec P' = P \Longrightarrow_{\tau} P'$

$\langle proof \rangle$

lemma *tauStepTransitionChain[simp]*:

fixes $P :: pi$

and $P' :: pi$

assumes $P \neq P'$

shows $P \Longrightarrow_{\tau} \prec P' = P \Longrightarrow_{\tau} P'$

$\langle proof \rangle$

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 $P \Longrightarrow \hat{\alpha} \prec P'' \Longrightarrow P'' \Longrightarrow_{\tau} P' \Longrightarrow P \Longrightarrow \hat{\alpha} \prec P'$

$\langle proof \rangle$

lemma *freshTauTransition*:

fixes $P :: pi$

and $c :: name$

assumes $P \Longrightarrow \hat{\tau} \prec P'$

and $c \# P$

shows $c \# P'$

<proof>

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 $c \# P'$

<proof>

lemma *eqvtI*:

fixes $P :: pi$
and $\alpha :: freeRes$
and $P' :: pi$
and $p :: name prm$

assumes $P \Longrightarrow \hat{\alpha} \prec P'$

shows $(p \cdot P) \Longrightarrow \hat{(p \cdot \alpha)} \prec (p \cdot P')$

<proof>

lemma *freshInputTransition*:

fixes $P :: pi$
and $a :: name$
and $b :: name$
and $P' :: pi$
and $c :: name$

assumes $P \Longrightarrow \hat{a} \prec b \succ P'$
and $c \# P$
and $c \neq b$

shows $c \# P'$

<proof>

lemmas *freshTransition = freshBoundOutputTransition freshOutputTransition
freshInputTransition freshTauTransition*

end

theory *Weak-Early-Sim*

imports *Weak-Early-Semantics Strong-Early-Sim-Pres*
begin

definition *weakSimulation* :: *pi* ⇒ (*pi* × *pi*) *set* ⇒ *pi* ⇒ *bool* (*⋄* - *↔* <*ν*> -> [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* × *pi*) *set*
and *B* :: (*pi* × *pi*) *set*
and *P* :: *pi*
and *P'* :: *pi*

assumes $P \rightsquigarrow \langle A \rangle P'$
and $A \subseteq B$

shows $P \rightsquigarrow \langle B \rangle P'$
<proof>

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*: $\bigwedge a\ x\ Q'.\ \llbracket Q \mapsto a \langle \nu x \rangle \prec Q';\ x \# P;\ x \# Q;\ x \neq a;\ x \# C \rrbracket \Longrightarrow \exists P'.\ P \Longrightarrow a \langle \nu x \rangle \prec P' \wedge (P', Q') \in \text{Rel}$
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>

lemma *simCases*[*case-names Bound Free*]:

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

assumes $\bigwedge Q'\ a\ x.\ \llbracket Q \mapsto a \langle \nu x \rangle \prec Q';\ x \# P \rrbracket \Longrightarrow \exists P'.\ P \Longrightarrow a \langle \nu x \rangle \prec P' \wedge (P', Q') \in \text{Rel}$
and $\bigwedge Q'\ \alpha.\ Q \mapsto \alpha \prec Q' \Longrightarrow \exists P'.\ P \Longrightarrow \hat{\alpha} \prec P' \wedge (P', Q') \in \text{Rel}$

shows $P \rightsquigarrow \langle \text{Rel} \rangle Q$
<proof>

lemma *simE*:

fixes *P* :: *pi*
and *Rel* :: (*pi* × *pi*) *set*

and $Q :: pi$
and $a :: name$
and $x :: name$
and $Q' :: pi$

assumes $P \rightsquigarrow_{\langle Rel \rangle} Q$

shows $Q \mapsto_{a \langle \nu x \rangle} \prec Q' \implies x \# P \implies \exists P'. P \implies_{a \langle \nu x \rangle} \prec P' \wedge (P', Q') \in Rel$

and $Q \mapsto_{\alpha} \prec Q' \implies \exists P'. P \implies_{\hat{\alpha}} \prec P' \wedge (P', Q') \in Rel$
<proof>

lemma *weakSimTauChain*:

fixes $P :: pi$
and $Rel :: (pi \times pi) \text{ set}$
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 $PSimQ: \bigwedge R S. (R, S) \in Rel \implies R \rightsquigarrow_{\langle Rel \rangle} S$

shows $\exists P'. P \implies_{\tau} P' \wedge (P', Q') \in Rel$
<proof>

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 $Sim: \bigwedge R S. (R, S) \in Rel \implies R \rightsquigarrow_{\langle Rel \rangle} S$
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 $Q \implies_{\hat{\alpha}} \prec Q' \implies \exists P'. P \implies_{\hat{\alpha}} \prec P' \wedge (P', Q') \in Rel$
<proof>

lemma *eqvtI*:

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

assumes $PSimQ: P \rightsquigarrow \langle Rel \rangle Q$
and $RelRel': Rel \subseteq Rel'$
and $EqvtRel': eqvt Rel'$

shows $(perm \cdot P) \rightsquigarrow \langle Rel' \rangle (perm \cdot Q)$
 $\langle proof \rangle$

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

assumes $Id \subseteq Rel$

shows $P \rightsquigarrow \langle Rel \rangle P$
 $\langle proof \rangle$

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 $Trans: Rel \circ Rel' \subseteq Rel''$
and $Sim: \bigwedge S T. (S, T) \in Rel \implies S \rightsquigarrow \langle Rel \rangle T$
and $PRelQ: (P, Q) \in Rel$

shows $P \rightsquigarrow \langle Rel'' \rangle R$
 $\langle proof \rangle$

lemma strongAppend:
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 $QSimR: Q \rightsquigarrow [Rel'] R$
and $Eqvt'': eqvt Rel''$
and $Trans: Rel \circ Rel' \subseteq Rel''$

shows $P \rightsquigarrow \langle Rel'' \rangle R$
 ⟨proof⟩

lemma *strongSimWeakSim*:

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

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

shows $P \rightsquigarrow \langle Rel \rangle Q$
 ⟨proof⟩

end

theory *Weak-Early-Bisim*

imports *Weak-Early-Sim Strong-Early-Bisim*

begin

lemma *monoAux*: $A \subseteq B \implies P \rightsquigarrow \langle A \rangle Q \longrightarrow P \rightsquigarrow \langle B \rangle Q$
 ⟨proof⟩

coinductive-set *weakBisim* :: $(pi \times pi)$ set

where

step: $\llbracket P \rightsquigarrow \langle weakBisim \rangle Q; (Q, P) \in weakBisim \rrbracket \implies (P, Q) \in weakBisim$

monos *monoAux*

abbreviation *weakEarlyBisimJudge* (**infixr** $\langle \approx \rangle$ 65) **where** $P \approx Q \equiv (P, Q) \in weakBisim$

lemma *weakBisimCoinductAux*[*case-names weakBisim, case-conclusion weakBisim step, consumes 1*]:

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

and *step*: $\bigwedge P Q. (P, Q) \in X \implies P \rightsquigarrow \langle X \cup weakBisim \rangle Q \wedge (Q, P) \in X \cup weakBisim$

shows $P \approx Q$
 ⟨proof⟩

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 $P \approx Q$
 ⟨proof⟩

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

fixes $P :: pi$

```

and  $Q :: pi$ 
and  $X :: (pi \times pi) \text{ set}$ 

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  $P \approx Q$ 
<proof>

lemma weakBisimWeakCoinduct[consumes 1, case-names cSim cSym]:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $X :: (pi \times pi) \text{ set}$ 

  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  $P \approx Q$ 
  <proof>

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

  assumes  $P \approx Q$ 

  shows  $P \rightsquigarrow \langle \text{weakBisim} \rangle Q$ 
  and  $Q \approx P$ 
  <proof>

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

  assumes  $P \rightsquigarrow \langle \text{weakBisim} \rangle Q$ 
  and  $Q \approx P$ 

  shows  $P \approx Q$ 
  <proof>

lemma eqvt[simp]:
  shows eqvt weakBisim
  <proof>

lemma eqvtI[eqvt]:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 

```

```

and perm :: name prm

assumes P ≈ Q

shows (perm · P) ≈ (perm · Q)
⟨proof⟩

lemma strongBisimWeakBisim:
  fixes P :: pi
  and Q :: pi

  assumes P ~ Q

  shows P ≈ Q
⟨proof⟩

lemma reflexive:
  fixes P :: pi

  shows P ≈ P
⟨proof⟩

lemma symetric:
  fixes P :: pi
  and Q :: pi

  assumes P ≈ Q

  shows Q ≈ P
⟨proof⟩

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

  assumes P ≈ Q
  and Q ≈ R

  shows P ≈ R
⟨proof⟩

lemma weakBisimWeakUpto[case-names cSim cSym, consumes 1]:
  assumes p: (P, Q) ∈ 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 ≈ Q

```

<proof>

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>

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>

end

theory *Weak-Early-Step-Sim*

imports *Weak-Early-Sim Strong-Early-Sim*

begin

definition *weakStepSimulation* :: $pi \Rightarrow (pi \times pi) \text{ set} \Rightarrow pi \Rightarrow bool \ (\prec \rightsquigarrow \langle - \rangle \rightarrow [80, 80, 80] \ 80)$ **where**

$P \rightsquigarrow \langle Rel \rangle Q \equiv (\forall Q' a x. Q \mapsto a \langle \nu x \rangle \prec Q' \longrightarrow x \# P \longrightarrow (\exists P'. P \Longrightarrow a \langle \nu x \rangle \prec P' \wedge (P', Q') \in Rel)) \wedge$

$(\forall Q' \alpha. Q \mapsto \alpha \prec Q' \longrightarrow (\exists P'. P \Longrightarrow \alpha \prec P' \wedge (P', Q') \in Rel))$

lemma *monotonic*:

fixes $A :: (pi \times pi) \text{ set}$

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 $P \rightsquigarrow \langle B \rangle P'$

<proof>

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

fixes $P :: pi$

and $Q :: pi$
and $Rel :: (pi \times pi) \text{ set}$
and $C :: 'a::fs\text{-name}$

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

shows $P \rightsquigarrow \langle Rel \rangle Q$
 $\langle proof \rangle$

lemma $simCases[consumes \ 0, \ \text{case-names } Bound \ Free]:$

fixes $P :: pi$
and $Q :: pi$
and $Rel :: (pi \times pi) \text{ set}$
and $C :: 'a::fs\text{-name}$

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

shows $P \rightsquigarrow \langle Rel \rangle Q$
 $\langle proof \rangle$

lemma $simE:$

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

assumes $P \rightsquigarrow \langle Rel \rangle Q$

shows $Q \mapsto a \langle \nu x \rangle \prec Q' \implies x \# P \implies \exists P'. P \implies a \langle \nu x \rangle \prec P' \wedge (P', Q') \in Rel$
and $Q \mapsto \alpha \prec Q' \implies \exists P'. P \implies \alpha \prec P' \wedge (P', Q') \in Rel$
 $\langle proof \rangle$

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>

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>

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

assumes $Id \subseteq Rel$

shows $P \rightsquigarrow \langle Rel \rangle P$
<proof>

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 $Sim: \bigwedge S T. (S, T) \in Rel \implies S \rightsquigarrow \langle Rel \rangle T$
and $PRelQ: (P, Q) \in Rel$

shows $P \rightsquigarrow\langle\langle Rel'' \rangle\rangle R$
 $\langle proof \rangle$

lemma *strongSimWeakSim*:

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

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

shows $P \rightsquigarrow\langle\langle Rel \rangle\rangle Q$
 $\langle proof \rangle$

lemma *weakSimWeakEqSim*:

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

assumes $P \rightsquigarrow\langle\langle Rel \rangle\rangle Q$

shows $P \rightsquigarrow\langle Rel \rangle Q$
 $\langle proof \rangle$

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 $P \simeq Q \equiv P \rightsquigarrow\langle\langle weakBisim \rangle\rangle Q \wedge Q \rightsquigarrow\langle\langle weakBisim \rangle\rangle P$

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 $\bigwedge R S. Prop R S \implies (F R) \rightsquigarrow\langle\langle weakBisim \rangle\rangle (F S)$

shows $F P \simeq F Q$
 $\langle proof \rangle$

lemma *weakCongISym2*[*consumes 1, case-names cSim*]:

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

assumes $P \simeq Q$

and $\bigwedge R S. R \simeq S \implies (F R) \rightsquigarrow\langle\langle weakBisim \rangle\rangle (F S)$

shows $F P \simeq F Q$
<proof>

lemma *weakCongEE*:
fixes $P :: pi$
and $Q :: pi$
and $s :: (name \times name) list$

assumes $P \simeq Q$

shows $P \rightsquigarrow\langle\langle weakBisim \rangle\rangle Q$
and $Q \rightsquigarrow\langle\langle weakBisim \rangle\rangle P$
<proof>

lemma *weakCongI*:

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

assumes $P \rightsquigarrow\langle\langle weakBisim \rangle\rangle Q$
and $Q \rightsquigarrow\langle\langle weakBisim \rangle\rangle P$

shows $P \simeq Q$
<proof>

lemma *eqvtI[eqvt]*:

fixes $P :: pi$
and $Q :: pi$
and $p :: name prm$

assumes $P \simeq Q$

shows $(p \cdot P) \simeq (p \cdot Q)$
<proof>

lemma *strongBisimWeakCong*:

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

assumes $P \sim Q$

shows $P \simeq Q$
<proof>

lemma *congruenceWeakBisim*:

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

assumes $P \simeq Q$

shows $P \approx Q$
 $\langle proof \rangle$

lemma *reflexive*:
fixes $P :: pi$

shows $P \simeq P$
 $\langle proof \rangle$

lemma *symetric*:
fixes $P :: pi$
and $Q :: pi$

assumes $P \simeq Q$

shows $Q \simeq P$
 $\langle proof \rangle$

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

assumes $P \simeq Q$
and $Q \simeq R$

shows $P \simeq R$
 $\langle proof \rangle$

end

theory *Weak-Early-Bisim-Subst*

imports *Weak-Early-Bisim Strong-Early-Bisim-Subst*

begin

consts *weakBisimSubst* :: $(pi \times pi)$ set

abbreviation *weakEarlyBisimSubstJudge* (**infix** $\langle \approx^s \rangle$ 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 $P \approx Q$
 $\langle proof \rangle$

lemma *strongBisimWeakBisim*:

fixes $P :: pi$

and $Q :: pi$

assumes $P \sim^s Q$

shows $P \approx^s Q$

$\langle proof \rangle$

lemma *eqvt*:

shows *eqvt* (*substClosed weakBisim*)

$\langle proof \rangle$

lemma *eqvtI*[*eqvt*]:

fixes $P :: pi$

and $Q :: pi$

and $perm :: name prm$

assumes $P \approx^s Q$

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

$\langle proof \rangle$

lemma *reflexive*:

fixes $P :: pi$

shows $P \approx^s P$

$\langle proof \rangle$

lemma *symetric*:

fixes $P :: pi$

and $Q :: pi$

assumes $P \approx^s Q$

shows $Q \approx^s P$

$\langle proof \rangle$

lemma *transitive*:

fixes $P :: pi$

and $Q :: pi$

and $R :: pi$

assumes $P \approx^s Q$

and $Q \approx^s R$

shows $P \approx^s R$

$\langle proof \rangle$

```

lemma partUnfold:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $s :: (name \times name) list$ 

  assumes  $P \approx^s Q$ 

  shows  $P[\langle s \rangle] \approx^s Q[\langle s \rangle]$ 
  <proof>

end

theory Weak-Early-Cong-Subst
  imports Weak-Early-Cong Weak-Early-Bisim-Subst Strong-Early-Bisim-Subst
begin

consts congruenceSubst ::  $(pi \times pi) set$ 

definition weakCongruenceSubst (infixr  $\langle \simeq^s \rangle$  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  $s :: (name \times name) list$ 

  assumes  $P \simeq^s Q$ 

  shows  $P[\langle s \rangle] \rightsquigarrow \langle \langle weakBisim \rangle \rangle Q[\langle s \rangle]$ 
  and  $Q[\langle s \rangle] \rightsquigarrow \langle \langle weakBisim \rangle \rangle P[\langle s \rangle]$ 
  <proof>

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

  assumes  $\bigwedge s. P[\langle s \rangle] \rightsquigarrow \langle \langle weakBisim \rangle \rangle Q[\langle s \rangle]$ 
  and  $\bigwedge s. Q[\langle s \rangle] \rightsquigarrow \langle \langle weakBisim \rangle \rangle P[\langle s \rangle]$ 

  shows  $P \simeq^s Q$ 
  <proof>

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

  assumes  $P \simeq^s Q$ 

  shows  $P \simeq Q$ 

```

<proof>

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>

lemma *strongEqWeakCong*:

fixes $P :: pi$

and $Q :: pi$

assumes $P \sim^s Q$

shows $P \simeq^s Q$

<proof>

lemma *congSubstBisimSubst*:

fixes $P :: pi$

and $Q :: pi$

assumes $P \simeq^s Q$

shows $P \approx^s Q$

<proof>

lemma *reflexive*:

fixes $P :: pi$

shows $P \simeq^s P$

<proof>

lemma *symetric*:

fixes $P :: pi$

and $Q :: pi$

assumes $P \simeq^s Q$

shows $Q \simeq^s P$

<proof>

lemma *transitive*:

fixes $P :: pi$

and $Q :: pi$

and $R :: pi$

```

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

shows  $P \simeq^s R$ 
<proof>

lemma partUnfold:
fixes  $P :: pi$ 
and  $Q :: pi$ 
and  $s :: (name \times name) list$ 

assumes  $P \simeq^s Q$ 

shows  $P[<s>] \simeq^s Q[<s>]$ 
<proof>

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>

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  $a <x>.P \rightsquigarrow \langle Rel \rangle a <x>.Q$ 
<proof>

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  $a\{b\}.P \rightsquigarrow_{\langle Rel \rangle} a\{b\}.Q$ 
 $\langle proof \rangle$ 

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  $[a \frown b]P \rightsquigarrow_{\langle Rel' \rangle} [a \frown b]Q$ 
 $\langle proof \rangle$ 

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  $[a \neq b]P \rightsquigarrow_{\langle Rel' \rangle} [a \neq b]Q$ 
 $\langle proof \rangle$ 

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>

lemma *parPres*:

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 $PSimQ$: $P \rightsquigarrow \langle Rel \rangle Q$
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>

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 $EqtRel$: *eqvt* Rel
and $EqtRel'$: *eqvt* Rel'

shows $\langle \nu x \rangle P \rightsquigarrow \langle Rel' \rangle \langle \nu x \rangle Q$

<proof>

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 $PRelQ$: $P \rightsquigarrow \langle Rel \rangle Q$

shows $(resChain\ lst)\ P \rightsquigarrow \langle Rel \rangle (resChain\ lst)\ Q$

<proof>

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    $C1$ :  $Rel \subseteq Rel'$ 
and    $eqvtRel$ :  $eqvt Rel'$ 

shows  $!P \rightsquigarrow\langle bangRel Rel' \rangle !Q$ 
 $\langle proof \rangle$ 

end

theory Weak-Early-Sim-Pres
imports Weak-Early-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  $\tau.(P) \rightsquigarrow\langle Rel \rangle \tau.(Q)$ 
 $\langle proof \rangle$ 

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  $a\langle x \rangle.P \rightsquigarrow\langle Rel \rangle a\langle x \rangle.Q$ 
 $\langle proof \rangle$ 

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 $a\{b\}.P \rightsquigarrow_{\langle Rel \rangle} a\{b\}.Q$
 $\langle proof \rangle$

lemma *matchPres*:
fixes $P :: pi$
and $Q :: pi$
and $a :: name$
and $b :: name$
and $Rel :: (pi \times pi) \text{ set}$
and $Rel' :: (pi \times pi) \text{ set}$

assumes $PSimQ: P \rightsquigarrow_{\langle Rel \rangle} Q$
and $RelRel': Rel \subseteq Rel'$
and $RelStay: \bigwedge R S c. (R, S) \in Rel \implies ([c \frown c]R, S) \in Rel$

shows $[a \frown b]P \rightsquigarrow_{\langle Rel' \rangle} [a \frown b]Q$
 $\langle proof \rangle$

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

assumes $PSimQ: P \rightsquigarrow_{\langle Rel \rangle} Q$
and $RelRel': Rel \subseteq Rel'$
and $RelStay: \bigwedge R S c d. [(R, S) \in Rel; c \neq d] \implies ([c \neq d]R, S) \in Rel$

shows $[a \neq b]P \rightsquigarrow_{\langle Rel' \rangle} [a \neq b]Q$
 $\langle proof \rangle$

lemma *parCompose*:
fixes $P :: pi$
and $Q :: pi$
and $R :: pi$
and $S :: pi$
and $Rel :: (pi \times pi) \text{ set}$
and $Rel' :: (pi \times pi) \text{ set}$
and $Rel'' :: (pi \times pi) \text{ set}$

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 $Par: \bigwedge P' Q' R' S'. [(P', Q') \in Rel; (R', S') \in Rel'] \implies (P' \parallel R',$

$Q' \parallel S') \in Rel''$
and $Res: \bigwedge T U x. (T, U) \in Rel'' \implies (\langle \nu x \rangle T, \langle \nu x \rangle U) \in Rel''$
shows $P \parallel R \rightsquigarrow_{\langle Rel'' \rangle} Q \parallel S$
 $\langle proof \rangle$

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 $P \parallel R \rightsquigarrow_{\langle Rel' \rangle} Q \parallel R$
 $\langle proof \rangle$

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 $\langle \nu x \rangle P \rightsquigarrow_{\langle Rel' \rangle} \langle \nu x \rangle Q$
 $\langle proof \rangle$

lemma *resChainI*:

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

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>

lemma *bangPres*:

fixes $P :: pi$

and $Q :: pi$

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

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

and $Sim: \bigwedge R S. (R, S) \in Rel \implies R \rightsquigarrow_{\langle Rel \rangle} S$

and $ParComp: \bigwedge R S T U. \llbracket (R, S) \in Rel; (T, U) \in Rel \rrbracket \implies (R \parallel T, S \parallel U) \in Rel'$

and $Res: \bigwedge R S x. (R, S) \in Rel' \implies (\nu x \rangle R, \langle \nu x \rangle S) \in Rel'$

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 $!P \rightsquigarrow_{\langle Rel' \rangle} !Q$

<proof>

lemma *bangRelSim*:

fixes $P :: pi$

and $Q :: pi$

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

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

assumes $PBangRelQ: (P, Q) \in bangRel Rel$

and $Sim: \bigwedge R S. (R, S) \in Rel \implies R \rightsquigarrow_{\langle Rel \rangle} S$

and $ParComp: \bigwedge R S T U. \llbracket (R, S) \in Rel; (T, U) \in Rel \rrbracket \implies (R \parallel T, S \parallel U) \in Rel'$

and $Res: \bigwedge R S x. (R, S) \in Rel' \implies (\nu x \rangle R, \langle \nu x \rangle S) \in Rel'$

and $RelStay: \bigwedge R S. (R \parallel !R, S) \in Rel' \implies (!R, S) \in Rel'$

and $BangRelRel': (bangRel Rel) \subseteq Rel'$

and $eqvtRel': eqvt Rel'$

and $Eqvt: eqvt Rel$

shows $P \rightsquigarrow_{\langle Rel' \rangle} Q$

<proof>

end

theory *Strong-Early-Late-Comp*

imports *Strong-Late-Bisim-Subst-SC Strong-Early-Bisim-Subst*

begin

abbreviation *TransitionsLate-judge* $(\langle \cdot \rangle \mapsto_l \cdot \rightarrow [80, 80] 80)$ **where** $P \mapsto_l Rs \equiv$

transitions $P R_s$

abbreviation *TransitionsEarly-judge* ($\langle \cdot \mapsto_e \cdot \rangle [80, 80] 80$) **where** $P \mapsto_e R_s$
 \equiv *TransitionsEarly* $P R_s$

abbreviation *Transitions-InputjudgeLate* ($\langle \cdot \langle \cdot \rangle \prec_l \cdot \rangle [80, 80] 80$) **where** $a \langle x \rangle$
 $\prec_l P' \equiv$ (*Late-Semantics.BoundR* (*Late-Semantics.InputS* a) $x P'$)

abbreviation *Transitions-OutputjudgeLate* ($\langle \cdot [-] \prec_l \cdot \rangle [80, 80] 80$) **where** $a[b] \prec_l$
 $P' \equiv$ (*Late-Semantics.FreeR* (*Late-Semantics.OutputR* a b) P')

abbreviation *Transitions-BoundOutputjudgeLate* ($\langle \cdot \langle \nu \rangle \prec_l \cdot \rangle [80, 80] 80$) **where**
 $a \langle \nu x \rangle \prec_l P' \equiv$ (*Late-Semantics.BoundR* (*Late-Semantics.BoundOutputS* a) $x P'$)

abbreviation *Transitions-TaujudgeLate* ($\langle \tau \prec_l \cdot \rangle 80$) **where** $\tau \prec_l P' \equiv$ (*Late-Semantics.FreeR*
Late-Semantics.TauR P')

abbreviation *Transitions-InputjudgeEarly* ($\langle \cdot \langle \cdot \rangle \prec_e \cdot \rangle [80, 80] 80$) **where** $a \langle x \rangle$
 $\prec_e P' \equiv$ (*Early-Semantics.FreeR* (*Early-Semantics.InputR* a x) P')

abbreviation *Transitions-OutputjudgeEarly* ($\langle \cdot [-] \prec_e \cdot \rangle [80, 80] 80$) **where** $a[b]$
 $\prec_e P' \equiv$ (*Early-Semantics.FreeR* (*Early-Semantics.OutputR* a b) P')

abbreviation *Transitions-BoundOutputjudgeEarly* ($\langle \cdot \langle \nu \rangle \prec_e \cdot \rangle [80, 80] 80$)
where $a \langle \nu x \rangle \prec_e P' \equiv$ (*Early-Semantics.BoundOutputR* a $x P'$)

abbreviation *Transitions-TaujudgeEarly* ($\langle \tau \prec_e \cdot \rangle 80$) **where** $\tau \prec_e P' \equiv$ (*Early-Semantics.FreeR*
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 $P \mapsto_l a[b] \prec_l P'$

<proof>

lemma *lateEarlyOutput*:

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

assumes $P \mapsto_l a[b] \prec_l P'$

shows $P \mapsto_e a[b] \prec_e P'$

<proof>

lemma *outputEq*:

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

shows $P \mapsto_e a[b] \prec_e P' = P \mapsto_l a[b] \prec_l P'$
<proof>

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>

lemma *earlyLateBoundOutput*:

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

assumes $P \mapsto_e a \langle \nu x \rangle \prec_e P'$

shows $P \mapsto_l a \langle \nu x \rangle \prec_l P'$
<proof>

lemma *BoundOutputEq*:

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

shows $P \mapsto_e a \langle \nu x \rangle \prec_e P' = P \mapsto_l a \langle \nu x \rangle \prec_l P'$
<proof>

lemma *lateEarlyInput*:

fixes $P :: pi$
and $a :: name$
and $x :: name$
and $P' :: pi$
and $u :: name$

assumes $PTrans: P \mapsto_l a \langle x \rangle \prec_l P'$

shows $P \mapsto_e a \langle u \rangle \prec_e (P'[x::=u])$
<proof>

lemma *earlyLateInput*:

fixes $P :: pi$

and $a :: name$
and $x :: name$
and $P' :: pi$
and $u :: name$
and $C :: 'a::fs-name$

assumes $P \mapsto_e a \langle u \rangle \prec_e P'$
and $x \# P$

shows $\exists P''. P \mapsto_l a \langle x \rangle \prec_l P'' \wedge P' = P''[x::=u]$
 $\langle proof \rangle$

lemma *lateEarlyTau*:

fixes $P :: pi$
and $P' :: pi$

assumes $P \mapsto_l \tau \prec_l P'$

shows $P \mapsto_e \tau \prec_e P'$
 $\langle proof \rangle$

lemma *earlyLateTau*:

fixes $P :: pi$
and $P' :: pi$

assumes $P \mapsto_e \tau \prec_e P'$

shows $P \mapsto_l \tau \prec_l P'$
 $\langle proof \rangle$

lemma *tauEq*:

fixes $P :: pi$
and $P' :: pi$

shows $P \mapsto_e (Early-Semantics.FreeR \ Early-Semantics.TauR \ P') = P \mapsto_\tau \prec_l P'$
 $\langle proof \rangle$

abbreviation *simLate-judge* $(\prec \rightsquigarrow_l [-] \rightarrow [80, 80, 80] \ 80)$ **where** $P \rightsquigarrow_l [Rel] \ Q \equiv Strong-Late-Sim.simulation \ P \ Rel \ Q$

abbreviation *simEarly-judge* $(\prec \rightsquigarrow_e [-] \rightarrow [80, 80, 80] \ 80)$ **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>

abbreviation $bisimLate-judge (\langle - \sim_l \rightarrow [80, 80] 80 \rangle)$ **where** $P \sim_l Q \equiv (P, Q) \in Strong-Late-Bisim.bisim$

abbreviation $bisimEarly-judge (\langle - \sim_e \rightarrow [80, 80] 80 \rangle)$ **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 $P \sim_e Q$
<proof>

abbreviation $congLate-judge (\langle - \sim^s_l \rightarrow [80, 80] 80 \rangle)$ **where** $P \sim^s_l Q \equiv (P, Q) \in (substClosed Strong-Late-Bisim.bisim)$

abbreviation $congEarly-judge (\langle - \sim^s_e \rightarrow [80, 80] 80 \rangle)$ **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 $P \sim^s_e Q$
<proof>

lemma $earlyCongStructCong:$

fixes $P :: pi$

and $Q :: pi$

assumes $P \equiv_s Q$

shows $P \sim^s_e Q$
<proof>

lemma *earlyBisimStructCong*:

fixes $P :: pi$

and $Q :: pi$

assumes $P \equiv_s Q$

shows $P \sim_e Q$

<proof>

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>

lemma *matchId*:

fixes $a :: name$

and $P :: pi$

shows $[a \curvearrowright a]P \sim_e P$

<proof>

lemma *mismatchId*:

fixes $a :: name$

and $b :: name$

and $P :: pi$

assumes $a \neq b$

shows $[a \neq b]P \sim_e P$

<proof>

lemma *mismatchNil*:

fixes $a :: name$

and $P :: pi$

shows $[a \neq a]P \sim_e \mathbf{0}$
<proof>

lemma *sumSym*:
fixes $P :: pi$
and $Q :: pi$

shows $P \oplus Q \sim_e Q \oplus P$
<proof>

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

shows $(P \oplus Q) \oplus R \sim_e P \oplus (Q \oplus R)$
<proof>

lemma *sumZero*:
fixes $P :: pi$

shows $P \oplus \mathbf{0} \sim_e P$
<proof>

lemma *parZero*:
fixes $P :: pi$

shows $P \parallel \mathbf{0} \sim_e P$
<proof>

lemma *parSym*:
fixes $P :: pi$
and $Q :: pi$

shows $P \parallel Q \sim_e Q \parallel P$
<proof>

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>

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>

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>

lemma *freshRes*:

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

assumes $aFreshP: a \# P$

shows $\langle \nu a \rangle P \sim_e P$
<proof>

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>

lemma *bangSC*:

fixes P

shows $!P \sim_e P \parallel !P$
<proof>

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 $P \approx Q$
<proof>

lemma *matchId*:

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

shows $[a \curvearrowright a]P \approx P$
<proof>

lemma *mismatchId*:

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

assumes $a \neq b$

shows $[a \neq b]P \approx P$
<proof>

lemma *mismatchNil*:

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

shows $[a \neq a]P \approx \mathbf{0}$
<proof>

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>

lemma *sumSym*:

```

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

shows  $P \oplus Q \approx Q \oplus P$ 
<proof>

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

shows  $(P \oplus Q) \oplus R \approx P \oplus (Q \oplus R)$ 
<proof>

lemma sumZero:
fixes  $P :: pi$ 

shows  $P \oplus \mathbf{0} \approx P$ 
<proof>

lemma parZero:
fixes  $P :: pi$ 

shows  $P \parallel \mathbf{0} \approx P$ 
<proof>

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

shows  $P \parallel Q \approx Q \parallel P$ 
<proof>

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>

lemma scopeExtPar':
fixes  $P :: pi$ 
and  $Q :: pi$ 
and  $x :: name$ 

```

```

assumes  $x \# Q$ 

shows  $\langle \nu x \rangle (P \parallel Q) \approx (\langle \nu x \rangle P) \parallel Q$ 
 $\langle proof \rangle$ 

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

  shows  $(P \parallel Q) \parallel R \approx P \parallel (Q \parallel R)$ 
 $\langle proof \rangle$ 

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

  assumes  $a \# P$ 

  shows  $\langle \nu a \rangle P \approx P$ 
 $\langle proof \rangle$ 

lemma scopeExtSum:
  fixes  $P :: pi$ 
  and  $Q :: pi$ 
  and  $x :: name$ 

  assumes  $x \# P$ 

  shows  $\langle \nu x \rangle (P \oplus Q) \approx P \oplus \langle \nu x \rangle Q$ 
 $\langle proof \rangle$ 

lemma bangSC:
  fixes  $P$ 

  shows  $!P \approx P \parallel !P$ 
 $\langle proof \rangle$ 

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  $\tau.(P) \approx \tau.(Q)$ 
 $\langle proof \rangle$ 

lemma outputPres:
fixes  $P :: pi$ 
and  $Q :: pi$ 
and  $a :: name$ 
and  $b :: name$ 

assumes  $P \approx Q$ 

shows  $a\{b\}.P \approx a\{b\}.Q$ 
 $\langle proof \rangle$ 

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  $a\langle x \rangle.P \approx a\langle x \rangle.Q$ 
 $\langle proof \rangle$ 

lemma resPres:
fixes  $P :: pi$ 
and  $Q :: pi$ 
and  $x :: name$ 

assumes  $P \approx Q$ 

shows  $\langle \nu x \rangle P \approx \langle \nu x \rangle Q$ 
 $\langle proof \rangle$ 

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>

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>

lemma *parPres*:

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

assumes $P \approx Q$

shows $P \parallel R \approx Q \parallel R$
<proof>

lemma *bangPres*:

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

assumes $P \text{Bisim} Q: P \approx Q$

shows $!P \approx !Q$
<proof>

lemma *bangRelSubWeakBisim*:

shows $\text{bangRel weakBisim} \subseteq \text{weakBisim}$
<proof>

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>

lemma *outputPres*:

fixes $P :: pi$

and $Q :: pi$

assumes $P \simeq Q$

shows $a\{b\}.P \simeq a\{b\}.Q$

<proof>

lemma *matchPres*:

fixes $P :: pi$

and $Q :: pi$

and $a :: name$

and $b :: name$

assumes $P \simeq Q$

shows $[a\curvearrowright b]P \simeq [a\curvearrowright b]Q$

<proof>

lemma *mismatchPres*:

fixes $P :: pi$

and $Q :: pi$

and $a :: name$

and $b :: name$

assumes $P \simeq Q$

shows $[a\neq b]P \simeq [a\neq b]Q$

<proof>

lemma *sumPres*:

fixes $P :: pi$

and $Q :: pi$

and $R :: pi$

assumes $P \simeq Q$

shows $P \oplus R \simeq Q \oplus R$

<proof>

lemma *parPres*:

fixes $P :: pi$

and $Q :: pi$

and $R :: pi$

assumes $P \simeq Q$

shows $P \parallel R \simeq Q \parallel R$
<proof>

lemma *resPres*:

fixes $P :: pi$
and $Q :: pi$
and $x :: name$

assumes $PeqQ: P \simeq Q$

shows $\langle \nu x \rangle P \simeq \langle \nu x \rangle Q$
<proof>

lemma *bangPres*:

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

assumes $P \simeq Q$

shows $!P \simeq !Q$
<proof>

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 $P \simeq^s Q$
<proof>

lemma *tauPres*:

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

assumes $P \simeq^s Q$

shows $\tau.(P) \simeq^s \tau.(Q)$
<proof>

lemma *inputPres*:

```

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

assumes  $P \text{eq} Q: P \simeq^s Q$ 

shows  $a \langle x \rangle . P \simeq^s a \langle x \rangle . Q$ 
<proof>

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

assumes  $P \simeq^s Q$ 

shows  $a\{b\} . P \simeq^s a\{b\} . Q$ 
<proof>

lemma matchPres:
fixes  $P :: pi$ 
and    $Q :: pi$ 
and    $a :: name$ 
and    $b :: name$ 

assumes  $P \simeq^s Q$ 

shows  $[a \frown b] P \simeq^s [a \frown b] Q$ 
<proof>

lemma mismatchPres:
fixes  $P :: pi$ 
and    $Q :: pi$ 
and    $a :: name$ 
and    $b :: name$ 

assumes  $P \simeq^s Q$ 

shows  $[a \neq b] P \simeq^s [a \neq b] Q$ 
<proof>

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

assumes  $P \simeq^s Q$ 

shows  $P \oplus R \simeq^s Q \oplus R$ 

```

$\langle proof \rangle$

lemma *parPres*:

fixes $P :: pi$

and $Q :: pi$

and $R :: pi$

assumes $P \simeq^s Q$

shows $P \parallel R \simeq^s Q \parallel R$

$\langle proof \rangle$

lemma *resPres*:

fixes $P :: pi$

and $Q :: pi$

and $x :: name$

assumes $P \text{ eq } Q: P \simeq^s Q$

shows $\langle \nu x \rangle P \simeq^s \langle \nu x \rangle Q$

$\langle proof \rangle$

lemma *bangPres*:

fixes $P :: pi$

and $Q :: pi$

assumes $P \simeq^s Q$

shows $!P \simeq^s !Q$

$\langle proof \rangle$

end

theory *Strong-Late-Expansion-Law*

imports *Strong-Late-Bisim-SC*

begin

nominal-primrec *summands* $:: pi \Rightarrow pi$ **set where**

$summands \mathbf{0} = \{\}$

| $summands (\tau.(P)) = \{\tau.(P)\}$

| $x \# a \Longrightarrow summands (a \langle x \rangle . P) = \{a \langle x \rangle . P\}$

| $summands (a\{b\}.P) = \{a\{b\}.P\}$

| $summands ([a \frown b]P) = \{\}$

| $summands ([a \neq b]P) = \{\}$

| $summands (P \oplus Q) = (summands P) \cup (summands Q)$

| $summands (P \parallel Q) = \{\}$

| $summands (\langle \nu x \rangle P) = (if (\exists a P'. a \neq x \wedge P = a\{x\}.P') then (\{\langle \nu x \rangle P\}) else \{\})$

| $summands (!P) = \{\}$

$\langle proof \rangle$

lemma *summandsInput*[*simp*]:

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

shows $summands (a\langle x \rangle.P) = \{a\langle x \rangle.P\}$

$\langle proof \rangle$

lemma *finiteSummands*:

fixes $P :: pi$

shows $finite(summands P)$

$\langle proof \rangle$

lemma *boundSummandDest*[*dest*]:

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

assumes $\langle \nu x \rangle x\{y\}.P' \in summands P$

shows $False$

$\langle proof \rangle$

lemma *summandFresh*:

fixes $P :: pi$
and $Q :: pi$
and $x :: name$

assumes $P \in summands Q$

and $x \# Q$

shows $x \# P$

$\langle proof \rangle$

nominal-primrec *hnf* :: $pi \Rightarrow bool$ **where**

$hnf \mathbf{0} = True$
 $| hnf (\tau.(P)) = True$
 $| x \# a \implies hnf (a\langle x \rangle.P) = True$
 $| hnf (a\{b\}.P) = True$
 $| hnf ([a\curvearrowright b]P) = False$
 $| hnf ([a\neq b]P) = False$
 $| hnf (P \oplus Q) = ((hnf P) \wedge (hnf Q) \wedge P \neq \mathbf{0} \wedge Q \neq \mathbf{0})$
 $| hnf (P \parallel Q) = False$
 $| hnf (\langle \nu x \rangle P) = (\exists a P'. a \neq x \wedge P = a\{x\}.P')$
 $| hnf (!P) = False$

<proof>

lemma *hnfInput[simp]*:
 fixes $a :: name$
 and $x :: name$
 and $P :: pi$

shows $hnf (a<x>.P)$
<proof>

lemma *summandTransition*:
 fixes $P :: pi$
 and $a :: name$
 and $x :: name$
 and $b :: name$
 and $P' :: pi$

assumes $hnf P$

shows $P \mapsto_{\tau} \prec P' = (\tau.(P') \in summands P)$
 and $P \mapsto a<x> \prec P' = (a<x>.P' \in summands P)$
 and $P \mapsto a[b] \prec P' = (a\{b\}.P' \in summands P)$
 and $a \neq x \implies P \mapsto a<\nu x> \prec P' = (<\nu x>a\{x\}.P' \in summands P)$
<proof>

definition *expandSet* :: $pi \Rightarrow pi \Rightarrow pi$ **set where**

$expandSet P Q \equiv \{\tau.(P' \parallel Q) \mid P'. \tau.(P') \in summands P\} \cup$
 $\{\tau.(P \parallel Q') \mid Q'. \tau.(Q') \in summands Q\} \cup$
 $\{a\{b\}.(P' \parallel Q) \mid a b P'. a\{b\}.P' \in summands P\} \cup$
 $\{a\{b\}.(P \parallel Q') \mid a b Q'. a\{b\}.Q' \in summands Q\} \cup$
 $\{a<x>.(P' \parallel Q) \mid a x P'. a<x>.P' \in summands P \wedge x \# Q\}$
 \cup
 $\{a<x>.(P \parallel Q') \mid a x Q'. a<x>.Q' \in summands Q \wedge x \#$
 $P\} \cup$
 $\{<\nu x>a\{x\}.(P' \parallel Q) \mid a x P'. <\nu x>a\{x\}.P' \in summands P$
 $\wedge x \# Q\} \cup$
 $\{<\nu x>a\{x\}.(P \parallel Q') \mid a x Q'. <\nu x>a\{x\}.Q' \in summands$
 $Q \wedge x \# P\} \cup$
 $\{\tau.(P'[x::=b] \parallel Q') \mid x P' b Q'. \exists a. a<x>.P' \in summands P$
 $\wedge a\{b\}.Q' \in summands Q\} \cup$
 $\{\tau.(P' \parallel (Q'[x::=b])) \mid b P' x Q'. \exists a. a\{b\}.P' \in summands$
 $P \wedge a<x>.Q' \in summands Q\} \cup$
 $\{\tau.(<\nu y>(P'[x::=y] \parallel Q')) \mid x P' y Q'. \exists a. a<x>.P' \in$
 $summands P \wedge <\nu y>a\{y\}.Q' \in summands Q \wedge y \# P\} \cup$
 $\{\tau.(<\nu y>(P' \parallel (Q'[x::=y]))) \mid y P' x Q'. \exists a. <\nu y>a\{y\}.P'$
 $\in summands P \wedge a<x>.Q' \in summands Q \wedge y \# Q\}$

lemma *finiteExpand*:
 fixes $P :: pi$

```

and   Q :: pi

shows finite(expandSet P Q)
⟨proof⟩

lemma expandHnf:
  fixes P :: pi
  and   Q :: pi

  shows  $\forall R \in (\text{expandSet } P \ Q). \text{hnf } R$ 
  ⟨proof⟩

inductive-set sumComposeSet :: (pi × pi set) set
where
  empty: (0, { }) ∈ sumComposeSet
| insert:  $\llbracket Q \in S; (P, S - \{Q\}) \in \text{sumComposeSet} \rrbracket \implies (P \oplus Q, S) \in \text{sumComposeSet}$ 

lemma expandAction:
  fixes P :: pi
  and   Q :: pi
  and   S :: pi set

  assumes (P, S) ∈ sumComposeSet
  and     Q ∈ S
  and     Q ↦ Rs

  shows P ↦ Rs
  ⟨proof⟩

lemma expandAction':
  fixes P :: pi
  and   Q :: pi
  and   R :: pi

  assumes (R, S) ∈ sumComposeSet
  and     R ↦ Rs

  shows  $\exists P \in S. P \mapsto Rs$ 
  ⟨proof⟩

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 $Phnf: \text{hnf } P$
and $Qhnf: \text{hnf } Q$

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 $(P \parallel Q \mapsto a\langle \nu x \rangle \prec P') = (R \mapsto a\langle \nu x \rangle \prec P')$
 $\langle \text{proof} \rangle$

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

assumes $Exp: (R, \text{expandSet } P \ Q) \in \text{sumComposeSet}$
and $Phnf: \text{hnf } P$
and $Qhnf: \text{hnf } Q$
and $Id: Id \subseteq Rel$

shows $P \parallel Q \rightsquigarrow[Rel] R$
 $\langle \text{proof} \rangle$

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

assumes $Exp: (R, \text{expandSet } P \ Q) \in \text{sumComposeSet}$
and $Phnf: \text{hnf } P$
and $Qhnf: \text{hnf } Q$
and $Id: Id \subseteq Rel$

shows $R \rightsquigarrow[Rel] P \parallel Q$
 $\langle \text{proof} \rangle$

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

assumes $(R, \text{expandSet } P \ Q) \in \text{sumComposeSet}$
and $\text{hnf } P$
and $\text{hnf } Q$

shows $P \parallel Q \sim R$
 $\langle \text{proof} \rangle$

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>

lemma *inputSuppPresBisim*:

fixes $P :: pi$
and $Q :: pi$
and $x :: name$

assumes $PSimQ: \bigwedge y. y \in supp(P, Q, x) \implies P[x::=y] \sim Q[x::=y]$

shows $a\langle x \rangle.P \sim a\langle x \rangle.Q$
<proof>

inductive *equiv* :: $pi \Rightarrow pi \Rightarrow bool$ (**infixr** $\langle \equiv_e \rangle$ 80)

where

$Refl: P \equiv_e P$
 $Sym: P \equiv_e Q \implies Q \equiv_e P$
 $Trans: \llbracket P \equiv_e Q; Q \equiv_e R \rrbracket \implies P \equiv_e R$

 $Match: [a \frown a]P \equiv_e P$
 $Match': a \neq b \implies [a \frown b]P \equiv_e \mathbf{0}$

 $Mismatch: a \neq b \implies [a \neq b]P \equiv_e P$
 $Mismatch': [a \neq a]P \equiv_e \mathbf{0}$

 $SumSym: P \oplus Q \equiv_e Q \oplus P$
 $SumAssoc: (P \oplus Q) \oplus R \equiv_e P \oplus (Q \oplus R)$
 $SumZero: P \oplus \mathbf{0} \equiv_e P$
 $SumIdemp: P \oplus P \equiv_e P$
 $SumRes: \langle \nu x \rangle (P \oplus Q) \equiv_e (\langle \nu x \rangle P) \oplus (\langle \nu x \rangle Q)$

 $ResNil: \langle \nu x \rangle \mathbf{0} \equiv_e \mathbf{0}$
 $ResInput: \llbracket x \neq a; x \neq y \rrbracket \implies \langle \nu x \rangle a \langle y \rangle . P \equiv_e a \langle y \rangle . (\langle \nu x \rangle P)$

| *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*: $\llbracket (R, \text{expandSet } P \ Q) \in \text{sumComposeSet}; \text{hnf } P; \text{hnf } Q \rrbracket \implies P$
| $Q \equiv_e R$

| *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>

lemma *SumPres'*:

fixes $P :: pi$
and $P' :: pi$
and $Q :: pi$
and $Q' :: pi$

assumes $P \text{ eq } P': P \equiv_e P'$

and $Q \text{ eq } Q': Q \equiv_e Q'$

shows $P \oplus Q \equiv_e P' \oplus Q'$

<proof>

lemma *sound*:

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

assumes $P \equiv_e Q$

shows $P \sim Q$

<proof>

lemma *zeroDest[dest]*:

```

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

shows  $(a\{b\}.P) \equiv_e \mathbf{0} \implies False$ 
and  $(a\langle x \rangle.P) \equiv_e \mathbf{0} \implies False$ 
and  $(\tau.(P)) \equiv_e \mathbf{0} \implies False$ 

and  $\mathbf{0} \equiv_e a\{b\}.P \implies False$ 
and  $\mathbf{0} \equiv_e a\langle x \rangle.P \implies False$ 
and  $\mathbf{0} \equiv_e \tau.(P) \implies False$ 
<proof>

```

```

lemma eq-eqv:
  fixes  $pi :: name\ prm$ 
  and  $x :: 'a :: pt-name$ 
  shows  $pi.(x=y) = ((pi.x)=(pi.y))$ 
<proof>

```

```

nominal-primrec depth ::  $pi \Rightarrow nat$  where
  depth  $\mathbf{0} = 0$ 
| depth  $(\tau.(P)) = 1 + (depth\ P)$ 
|  $a \# x \implies depth\ (a\langle x \rangle.P) = 1 + (depth\ P)$ 
| depth  $(a\{b\}.P) = 1 + (depth\ P)$ 
| depth  $([a\ \neg\ b]P) = (depth\ P)$ 
| depth  $([a\ \neq\ b]P) = (depth\ P)$ 
| depth  $(P \oplus Q) = max\ (depth\ P)\ (depth\ Q)$ 
| depth  $(P \parallel Q) = ((depth\ P) + (depth\ Q))$ 
| depth  $(\langle \nu x \rangle P) = (depth\ P)$ 
| depth  $(!P) = (depth\ P)$ 
<proof>

```

```

lemma depthEqvt[simp]:
  fixes  $P :: pi$ 
  and  $p :: name\ prm$ 

  shows  $depth(p \cdot P) = depth\ P$ 
<proof>

```

```

lemma depthInput[simp]:
  fixes  $a :: name$ 
  and  $x :: name$ 
  and  $P :: pi$ 

  shows  $depth\ (a\langle x \rangle.P) = 1 + (depth\ P)$ 
<proof>

```

```

nominal-primrec valid ::  $pi \Rightarrow bool$  where

```

```

  valid 0 = True
| valid ( $\tau.(P)$ ) = valid P
|  $x \# a \implies \text{valid } (a \langle x \rangle . P) = \text{valid } P$ 
| valid ( $a\{b\}.P$ ) = valid P
| valid ( $[a \frown b]P$ ) = valid P
| valid ( $[a \neq b]P$ ) = valid P
| valid ( $P \oplus Q$ ) = ((valid P)  $\wedge$  (valid Q))
| valid ( $P \parallel Q$ ) = ((valid P)  $\wedge$  (valid Q))
| valid ( $\langle \nu x \rangle P$ ) = valid P
| valid ( $!P$ ) = False
<proof>

```

```

lemma validEqvt[simp]:
  fixes P :: pi
  and p :: name prm

  shows valid(p · P) = valid P
<proof>

```

```

lemma validInput[simp]:
  fixes a :: name
  and x :: name
  and P :: pi

  shows valid (a  $\langle x \rangle$  . P) = valid P
<proof>

```

```

lemma depthMin[intro]:
  fixes P

  shows 0  $\leq$  depth P
<proof>

```

```

lemma hnfTransition:
  fixes P :: pi

  assumes hnf P
  and P  $\neq$  0

  shows  $\exists Rs. P \longmapsto Rs$ 
<proof>

```

```

definition uhnf :: pi  $\Rightarrow$  bool where
  uhnf P  $\equiv$  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 $P \oplus Q' \equiv_e P$
 $\langle \text{proof} \rangle$

lemma *uhnfSum*:
fixes $P :: pi$
and $Q :: pi$

assumes $Phnf: \text{uhnf } P$
and $Qhnf: \text{uhnf } Q$
and $\text{valid}P: \text{valid } P$
and $\text{valid}Q: \text{valid } Q$

shows $\exists R. \text{uhnf } R \wedge \text{valid } R \wedge P \oplus Q \equiv_e R \wedge (\text{depth } R) \leq (\text{depth } (P \oplus Q))$
 $\langle \text{proof} \rangle$

lemma *uhnfRes*:
fixes $x :: name$
and $P :: pi$

assumes $Phnf: \text{uhnf } P$
and $\text{valid}P: \text{valid } P$

shows $\exists P'. \text{uhnf } P' \wedge \text{valid } P' \wedge \langle \nu x \rangle P \equiv_e P' \wedge \text{depth } P' \leq \text{depth}(\langle \nu x \rangle P)$
 $\langle \text{proof} \rangle$

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 $\exists P'. \text{uhnf } P' \wedge \text{valid } P' \wedge P \equiv_e P' \wedge \text{depth } P' \leq \text{depth } P$
 $\langle \text{proof} \rangle$

lemma *hnfSummandsRemove*:
fixes $P :: pi$
and $Q :: pi$

assumes $P \in \text{summands } Q$
and $\text{uhnf } Q$

shows $(\text{summands } Q) - \{P' \mid P'. P' \in \text{summands } Q \wedge P' \equiv_e P\} = (\text{summands } Q) - \{P\}$
 $\langle \text{proof} \rangle$

lemma *pullSummand*:

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

assumes $PsummQ: P \in summands\ Q$
and $Qhnf: uhnf\ Q$

shows $\exists Q'. P \oplus Q' \equiv_e Q \wedge (summands\ Q') = ((summands\ Q) - \{x. \exists P'. x = P' \wedge P' \in (summands\ Q) \wedge P' \equiv_e P\}) \wedge uhnf\ Q'$
<proof>

lemma *nSym*:

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

assumes $\neg(P \equiv_e Q)$

shows $\neg(Q \equiv_e P)$
<proof>

lemma *summandsZero*:

fixes $P :: pi$

assumes $summands\ P = \{\}$
and $hnf\ P$

shows $P = \mathbf{0}$
<proof>

lemma *summandsZero'*:

fixes $P :: pi$

assumes $summP: summands\ P = \{\}$
and $Puhnf: uhnf\ P$

shows $P = \mathbf{0}$
<proof>

lemma *summandEquiv*:

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

assumes $Phnf: uhnf\ P$
and $Qhnf: uhnf\ Q$
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>

lemma *validSubst[simp]*:
 fixes $P :: pi$
 and $a :: name$
 and $b :: name$
 and $p :: pi$

 shows $valid(P[a:=b]) = valid P$
<proof>

lemma *validOutputTransition*:
 fixes $P :: pi$
 and $a :: name$
 and $b :: name$
 and $P' :: pi$

 assumes $P \mapsto a[b] \prec P'$
 and $valid P$

 shows $valid P'$
<proof>

lemma *validInputTransition*:
 fixes $P :: pi$
 and $a :: name$
 and $x :: name$
 and $P' :: pi$

 assumes $PTrans: P \mapsto a\langle x \rangle \prec P'$
 and $validP: valid P$

 shows $valid P'$
<proof>

lemma *validBoundOutputTransition*:
 fixes $P :: pi$
 and $a :: name$
 and $x :: name$
 and $P' :: pi$

 assumes $PTrans: P \mapsto a\langle \nu x \rangle \prec P'$
 and $validP: valid P$

 shows $valid P'$
<proof>

lemma *validTauTransition*:

```

fixes  $P :: pi$ 
and  $P' :: pi$ 

assumes  $PTrans: P \mapsto \tau \prec P'$ 
and  $validP: valid\ P$ 

shows  $valid\ P'$ 
⟨proof⟩

lemmas  $validTransition = validInputTransition\ validOutputTransition\ validTau-$ 
 $Transition\ validBoundOutputTransition$ 

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  $a\langle x \rangle.P' \in summands\ P \implies valid\ P'$ 
and  $\llbracket a \neq x; \langle \nu x \rangle a\{x\}.P' \in summands\ P \rrbracket \implies valid\ P'$ 
⟨proof⟩

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

assumes  $valid\ P$ 
and  $valid\ Q$ 
and  $uhnf\ P$ 
and  $uhnf\ Q$ 

shows  $\forall R \in (expandSet\ P\ Q). uhnf\ R \wedge valid\ R$ 
⟨proof⟩

lemma  $expandComplete:$ 
fixes  $F :: pi\ set$ 

assumes  $finite\ F$ 

shows  $\exists P. (P, F) \in sumComposeSet$ 
⟨proof⟩

lemma  $expandDepth:$ 

```

```

fixes  $F :: pi\ set$ 
and  $P :: pi$ 
and  $Q :: pi$ 

assumes  $(P, F) \in sumComposeSet$ 
and  $F \neq \{\}$ 

shows  $\exists Q \in F. depth\ P \leq depth\ Q \wedge (\forall R \in F. depth\ R \leq depth\ Q)$ 
<proof>

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

shows  $depth(P[a::=b]) = depth\ P$ 
<proof>

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>

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

assumes  $R \in expandSet\ P\ Q$ 
and  $hnf\ P$ 
and  $hnf\ Q$ 

shows  $depth\ R \leq depth(P \parallel Q)$ 
<proof>

lemma expandDepth':
fixes  $P :: pi$ 
and  $Q :: pi$ 

assumes  $Phnf: hnf\ P$ 

```

and $Qhnf: hnf\ Q$

shows $\exists R. (R, expandSet\ P\ Q) \in sumComposeSet \wedge depth\ R \leq depth(P \parallel Q)$
 $\langle proof \rangle$

lemma *validToHnf*:
fixes $P :: pi$

assumes *valid P*

shows $\exists Q. uhnf\ Q \wedge valid\ Q \wedge Q \equiv_e P \wedge (depth\ Q) \leq (depth\ P)$
 $\langle proof \rangle$

lemma *depthZero*:
fixes $P :: pi$

assumes $depth\ P = 0$
and *uhnf P*

shows $P = \mathbf{0}$
 $\langle proof \rangle$

lemma *completeAux*:
fixes $n :: nat$
and $P :: pi$
and $Q :: pi$

assumes $depth\ P + depth\ Q \leq n$
and *valid P*
and *valid Q*
and *uhnf P*
and *uhnf Q*
and $P \sim Q$

shows $P \equiv_e Q$
 $\langle proof \rangle$

lemma *complete*:
fixes $P :: pi$
and $Q :: pi$

assumes *validP: valid P*
and *validQ: valid Q*
and *PBisimQ: P ~ Q*

shows $P \equiv_e Q$
 $\langle proof \rangle$

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.