The pi-calculus

Jesper Bengtson

September 13, 2023

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

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 contians *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* (**0**)
  | *Output name name pi* (-{-}.- [*120*, *120*, *110*] *110*)
  | *Tau pi* (τ.- [*120*] *110*)
  | *Input name «name» pi* (-<->.- [*120*, *120*, *110*] *110*)
  | *Match name name pi* ([-⌢-]- [*120*, *120*, *110*] *110*)
  | *Mismatch name name pi* ([-≠-]- [*120*, *120*, *110*] *110*)

| *Sum pi pi* (**infixr** ⊕ *90*)
| *Par pi pi* (**infixr** ∥ *85*)
| *Res* «*name*» *pi* (<ν->- [*100*, *100*] *100*)
| *Bang pi* (!- [*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*<*x*>.*P* = *a*<*c*>.([(*x*, *c*)] • *P*)
⟨*proof*⟩

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

assumes *A1*: *c* ♯ *P*

shows <ν*a*>*P* = <ν*c*>([(*a*, *c*)] • *P*)
⟨*proof*⟩

definition *subst-name* :: *name* ⇒ *name* ⇒ *name* ⇒ *name* (-[-::=-] [*110*, *110*, *110*] *110*)
where
a[*b*::=*c*] ≡ *if* (*a* = *b*) *then c else a*

declare *subst-name-def*[*simp*]

lemma *subst-name-eqvt*[*eqvt*]:
fixes *p* :: *name prm*
and *a* :: *name*
and *b* :: *name*
and *c* :: *name*

shows *p* • (*a*[*b*::=*c*]) = (*p*• *a*)[(*p* • *b*)::=(*p* • *c*)]
⟨*proof*⟩

nominal-primrec (*freshness-context*: (*c*::*name*, *d*::*name*))
subs :: *pi* ⇒ *name* ⇒ *name* ⇒ *pi* (-[-::=-] [*100*,*100*,*100*] *100*)

where
  $\mathbf{0}[c::=d] = \mathbf{0}$
| $\tau.(P)[c::=d] = \tau.(P[c::=d])$
| $a\{b\}.P[c::=d] = (a[c::=d])\{(b[c::=d])\}.(P[c::=d])$
| $[\![x \neq a;\ x \neq c;\ x \neq d]\!] \Longrightarrow (a{<}x{>}.P)[c::=d] = (a[c::=d]){<}x{>}.(P[c::=d])$
| $[a\frown b]P[c::=d] = [(a[c::=d])\frown(b[c::=d])](P[c::=d])$
| $[a{\neq}b]P[c::=d] = [(a[c::=d]){\neq}(b[c::=d])](P[c::=d])$
| $(P \oplus Q)[c::=d] = (P[c::=d]) \oplus (Q[c::=d])$
| $(P \parallel Q)[c::=d] = (P[c::=d]) \parallel (Q[c::=d])$
| $[\![x \neq c;\ x \neq d]\!] \Longrightarrow ({<}\nu x{>}P)[c::=d] = {<}\nu x{>}(P[c::=d])$
| $!P[c::=d] = !(P[c::=d])$
⟨*proof*⟩

lemma *forget*:
  fixes a :: *name*
  and P :: *pi*
  and b :: *name*

  assumes $a \sharp P$

  shows $P[a::=b] = P$
⟨*proof*⟩

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

  assumes $a \neq b$

  shows $a \sharp P[a::=b]$
⟨*proof*⟩

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

  shows $P[a::=a] = P$
⟨*proof*⟩

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

  assumes $c \sharp P$

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

⟨*proof*⟩

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

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

shows $a \sharp P[b::=c]$
⟨*proof*⟩

lemma *eqvt-subs*[*eqvt*]:
fixes p :: *name prm*
and P :: *pi*
and a :: *name*
and b :: *name*

shows $p \cdot (P[a::=b]) = (p \cdot P)[(p \cdot a)::=(p \cdot b)]$
⟨*proof*⟩

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 $(a<x>.P)[b::=c] = (a[b::=c])<x>.(P[b::=c])$
⟨*proof*⟩

lemma *injPermSubst*:
fixes P :: *pi*
and a :: *name*
and b :: *name*

assumes $b \sharp P$

shows $[(a, b)] \cdot P = P[a::=b]$
⟨*proof*⟩

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

assumes $b \sharp P$

shows $<\nu a>P = <\nu b>(P[a::=b])$
⟨*proof*⟩

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

shows $a \sharp <\nu a>P$
⟨*proof*⟩

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

assumes $b \sharp P$

shows $(<\nu a>P)[a::=b] = <\nu b>(P[a::=b])$
⟨*proof*⟩

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

shows $supp(P[a::=b]) \subseteq insert\ b\ ((supp\ P) - \{a\})$
⟨*proof*⟩

primrec $seqSubs :: pi \Rightarrow (name \times name)\ list \Rightarrow pi$ (-[<->] [*100*,*100*] *100*) **where**
$P[<[]>] = P$
$|\ P[<(x\#\sigma)>] = (P[(fst\ x)::=(snd\ x)])[<\sigma>]$

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

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

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

shows $x \neq$ *seq-subst-name a s*
⟨*proof*⟩

lemma *seqSubstZero*[*simp*]:
fixes σ :: (*name* × *name*) *list*

shows $\mathbf{0}[<\sigma>] = \mathbf{0}$
⟨*proof*⟩

lemma *seqSubstTau*[*simp*]:
fixes P :: *pi*
and σ :: (*name* × *name*) *list*

shows $(\tau.(P))[<\sigma>] = \tau.(P[<\sigma>])$
⟨*proof*⟩

lemma *seqSubstOutput*[*simp*]:
fixes a :: *name*
and b :: *name*
and P :: *pi*
and σ :: (*name* × *name*) *list*

shows $(a\{b\}.P)[<\sigma>] =$ (*seq-subst-name a* σ){(*seq-subst-name b* σ)}.$(P[<\sigma>])$
⟨*proof*⟩

lemma *seqSubstInput*[*simp*]:
fixes a :: *name*
and x :: *name*
and P :: *pi*
and σ :: (*name* × *name*) *list*

assumes $x \sharp \sigma$

shows $(a<x>.P)[<\sigma>] =$ (*seq-subst-name a* σ)$<x>.(P[<\sigma>])$
⟨*proof*⟩

lemma *seqSubstMatch*[*simp*]:
fixes a :: *name*
and b :: *name*
and P :: *pi*
and σ :: (*name* × *name*) *list*

shows $([a \frown b]P)[<\sigma>] =$ [(*seq-subst-name a* σ)⌢(*seq-subst-name b* σ)]$(P[<\sigma>])$
⟨*proof*⟩

lemma *seqSubstMismatch*[*simp*]:
  fixes a :: *name*
  and b :: *name*
  and P :: *pi*
  and σ :: (*name* × *name*) *list*

  shows $([a \neq b]P)[<\sigma>] = [(\textit{seq-subst-name}\ a\ \sigma) \neq (\textit{seq-subst-name}\ b\ \sigma)](P[<\sigma>])$
⟨*proof*⟩

lemma *seqSubstSum*[*simp*]:
  fixes P :: *pi*
  and Q :: *pi*
  and σ :: (*name* × *name*) *list*

  shows $(P \oplus Q)[<\sigma>] = (P[<\sigma>]) \oplus (Q[<\sigma>])$
⟨*proof*⟩

lemma *seqSubstPar*[*simp*]:
  fixes P :: *pi*
  and Q :: *pi*
  and σ :: (*name* × *name*) *list*

  shows $(P \parallel Q)[<\sigma>] = (P[<\sigma>]) \parallel (Q[<\sigma>])$
⟨*proof*⟩

lemma *seqSubstRes*[*simp*]:
  fixes x :: *name*
  and P :: *pi*
  and σ :: (*name* × *name*) *list*

  assumes $x \sharp \sigma$

  shows $(<\nu x>P)[<\sigma>] = <\nu x>(P[<\sigma>])$
⟨*proof*⟩

lemma *seqSubstBang*[*simp*]:
  fixes P :: *pi*
  and s :: (*name* × *name*) *list*

  shows $(!P)[<\sigma>] = !(P[<\sigma>])$
⟨*proof*⟩

lemma *seqSubstEqvt*[*eqvt*, *simp*]:
  fixes P :: *pi*
  and σ :: (*name* × *name*) *list*
  and p :: *name prm*

  shows $p \cdot (P[<\sigma>]) = (p \cdot P)[<(p \cdot \sigma)>]$

```
⟨proof⟩

lemma seqSubstAppend[simp]:
  fixes P  :: pi
  and   σ  :: (name × name) list
  and   σ' :: (name × name) list

  shows P[<(σ@σ')>] = (P[<σ>])[<σ'>]
⟨proof⟩

lemma freshSubstChain[intro]:
  fixes P :: pi
  and   σ :: (name × name) list
  and   a :: name

  assumes a ♯ P
  and     a ♯ σ

  shows a ♯ P[<σ>]
⟨proof⟩

end

theory Late-Semantics
  imports Agent
begin

nominal-datatype subject = InputS name
                         | BoundOutputS name

nominal-datatype freeRes = OutputR name name             (-[-] [130, 130] 110)
                         | TauR                          (τ 130)

nominal-datatype residual = BoundR subject «name» pi (-«-» ≺ - [80, 80, 80]
80)
                          | FreeR freeRes pi            (- ≺ - [80, 80] 80)

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

abbreviation Transitions-Inputjudge :: name ⇒ name ⇒ pi ⇒ residual (-<-> ≺
- [80, 80, 80] 80)
where a<x> ≺ P' ≡ ((InputS a)«x» ≺ P')

abbreviation Transitions-BoundOutputjudge :: name ⇒ name ⇒ pi ⇒ residual
(-<ν-> ≺ - [80, 80, 80] 80)
where a<νx> ≺ P' ≡ (BoundR (BoundOutputS a) x P')

inductive transitions :: pi ⇒ residual ⇒ bool     (- ⟼ - [80, 80] 80)
where
```

Tau: $\tau.(P) \longmapsto \tau \prec P$
| *Input*: $x \neq a \Longrightarrow a{<}x{>}.P \longmapsto a{<}x{>} \prec P$
| *Output*: $a\{b\}.P \longmapsto a[b] \prec P$

| *Match*: $[\![P \longmapsto Rs]\!] \Longrightarrow [b\frown b]P \longmapsto Rs$
| *Mismatch*: $[\![P \longmapsto Rs;\ a \neq b]\!] \Longrightarrow [a{\neq}b]P \longmapsto Rs$

| *Open*: $[\![P \longmapsto a[b] \prec P';\ a \neq b]\!] \Longrightarrow {<}\nu b{>}P \longmapsto a{<}\nu b{>} \prec P'$
| *Sum1*: $[\![P \longmapsto Rs]\!] \Longrightarrow (P \oplus Q) \longmapsto Rs$
| *Sum2*: $[\![Q \longmapsto Rs]\!] \Longrightarrow (P \oplus Q) \longmapsto Rs$

| *Par1B*: $[\![P \longmapsto a«x» \prec P';\ x \sharp P;\ x \sharp Q;\ x \sharp a]\!] \Longrightarrow P \parallel Q \longmapsto a«x» \prec (P' \parallel Q)$
| *Par1F*: $[\![P \longmapsto \alpha \prec P']\!] \Longrightarrow P \parallel Q \longmapsto \alpha \prec (P' \parallel Q)$
| *Par2B*: $[\![Q \longmapsto a«x» \prec Q';\ x \sharp P;\ x \sharp Q;\ x \sharp a]\!] \Longrightarrow P \parallel Q \longmapsto a«x» \prec (P \parallel Q')$
| *Par2F*: $[\![Q \longmapsto \alpha \prec Q']\!] \Longrightarrow P \parallel Q \longmapsto \alpha \prec (P \parallel Q')$

| *Comm1*: $[\![P \longmapsto a{<}x{>} \prec P';\ Q \longmapsto a[b] \prec Q';\ x \sharp P;\ x \sharp Q;\ x \neq a;\ x \neq b;\ x \sharp Q']\!] \Longrightarrow P \parallel Q \longmapsto \tau \prec P'[x{::=}b] \parallel Q'$
| *Comm2*: $[\![P \longmapsto a[b] \prec P';\ Q \longmapsto a{<}x{>} \prec Q';\ x \sharp P;\ x \sharp Q;\ x \neq a;\ x \neq b;\ x \sharp P']\!] \Longrightarrow P \parallel Q \longmapsto \tau \prec P' \parallel Q'[x{::=}b]$
| *Close1*: $[\![P \longmapsto a{<}x{>} \prec P';\ Q \longmapsto a{<}\nu y{>} \prec Q';\ x \sharp P;\ x \sharp Q;\ y \sharp P;\ y \sharp Q;\ x \neq a;\ x \sharp Q';\ y \neq a;\ y \sharp P';\ x \neq y]\!] \Longrightarrow P \parallel Q \longmapsto \tau \prec {<}\nu y{>}(P'[x{::=}y] \parallel Q')$
| *Close2*: $[\![P \longmapsto a{<}\nu y{>} \prec P';\ Q \longmapsto a{<}x{>} \prec Q';\ x \sharp P;\ x \sharp Q;\ y \sharp P;\ y \sharp Q;\ x \neq a;\ x \sharp P';\ y \neq a;\ y \sharp Q';\ x \neq y]\!] \Longrightarrow P \parallel Q \longmapsto \tau \prec {<}\nu y{>}(P' \parallel Q'[x{::=}y])$

| *ResB*: $[\![P \longmapsto a«x» \prec P';\ y \sharp a;\ y \neq x;\ x \sharp P;\ x \sharp a]\!] \Longrightarrow {<}\nu y{>}P \longmapsto a«x» \prec {<}\nu y{>}P'$
| *ResF*: $[\![P \longmapsto \alpha \prec P';\ y \sharp \alpha]\!] \Longrightarrow {<}\nu y{>}P \longmapsto \alpha \prec {<}\nu y{>}P'$

| *Bang*: $[\![P \parallel !P \longmapsto Rs]\!] \Longrightarrow !P \longmapsto Rs$

equivariance *transitions*
nominal-inductive *transitions*
⟨*proof*⟩

lemma *alphaBoundResidual*:
fixes a :: *subject*
and x :: *name*
and P :: *pi*
and x' :: *name*

assumes *A1*: $x' \sharp P$

shows $a«x» \prec P = a«x'» \prec ([(x, x')] \cdot P)$
⟨*proof*⟩

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

assumes $P \longmapsto Rs$
and $x \sharp P$

shows $x \sharp Rs$
⟨*proof*⟩

lemma *freshBoundDerivative*:
assumes $P \longmapsto a«x» \prec P'$
and $y \sharp P$

shows $y \sharp a$
and $y \neq x \Longrightarrow y \sharp P'$
⟨*proof*⟩

lemma *freshFreeDerivative*:
fixes P :: *pi*
and α :: *freeRes*
and P' :: *pi*
and y :: *name*

assumes $P \longmapsto \alpha \prec P'$
and $y \sharp P$

shows $y \sharp \alpha$
and $y \sharp P'$
⟨*proof*⟩

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

assumes $b \sharp P$

shows $(P[a::=b])[b::=c] = P[a::=c]$
⟨*proof*⟩

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

shows $a<x>.P \longmapsto a<x> \prec P$
⟨*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 $P \longmapsto a«x» \prec P'$
and $x \sharp Q$

shows $P \parallel Q \longmapsto a«x» \prec P' \parallel Q$
⟨*proof*⟩

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

assumes *QTrans*: $Q \longmapsto a«x» \prec Q'$
and $x \sharp P$

shows $P \parallel Q \longmapsto a«x» \prec P \parallel Q'$
⟨*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*: $P \longmapsto a<x> \prec P'$
and *QTrans*: $Q \longmapsto a[b] \prec Q'$

shows $P \parallel Q \longmapsto \tau \prec P'[x::=b] \parallel Q'$
⟨*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*: $P \longmapsto a[b] \prec P'$
and *QTrans*: $Q \longmapsto a{<}x{>} \prec Q'$

shows $P \parallel Q \longmapsto \tau \prec P' \parallel (Q'[x{::=}b])$
⟨*proof*⟩

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*: $P \longmapsto a{<}x{>} \prec P'$
and *QTrans*: $Q \longmapsto a{<}\nu y{>} \prec Q'$
and $y \sharp P$

shows $P \parallel Q \longmapsto \tau \prec {<}\nu y{>}(P'[x{::=}y] \parallel Q')$
⟨*proof*⟩

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*: $P \longmapsto a{<}\nu y{>} \prec P'$
and *QTrans*: $Q \longmapsto a{<}x{>} \prec Q'$
and $y \sharp Q$

shows $P \parallel Q \longmapsto \tau \prec {<}\nu y{>}(P' \parallel (Q'[x{::=}y]))$
⟨*proof*⟩

lemma *ResB*:
fixes P :: *pi*
and a :: *subject*
and x :: *name*
and P' :: *pi*

and y :: *name*

assumes *PTrans*: $P \longmapsto a«x» \prec P'$
and $y \sharp a$
and $y \neq x$

shows $<\nu y>P \longmapsto a«x» \prec <\nu y>P'$
⟨*proof*⟩

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* ⇒ *pi* ⇒ *name* ⇒ *name* ⇒ *pi* ⇒ *bool*
and C :: *'a::fs-name*

assumes *Trans*: $P \longmapsto a[b] \prec P'$
and $\bigwedge a\ b\ P\ C.\ F\ C\ (a\{b\}.P)\ a\ b\ P$
and $\bigwedge P\ a\ b\ P'\ c\ C.\ [\![P \longmapsto OutputR\ a\ b \prec P';\ \bigwedge C.\ F\ C\ P\ a\ b\ P']\!] \Longrightarrow F\ C\ ([c\frown c]P)\ a\ b\ P'$
and $\bigwedge P\ a\ b\ P'\ c\ d\ C.\ [\![P \longmapsto OutputR\ a\ b \prec P';\ \bigwedge C.\ F\ C\ P\ a\ b\ P';\ c \neq d]\!] \Longrightarrow F\ C\ ([c{\neq}d]P)\ a\ b\ P'$
and $\bigwedge P\ a\ b\ P'\ Q\ C.\ [\![P \longmapsto OutputR\ a\ b \prec P';\ \bigwedge C.\ F\ C\ P\ a\ b\ P']\!] \Longrightarrow F\ C\ (P \oplus Q)\ a\ b\ P'$
and $\bigwedge Q\ a\ b\ Q'\ P\ C.\ [\![Q \longmapsto OutputR\ a\ b \prec Q';\ \bigwedge C.\ F\ C\ Q\ a\ b\ Q']\!] \Longrightarrow F\ C\ (P \oplus Q)\ a\ b\ Q'$
and $\bigwedge P\ a\ b\ P'\ Q\ C.\ [\![P \longmapsto OutputR\ a\ b \prec P';\ \bigwedge C.\ F\ C\ P\ a\ b\ P']\!] \Longrightarrow F\ C\ (P \parallel Q)\ a\ b\ (P' \parallel Q)$
and $\bigwedge Q\ a\ b\ Q'\ P\ C.\ [\![Q \longmapsto OutputR\ a\ b \prec Q';\ \bigwedge C.\ F\ C\ Q\ a\ b\ Q']\!] \Longrightarrow F\ C\ (P \parallel Q)\ a\ b\ (P \parallel Q')$
and $\bigwedge P\ a\ b\ P'\ x\ C.\ [\![P \longmapsto OutputR\ a\ b \prec P';\ x \neq a;\ x \neq b;\ x \sharp C;\ \bigwedge C.\ F\ C\ P\ a\ b\ P']\!] \Longrightarrow$
$F\ C\ (<\nu x>P)\ a\ b\ (<\nu x>P')$
and $\bigwedge P\ a\ b\ P'\ C.\ [\![P \parallel !P \longmapsto OutputR\ a\ b \prec P';\ \bigwedge C.\ F\ C\ (P \parallel !P)\ a\ b\ P']\!] \Longrightarrow F\ C\ (!P)\ a\ b\ P'$

shows $F\ C\ P\ a\ b\ P'$
⟨*proof*⟩

lemma *inputInduct*[*consumes 2*, *case-names Input Match Mismatch Sum1 Sum2 Par1 Par2 Res Bang*]:
fixes P :: *pi*
and a :: *name*
and x :: *name*
and P' :: *pi*
and F :: (*'a::fs-name*) ⇒ *pi* ⇒ *name* ⇒ *name* ⇒ *pi* ⇒ *bool*
and C :: *'a::fs-name*

assumes a: $P \longmapsto a{<}x{>} \prec P'$
and $x \sharp P$
and *cInput*: $\bigwedge a\ x\ P\ C.\ F\ C\ (a{<}x{>}.P)\ a\ x\ P$
and *cMatch*: $\bigwedge P\ a\ x\ P'\ b\ C.\ [\![P \longmapsto a{<}x{>} \prec P';\ \bigwedge C.\ F\ C\ P\ a\ x\ P']\!] \Longrightarrow F\ C\ ([b\frown b]P)\ a\ x\ P'$
and *cMismatch*: $\bigwedge P\ a\ x\ P'\ b\ c\ C.\ [\![P \longmapsto a{<}x{>} \prec P';\ \bigwedge C.\ F\ C\ P\ a\ x\ P';\ b \neq c]\!] \Longrightarrow F\ C\ ([b{\neq}c]P)\ a\ x\ P'$
and *cSum1*: $\bigwedge P\ Q\ a\ x\ P'\ C.\ [\![P \longmapsto a{<}x{>} \prec P';\ \bigwedge C.\ F\ C\ P\ a\ x\ P']\!] \Longrightarrow F\ C\ (P \oplus Q)\ a\ x\ P'$
and *cSum2*: $\bigwedge P\ Q\ a\ x\ Q'\ C.\ [\![Q \longmapsto a{<}x{>} \prec Q';\ \bigwedge C.\ F\ C\ Q\ a\ x\ Q']\!] \Longrightarrow F\ C\ (P \oplus Q)\ a\ x\ Q'$
and *cPar1B*: $\bigwedge P\ P'\ Q\ a\ x\ C.\ [\![P \longmapsto a{<}x{>} \prec P';\ x \sharp P;\ x \sharp Q;\ x \neq a;\ \bigwedge C.\ F\ C\ P\ a\ x\ P']\!] \Longrightarrow$
$F\ C\ (P \parallel Q)\ a\ x\ (P' \parallel Q)$
and *cPar2B*: $\bigwedge P\ Q\ Q'\ a\ x\ C.\ [\![Q \longmapsto a{<}x{>} \prec Q';\ x \sharp P;\ x \sharp Q;\ x \neq a;\ \bigwedge C.\ F\ C\ Q\ a\ x\ Q']\!] \Longrightarrow$
$F\ C\ (P \parallel Q)\ a\ x\ (P \parallel Q')$
and *cResB*: $\bigwedge P\ P'\ a\ x\ y\ C.\ [\![P \longmapsto a{<}x{>} \prec P';\ y \neq a;\ y \neq x;\ y \sharp C;$
$\bigwedge C.\ F\ C\ P\ a\ x\ P']\!] \Longrightarrow F\ C\ ({<}\nu y{>}P)\ a\ x\ ({<}\nu y{>}P')$
and *cBang*: $\bigwedge P\ a\ x\ P'\ C.\ [\![P \parallel !P \longmapsto a{<}x{>} \prec P';\ \bigwedge C.\ F\ C\ (P \parallel !P)\ a\ x\ P']\!] \Longrightarrow$
$F\ C\ (!P)\ a\ x\ P'$
shows $F\ C\ P\ a\ x\ P'$
⟨*proof*⟩

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*) ⇒ *pi* ⇒ *name* ⇒ *name* ⇒ *pi* ⇒ *bool*
and C :: '*a*::*fs-name*

assumes a: $P \longmapsto a{<}\nu x{>} \prec P'$
and $x \sharp P$
and *cMatch*: $\bigwedge P\ a\ x\ P'\ b\ C.\ [\![P \longmapsto a{<}\nu x{>} \prec P';\ \bigwedge C.\ F\ C\ P\ a\ x\ P']\!] \Longrightarrow F\ C\ ([b\frown b]P)\ a\ x\ P'$
and *cMismatch*: $\bigwedge P\ a\ x\ P'\ b\ c\ C.\ [\![P \longmapsto a{<}\nu x{>} \prec P';\ \bigwedge C.\ F\ C\ P\ a\ x\ P';\ b \neq c]\!] \Longrightarrow F\ C\ ([b{\neq}c]P)\ a\ x\ P'$
and *cOpen*: $\bigwedge P\ a\ x\ P'\ C.\ [\![P \longmapsto (OutputR\ a\ x) \prec P';\ a \neq x]\!] \Longrightarrow F\ C\ ({<}\nu x{>}P)\ a\ x\ P'$
and *cSum1*: $\bigwedge P\ Q\ a\ x\ P'\ C.\ [\![P \longmapsto a{<}\nu x{>} \prec P';\ \bigwedge C.\ F\ C\ P\ a\ x\ P']\!] \Longrightarrow F\ C\ (P \oplus Q)\ a\ x\ P'$
and *cSum2*: $\bigwedge P\ Q\ a\ x\ Q'\ C.\ [\![Q \longmapsto a{<}\nu x{>} \prec Q';\ \bigwedge C.\ F\ C\ Q\ a\ x\ Q']\!] \Longrightarrow F\ C\ (P \oplus Q)\ a\ x\ Q'$
and *cPar1B*: $\bigwedge P\ P'\ Q\ a\ x\ C.\ [\![P \longmapsto a{<}\nu x{>} \prec P';\ x \sharp Q;\ \bigwedge C.\ F\ C\ P\ a\ x\ P']\!] \Longrightarrow$

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

and *cPar2B*: $\bigwedge P\ Q\ Q'\ a\ x\ C.\ [\![Q \longmapsto a{<}\nu x{>} \prec Q';\ x \sharp P;\ \bigwedge C.\ F\ C\ Q\ a\ x\ Q']\!] \Longrightarrow$

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

and *cResB*: $\bigwedge P\ P'\ a\ x\ y\ C.\ [\![P \longmapsto a{<}\nu x{>} \prec P';\ y \neq a;\ y \neq x;\ y \sharp C;$
$\bigwedge C.\ F\ C\ P\ a\ x\ P']\!] \Longrightarrow F\ C\ ({<}\nu y{>}P)\ a\ x\ ({<}\nu y{>}P')$

and *cBang*: $\bigwedge P\ a\ x\ P'\ C.\ [\![P \parallel !P \longmapsto a{<}\nu x{>} \prec P';\ \bigwedge C.\ F\ C\ (P \parallel !P)\ a\ x\ P']\!] \Longrightarrow$

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

shows $F\ C\ P\ a\ x\ P'$

⟨*proof*⟩

lemma *tauInduct*[*consumes 1*, *case-names Tau Match Mismatch Sum1 Sum2 Par1 Par2 Comm1 Comm2 Close1 Close2 Res Bang*]:
fixes P :: *pi*
and P' :: *pi*
and F :: *'a::fs-name* ⇒ *pi* ⇒ *pi* ⇒ *bool*
and C :: *'a::fs-name*

assumes *Trans*: $P \longmapsto \tau \prec P'$
and $\bigwedge P\ C.\ F\ C\ (\tau.(P))\ P$
and $\bigwedge P\ P'\ c\ C.\ [\![P \longmapsto \tau \prec P';\ \bigwedge C.\ F\ C\ P\ P']\!] \Longrightarrow F\ C\ ([c\frown c]P)\ P'$
and $\bigwedge P\ P'\ c\ d\ C.\ [\![P \longmapsto \tau \prec P';\ \bigwedge C.\ F\ C\ P\ P';\ c \neq d]\!] \Longrightarrow F\ C\ ([c{\neq}d]P)\ P'$
and $\bigwedge P\ P'\ Q\ C.\ [\![P \longmapsto \tau \prec P';\ \bigwedge C.\ F\ C\ P\ P']\!] \Longrightarrow F\ C\ (P \oplus Q)\ P'$
and $\bigwedge Q\ Q'\ P\ C.\ [\![Q \longmapsto \tau \prec Q';\ \bigwedge C.\ F\ C\ Q\ Q']\!] \Longrightarrow F\ C\ (P \oplus Q)\ Q'$
and $\bigwedge P\ P'\ Q\ C.\ [\![P \longmapsto \tau \prec P';\ \bigwedge C.\ F\ C\ P\ P']\!] \Longrightarrow F\ C\ (P \parallel Q)\ (P' \parallel Q)$
and $\bigwedge Q\ Q'\ P\ C.\ [\![Q \longmapsto \tau \prec Q';\ \bigwedge C.\ F\ C\ Q\ Q']\!] \Longrightarrow F\ C\ (P \parallel Q)\ (P \parallel Q')$
and $\bigwedge P\ a\ x\ P'\ Q\ b\ Q'\ C.\ [\![P \longmapsto (\mathit{BoundR}\ (\mathit{InputS}\ a)\ x\ P');\ Q \longmapsto \mathit{OutputR}\ a\ b \prec Q';\ x \sharp P;\ x \sharp Q;\ x \sharp C]\!] \Longrightarrow F\ C\ (P \parallel Q)\ (P'[x{::=}b] \parallel Q')$
and $\bigwedge P\ a\ b\ P'\ Q\ x\ Q'\ C.\ [\![P \longmapsto \mathit{OutputR}\ a\ b \prec P';\ Q \longmapsto (\mathit{BoundR}\ (\mathit{InputS}\ a)\ x\ Q');\ x \sharp P;\ x \sharp Q;\ x \sharp C]\!] \Longrightarrow F\ C\ (P \parallel Q)\ (P' \parallel Q'[x{::=}b])$
and $\bigwedge P\ a\ x\ P'\ Q\ y\ Q'\ C.\ [\![P \longmapsto (\mathit{BoundR}\ (\mathit{InputS}\ a)\ x\ P');\ Q \longmapsto a{<}\nu y{>} \prec Q';\ x \sharp P;\ x \sharp Q;\ x \sharp C;\ y \sharp P;\ y \sharp Q;\ y \sharp C;\ x \neq y]\!] \Longrightarrow F\ C\ (P \parallel Q)\ ({<}\nu y{>}(P'[x{::=}y] \parallel Q'))$
and $\bigwedge P\ a\ y\ P'\ Q\ x\ Q'\ C.\ [\![P \longmapsto a{<}\nu y{>} \prec P';\ Q \longmapsto (\mathit{BoundR}\ (\mathit{InputS}\ a)\ x\ Q');\ x \sharp P;\ x \sharp Q;\ x \sharp C;\ y \sharp P;\ y \sharp Q;\ y \sharp C;\ x \neq y]\!] \Longrightarrow F\ C\ (P \parallel Q)\ ({<}\nu y{>}(P' \parallel Q'[x{::=}y]))$
and $\bigwedge P\ P'\ x\ C.\ [\![P \longmapsto \tau \prec P';\ x \sharp C;\ \bigwedge C.\ F\ C\ P\ P']\!] \Longrightarrow$
$F\ C\ ({<}\nu x{>}P)\ ({<}\nu x{>}P')$
and $\bigwedge P\ P'\ C.\ [\![P \parallel !P \longmapsto \tau \prec P';\ \bigwedge C.\ F\ C\ (P \parallel !P)\ P']\!] \Longrightarrow F\ C\ (!P)\ P'$

shows $F\ C\ P\ P'$

⟨*proof*⟩

inductive *bangPred* :: *pi* ⇒ *pi* ⇒ *bool*
where
aux1: *bangPred* P $(!P)$
| *aux2*: *bangPred* P $(P \parallel !P)$

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

lemma *tauCases*[*consumes 1*, *case-names cTau*]:
 fixes P :: *pi*
 and α :: *freeRes*
 and P' :: *pi*

 assumes $\tau.(P) \longmapsto \alpha \prec P'$
 and $[\![\alpha = \tau;\ P = P']\!] \Longrightarrow Prop\ (\tau)\ P$

 shows $Prop\ \alpha\ P'$
⟨*proof*⟩

lemma *outputCases*[*consumes 1*, *case-names cOutput*]:
 fixes a :: *name*
 and b :: *name*
 and P :: *pi*
 and α :: *freeRes*
 and P' :: *pi*

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

 shows $Prop\ \alpha\ P'$
⟨*proof*⟩

lemma *zeroTrans*[*dest*]:
 fixes Rs :: *residual*

 assumes $\mathbf{0} \longmapsto Rs$

shows *False*
⟨*proof*⟩

lemma *resZeroTrans*[*dest*]:
fixes x :: *name*
and *Rs* :: *residual*

assumes $<\nu x>\mathbf{0} \longmapsto Rs$

shows *False*
⟨*proof*⟩

lemma *matchTrans*[*dest*]:
fixes a :: *name*
and b :: *name*
and P :: *pi*
and *Rs* :: *residual*

assumes $[a\frown b]P \longmapsto Rs$
and $a\neq b$

shows *False*
⟨*proof*⟩

lemma *mismatchTrans*[*dest*]:
fixes a :: *name*
and P :: *pi*
and *Rs* :: *residual*

assumes $[a\neq a]P \longmapsto Rs$

shows *False*
⟨*proof*⟩

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

assumes *Input*: $a<x>.P \longmapsto b«y» \prec yP'$
and $y \neq a$
and $y \neq x$
and $y \sharp P$
and A: $[\![b = InputS\ a;\ yP' = ([(x,\ y)] \cdot P)]\!] \Longrightarrow Prop\ (InputS\ a)\ y\ ([(x,\ y)] \cdot P)$

shows $Prop\ b\ y\ yP'$
⟨*proof*⟩

lemma *tauBoundTrans*[*dest*]:
fixes P :: *pi*
and a :: *subject*
and x :: *name*
and P' :: *pi*

assumes $\tau.(P) \longmapsto a«x» \prec P'$

shows *False*
⟨*proof*⟩

lemma *tauOutputTrans*[*dest*]:
fixes P :: *pi*
and a :: *name*
and b :: *name*
and P' :: *pi*

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

shows *False*
⟨*proof*⟩

lemma *inputFreeTrans*[*dest*]:
fixes a :: *name*
and x :: *name*
and P :: *pi*
and α :: *freeRes*
and P' :: *pi*

assumes $a<x>.P \longmapsto \alpha \prec P'$

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 $a<x>.P \longmapsto b<\nu y> \prec P'$

shows *False*
⟨*proof*⟩

lemma *outputTauTrans*[*dest*]:

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

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

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 $a\{b\}.P \longmapsto c«x» \prec P'$

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 $a\{b\}.P \longmapsto c[d] \prec P'$
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 $a\{b\}.P \longmapsto \alpha \prec P'$
and $a \sharp \alpha \vee b \sharp \alpha$

shows *False*
⟨*proof*⟩

lemma *inputIneqTrans*[*dest*]:
  fixes a :: *name*
  and x :: *name*
  and P :: *pi*
  and b :: *subject*
  and y :: *name*
  and P' :: *pi*

  assumes $a{<}x{>}.P \longmapsto b«y» \prec P'$
  and $a \sharp b$

  shows *False*
$\langle proof \rangle$

lemma *resTauBoundTrans*[*dest*]:
  fixes x :: *name*
  and P :: *pi*
  and a :: *subject*
  and y :: *name*
  and P' :: *pi*

  assumes ${<}\nu x{>}\tau.(P) \longmapsto a«y» \prec P'$

  shows *False*
$\langle proof \rangle$

lemma *resTauOutputTrans*[*dest*]:
  fixes x :: *name*
  and P :: *pi*
  and a :: *name*
  and b :: *name*
  and P' :: *pi*

  assumes ${<}\nu x{>}\tau.(P) \longmapsto a[b] \prec P'$

  shows *False*
$\langle proof \rangle$

lemma *resInputFreeTrans*[*dest*]:
  fixes x :: *name*
  fixes a :: *name*
  and y :: *name*
  and P :: *pi*
  and α :: *freeRes*
  and P' :: *pi*

  assumes ${<}\nu x{>}a{<}y{>}.P \longmapsto \alpha \prec P'$

shows *False*
⟨*proof*⟩

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 $<\nu x>a<y>.P \longmapsto b<\nu z> \prec P'$

shows *False*
⟨*proof*⟩

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

assumes $<\nu x>a\{b\}.P \longmapsto \tau \prec P'$

shows *False*
⟨*proof*⟩

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 $<\nu x>a\{b\}.P \longmapsto c<y> \prec P'$

shows *False*
⟨*proof*⟩

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

and $P' :: pi$

assumes $<\nu x>a\{x\}.P \longmapsto b[y] \prec P'$

shows *False*
⟨*proof*⟩

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

shows $<\nu x>x\{b\}.P \longmapsto Rs \Longrightarrow False$
and $<\nu x>x<y>.P \longmapsto Rs \Longrightarrow False$
⟨*proof*⟩

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 $[a\frown b]P \longmapsto Rs$
and $[\![P \longmapsto Rs;\ a = b]\!] \Longrightarrow F\ a\ a$

shows $F\ a\ b$
⟨*proof*⟩

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*: $[a{\neq}b]P \longmapsto Rs$
and *cMatch*: $[\![P \longmapsto Rs;\ a \neq b]\!] \Longrightarrow F\ a\ b$

shows $F\ a\ b$
⟨*proof*⟩

lemma *sumCases*[*consumes 1*, *case-names cSum1 cSum2*]:
fixes $P :: pi$
and $Q :: pi$
and $Rs :: residual$

assumes *Trans*: $P \oplus Q \longmapsto Rs$

and *cSum1*: $P \longmapsto Rs \Longrightarrow Prop$
and *cSum2*: $Q \longmapsto Rs \Longrightarrow Prop$

shows *Prop*
⟨*proof*⟩

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

assumes $b \sharp P$

shows $[a].P = [b].([(a, b)] \cdot P)$
⟨*proof*⟩

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 $P \parallel Q \longmapsto a«x» \prec PQ'$
and $x \sharp P$
and $x \sharp Q$
and $\bigwedge P'. P \longmapsto a«x» \prec P' \Longrightarrow Prop\ (P' \parallel Q)$
and $\bigwedge Q'. Q \longmapsto a«x» \prec Q' \Longrightarrow Prop\ (P \parallel Q')$

shows *Prop* PQ'
⟨*proof*⟩

lemma *parCasesF*[*consumes 1*, *case-names cPar1 cPar2 cComm1 cComm2 cClose1 cClose2*]:
fixes P :: *pi*
and Q :: *pi*
and α :: *freeRes*
and P' :: *pi*
and C :: *'a::fs-name*
and F :: *freeRes* ⇒ *pi* ⇒ *bool*

assumes *Trans*: $P \parallel Q \longmapsto \alpha \prec PQ'$
and *icPar1F*: $\bigwedge P'. [\![P \longmapsto \alpha \prec P']\!] \Longrightarrow F\ \alpha\ (P' \parallel Q)$
and *icPar2F*: $\bigwedge Q'. [\![Q \longmapsto \alpha \prec Q']\!] \Longrightarrow F\ \alpha\ (P \parallel Q')$
and *icComm1*: $\bigwedge P'\ Q'\ a\ b\ x. [\![P \longmapsto a{<}x{>} \prec P'; Q \longmapsto a[b] \prec Q'; x \sharp P; x\sharp Q; x \neq a; x \neq b; x \sharp Q'; x \sharp C; \alpha = \tau]\!] \Longrightarrow F\ (\tau)\ (P'[x{::=}b] \parallel Q')$
and *icComm2*: $\bigwedge P'\ Q'\ a\ b\ x. [\![P \longmapsto a[b] \prec P'; Q \longmapsto a{<}x{>} \prec Q'; x \sharp P;$

$x \sharp Q$; $x \neq a$; $x \neq b$; $x \sharp P'$; $x \sharp C$; $\alpha = \tau$⟧ $\Longrightarrow$ F (τ) $(P' \parallel Q'[x{::=}b])$
and *icClose1*: $\bigwedge P'\ Q'\ a\ x\ y$. ⟦$P \longmapsto a{<}x{>} \prec P'$; $Q \longmapsto a{<}\nu y{>} \prec Q'$; $x \sharp P$; $x \sharp Q$; $x \neq a$; $x \neq y$; $x \sharp Q'$; $y \sharp P$; $y \sharp Q$; $y \neq a$; $y \sharp P'$; $x \sharp C$; $y \sharp C$; $\alpha = \tau$⟧ $\Longrightarrow$

F (τ) $({<}\nu y{>}(P'[x{::=}y] \parallel Q'))$

and *icClose2*: $\bigwedge P'\ Q'\ a\ x\ y$. ⟦$P \longmapsto a{<}\nu y{>} \prec P'$; $Q \longmapsto a{<}x{>} \prec Q'$; $x \sharp P$; $x \sharp Q$; $x \neq a$; $x \neq y$; $x \sharp P'$; $y \sharp P$; $y \sharp Q$; $y \neq a$; $y \sharp Q'$; $x \sharp C$; $y \sharp C$; $\alpha = \tau$⟧ $\Longrightarrow$

F (τ) $({<}\nu y{>}(P' \parallel Q'[x{::=}y]))$

shows $F\ \alpha\ PQ'$
⟨*proof*⟩

lemma *resCasesF*[*consumes 1*, *case-names cRes*]:
fixes x :: *name*
and P :: *pi*
and α :: *freeRes*
and P' :: *pi*
and C :: *'a::fs-name*

assumes ${<}\nu x{>}P \longmapsto \alpha \prec xP'$
and $\bigwedge P'$. ⟦$P \longmapsto \alpha \prec P'$; $x \sharp \alpha$⟧ $\Longrightarrow$ F $({<}\nu x{>}P')$

shows $F\ xP'$
⟨*proof*⟩

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*: ${<}\nu y{>}P \longmapsto a«x» \prec yP'$
and *xineqy*: $x \neq y$
and *xineqy*: $x \sharp P$
and *rcOpen*: $\bigwedge b\ P'$. ⟦$P \longmapsto b[y] \prec P'$; $b \neq y$; $a = BoundOutputS\ b$⟧ $\Longrightarrow$ F $(BoundOutputS\ b)$ $([(x, y)] \cdot P')$
and *rcResB*: $\bigwedge P'$. ⟦$P \longmapsto a«x» \prec P'$; $y \sharp a$⟧ $\Longrightarrow$ $F\ a$ $({<}\nu y{>}P')$

shows $F\ a\ yP'$
⟨*proof*⟩

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*: $!P \longmapsto Rs$
and *cPar1B*: $\bigwedge a\ x\ P'\ C.\ [\![P \longmapsto a«x» \prec P';\ x \sharp P;\ x \sharp C]\!] \Longrightarrow F\ C\ (P \parallel !P)\ (a«x» \prec P' \parallel !P)$
and *cPar1F*: $\bigwedge \alpha\ P'\ C.\ [\![P \longmapsto \alpha \prec P']\!] \Longrightarrow F\ C\ (P \parallel !P)\ (\alpha \prec P' \parallel !P)$
and *cPar2B*: $\bigwedge a\ x\ P'\ C.\ [\![!P \longmapsto a«x» \prec P';\ x \sharp P;\ x \sharp C;\ \bigwedge C.\ F\ C\ (!P)\ (a«x» \prec P')]\!] \Longrightarrow$
$F\ C\ (P \parallel !P)\ (a«x» \prec P \parallel P')$
and *cPar2F*: $\bigwedge \alpha\ P'\ C.\ [\![!P \longmapsto \alpha \prec P';\ \bigwedge C.\ F\ C\ (!P)\ (\alpha \prec P')]\!] \Longrightarrow F\ C\ (P \parallel !P)\ (\alpha \prec P \parallel P')$
and *cComm1*: $\bigwedge a\ x\ P'\ b\ P''\ C.\ [\![P \longmapsto a<x> \prec P';\ !P \longmapsto (OutputR\ a\ b) \prec P'';\ x \sharp C;$
$\bigwedge C.\ F\ C\ (!P)\ ((OutputR\ a\ b) \prec P'')]\!] \Longrightarrow$
$F\ C\ (P \parallel !P)\ (\tau \prec (P'[x::=b]) \parallel P'')$
and *cComm2*: $\bigwedge a\ b\ P'\ x\ P''\ C.\ [\![P \longmapsto (OutputR\ a\ b) \prec P';\ !P \longmapsto a<x> \prec P'';\ x \sharp C;$
$\bigwedge C.\ F\ C\ (!P)\ (a<x> \prec P'')]\!] \Longrightarrow$
$F\ C\ (P \parallel !P)\ (\tau \prec P' \parallel (P''[x::=b]))$
and *cClose1*: $\bigwedge a\ x\ P'\ y\ P''\ C.\ [\![P \longmapsto a<x> \prec P';\ !P \longmapsto a<\nu y> \prec P'';\ y \sharp P;\ x \sharp C;\ y \sharp C;$
$\bigwedge C.\ F\ C\ (!P)\ (a<\nu y> \prec P'')]\!] \Longrightarrow$
$F\ C\ (P \parallel !P)\ (\tau \prec <\nu y>((P'[x::=y]) \parallel P''))$
and *cClose2*: $\bigwedge a\ y\ P'\ x\ P''\ C.\ [\![P \longmapsto a<\nu y> \prec P';\ !P \longmapsto a<x> \prec P'';\ y \sharp P;\ x \sharp C;\ y \sharp C;$
$\bigwedge C.\ F\ C\ (!P)\ (a<x> \prec P'')]\!] \Longrightarrow$
$F\ C\ (P \parallel !P)\ (\tau \prec <\nu y>(P' \parallel (P''[x::=y])))$
and *cBang*: $\bigwedge Rs\ C.\ [\![P \parallel !P \longmapsto Rs;\ \bigwedge C.\ F\ C\ (P \parallel !P)\ Rs]\!] \Longrightarrow F\ C\ (!P)\ Rs$

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

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

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

  shows $(perm \cdot P, perm \cdot Q) \in Rel$
⟨*proof*⟩

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

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

  shows $(P, Q) \in Rel$
⟨*proof*⟩

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

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

  shows *eqvt* (*Rel O Rel'*)
⟨*proof*⟩

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

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

shows *eqvt* (*Rel* ∪ *Rel'*)
⟨*proof*⟩

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

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

assumes *eqvtRel*: *eqvt Rel*

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

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

shows *substClosed Rel* ⊆ *Rel*
⟨*proof*⟩

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

assumes (*P*, *Q*) ∈ *substClosed Rel*

shows (*P*[<*σ*>], *Q*[<*σ*>]) ∈ *substClosed Rel*
⟨*proof*⟩

inductive-set *bangRel* :: (*pi* × *pi*) *set* ⇒ (*pi* × *pi*) *set*
for *Rel* :: (*pi* × *pi*) *set*
where
BRBang: (*P*, *Q*) ∈ *Rel* ⟹ (!*P*, !*Q*) ∈ *bangRel Rel*
| *BRPar*: (*R*, *T*) ∈ *Rel* ⟹ (*P*, *Q*) ∈ (*bangRel Rel*) ⟹ (*R* ∥ *P*, *T* ∥ *Q*) ∈ (*bangRel Rel*)
| *BRRes*: (*P*, *Q*) ∈ *bangRel Rel* ⟹ (<*νa*>*P*, <*νa*>*Q*) ∈ *bangRel Rel*

inductive-cases *BRBangCases'*: (*P*, !*Q*) ∈ *bangRel Rel*
inductive-cases *BRParCases'*: (*P*, *Q* ∥ !*Q*) ∈ *bangRel Rel*
inductive-cases *BRResCases'*: (*P*, <*νx*>*Q*) ∈ *bangRel Rel*

lemma *eqvtBangRel*:
fixes *Rel* :: (*pi* × *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 $(P, !Q) \in$ *bangRel Rel*
and $\bigwedge P.\ (P, Q) \in Rel \Longrightarrow F\ (!P)$

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

shows *F P*
⟨*proof*⟩

lemma *bangRelSubset*:
fixes *Rel* :: (*pi* × *pi*) *set*
and *Rel′* :: (*pi* × *pi*) *set*

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

shows $(P, Q) \in$ *bangRel Rel′*
⟨*proof*⟩

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

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

shows $(Q, P) \in$ *bangRel Rel*
⟨*proof*⟩

primrec *resChain* :: *name list* ⇒ *pi* ⇒ *pi* **where**

base: *resChain* [] *P* = *P*
| *step*: *resChain* (*x*#*xs*) *P* = <ν*x*>(*resChain xs P*)

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

shows *perm* • (*resChain lst P*) = *resChain* (*perm* • *lst*) (*perm* • *P*)
⟨*proof*⟩

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

assumes *a* ♯ (*lst*, *P*)

shows *a* ♯ (*resChain lst P*)
⟨*proof*⟩

end

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

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

definition *simulation* :: *pi* ⇒ (*pi* × *pi*) *set* ⇒ *pi* ⇒ *bool* (- ⇝[-] - [*80*, *80*, *80*] *80*) **where**
P ⇝[*Rel*] *Q* ≡ (∀ *a x Q'*. *Q* ⟼*a*«*x*» ≺ *Q'* ∧ *x* ♯ *P* ⟶ (∃ *P'*. *P* ⟼*a*«*x*» ≺ *P'* ∧ *derivative P' Q' a x Rel*)) ∧
(∀ α *Q'*. *Q* ⟼α ≺ *Q'* ⟶ (∃ *P'*. *P* ⟼α ≺ *P'* ∧ (*P'*, *Q'*) ∈ *Rel*))

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

assumes *P* ⇝[*A*] *P'*
and *A* ⊆ *B*

shows *P* ⇝[*B*] *P'*
⟨*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* $(perm \cdot P)$ $(perm \cdot Q)$ $(perm \cdot a)$ $(perm \cdot x)$ *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* $(perm \cdot P)$ $(perm \cdot Q)$ *a* $(perm \cdot x)$ *Rel*
⟨*proof*⟩

lemma *freshUnit*[*simp*]:
fixes $y :: name$

shows $y \sharp ()$
⟨*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*: $\bigwedge a\ x\ Q'.\ [\![Q \longmapsto a«x» \prec Q';\ x \sharp P;\ x \sharp Q;\ x \sharp a;\ x \sharp C]\!] \Longrightarrow \exists P'.\ P \longmapsto a«x» \prec P' \wedge \textit{derivative}\ P'\ Q'\ a\ x\ \textit{Rel}$
and *Free*: $\bigwedge \alpha\ Q'.\ Q \longmapsto \alpha \prec Q' \Longrightarrow \exists P'.\ P \longmapsto \alpha \prec P' \wedge (P', Q') \in \textit{Rel}$

shows $P \leadsto[\textit{Rel}]\ Q$
⟨*proof*⟩

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

assumes *Bound*: $\bigwedge a\ y\ Q'.\ [\![Q \longmapsto a«y» \prec Q';\ y \sharp P]\!] \Longrightarrow \exists P'.\ P \longmapsto a«y» \prec P' \wedge \textit{derivative}\ P'\ Q'\ a\ y\ \textit{Rel}$
and *Free*: $\bigwedge \alpha\ Q'.\ Q \longmapsto \alpha \prec Q' \Longrightarrow \exists P'.\ P \longmapsto \alpha \prec P' \wedge (P', Q') \in \textit{Rel}$

shows $P \leadsto[\textit{Rel}]\ Q$
⟨*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*: $\bigwedge Q'\ a.\ [\![Q \longmapsto a[x] \prec Q';\ a \neq x]\!] \Longrightarrow \exists P'.\ P \longmapsto a<\nu x> \prec P' \wedge (P', Q') \in \textit{Rel}$
and *BR*: $\bigwedge Q'\ a\ y.\ [\![Q \longmapsto a«y» \prec Q';\ x \sharp a;\ x \neq y;\ y \sharp C]\!] \Longrightarrow \exists P'.\ P \longmapsto a«y» \prec P' \wedge \textit{derivative}\ P'\ (<\nu x>Q')\ a\ y\ \textit{Rel}$
and *BF*: $\bigwedge Q'\ \alpha.\ [\![Q \longmapsto \alpha \prec Q';\ x \sharp \alpha]\!] \Longrightarrow \exists P'.\ P \longmapsto \alpha \prec P' \wedge (P', <\nu x>Q') \in \textit{Rel}$

shows $P \leadsto[\textit{Rel}]\ <\nu x>Q$
⟨*proof*⟩

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

and Q' :: *pi*

assumes $P \rightsquigarrow[Rel]\ Q$

shows $Q \longmapsto a«x» \prec Q' \Longrightarrow x \sharp P \Longrightarrow \exists P'.\ P \longmapsto a«x» \prec P' \wedge (derivative\ P'$ $Q'\ a\ x\ Rel)$
and $Q \longmapsto \alpha \prec Q' \Longrightarrow \exists P'.\ P \longmapsto \alpha \prec P' \wedge (P', Q') \in Rel$
⟨*proof*⟩

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

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

shows $(perm \cdot P) \rightsquigarrow[Rel']\ (perm \cdot Q)$
⟨*proof*⟩

lemma *derivativeReflexive*:
fixes P :: *pi*
and a :: *subject*
and x :: *name*
and *Rel* :: (*pi* × *pi*) *set*

assumes $Id \subseteq Rel$

shows *derivative P P a x Rel*
⟨*proof*⟩

lemma *reflexive*:
fixes P :: *pi*
and *Rel* :: (*pi* × *pi*) *set*

assumes $Id \subseteq Rel$

shows $P \rightsquigarrow[Rel]\ P$
⟨*proof*⟩

lemma *transitive*:
fixes P :: *pi*
and Q :: *pi*
and R :: *pi*
and *Rel* :: (*pi* × *pi*) *set*

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

assumes *PSimQ*: $P \leadsto[Rel]\ Q$
and *QSimR*: $Q \leadsto[Rel']\ R$
and *Eqvt'*: *eqvt* Rel''
and *Trans*: $Rel\ O\ Rel' \subseteq Rel''$

shows $P \leadsto[Rel'']\ R$
⟨*proof*⟩

end

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

lemma *monoAux*: $A \subseteq B \Longrightarrow P \leadsto[A]\ Q \longrightarrow P \leadsto[B]\ Q$
⟨*proof*⟩

coinductive-set *bisim* :: $(pi \times pi)$ *set*
where
step: $[\![P \leadsto[bisim]\ Q;\ (Q,\ P) \in bisim]\!] \Longrightarrow (P,\ Q) \in bisim$
monos *monoAux*

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

lemma *monotonic'*: $mono(\lambda S.\ \{(P,\ Q)\ |P\ Q.\ P \leadsto[S]\ Q \land Q \leadsto[S]\ P\})$
⟨*proof*⟩

lemma *monotonic*: $mono(\lambda p\ x1\ x2.$
$\exists P\ Q.\ x1 = P \land$
$x2 = Q \land P \leadsto[\{(xa,\ x).\ p\ xa\ x\}]\ Q \land Q \leadsto[\{(xa,\ x).\ p\ xa\ x\}]\ P)$
⟨*proof*⟩

lemma *bisimCoinduct*[*case-names cSim cSym* , *consumes 1*]:
assumes *p*: $(P,\ Q) \in X$
and *rSim*: $\bigwedge R\ S.\ (R,\ S) \in X \Longrightarrow R \leadsto[(X \cup bisim)]\ S$
and *rSym*: $\bigwedge R\ S.\ (R,\ S) \in X \Longrightarrow (S,\ R) \in X$

shows $P \sim Q$
⟨*proof*⟩

lemma *bisimE*:
fixes P :: *pi*
and Q :: *pi*

assumes $P \sim Q$

shows $P \leadsto[bisim]\ Q$
⟨*proof*⟩

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

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

shows $P \sim Q$
⟨*proof*⟩

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

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

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

shows $P \sim P$
⟨*proof*⟩

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

assumes $P \sim Q$

shows $Q \sim P$
⟨*proof*⟩

lemma *bisimClosed*:
fixes $P :: pi$
and $Q :: pi$
and $p :: name\ prm$

assumes $P \sim Q$

shows $(p \cdot P) \sim (p \cdot Q)$
⟨*proof*⟩

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*: $\bigwedge R\ S.\ (R, S) \in X \Longrightarrow R \leadsto[(bisim\ O\ (X \cup bisim)\ O\ bisim)]\ S$
  and *rSym*: $\bigwedge R\ S.\ (R, S) \in X \Longrightarrow (S, R) \in bisim\ O\ (X \cup bisim)\ O\ bisim$

  shows $P \sim Q$
⟨*proof*⟩

end

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

abbreviation
  StrongEqJudge (**infixr** $\sim^s$ *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 $(perm \cdot P) \sim^s (perm \cdot Q)$
⟨*proof*⟩

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

shows $P \sim^s P$
⟨*proof*⟩

lemma *symmetric*:
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*⟩

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 $\tau.(P) \leadsto[Rel]\ \tau.(Q)$
⟨*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* × *pi*) *set*
and *Rel'* :: (*pi* × *pi*) *set*

assumes *PSimQ*: *P* ⇝[*Rel*] *Q*
and *Rel* ⊆ *Rel'*

shows [*a*⌢*b*]*P* ⇝[*Rel'*] [*a*⌢*b*]*Q*
⟨*proof*⟩

lemma *mismatchPres*:
fixes *P* :: *pi*
and *Q* :: *pi*
and *a* :: *name*
and *b* :: *name*
and *Rel* :: (*pi* × *pi*) *set*
and *Rel'* :: (*pi* × *pi*) *set*

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

shows $[a{\neq}b]P \leadsto[Rel']\ [a{\neq}b]Q$
⟨*proof*⟩

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

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

shows $P \oplus R \leadsto[Rel']\ Q \oplus R$
⟨*proof*⟩

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*: $P \leadsto[Rel]\ Q$
and *RSimT*: $R \leadsto[Rel']\ T$
and *PRelQ*: $(P,\ Q) \in Rel$
and *RRel'T*: $(R,\ T) \in Rel'$
and *Par*: $\bigwedge P\ Q\ R\ T.\ [\![(P,\ Q) \in Rel;\ (R,\ T) \in Rel']\!] \Longrightarrow (P \parallel R,\ Q \parallel T)$
$\in Rel''$
and *Res*: $\bigwedge P\ Q\ a.\ (P,\ Q) \in Rel'' \Longrightarrow (<\nu a>P,\ <\nu a>Q) \in Rel''$
and *EqvtRel*: *eqvt Rel*
and *EqvtRel'*: *eqvt Rel'*
and *EqvtRel''*: *eqvt Rel''*

shows $P \parallel R \leadsto[Rel'']\ Q \parallel T$
⟨*proof*⟩

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*: $P \leadsto[Rel]\ Q$
and *PRelQ*: $(P,\ Q) \in Rel$
and *Par*: $\bigwedge P\ Q\ R.\ (P,\ Q) \in Rel \Longrightarrow (P \parallel R,\ Q \parallel R) \in Rel'$
and *Res*: $\bigwedge P\ Q\ a.\ (P,\ Q) \in Rel' \Longrightarrow (<\nu a>P,\ <\nu a>Q) \in Rel'$
and *EqvtRel*: *eqvt Rel*
and *EqvtRel'*: *eqvt Rel'*

shows $P \parallel R \leadsto[Rel']\ Q \parallel R$
⟨*proof*⟩

lemma *resDerivative*:
fixes P :: *pi*
and Q :: *pi*
and a :: *subject*
and x :: *name*
and y :: *name*
and *Rel* :: (*pi* × *pi*) *set*
and *Rel'* :: (*pi* × *pi*) *set*

assumes *Der*: *derivative P Q a x Rel*
and *Rel*: $\bigwedge(P::pi)\ (Q::pi)\ (x::name).\ (P,\ Q) \in Rel \Longrightarrow (<\nu x>P,\ <\nu x>Q) \in Rel'$
and *Eqv*: *eqvt Rel*

shows *derivative* $(<\nu y>P)\ (<\nu y>Q)$ *a x Rel'*
⟨*proof*⟩

lemma *resPres*:
fixes P :: *pi*
and Q :: *pi*
and *Rel* :: (*pi* × *pi*) *set*
and x :: *name*
and *Rel'* :: (*pi* × *pi*) *set*

assumes *PSimQ*: $P \leadsto[Rel]\ Q$
and *ResRel*: $\bigwedge(P::pi)\ (Q::pi)\ (x::name).\ (P,\ Q) \in Rel \Longrightarrow (<\nu x>P,\ <\nu x>Q) \in Rel'$
and *RelRel'*: $Rel \subseteq Rel'$
and *EqvtRel*: *eqvt Rel*
and *EqvtRel'*: *eqvt Rel'*

shows $<\nu x>P \leadsto[Rel']\ <\nu x>Q$
⟨*proof*⟩

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

and *xs* :: *name list*

assumes *PRelQ*: $P \rightsquigarrow[Rel]\ Q$
and *eqvtRel*: *eqvt Rel*
and *Res*: $\bigwedge P\ Q\ x.\ (P,\ Q) \in Rel \Longrightarrow (<\nu x>P,\ <\nu x>Q) \in Rel$

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*: $\bigwedge P\ Q.\ (P,\ Q) \in Rel \Longrightarrow P \rightsquigarrow[Rel]\ Q$
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 $a<x>.P \sim a<x>.Q$
⟨*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 $[a\frown b]P \sim [a\frown b]Q$
⟨*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 $<\nu x>P \sim <\nu x>Q$

⟨*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 $a<x>.P \sim^s a<x>.Q$
⟨*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 *PeqQ*: $P \sim^s Q$

shows $<\nu x>P \sim^s <\nu x>Q$
⟨*proof*⟩

lemma *bangPres*:
fixes P :: *pi*
and Q :: *pi*

assumes $P \sim^s Q$

shows $!P \sim^s !Q$
⟨*proof*⟩

end

theory *Late-Tau-Chain*
imports *Late-Semantics1*
begin

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

lemma *singleTauChain*:
fixes P :: *pi*
and P' :: *pi*

assumes $P \longmapsto \tau \prec P'$

shows $P \Longrightarrow_\tau P'$
⟨*proof*⟩

lemma *tauChainAddTau*[*dest*]:
fixes P :: *pi*
and P' :: *pi*
and P'' :: *pi*

shows $P \Longrightarrow_\tau P' \Longrightarrow P' \longmapsto \tau \prec P'' \Longrightarrow P \Longrightarrow_\tau P''$
and $P \longmapsto \tau \prec P' \Longrightarrow P' \Longrightarrow_\tau P'' \Longrightarrow P \Longrightarrow_\tau P''$
⟨*proof*⟩

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

and $P' :: pi$

assumes $P \Longrightarrow_\tau P'$
and $F\ P$
and $\bigwedge P'\ P''.\ [\![P \Longrightarrow_\tau P';\ P' \longmapsto \tau \prec P'';\ F\ P']\!] \Longrightarrow F\ P''$

shows $F\ P'$
⟨*proof*⟩

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

assumes $P \Longrightarrow_\tau P'$

shows $(perm \cdot P) \Longrightarrow_\tau (perm \cdot P')$
⟨*proof*⟩

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

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

shows $P \Longrightarrow_\tau P'$
⟨*proof*⟩

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

shows $P \Longrightarrow_\tau P' = (perm \cdot P) \Longrightarrow_\tau (perm \cdot P')$
⟨*proof*⟩

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

assumes $P \Longrightarrow_\tau P'$
and $x \sharp P$

shows $x \sharp P'$
⟨*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 *aineqb*: $a \neq b$
and *PineqP'*: $P \neq P'$

shows $[a \neq b]P \Longrightarrow_\tau P'$
⟨*proof*⟩

lemma *sum1Chain*[*rule-format*]:
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*[*rule-format*]:
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 $<\nu a>P \Longrightarrow_\tau <\nu a>P'$
⟨*proof*⟩

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* ⇒ *name* ⇒ *pi* ⇒ *name* ⇒ *name* ⇒ *pi* ⇒ *bool* (- $\Longrightarrow_l$- *in* -→-<-> ≺ - [*80*, *80*, *80*, *80*, *80*] *80*)
where $P \Longrightarrow_l u$ *in* $P'' \rightarrow a<x> \prec P' \equiv \exists P'''.\ P \Longrightarrow_\tau P''' \wedge P''' \longmapsto a<x> \prec P'' \wedge P''[x::=u] \Longrightarrow_\tau P'$

definition *transition* :: (*pi* × *Late-Semantics.residual*) *set* **where**
  transition $\equiv \{x.\ \exists P\ P'\ \alpha\ P''\ P'''.\ P \Longrightarrow_\tau P' \wedge P' \longmapsto \alpha \prec P'' \wedge P'' \Longrightarrow_\tau P''' \wedge x = (P, \alpha \prec P''')\} \cup$
  $\{x.\ \exists P\ P'\ a\ y\ P''\ P'''.\ P \Longrightarrow_\tau P' \wedge (P' \longmapsto (a<\nu y> \prec P'')) \wedge P'' \Longrightarrow_\tau P''' \wedge x = (P, (a<\nu y> \prec P'''))\}$

abbreviation *weakTransition-judge* :: *pi* ⇒ *Late-Semantics.residual* ⇒ *bool* (- $\Longrightarrow_l$ - [*80*, *80*] *80*)
  where $P \Longrightarrow_l Rs \equiv (P, Rs) \in$ *transition*

lemma *weakNonInput*[*dest*]:
  fixes P :: *pi*
  and a :: *name*
  and x :: *name*
  and P' :: *pi*

  assumes $P \Longrightarrow_l a<x> \prec P'$

  shows *False*
⟨*proof*⟩

lemma *transitionI*:
  fixes P :: *pi*
  and P''' :: *pi*
  and α :: *freeRes*
  and P'' :: *pi*

and P' :: *pi*
and a :: *name*
and x :: *name*
and u :: *name*

shows $[\![P \Longrightarrow_\tau P'''; P''' \longmapsto \alpha \prec P''; P'' \Longrightarrow_\tau P']\!] \Longrightarrow P \Longrightarrow_l \alpha \prec P'$
and $[\![P \Longrightarrow_\tau P'''; P''' \longmapsto a{<}\nu x{>} \prec P''; P'' \Longrightarrow_\tau P']\!] \Longrightarrow P \Longrightarrow_l a{<}\nu x{>} \prec P'$
and $[\![P \Longrightarrow_\tau P'''; P''' \longmapsto a{<}x{>} \prec P''; P''[x{::=}u] \Longrightarrow_\tau P']\!] \Longrightarrow P \Longrightarrow_l u \textit{ in } P''{\rightarrow}a{<}x{>} \prec P'$
⟨*proof*⟩

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

shows $P \Longrightarrow_l \alpha \prec P' \Longrightarrow \exists P''\, P'''.\ P \Longrightarrow_\tau P'' \wedge P'' \longmapsto \alpha \prec P''' \wedge P''' \Longrightarrow_\tau P'$ (**is** - $\Longrightarrow$ *?thesis1*)
and $[\![P \Longrightarrow_l a{<}\nu x{>} \prec P'; x \sharp P]\!] \Longrightarrow \exists P''\, P'''.\ P \Longrightarrow_\tau P''' \wedge P''' \longmapsto a{<}\nu x{>} \prec P'' \wedge P'' \Longrightarrow_\tau P'$
and $[\![P \Longrightarrow_l u \textit{ in } P''{\rightarrow}a{<}x{>} \prec P']\!] \Longrightarrow \exists P'''.\ P \Longrightarrow_\tau P''' \wedge P''' \longmapsto a{<}x{>} \prec P'' \wedge P''[x{::=}u] \Longrightarrow_\tau P'$
⟨*proof*⟩

lemma *alphaInput*:
fixes P :: *pi*
and u :: *name*
and P'' :: *pi*
and a :: *name*
and x :: *name*
and P' :: *pi*
and y :: *name*

assumes *PTrans*: $P \Longrightarrow_l u \textit{ in } P''{\rightarrow}a{<}x{>} \prec P'$
and *yFreshP*: $y \sharp P$

shows $P \Longrightarrow_l u \textit{ in } ([(x, y)] \cdot P''){\rightarrow}a{<}y{>} \prec P'$
⟨*proof*⟩

lemma *tauActionChain*:
fixes P :: *pi*
and P' :: *pi*

shows $P \Longrightarrow_l \tau \prec P' \Longrightarrow P \Longrightarrow_\tau P'$

and $P \neq P' \Longrightarrow P \Longrightarrow_\tau P' \Longrightarrow P \Longrightarrow_l \tau \prec P'$
⟨*proof*⟩

lemma *singleActionChain*:
fixes P :: *pi*
and a :: *name*
and x :: *name*
and α :: *freeRes*
and u :: *name*

shows $P \longmapsto a{<}\nu x{>} \prec P' \Longrightarrow P \Longrightarrow_l a{<}\nu x{>} \prec P'$
and $[\![P \longmapsto a{<}x{>} \prec P']\!] \Longrightarrow P \Longrightarrow_l u\ in\ P' \rightarrow a{<}x{>} \prec P'[x{::=}u]$
and $P \longmapsto \alpha \prec P' \Longrightarrow P \Longrightarrow_l \alpha \prec P'$
⟨*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 $a{<}x{>}.P \Longrightarrow_l u\ in\ P \rightarrow a{<}x{>} \prec P[x{::=}u]$
⟨*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 $P \Longrightarrow_l Rs \Longrightarrow [a \frown a]P \Longrightarrow_l Rs$
and $P \Longrightarrow_l u\ in\ P'' \rightarrow b{<}x{>} \prec P' \Longrightarrow [a \frown a]P \Longrightarrow_l u\ in\ P'' \rightarrow b{<}x{>} \prec P'$

⟨proof⟩

lemma *Mismatch*:
  fixes P :: *pi*
  and Rs :: *residual*
  and a :: *name*
  and c :: *name*
  and u :: *name*
  and b :: *name*
  and x :: *name*
  and P' :: *pi*

  shows $[\![P \Longrightarrow_l Rs;\ a \neq c]\!] \Longrightarrow [a{\neq}c]P \Longrightarrow_l Rs$
  and $[\![P \Longrightarrow_l u\ in\ P''{\rightarrow}b{<}x{>} \prec P';\ a \neq c]\!] \Longrightarrow [a{\neq}c]P \Longrightarrow_l u\ in\ P''{\rightarrow}b{<}x{>} \prec$
P'
⟨proof⟩

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

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

  shows ${<}\nu b{>}P \Longrightarrow_l a{<}\nu b{>} \prec P'$
⟨proof⟩

lemma *Sum1*:
  fixes P :: *pi*
  and Rs :: *residual*
  and Q :: *pi*
  and u :: *name*
  and P'' :: *pi*
  and a :: *name*
  and x :: *name*
  and P' :: *pi*

  shows $P \Longrightarrow_l Rs \Longrightarrow P \oplus Q \Longrightarrow_l Rs$
  and $P \Longrightarrow_l u\ in\ P''{\rightarrow}a{<}x{>} \prec P' \Longrightarrow P \oplus Q \Longrightarrow_l u\ in\ P''{\rightarrow}a{<}x{>} \prec P'$
⟨proof⟩

lemma *Sum2*:
  fixes Q :: *pi*
  and Rs :: *residual*
  and P :: *pi*
  and u :: *name*
  and Q'' :: *pi*

and a :: *name*
and x :: *name*
and Q' :: *pi*

shows $Q \Longrightarrow_l Rs \Longrightarrow P \oplus Q \Longrightarrow_l Rs$
and $Q \Longrightarrow_l u\ in\ Q''{\rightarrow}a{<}x{>} \prec Q' \Longrightarrow P \oplus Q \Longrightarrow_l u\ in\ Q''{\rightarrow}a{<}x{>} \prec Q'$
⟨*proof*⟩

lemma *Par1B*:
fixes P :: *pi*
and a :: *name*
and x :: *name*
and P' :: *pi*
and u :: *name*
and P'' :: *pi*

shows $[\![P \Longrightarrow_l a{<}\nu x{>} \prec P'; x \sharp Q]\!] \Longrightarrow P \parallel Q \Longrightarrow_l a{<}\nu x{>} \prec (P' \parallel Q)$
and $[\![P \Longrightarrow_l u\ in\ P''{\rightarrow}a{<}x{>} \prec P'; x \sharp Q]\!] \Longrightarrow P \parallel Q \Longrightarrow_l u\ in\ (P'' \parallel Q){\rightarrow}a{<}x{>} \prec P' \parallel Q$
⟨*proof*⟩

lemma *Par1F*:
fixes P :: *pi*
and α :: *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 $Q \Longrightarrow_l a{<}\nu x{>} \prec Q' \Longrightarrow x \sharp P \Longrightarrow P \parallel Q \Longrightarrow_l a{<}\nu x{>} \prec (P \parallel Q')$
and $Q \Longrightarrow_l u\ in\ Q''{\rightarrow}a{<}x{>} \prec Q' \Longrightarrow x \sharp P \Longrightarrow P \parallel Q \Longrightarrow_l u\ in\ (P \parallel Q''){\rightarrow}a{<}x{>} \prec P \parallel Q'$
⟨*proof*⟩

lemma *Par2F*:
fixes Q :: *pi*
and α :: *freeRes*
and Q' :: *pi*

assumes *QTrans*: $Q \Longrightarrow_l \alpha \prec Q'$

shows $P \parallel Q \Longrightarrow_l \alpha \prec (P \parallel Q')$
⟨*proof*⟩

lemma *Comm1*:
fixes P :: *pi*
and b :: *name*
and P'' :: *pi*
and a :: *name*
and x :: *name*
and P' :: *pi*
and Q :: *pi*
and Q' :: *pi*

assumes *PTrans*: $P \Longrightarrow_l b\ in\ P'' \rightarrow a{<}x{>} \prec P'$
and *QTrans*: $Q \Longrightarrow_l a[b] \prec Q'$

shows $P \parallel Q \Longrightarrow_l \tau \prec P' \parallel Q'$
⟨*proof*⟩

lemma *Comm2*:
fixes P :: *pi*
and a :: *name*
and b :: *name*
and P' :: *pi*
and Q :: *pi*
and x :: *name*
and Q'' :: *pi*
and Q' :: *pi*

assumes *PTrans*: $P \Longrightarrow_l a[b] \prec P'$
and *QTrans*: $Q \Longrightarrow_l b\ in\ Q'' \rightarrow a{<}x{>} \prec Q'$

shows $P \parallel Q \Longrightarrow_l \tau \prec P' \parallel Q'$
⟨*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*: $P \Longrightarrow_l y\ in\ P'' \rightarrow a{<}x{>} \prec P'$

and *QTrans*: $Q \Longrightarrow_l a{<}\nu y{>} \prec Q'$
and *yFreshP*: $y \sharp P$
and *yFreshQ*: $y \sharp Q$

shows $P \parallel Q \Longrightarrow_l \tau \prec {<}\nu y{>}(P' \parallel Q')$
⟨*proof*⟩

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*: $P \Longrightarrow_l a{<}\nu y{>} \prec P'$
and *QTrans*: $Q \Longrightarrow_l y \text{ in } Q'' \rightarrow a{<}x{>} \prec Q'$
and *yFreshP*: $y \sharp P$
and *yFreshQ*: $y \sharp Q$

shows $P \parallel Q \Longrightarrow_l \tau \prec {<}\nu y{>}(P' \parallel Q')$
⟨*proof*⟩

lemma *ResF*:
fixes P :: *pi*
and α :: *freeRes*
and P' :: *pi*
and x :: *name*

assumes *PTrans*: $P \Longrightarrow_l \alpha \prec P'$
and *xFreshAlpha*: $x \sharp \alpha$

shows ${<}\nu x{>}P \Longrightarrow_l \alpha \prec {<}\nu x{>}P'$
⟨*proof*⟩

lemma *ResB*:
fixes P :: *pi*
and a :: *name*
and x :: *name*
and P' :: *pi*
and y :: *name*
and u :: *name*
and P'' :: *pi*

shows $[\![P \Longrightarrow_l a{<}\nu x{>} \prec P'; y \neq a; y \neq x; x \sharp P]\!] \Longrightarrow {<}\nu y{>}P \Longrightarrow_l a{<}\nu x{>} \prec ({<}\nu y{>}P')$
and $[\![P \Longrightarrow_l u \text{ in } P'' \rightarrow a{<}x{>} \prec P'; y \neq a; y \neq x; y \neq u]\!] \Longrightarrow {<}\nu y{>}P \Longrightarrow_l u \text{ in}$

$(<\nu y>P'')\rightarrow a<x> \prec (<\nu y>P')$
⟨*proof*⟩

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 $P \parallel !P \Longrightarrow_l u \ in \ P''\rightarrow a<x> \prec P' \Longrightarrow !P \Longrightarrow_l u \ in \ P''\rightarrow a<x> \prec P'$
⟨*proof*⟩

lemma *tauTransitionChain*:
fixes P :: *pi*
and P' :: *pi*

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

shows $P \Longrightarrow_\tau P'$
⟨*proof*⟩

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 α :: *freeRes*

shows $P \Longrightarrow_\tau P' \Longrightarrow P' \Longrightarrow_l Rs \Longrightarrow P \Longrightarrow_l Rs$
and $P \Longrightarrow_\tau P'' \Longrightarrow P'' \Longrightarrow_l u \ in \ P'''\rightarrow a<x> \prec P' \Longrightarrow P \Longrightarrow_l u \ in \ P'''\rightarrow a<x> \prec P'$
and $P \Longrightarrow_l a<\nu x> \prec P'' \Longrightarrow P'' \Longrightarrow_\tau P' \Longrightarrow x \sharp P \Longrightarrow P \Longrightarrow_l a<\nu x> \prec P'$
and $P \Longrightarrow_l u \ in \ P'''\rightarrow a<x> \prec P'' \Longrightarrow P'' \Longrightarrow_\tau P' \Longrightarrow P \Longrightarrow_l u \ in \ P'''\rightarrow a<x> \prec P'$
and $P \Longrightarrow_l \alpha \prec P'' \Longrightarrow P'' \Longrightarrow_\tau P' \Longrightarrow P \Longrightarrow_l \alpha \prec P'$
⟨*proof*⟩

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*: $P \Longrightarrow_l u\ in\ P'' \rightarrow a{<}x{>} \prec P'$
and *cFreshP*: $c \sharp P$
and *cinequ*: $c \neq u$

shows $c \sharp P'$
⟨*proof*⟩

lemma *freshBoundOutputTransition*:
fixes P :: *pi*
and a :: *name*
and x :: *name*
and P' :: *pi*
and c :: *name*

assumes *PTrans*: $P \Longrightarrow_l a{<}\nu x{>} \prec P'$
and *cFreshP*: $c \sharp P$
and *cineqx*: $c \neq x$

shows $c \sharp P'$
⟨*proof*⟩

lemma *freshTauTransition*:
fixes P :: *pi*
and c :: *name*

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

shows $c \sharp P'$
⟨*proof*⟩

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*: $c \sharp P$

shows $c \sharp P'$
⟨*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 $P \Longrightarrow_l u\ in\ P''{\rightarrow}a{<}x{>} \prec P' \Longrightarrow (perm \cdot P) \Longrightarrow_l (perm \cdot u)\ in\ (perm \cdot P''){\rightarrow}(perm \cdot a){<}(perm \cdot x){>} \prec (perm \cdot P')$
⟨*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* :: $pi \Rightarrow residual \Rightarrow bool$ (- $\Longrightarrow_l\hat{}$ - [*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 $P \Longrightarrow_l\hat{}\ Rs$
and $P \Longrightarrow_l Rs \Longrightarrow F\ Rs$

and $Rs = \tau \prec P \Longrightarrow F\ (\tau \prec P)$

shows $F\ Rs$
⟨*proof*⟩

lemma *singleActionChain*:
fixes P :: *pi*
and α :: *freeRes*
and P' :: *pi*

assumes $P \longmapsto \alpha \prec P'$

shows $P \Longrightarrow_l\hat{}(\alpha \prec P')$
⟨*proof*⟩

lemma *Tau*:
fixes P :: *pi*

shows $\tau.(P) \Longrightarrow_l\hat{}\ \tau \prec\ P$
⟨*proof*⟩

lemma *Output*:
fixes a :: *name*
and b :: *name*
and P :: *pi*

shows $a\{b\}.P \Longrightarrow_l\hat{}\ a[b] \prec P$
⟨*proof*⟩

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

shows $P \Longrightarrow_l\hat{}\ b{<}\nu x{>} \prec P' \Longrightarrow [a\frown a]P \Longrightarrow_l\hat{}\ b{<}\nu x{>} \prec P'$
and $P \Longrightarrow_l\hat{}\ \alpha \prec P' \Longrightarrow P \neq P' \Longrightarrow [a\frown a]P \Longrightarrow_l\hat{}\ \alpha \prec P'$
⟨*proof*⟩

lemma *Mismatch*:
fixes a :: *name*
and c :: *name*
and P :: *pi*
and b :: *name*
and x :: *name*
and P' :: *pi*
and α :: *freeRes*

shows $[\![P \Longrightarrow_l\hat{}\ b{<}\nu x{>} \prec P'; a \neq c]\!] \Longrightarrow [a{\neq}c]P \Longrightarrow_l\hat{}\ b{<}\nu x{>} \prec P'$
and $P \Longrightarrow_l\hat{}\ \alpha \prec P' \Longrightarrow P \neq P' \Longrightarrow a \neq c \Longrightarrow [a{\neq}c]P \Longrightarrow_l\hat{}\ \alpha \prec P'$
⟨*proof*⟩

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

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

shows $<\nu b>P \Longrightarrow_l\hat{}\ a{<}\nu b{>} \prec P'$
⟨*proof*⟩

lemma *Par1B*:
fixes P :: *pi*
and a :: *name*
and x :: *name*
and P' :: *pi*

assumes *PTrans*: $P \Longrightarrow_l\hat{}\ a{<}\nu x{>} \prec P'$
and *xFreshQ*: $x \sharp Q$

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

lemma *Par1F*:
fixes P :: *pi*
and α :: *freeRes*
and P' :: *pi*

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

shows $P \parallel Q \Longrightarrow_l\hat{}\ \alpha \prec (P' \parallel Q)$
⟨*proof*⟩

lemma *Par2B*:
fixes Q :: *pi*
and a :: *name*
and x :: *name*
and Q' :: *pi*

assumes *QTrans*: $Q \Longrightarrow_l\hat{}\ a{<}\nu x{>} \prec Q'$
and *xFreshP*: $x \sharp P$

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

⟨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*: $P \Longrightarrow_l b\ in\ P''\rightarrow a<x> \prec P'$
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*: $Q \Longrightarrow_l b\ in\ Q''\rightarrow a<x> \prec Q'$

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*: $P \Longrightarrow_l y$ *in* $P''\rightarrow a{<}x{>} \prec P'$
and *QTrans*: $Q \Longrightarrow_l\hat{}\ a{<}\nu y{>} \prec Q'$
and *xFreshP*: $y \sharp P$
and *xFreshQ*: $y \sharp Q$

shows $P \parallel Q \Longrightarrow_l\hat{}\ \tau \prec {<}\nu y{>}(P' \parallel Q')$
⟨*proof*⟩

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*: $P \Longrightarrow_l\hat{}\ a{<}\nu y{>} \prec P'$
and *QTrans*: $Q \Longrightarrow_l y$ *in* $Q''\rightarrow a{<}x{>} \prec Q'$
and *xFreshP*: $y \sharp P$
and *xFreshQ*: $y \sharp Q$

shows $P \parallel Q \Longrightarrow_l\hat{}\ \tau \prec {<}\nu y{>}(P' \parallel Q')$
⟨*proof*⟩

lemma *ResF*:
fixes P :: *pi*
and α :: *freeRes*
and P' :: *pi*
and x :: *name*

assumes *PTrans*: $P \Longrightarrow_l\hat{}\ \alpha \prec P'$
and *xFreshAlpha*: $x \sharp \alpha$

shows ${<}\nu x{>}P \Longrightarrow_l\hat{}\ \alpha \prec {<}\nu x{>}P'$
⟨*proof*⟩

lemma *ResB*:
fixes P :: *pi*
and a :: *name*
and x :: *name*
and P' :: *pi*
and y :: *name*

assumes *PTrans*: $P \Longrightarrow_l\hat{}\ a{<}\nu x{>} \prec P'$

and *yineqa*: $y \neq a$
and *yineqx*: $y \neq x$
and *xFreshP*: $x \sharp P$

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

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 α :: *freeRes*

shows $P \Longrightarrow_\tau P' \Longrightarrow P' \Longrightarrow_l\hat{}\ Rs \Longrightarrow P \Longrightarrow_l\hat{}\ Rs$
and $P \Longrightarrow_l\hat{}\, a<\nu x> \prec P'' \Longrightarrow P'' \Longrightarrow_\tau P' \Longrightarrow x \sharp P \Longrightarrow P \Longrightarrow_l\hat{}\, a<\nu x> \prec$
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 α :: *freeRes*
and P'' :: *pi*

assumes *PTrans*: $P \Longrightarrow_l\tau \prec P'$
and *P'Trans*: $P' \Longrightarrow_l\hat{}\,\alpha \prec P''$

shows $P \Longrightarrow_l \alpha \prec P''$
⟨*proof*⟩

lemma *freshBoundOutputTransition*:
fixes P :: *pi*
and a :: *name*
and x :: *name*
and P' :: *pi*
and c :: *name*

assumes *PTrans*: $P \Longrightarrow_l\hat{}\ a{<}\nu x{>} \prec P'$
and *cFreshP*: $c \sharp P$
and *cineqx*: $c \neq x$

shows $c \sharp P'$
⟨*proof*⟩

lemma *freshTauTransition*:
fixes P :: *pi*
and c :: *name*

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

shows $c \sharp P'$
⟨*proof*⟩

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*: $c \sharp P$

shows $c \sharp P'$
⟨*proof*⟩

lemma *eqvtI*:
fixes P :: *pi*
and *Rs* :: *residual*
and *perm* :: *name prm*

assumes $P \Longrightarrow_l\hat{}\ Rs$

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

⟨*proof*⟩

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*: $c \sharp P$
  and *cineqb*: $c \neq b$

  shows $c \sharp P'$
⟨*proof*⟩

lemmas *freshTransition* = *freshBoundOutputTransition freshOutputTransition freshInputTransition freshTauTransition*

end

theory *Weak-Late-Sim*
  imports *Weak-Late-Semantics Strong-Late-Sim*
begin

definition *weakSimAct* :: *pi* ⇒ *residual* ⇒ (*'a::fs-name*) ⇒ (*pi* × *pi*) *set* ⇒ *bool*
where
  $weakSimAct\ P\ Rs\ C\ Rel \equiv (\forall Q'\ a\ x.\ Rs = a{<}\nu x{>} \prec Q' \longrightarrow x \sharp C \longrightarrow (\exists P'.\ P \Longrightarrow_l\hat{}\, a{<}\nu x{>} \prec P' \wedge (P', Q') \in Rel)) \wedge$
  $(\forall Q'\ a\ x.\ Rs = a{<}x{>} \prec Q' \longrightarrow x \sharp C \longrightarrow (\exists P''.\ \forall u.\ \exists P'.\ P \Longrightarrow_l u\ in\ P''{\rightarrow}a{<}x{>} \prec P' \wedge (P', Q'[x{::=}u]) \in Rel)) \wedge$
  $(\forall Q'\ \alpha.\ Rs = \alpha \prec Q' \longrightarrow (\exists P'.\ P \Longrightarrow_l\hat{}\, \alpha \prec P' \wedge (P', Q') \in Rel))$

definition *weakSimAux* :: *pi* ⇒ (*pi* × *pi*) *set* ⇒ *pi* ⇒ *bool* **where**
  $weakSimAux\ P\ Rel\ Q \equiv (\forall Q'\ a\ x.\ (Q \longmapsto a{<}\nu x{>} \prec Q' \wedge x \sharp P) \longrightarrow (\exists P'.\ P \Longrightarrow_l\hat{}\, a{<}\nu x{>} \prec P' \wedge (P', Q') \in Rel)) \wedge$
  $(\forall Q'\ a\ x.\ (Q \longmapsto a{<}x{>} \prec Q' \wedge x \sharp P) \longrightarrow (\exists P''.\ \forall u.\ \exists P'.\ P \Longrightarrow_l u\ in\ P''{\rightarrow}a{<}x{>} \prec P' \wedge (P', Q'[x{::=}u]) \in Rel)) \wedge$
  $(\forall Q'\ \alpha.\ Q \longmapsto \alpha \prec Q' \longrightarrow (\exists P'.\ P \Longrightarrow_l\hat{}\, \alpha \prec P' \wedge (P', Q') \in Rel))$

definition *weakSimulation* :: *pi* ⇒ (*pi* × *pi*) *set* ⇒ *pi* ⇒ *bool* (- ⇝^<-> - [*80*, *80*, *80*] *80*) **where**
  $P \leadsto\hat{}{<}Rel{>}\ Q \equiv (\forall Rs.\ Q \longmapsto Rs \longrightarrow weakSimAct\ P\ Rs\ P\ Rel)$

lemmas *simDef* = *weakSimAct-def weakSimulation-def*

lemma *weakSimAux P Rel Q* = *weakSimulation 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 $P \rightsquigarrow\hat{} <A>\ P'$
  and $A \subseteq B$

  shows $P \rightsquigarrow\hat{} <B>\ P'$
⟨*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-name*

  assumes *Eqvt*: *eqvt Rel*
  and *Bound*: $\bigwedge Q'\ a\ x.\ [\![x \sharp C;\ Q \longmapsto a<\nu x> \prec Q']\!] \Longrightarrow \exists P'.\ P \Longrightarrow_l\hat{}\ a<\nu x> \prec P' \wedge (P', Q') \in Rel$
  and *Input*: $\bigwedge Q'\ a\ x.\ [\![x \sharp C;\ Q \longmapsto a<x> \prec Q']\!] \Longrightarrow \exists P''.\ \forall u.\ \exists P'.\ P \Longrightarrow_l u\ in\ P'' \rightarrow a<x> \prec P' \wedge (P', Q'[x::=u]) \in Rel$
  and *Free*: $\bigwedge Q'\ \alpha.\ Q \longmapsto \alpha \prec Q' \Longrightarrow (\exists P'.\ P \Longrightarrow_l\hat{}\ \alpha \prec P' \wedge (P', Q') \in Rel)$

  shows $P \rightsquigarrow\hat{} <Rel>\ Q$
⟨*proof*⟩

lemma *simCases*[*case-names Bound Input Free*]:
  fixes P :: *pi*
  and Q :: *pi*
  and *Rel* :: $(pi \times pi)$ *set*
  and C :: *'a::fs-name*

  assumes *Bound*: $\bigwedge Q'\ a\ x.\ [\![Q \longmapsto a<\nu x> \prec Q';\ x \sharp P]\!] \Longrightarrow \exists P'.\ P \Longrightarrow_l\hat{}\ a<\nu x> \prec P' \wedge (P', Q') \in Rel$
  and *Input*: $\bigwedge Q'\ a\ x.\ [\![Q \longmapsto a<x> \prec Q';\ x \sharp P]\!] \Longrightarrow \exists P''.\ \forall u.\ \exists P'.\ P \Longrightarrow_l u\ in\ P'' \rightarrow a<x> \prec P' \wedge (P', Q'[x::=u]) \in Rel$
  and *Free*: $\bigwedge Q'\ \alpha.\ Q \longmapsto \alpha \prec Q' \Longrightarrow (\exists P'.\ P \Longrightarrow_l\hat{}\ \alpha \prec P' \wedge (P', Q') \in Rel)$

  shows $P \rightsquigarrow\hat{} <Rel>\ Q$
⟨*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* × *pi*) *set*

assumes *EqvtRel*: *eqvt Rel*
and *DerInput*: $\bigwedge b.\ a = InputS\ b \Longrightarrow (\exists P''.\ \forall u.\ \exists P'.\ (P \Longrightarrow_l u\ in\ P''\rightarrow b\text{<}x\text{>} \prec P') \wedge (P',\ Q'[x::=u]) \in Rel)$
and *DerBoundOutput*: $\bigwedge b.\ a = BoundOutputS\ b \Longrightarrow (\exists P'.\ (P \Longrightarrow_l\hat{}\ b\text{<}\nu x\text{>} \prec P') \wedge (P',\ Q') \in Rel)$

shows *weakSimAct* P (a«x» $\prec Q'$) P *Rel*
⟨*proof*⟩

lemma *simActFreeCases*[*consumes 0*, *case-names Der*]:
fixes P :: *pi*
and α :: *freeRes*
and Q' :: *pi*
and *Rel* :: (*pi* × *pi*) *set*

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

shows *weakSimAct* P ($\alpha \prec Q'$) P *Rel*
⟨*proof*⟩

lemma *simE*:
fixes P :: *pi*
and *Rel* :: (*pi* × *pi*) *set*
and Q :: *pi*
and a :: *name*
and x :: *name*
and u :: *name*
and Q' :: *pi*

assumes $P \leadsto\hat{}$<*Rel*> Q

shows $Q \longmapsto a\text{<}\nu x\text{>} \prec Q' \Longrightarrow x \sharp P \Longrightarrow \exists P'.\ P \Longrightarrow_l\hat{}\ a\text{<}\nu x\text{>} \prec P' \wedge (P',\ Q') \in Rel$
and $Q \longmapsto a\text{<}x\text{>} \prec Q' \Longrightarrow x \sharp P \Longrightarrow \exists P''.\ \forall u.\ \exists P'.\ P \Longrightarrow_l u\ in\ P''\rightarrow a\text{<}x\text{>} \prec P' \wedge (P',\ Q'[x::=u]) \in Rel$
and $Q \longmapsto \alpha \prec Q' \Longrightarrow (\exists P'.\ P \Longrightarrow_l\hat{}\ \alpha \prec P' \wedge (P',\ Q') \in Rel)$
⟨*proof*⟩

lemma *weakSimTauChain*:
fixes P :: *pi*
and *Rel* :: (*pi* × *pi*) *set*
and Q :: *pi*
and Q' :: *pi*

assumes *QChain*: $Q \Longrightarrow_\tau Q'$
and *PRelQ*: $(P, Q) \in Rel$
and *Sim*: $\bigwedge P\ Q.\ (P, Q) \in Rel \Longrightarrow P \leadsto\hat{}<Rel> Q$

shows $\exists P'.\ P \Longrightarrow_\tau P' \wedge (P', Q') \in Rel$
⟨*proof*⟩

lemma *simE2*:
fixes P :: *pi*
and *Rel* :: (*pi* × *pi*) *set*
and Q :: *pi*
and a :: *name*
and x :: *name*
and Q' :: *pi*

assumes *PSimQ*: $P \leadsto\hat{}<Rel> Q$
and *Sim*: $\bigwedge P\ Q.\ (P, Q) \in Rel \Longrightarrow P \leadsto\hat{}<Rel> Q$
and *Eqvt*: *eqvt Rel*
and *PRelQ*: $(P, Q) \in Rel$

shows $Q \Longrightarrow_l\hat{}\ a<\nu x> \prec Q' \Longrightarrow x \sharp P \Longrightarrow \exists P'.\ P \Longrightarrow_l\hat{}\ a<\nu x> \prec P' \wedge (P', Q') \in Rel$
and $Q \Longrightarrow_l\hat{}\ \alpha \prec Q' \Longrightarrow \exists P'.\ P \Longrightarrow_l\hat{}\ \alpha \prec P' \wedge (P', Q') \in Rel$
⟨*proof*⟩

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

assumes *Sim*: $P \leadsto\hat{}<Rel> Q$
and *RelRel'*: $Rel \subseteq Rel'$
and *EqvtRel'*: *eqvt Rel'*

shows $(perm \cdot P) \leadsto\hat{}<Rel'> (perm \cdot Q)$
⟨*proof*⟩

lemma *reflexive*:
fixes P :: *pi*
and *Rel* :: (*pi* × *pi*) *set*

assumes $Id \subseteq Rel$

shows $P \leadsto\hat{}<Rel> P$
⟨*proof*⟩

lemma *transitive*:
fixes P :: *pi*
and Q :: *pi*
and R :: *pi*
and *Rel* :: (*pi* × *pi*) *set*
and *Rel′* :: (*pi* × *pi*) *set*
and *Rel′′* :: (*pi* × *pi*) *set*

assumes *QSimR*: Q ⇝ˆ<*Rel′*> R
and *Eqvt*: *eqvt Rel*
and *Eqvt′*: *eqvt Rel′′*
and *Trans*: *Rel O Rel′* ⊆ *Rel′′*
and *Sim*: ⋀P Q. (P, Q) ∈ *Rel* ⟹ P ⇝ˆ<*Rel*> Q
and *PRelQ*: (P, Q) ∈ *Rel*

shows P ⇝ˆ<*Rel′′*> R
⟨*proof*⟩

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

assumes *PSimQ*: P ⇝[*Rel*] Q

shows P ⇝ˆ<*Rel*> Q
⟨*proof*⟩

lemma *strongAppend*:
fixes P :: *pi*
and Q :: *pi*
and R :: *pi*
and *Rel* :: (*pi* × *pi*) *set*
and *Rel′* :: (*pi* × *pi*) *set*
and *Rel′′* :: (*pi* × *pi*) *set*

assumes *PSimQ*: P ⇝ˆ<*Rel*> Q
and *QSimR*: Q ⇝[*Rel′*] R
and *Eqvt′′*: *eqvt Rel′′*
and *Trans*: *Rel O Rel′* ⊆ *Rel′′*

shows P ⇝ˆ<*Rel′′*> R
⟨*proof*⟩

end

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

begin

lemma *monoAux*: $A \subseteq B \Longrightarrow P \rightsquigarrow\hat{}<A> Q \longrightarrow P \rightsquigarrow\hat{}<B> Q$
⟨*proof*⟩

coinductive-set *weakBisim* :: (*pi* × *pi*) *set*
where
step: $[\![P \rightsquigarrow\hat{}<weakBisim> Q; (Q, P) \in weakBisim]\!] \Longrightarrow (P, Q) \in weakBisim$
monos *monoAux*

abbreviation
weakBisimJudge (**infixr** ≈ *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*: $\bigwedge P\ Q.\ (P, Q) \in X \Longrightarrow P \rightsquigarrow\hat{}<(X \cup weakBisim)> Q \wedge ((Q, P) \in X \vee Q \approx P)$

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 $\bigwedge P\ Q.\ (P, Q) \in X \Longrightarrow P \rightsquigarrow\hat{}<(X \cup weakBisim)> Q$
and $\bigwedge P\ Q.\ (P, Q) \in X \Longrightarrow (Q, P) \in X$

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*: $\bigwedge P\ Q.\ (P, Q) \in X \Longrightarrow P \rightsquigarrow\hat{}<X> Q \wedge (Q, P) \in X$

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 $\bigwedge P\ Q.\ (P, Q) \in X \Longrightarrow P \rightsquigarrow\hat{}<X> Q$
and $\bigwedge P\ Q.\ (P, Q) \in X \Longrightarrow (Q, P) \in X$

shows $P \approx Q$
$\langle proof \rangle$
lemma *monotonic*: $mono(\lambda p\ x1\ x2.\ \exists P\ Q.\ x1 = P \wedge x2 = Q \wedge P \leadsto\hat{}<\{(xa, x).\ p\ xa\ x\}> Q \wedge Q \leadsto\hat{}<\{(xa, x).\ p\ xa\ x\}> P)$
$\langle proof \rangle$

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

assumes $P \approx Q$

shows $P \leadsto\hat{}<weakBisim> Q$
and $Q \approx P$
$\langle proof \rangle$

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

assumes $P \leadsto\hat{}<weakBisim> Q$
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 $P \approx Q$
⟨*proof*⟩

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

shows $P \approx P$
⟨*proof*⟩

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

assumes $P \approx Q$

shows $Q \approx P$
⟨*proof*⟩

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

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

shows $P \approx R$
⟨*proof*⟩

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*: $\bigwedge P\ Q.\ (P, Q) \in X \Longrightarrow P \rightsquigarrow\hat{}<(bisim\ O\ X\ O\ bisim)>\ Q \wedge (Q, P) \in X$

shows $P \approx Q$
⟨*proof*⟩

lemma *weakBisimTransitiveCoinduct*[*case-names cSim cSym*, *consumes 2*]:
assumes *p*: $(P, Q) \in X$
and *Eqvt*: *eqvt X*
and *rSim*: $\bigwedge P\ Q.\ (P, Q) \in X \Longrightarrow P \rightsquigarrow\hat{}<(bisim\ O\ X\ O\ bisim)>\ Q$
and *rSym*: $\bigwedge P\ Q.\ (P, Q) \in X \Longrightarrow (Q, P) \in X$

shows $P \approx Q$
⟨*proof*⟩

end

theory *Weak-Late-Step-Sim*
 imports *Weak-Late-Step-Semantics Weak-Late-Sim Strong-Late-Sim*
begin

definition $weakStepSimAct :: pi \Rightarrow residual \Rightarrow ('a::fs\text{-}name) \Rightarrow (pi \times pi)\ set \Rightarrow bool$ **where**
 $weakStepSimAct\ P\ Rs\ C\ Rel \equiv (\forall Q'\ a\ x.\ Rs = a{<}\nu x{>} \prec Q' \longrightarrow x \sharp C \longrightarrow (\exists P'.\ P \Longrightarrow_l a{<}\nu x{>} \prec P' \wedge (P', Q') \in Rel)) \wedge$
 $(\forall Q'\ a\ x.\ Rs = a{<}x{>} \prec Q' \longrightarrow x \sharp C \longrightarrow (\exists P''.\ \forall u.\ \exists P'.\ P \Longrightarrow_l u\ in\ P''{\rightarrow}a{<}x{>} \prec P' \wedge (P', Q'[x::=u]) \in Rel)) \wedge$
 $(\forall Q'\ \alpha.\ Rs = \alpha \prec Q' \longrightarrow (\exists P'.\ P \Longrightarrow_l \alpha \prec P' \wedge (P', Q') \in Rel))$

definition $weakStepSimAux :: pi \Rightarrow (pi \times pi)\ set \Rightarrow pi \Rightarrow bool$ **where**
 $weakStepSimAux\ P\ Rel\ Q \equiv (\forall Q'\ a\ x.\ (Q \longmapsto a{<}\nu x{>} \prec Q' \wedge x \sharp P) \longrightarrow (\exists P'.\ P \Longrightarrow_l a{<}\nu x{>} \prec P' \wedge (P', Q') \in Rel)) \wedge$
 $(\forall Q'\ a\ x.\ (Q \longmapsto a{<}x{>} \prec Q' \wedge x \sharp P) \longrightarrow (\exists P''.\ \forall u.\ \exists P'.\ P \Longrightarrow_l u\ in\ P''{\rightarrow}a{<}x{>} \prec P' \wedge (P', Q'[x::=u]) \in Rel)) \wedge$
 $(\forall Q'\ \alpha.\ Q \longmapsto \alpha \prec Q' \longrightarrow (\exists P'.\ P \Longrightarrow_l \alpha \prec P' \wedge (P', Q') \in Rel))$

definition $weakStepSim :: pi \Rightarrow (pi \times pi)\ set \Rightarrow pi \Rightarrow bool$ (- $\rightsquigarrow$<-> - [80, 80, 80] 80) **where**
 $P \rightsquigarrow{<}Rel{>}\ Q \equiv (\forall Rs.\ Q \longmapsto Rs \longrightarrow weakStepSimAct\ P\ Rs\ P\ Rel)$

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 $P \rightsquigarrow{<}A{>}\ P'$
 and $A \subseteq B$

 shows $P \rightsquigarrow{<}B{>}\ P'$
⟨*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*: $\bigwedge Q'\ a\ x.\ [\![x \sharp C;\ Q \longmapsto a{<}\nu x{>} \prec Q']\!] \Longrightarrow \exists P'.\ P \Longrightarrow_l a{<}\nu x{>} \prec P' \wedge (P', Q') \in Rel$
and *Input*: $\bigwedge Q'\ a\ x.\ [\![x \sharp C;\ Q \longmapsto a{<}x{>} \prec Q']\!] \Longrightarrow \exists P''.\ \forall u.\ \exists P'.\ P \Longrightarrow_l u\ in\ P''{\rightarrow}a{<}x{>} \prec P' \wedge (P', Q'[x{::=}u]) \in Rel$
and *Free*: $\bigwedge Q'\ \alpha.\ Q \longmapsto \alpha \prec Q' \Longrightarrow (\exists P'.\ P \Longrightarrow_l \alpha \prec P' \wedge (P', Q') \in Rel)$

shows $P \leadsto{<}Rel{>}\ Q$
⟨*proof*⟩

lemma *simCases*[*consumes 0*, *case-names Bound Input Free*]:
fixes P :: *pi*
and Q :: *pi*
and *Rel* :: (*pi* × *pi*) *set*
and C :: *'a::fs-name*

assumes *Bound*: $\bigwedge Q'\ a\ x.\ [\![Q \longmapsto a{<}\nu x{>} \prec Q';\ x \sharp P]\!] \Longrightarrow \exists P'.\ P \Longrightarrow_l a{<}\nu x{>} \prec P' \wedge (P', Q') \in Rel$
and *Input*: $\bigwedge Q'\ a\ x.\ [\![Q \longmapsto a{<}x{>} \prec Q';\ x \sharp P]\!] \Longrightarrow \exists P''.\ \forall u.\ \exists P'.\ P \Longrightarrow_l u\ in\ P''{\rightarrow}a{<}x{>} \prec P' \wedge (P', Q'[x{::=}u]) \in Rel$
and *Free*: $\bigwedge Q'\ \alpha.\ Q \longmapsto \alpha \prec Q' \Longrightarrow (\exists P'.\ P \Longrightarrow_l \alpha \prec P' \wedge (P', Q') \in Rel)$

shows $P \leadsto{<}Rel{>}\ Q$
⟨*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* × *pi*) *set*

assumes *EqvtRel*: *eqvt Rel*
and *DerInput*: $\bigwedge b.\ a = InputS\ b \Longrightarrow (\exists P''.\ \forall u.\ \exists P'.\ (P \Longrightarrow_l u\ in\ P''{\rightarrow}b{<}x{>} \prec P') \wedge (P', Q'[x{::=}u]) \in Rel)$
and *DerBoundOutput*: $\bigwedge b.\ a = BoundOutputS\ b \Longrightarrow (\exists P'.\ (P \Longrightarrow_l b{<}\nu x{>} \prec P') \wedge (P', Q') \in Rel)$

shows *weakStepSimAct P* (*a*«*x*» ≺ *Q'*) *P Rel*
⟨*proof*⟩

lemma *simActFreeCases*[*consumes 0*, *case-names Free*]:
fixes P :: *pi*
and α :: *freeRes*
and C :: *'a::fs-name*
and *Rel* :: (*pi* × *pi*) *set*

assumes *Der*: $\exists P'. (P \Longrightarrow_l \alpha \prec P') \wedge (P', Q') \in Rel$

shows *weakStepSimAct* P $(\alpha \prec Q')$ P *Rel*
⟨*proof*⟩

lemma *simE*:
fixes P :: *pi*
and *Rel* :: (*pi* × *pi*) *set*
and Q :: *pi*
and a :: *name*
and x :: *name*
and u :: *name*
and Q' :: *pi*

assumes $P \leadsto$<*Rel*> Q

shows $Q \longmapsto a{<}\nu x{>} \prec Q' \Longrightarrow x \sharp P \Longrightarrow \exists P'. P \Longrightarrow_l a{<}\nu x{>} \prec P' \wedge (P', Q') \in Rel$
and $Q \longmapsto a{<}x{>} \prec Q' \Longrightarrow x \sharp P \Longrightarrow \exists P''. \forall u. \exists P'. P \Longrightarrow_l u$ *in* $P'' \rightarrow a{<}x{>} \prec P' \wedge (P', Q'[x::=u]) \in Rel$
and $Q \longmapsto \alpha \prec Q' \Longrightarrow (\exists P'. P \Longrightarrow_l \alpha \prec P' \wedge (P', Q') \in Rel)$
⟨*proof*⟩

lemma *weakSimTauChain*:
fixes P :: *pi*
and *Rel* :: (*pi* × *pi*) *set*
and Q :: *pi*
and Q' :: *pi*

assumes *QChain*: $Q \Longrightarrow_\tau Q'$
and *PRelQ*: $(P, Q) \in Rel$
and *Sim*: $\bigwedge P\ Q. (P, Q) \in Rel \Longrightarrow P \leadsto$<*Rel*> Q

shows $\exists P'. P \Longrightarrow_\tau P' \wedge (P', Q') \in Rel$
⟨*proof*⟩

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

assumes *PSimQ*: $P \leadsto[Rel]\ Q$

shows $P \leadsto$<*Rel*> Q
⟨*proof*⟩

lemma *weakSimWeakEqSim*:
fixes P :: *pi*
and Q :: *pi*

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

assumes $P \rightsquigarrow <Rel>\ Q$

shows $P \rightsquigarrow\hat{}\ <Rel>\ Q$
⟨*proof*⟩

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

assumes *Sim*: $P \rightsquigarrow <Rel>\ Q$
and *RelRel'*: $Rel \subseteq Rel'$
and *EqvtRel'*: $eqvt\ Rel'$

shows $(perm \cdot P) \rightsquigarrow <Rel'>\ (perm \cdot Q)$
⟨*proof*⟩

lemma *simE2*:
fixes $P :: pi$
and $Rel :: (pi \times pi)\ set$
and $Q :: pi$
and $a :: name$
and $x :: name$
and $Q' :: pi$

assumes *PSimQ*: $P \rightsquigarrow <Rel>\ Q$
and *Sim*: $\bigwedge P\ Q.\ (P, Q) \in Rel \Longrightarrow P \rightsquigarrow\hat{}\ <Rel>\ Q$
and *Eqvt*: $eqvt\ Rel$
and *PRelQ*: $(P, Q) \in Rel$

shows $Q \Longrightarrow_l a<\nu x> \prec Q' \Longrightarrow x \sharp P \Longrightarrow \exists P'.\ P \Longrightarrow_l a<\nu x> \prec P' \wedge (P', Q') \in Rel$
and $Q \Longrightarrow_l \alpha \prec Q' \Longrightarrow \exists P'.\ P \Longrightarrow_l \alpha \prec P' \wedge (P', Q') \in Rel$
⟨*proof*⟩

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

assumes $Id \subseteq Rel$

shows $P \rightsquigarrow <Rel>\ P$
⟨*proof*⟩

lemma *transitive*:
fixes *P* :: *pi*
and *Q* :: *pi*
and *R* :: *pi*
and *Rel* :: (*pi* × *pi*) *set*
and *Rel′* :: (*pi* × *pi*) *set*
and *Rel′′* :: (*pi* × *pi*) *set*

assumes *PSimQ*: *P* ⇝<*Rel*> *Q*
and *QSimR*: *Q* ⇝<*Rel′*> *R*
and *Eqvt*: *eqvt Rel*
and *Eqvt′*: *eqvt Rel′′*
and *Trans*: *Rel O Rel′* ⊆ *Rel′′*
and *Sim*: ⋀*P Q*. (*P*, *Q*) ∈ *Rel* ⟹ *P* ⇝^<*Rel*> *Q*
and *PRelQ*: (*P*, *Q*) ∈ *Rel*

shows *P* ⇝<*Rel′′*> *R*
⟨*proof*⟩

end

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

definition *congruence* :: (*pi* × *pi*) *set* **where**
congruence ≡ {(*P*, *Q*) |*P Q*. *P* ⇝<*weakBisim*> *Q* ∧ *Q* ⇝<*weakBisim*> *P*}
abbreviation *congruenceJudge* (**infixr** ≃ *65*) **where** *P* ≃ *Q* ≡ (*P*, *Q*) ∈ *congruence*

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

assumes *P* ≃ *Q*

shows *P* ⇝<*weakBisim*> *Q*
and *Q* ⇝<*weakBisim*> *P*
⟨*proof*⟩

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

assumes *P* ⇝<*weakBisim*> *Q*
and *Q* ⇝<*weakBisim*> *P*

shows *P* ≃ *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* ⊆ *weakBisim*
⟨*proof*⟩

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

  shows $P \simeq P$
⟨*proof*⟩

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

assumes $P \simeq Q$

shows $Q \simeq P$
⟨*proof*⟩

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

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

shows $P \simeq R$
⟨*proof*⟩

end

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

consts *weakBisimSubst* :: (*pi* × *pi*) *set*
abbreviation
weakBisimSubstJudge (**infixr** $\approx^s$ *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 $P \approx Q$
⟨*proof*⟩

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

assumes $P \sim^s Q$

shows $P \approx^s Q$
⟨*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** $\simeq^s$ *65*) **where**
$P \simeq^s Q \equiv (P, Q) \in (substClosed\ congruence)$

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

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

assumes $P \simeq^s Q$

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 $P \simeq^s Q$
⟨*proof*⟩

lemma *weakEqSubset*:
shows *substClosed congruence* ⊆ *weakBisim*
⟨*proof*⟩

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

assumes $P \simeq^s Q$

shows $P \simeq Q$
⟨*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 $(perm \cdot P) \simeq^s (perm \cdot Q)$
⟨*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* × *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 $\mathbf{0} \leadsto[Rel]\ \tau.(P) \Longrightarrow False$
and $\mathbf{0} \leadsto[Rel]\ a<x>.P \Longrightarrow False$
and $\mathbf{0} \leadsto[Rel]\ a\{b\}.P \Longrightarrow False$
⟨*proof*⟩

lemma *nilSimRight*:
fixes P :: *pi*
and *Rel* :: (*pi* × *pi*) *set*

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

lemma *matchIdLeft*:
fixes a :: *name*
and P :: *pi*
and *Rel* :: (*pi* × *pi*) *set*

assumes $Id \subseteq Rel$

shows $[a\frown a]P \leadsto[Rel]\ P$
⟨*proof*⟩

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

assumes *IdRel*: $Id \subseteq Rel$

shows $P \leadsto[Rel]\ [a\frown a]P$
⟨*proof*⟩

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

assumes $a \neq b$

shows $\mathbf{0} \leadsto[Rel]\ [a\frown b]P$
⟨*proof*⟩

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 $[a\neq b]P \leadsto[Rel]\ P$
⟨*proof*⟩

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 $P \leadsto[Rel]\ [a\neq b]P$
⟨*proof*⟩

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

shows $\mathbf{0} \leadsto[Rel]\ [a{\neq}a]P$
⟨*proof*⟩

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

assumes *Id*: $Id \subseteq Rel$

shows $P \oplus Q \leadsto[Rel]\ Q \oplus P$
⟨*proof*⟩

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

assumes $Id \subseteq Rel$

shows $P \leadsto[Rel]\ P \oplus P$
⟨*proof*⟩

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

assumes *I*: $Id \subseteq Rel$

shows $P \oplus P \leadsto[Rel]\ P$
⟨*proof*⟩

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

assumes *Id*: $Id \subseteq Rel$

shows $(P \oplus Q) \oplus R \leadsto[Rel]\ P \oplus (Q \oplus R)$
⟨*proof*⟩

lemma *sumAssocRight*:
fixes P :: *pi*
and Q :: *pi*
and R :: *pi*
and *Rel* :: (*pi* × *pi*) *set*

assumes *Id*: *Id* ⊆ *Rel*

shows $P \oplus (Q \oplus R) \rightsquigarrow[Rel]\ (P \oplus Q) \oplus R$
⟨*proof*⟩

lemma *sumZeroLeft*:
fixes P :: *pi*
and *Rel* :: (*pi* × *pi*) *set*

assumes *Id*: *Id* ⊆ *Rel*

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

lemma *sumZeroRight*:
fixes P :: *pi*
and *Rel* :: (*pi* × *pi*) *set*

assumes *Id*: *Id* ⊆ *Rel*

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

lemma *sumResLeft*:
fixes x :: *name*
and P :: *pi*
and Q :: *pi*

assumes *Id*: *Id* ⊆ *Rel*
and *Eqvt*: *eqvt Rel*

shows $(<\nu x>P) \oplus (<\nu x>Q) \rightsquigarrow[Rel]\ <\nu x>(P \oplus Q)$
⟨*proof*⟩

lemma *sumResRight*:
fixes x :: *name*
and P :: *pi*
and Q :: *pi*

assumes *Id*: *Id* ⊆ *Rel*
and *Eqvt*: *eqvt Rel*

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

⟨*proof*⟩

lemma *parZeroLeft*:
  fixes P :: *pi*
  and *Rel* :: (*pi* × *pi*) *set*

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

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

lemma *parZeroRight*:
  fixes P :: *pi*
  and *Rel* :: (*pi* × *pi*) *set*

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

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

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

  assumes *Sym*: $\bigwedge R\ S.\ (R \parallel S,\ S \parallel R) \in Rel$
  and *Res*: $\bigwedge R\ S\ x.\ (R,\ S) \in Rel \Longrightarrow (<\nu x>R,\ <\nu x>S) \in Rel$

  shows $P \parallel Q \rightsquigarrow[Rel]\ Q \parallel P$
⟨*proof*⟩

lemma *parAssocLeft*:
  fixes P :: *pi*
  and Q :: *pi*
  and R :: *pi*
  and *Rel* :: (*pi* × *pi*) *set*

  assumes *Ass*: $\bigwedge S\ T\ U.\ ((S \parallel T) \parallel U,\ S \parallel (T \parallel U)) \in Rel$
  and *Res*: $\bigwedge S\ T\ x.\ (S,\ T) \in Rel \Longrightarrow (<\nu x>S,\ <\nu x>T) \in Rel$
  and *FreshExt*: $\bigwedge S\ T\ U\ x.\ x \sharp S \Longrightarrow (<\nu x>((S \parallel T) \parallel U),\ S \parallel <\nu x>(T \parallel U)) \in Rel$
  and *FreshExt'*: $\bigwedge S\ T\ U\ x.\ x \sharp U \Longrightarrow ((<\nu x>(S \parallel T)) \parallel U,\ <\nu x>(S \parallel (T \parallel U))) \in Rel$

  shows $(P \parallel Q) \parallel R \rightsquigarrow[Rel]\ P \parallel (Q \parallel R)$
⟨*proof*⟩

lemma *substRes3*:
fixes $a :: name$
and $P :: pi$
and $x :: name$

shows $(<\nu a>P)[x::=a] = <\nu x>([(x, a)] \cdot P)$
⟨*proof*⟩

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

assumes $x \sharp P$
and *Id*: $Id \subseteq Rel$
and *EqvtRel*: $eqvt\ Rel$
and *Res*: $\bigwedge R\ S\ y.\ y \sharp R \Longrightarrow (<\nu y>(R \parallel S),\ R \parallel <\nu y>S) \in Rel$
and *ScopeExt*: $\bigwedge R\ S\ y\ z.\ y \sharp R \Longrightarrow (<\nu y><\nu z>(R \parallel S),\ <\nu z>(R \parallel <\nu y>S)) \in Rel$

shows $<\nu x>(P \parallel Q) \leadsto[Rel]\ P \parallel <\nu x>Q$
⟨*proof*⟩

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

assumes $x \sharp P$
and *Id*: $Id \subseteq Rel$
and $eqvt\ Rel$
and *Res*: $\bigwedge R\ S\ y.\ y \sharp R \Longrightarrow (R \parallel <\nu y>S,\ <\nu y>(R \parallel S)) \in Rel$
and *ScopeExt*: $\bigwedge R\ S\ y\ z.\ y \sharp R \Longrightarrow (<\nu z>(R \parallel <\nu y>S),\ <\nu y><\nu z>(R \parallel S)) \in Rel$

shows $P \parallel <\nu x>Q \leadsto[Rel]\ <\nu x>(P \parallel Q)$
⟨*proof*⟩

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

shows $\mathbf{0} \leadsto[Rel]\ <\nu x>\mathbf{0}$
⟨*proof*⟩

lemma *resComm*:

fixes a :: *name*
and b :: *name*
and P :: *pi*
and *Rel* :: (*pi* × *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* × *pi*) *set*

assumes $Id \subseteq Rel$

shows $!P \rightsquigarrow[Rel]\ P \parallel !P$
⟨*proof*⟩

lemma *bangRightSC*:
fixes P :: *pi*
and *Rel* :: (*pi* × *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* × *pi*) *set*
and b :: *name*

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

lemma *resInputLeft*:
fixes x :: *name*
and a :: *name*
and y :: *name*
and P :: *pi*
and *Rel* :: (*pi* × *pi*) *set*

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

shows <νx>a<y>.P ⇝[Rel] a<y>.(<νx>P)
⟨*proof*⟩

lemma *resInputRight*:
fixes *a* :: *name*
and *y* :: *name*
and *x* :: *name*
and *P* :: *pi*
and *Rel* :: (*pi* × *pi*) *set*

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

shows a<y>.(<νx>P) ⇝[Rel] <νx>a<y>.P
⟨*proof*⟩

lemma *resOutputLeft*:
fixes *x* :: *name*
and *a* :: *name*
and *b* :: *name*
and *P* :: *pi*
and *Rel* :: (*pi* × *pi*) *set*

assumes *xineqa*: $x \neq a$
and *xineqb*: $x \neq b$
and *Id*: $Id \subseteq Rel$

shows <νx>a{b}.P ⇝[Rel] a{b}.(<νx>P)
⟨*proof*⟩

lemma *resOutputRight*:
fixes *x* :: *name*
and *a* :: *name*
and *b* :: *name*
and *P* :: *pi*
and *Rel* :: (*pi* × *pi*) *set*

assumes *xineqa*: $x \neq a$
and *xineqb*: $x \neq b$
and *Id*: $Id \subseteq Rel$
and *Eqvt*: *eqvt Rel*

shows $a\{b\}.(<\nu x>P) \leadsto[Rel] <\nu x>a\{b\}.P$
⟨*proof*⟩

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

assumes *Id*: $Id \subseteq Rel$

shows $<\nu x>(\tau.(P)) \leadsto[Rel]\ \tau.(<\nu x>P)$
⟨*proof*⟩

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

assumes *Id*: $Id \subseteq Rel$

shows $\tau.(<\nu x>P) \leadsto[Rel] <\nu x>(\tau.(P))$
⟨*proof*⟩

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 $\tau.(P) \sim \mathbf{0} \Longrightarrow False$
and $a<x>.P \sim \mathbf{0} \Longrightarrow False$
and $a\{b\}.P \sim \mathbf{0} \Longrightarrow False$
and $\mathbf{0} \sim \tau.(P) \Longrightarrow False$
and $\mathbf{0} \sim a<x>.P \Longrightarrow False$
and $\mathbf{0} \sim a\{b\}.P \Longrightarrow False$
⟨*proof*⟩

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

shows $[a\frown a]P \sim P$
⟨*proof*⟩

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

assumes $a \neq b$

shows $[a\frown b]P \sim \mathbf{0}$
⟨*proof*⟩

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

assumes $a \neq b$

shows $[a{\neq}b]P \sim P$
⟨*proof*⟩

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

shows $[a{\neq}a]P \sim \mathbf{0}$
⟨*proof*⟩

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

shows $<\nu x>\mathbf{0} \sim \mathbf{0}$
⟨*proof*⟩

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

shows $<\nu x><\nu y>P \sim <\nu y><\nu x>P$
⟨*proof*⟩

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 $x \sharp P$

shows $<\nu x>(P \parallel Q) \sim P \parallel <\nu x>Q$
⟨*proof*⟩

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

assumes *xFreshQ*: $x \sharp Q$

shows $<\nu x>(P \parallel Q) \sim (<\nu x>P) \parallel Q$
⟨*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 $x \sharp P$

shows $<\nu x>P \sim P$
⟨*proof*⟩

lemma *sumRes*:
fixes x :: *name*
and P :: *pi*
and Q :: *pi*

shows $<\nu x>(P \oplus Q) \sim (<\nu x>P) \oplus (<\nu x>Q)$
⟨*proof*⟩

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

assumes $x \sharp P$

shows $<\nu x>(P \oplus Q) \sim P \oplus <\nu x>Q$
⟨*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 $<\nu x>x<y>.P \sim \mathbf{0}$
and $<\nu x>x\{b\}.P \sim \mathbf{0}$
⟨*proof*⟩

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

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

shows $<\nu x>a<y>.P \sim a<y>.(<\nu x>P)$
⟨*proof*⟩

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

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

shows $<\nu x>a\{b\}.P \sim a\{b\}.(<\nu x>P)$
⟨*proof*⟩

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

shows $<\nu x>\tau.(P) \sim \tau.(<\nu x>P)$
⟨*proof*⟩

inductive *structCong* $:: pi \Rightarrow pi \Rightarrow bool$ (- $\equiv_s$ - [*70*, *70*] *70*)
where
Refl: $P \equiv_s P$
| *Sym*: $P \equiv_s Q \Longrightarrow Q \equiv_s P$

| *Trans*: $[\![P \equiv_s Q; Q \equiv_s R]\!] \Longrightarrow P \equiv_s R$

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

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

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

| *ResNil*: $<\nu x>\mathbf{0} \equiv_s \mathbf{0}$
| *ResComm*: $<\nu x><\nu y>P \equiv_s <\nu y><\nu x>P$
| *ResSum*: $<\nu x>(P \oplus Q) \equiv_s <\nu x>P \oplus <\nu x>Q$
| *ScopeExtPar*: $x \sharp P \Longrightarrow <\nu x>(P \parallel Q) \equiv_s P \parallel <\nu x>Q$
| *InputRes*: $[\![x \neq a; x \neq y]\!] \Longrightarrow <\nu x>a<y>.P \equiv_s a<y>.(<\nu x>P)$
| *OutputRes*: $[\![x \neq a; x \neq b]\!] \Longrightarrow <\nu x>a\{b\}.P \equiv_s a\{b\}.(<\nu x>P)$
| *TauRes*: $<\nu x>\tau.(P) \equiv_s \tau.(<\nu x>P)$

| *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*: $x \sharp P$

shows $<\nu x>P \sim^s P$
⟨*proof*⟩

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

shows $<\nu x><\nu y>P \sim^s <\nu y><\nu x>P$
⟨*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 $<\nu x>(P \oplus Q) \sim^s <\nu x>P \oplus <\nu x>Q$
⟨*proof*⟩

lemma *scopeExtSum*:

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

assumes *xFreshP*: $x \sharp P$

shows $<\nu x>(P \oplus Q) \sim^s P \oplus <\nu x>Q$
⟨*proof*⟩

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

shows $P \parallel \mathbf{0} \sim^s P$
⟨*proof*⟩

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

shows $P \parallel Q \sim^s Q \parallel P$
⟨*proof*⟩

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

shows $(P \parallel Q) \parallel R \sim^s P \parallel (Q \parallel R)$
⟨*proof*⟩

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

assumes *xFreshP*: $x \sharp P$

shows $<\nu x>(P \parallel Q) \sim^s P \parallel <\nu x>Q$
⟨*proof*⟩

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

assumes *xFreshP*: $x \sharp Q$

shows $<\nu x>(P \parallel Q) \sim^s (<\nu x>P) \parallel Q$
⟨*proof*⟩

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

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

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

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

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

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

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

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

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

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

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

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

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 $<\nu a><\nu b>P \simeq^s <\nu b><\nu a>P$
⟨*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 $(P \oplus Q) \oplus R \simeq^s P \oplus (Q \oplus R)$
⟨*proof*⟩

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

shows $P \oplus \mathbf{0} \simeq^s P$
⟨*proof*⟩

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

shows $P \parallel \mathbf{0} \simeq^s P$
⟨*proof*⟩

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

shows $P \parallel Q \simeq^s Q \parallel P$
⟨*proof*⟩

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

assumes $x \sharp P$

shows $<\nu x>(P \parallel Q) \simeq^s P \parallel <\nu x>Q$
⟨*proof*⟩

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

assumes *xFreshQ*: $x \sharp Q$

shows $<\nu x>(P \parallel Q) \simeq^s (<\nu x>P) \parallel Q$
⟨*proof*⟩

lemma *parAssoc*:

fixes P :: *pi*
and Q :: *pi*
and R :: *pi*

shows $(P \parallel Q) \parallel R \simeq^s P \parallel (Q \parallel R)$
⟨*proof*⟩

lemma *scopeFresh*:
fixes P :: *pi*
and a :: *name*

assumes *aFreshP*: $a \sharp P$

shows $<\nu a>P \simeq^s P$
⟨*proof*⟩

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

assumes $x \sharp P$

shows $<\nu x>(P \oplus Q) \simeq^s P \oplus <\nu x>Q$
⟨*proof*⟩

lemma *bangSC*:
fixes P

shows $!P \simeq^s P \parallel !P$
⟨*proof*⟩

end

theory *Weak-Late-Step-Sim-Pres*
imports *Weak-Late-Step-Sim*
begin

lemma *tauPres*:
fixes P :: *pi*
and Q :: *pi*
and *Rel* :: (*pi* × *pi*) *set*
and *Rel'* :: (*pi* × *pi*) *set*

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

shows $\tau.(P) \leadsto<Rel> \tau.(Q)$
⟨*proof*⟩

lemma *inputPres*:
fixes P :: *pi*
and Q :: *pi*
and a :: *name*
and x :: *name*
and *Rel* :: (*pi* × *pi*) *set*

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

shows $a{<}x{>}.P \rightsquigarrow{<}Rel{>}\ a{<}x{>}.Q$
⟨*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*: $(P,\ Q) \in Rel$

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* × *pi*) *set*
and *Rel′* :: (*pi* × *pi*) *set*

assumes *PSimQ*: $P \rightsquigarrow{<}Rel{>}\ Q$
and *RelRel′*: $Rel \subseteq Rel'$

shows $[a \frown b]P \rightsquigarrow{<}Rel'{>}\ [a \frown b]Q$
⟨*proof*⟩

lemma *mismatchPres*:
fixes P :: *pi*
and Q :: *pi*
and a :: *name*
and b :: *name*
and *Rel* :: (*pi* × *pi*) *set*
and *Rel′* :: (*pi* × *pi*) *set*

assumes *PSimQ*: $P \rightsquigarrow{<}Rel{>}\ Q$

and *RelRel'*: $Rel \subseteq Rel'$

shows $[a{\neq}b]P \leadsto<Rel'> [a{\neq}b]Q$
⟨*proof*⟩

lemma *sumCompose*:
fixes P :: *pi*
and Q :: *pi*
and R :: *pi*
and T :: *pi*

assumes *PSimQ*: $P \leadsto<Rel> Q$
and *RSimT*: $R \leadsto<Rel> T$
and *RelRel'*: $Rel \subseteq Rel'$

shows $P \oplus R \leadsto<Rel'> Q \oplus T$
⟨*proof*⟩

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

assumes *PSimQ*: $P \leadsto<Rel> Q$
and *Id*: $Id \subseteq Rel$
and *RelRel'*: $Rel \subseteq Rel'$

shows $P \oplus R \leadsto<Rel'> Q \oplus R$
⟨*proof*⟩

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

assumes *PSimQ*: $P \leadsto<Rel> Q$
and *PRelQ*: $(P, Q) \in Rel$
and *Par*: $\bigwedge P\ Q\ R.\ (P, Q) \in Rel \Longrightarrow (P \parallel R, Q \parallel R) \in Rel'$
and *Res*: $\bigwedge P\ Q\ a.\ (P, Q) \in Rel' \Longrightarrow (<\nu a>P, <\nu a>Q) \in Rel'$
and *EqvtRel*: *eqvt Rel*
and *EqvtRel'*: *eqvt Rel'*

shows $P \parallel R \leadsto<Rel'> Q \parallel R$
⟨*proof*⟩

lemma *resPres*:
fixes P :: *pi*

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

assumes *PSimQ*: $P \rightsquigarrow$<*Rel*> Q
and *ResRel*: $\bigwedge(P::pi)\ (Q::pi)\ (x::name).\ (P, Q) \in Rel \Longrightarrow (<\nu x>P, <\nu x>Q) \in Rel'$
and *RelRel'*: $Rel \subseteq Rel'$
and *EqvtRel*: *eqvt Rel*
and *EqvtRel'*: *eqvt Rel'*

shows $<\nu x>P \rightsquigarrow <Rel'> <\nu x>Q$
⟨*proof*⟩

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

assumes *PSimQ*: $P \rightsquigarrow <Rel'> Q$
and *PRelQ*: $(P, Q) \in Rel$
and *Sim*: $\bigwedge P\ Q.\ (P, Q) \in Rel \Longrightarrow P \rightsquigarrow <Rel'> Q$
and *RelRel'*: $\bigwedge P\ Q.\ (P, Q) \in Rel \Longrightarrow (P, Q) \in Rel'$
and *eqvtRel'*: *eqvt Rel'*

shows $!P \rightsquigarrow <bangRel\ Rel'> !Q$
⟨*proof*⟩

end

theory *Weak-Late-Bisim-SC*
imports *Weak-Late-Bisim Strong-Late-Bisim-SC*
begin

lemma *resComm*:
fixes P :: *pi*

shows $<\nu a><\nu b>P \approx <\nu b><\nu a>P$
⟨*proof*⟩

lemma *matchId*:
fixes a :: *name*

and $P :: pi$

shows $[a \frown a]P \approx P$
⟨*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 *mismatchZero*:
fixes $a :: name$
and $P :: pi$

shows $[a \neq a]P \approx \mathbf{0}$
⟨*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 $P \parallel \mathbf{0} \approx P$
$\langle proof \rangle$

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

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

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

  assumes $x \sharp P$

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

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

  assumes *xFreshQ*: $x \sharp Q$

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

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 *aFreshP*: $a \sharp P$

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

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

  assumes $x \sharp P$

  shows $<\nu x>(P \oplus Q) \approx P \oplus <\nu x>Q$
⟨*proof*⟩

lemma *bangSC*:
  fixes P

  shows $!P \approx P \parallel !P$
⟨*proof*⟩

end

theory *Weak-Late-Sim-Pres*
  imports *Weak-Late-Sim*
begin

lemma *tauPres*:
  fixes P :: *pi*
  and Q :: *pi*
  and *Rel* :: (*pi* × *pi*) *set*
  and *Rel'* :: (*pi* × *pi*) *set*

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

  shows $\tau.(P) \leadsto\hat{} <Rel> \tau.(Q)$
⟨*proof*⟩

lemma *inputPres*:
  fixes P :: *pi*
  and Q :: *pi*
  and a :: *name*
  and x :: *name*
  and *Rel* :: (*pi* × *pi*) *set*

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

  shows $a<x>.P \leadsto\hat{} <Rel> a<x>.Q$
⟨*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*: $(P,\ Q) \in Rel$

shows $a\{b\}.P \leadsto\hat{}<Rel>\ a\{b\}.Q$
⟨*proof*⟩

lemma *matchPres*:
fixes P :: *pi*
and Q :: *pi*
and a :: *name*
and b :: *name*
and *Rel* :: (*pi* × *pi*) *set*
and *Rel'* :: (*pi* × *pi*) *set*

assumes *PSimQ*: $P \leadsto\hat{}<Rel>\ Q$
and *RelStay*: $\bigwedge P\ Q\ a.\ (P,\ Q) \in Rel \Longrightarrow ([a\frown a]P,\ Q) \in Rel$
and *RelRel'*: $Rel \subseteq Rel'$

shows $[a\frown b]P \leadsto\hat{}<Rel'>\ [a\frown b]Q$
⟨*proof*⟩

lemma *mismatchPres*:
fixes P :: *pi*
and Q :: *pi*
and a :: *name*
and b :: *name*
and *Rel* :: (*pi* × *pi*) *set*
and *Rel'* :: (*pi* × *pi*) *set*

assumes *PSimQ*: $P \leadsto\hat{}<Rel>\ Q$
and *RelStay*: $\bigwedge P\ Q\ a\ b.\ [\![(P,\ Q) \in Rel;\ a \neq b]\!] \Longrightarrow ([a{\neq}b]P,\ Q) \in Rel$
and *RelRel'*: $Rel \subseteq Rel'$

shows $[a{\neq}b]P \leadsto\hat{}<Rel'>\ [a{\neq}b]Q$
⟨*proof*⟩

lemma *parCompose*:
fixes P :: *pi*
and Q :: *pi*
and R :: *pi*
and T :: *pi*
and *Rel* :: (*pi* × *pi*) *set*
and *Rel'* :: (*pi* × *pi*) *set*
and *Rel''* :: (*pi* × *pi*) *set*

assumes *PSimQ*: *P* ⇝^ <*Rel*> *Q*
and *RSimT*: *R* ⇝^ <*Rel'*> *T*
and *PRelQ*: (*P*, *Q*) ∈ *Rel*
and *RRel'T*: (*R*, *T*) ∈ *Rel'*
and *Par*: ⋀*P Q R T*. ⟦(*P*, *Q*) ∈ *Rel*; (*R*, *T*) ∈ *Rel'*⟧ ⟹ (*P* ∥ *R*, *Q* ∥ *T*) ∈ *Rel''*
and *Res*: ⋀*P Q a*. (*P*, *Q*) ∈ *Rel''* ⟹ (<ν*a*>*P*, <ν*a*>*Q*) ∈ *Rel''*
and *EqvtRel*: *eqvt Rel*
and *EqvtRel'*: *eqvt Rel'*
and *EqvtRel''*: *eqvt Rel''*

shows *P* ∥ *R* ⇝^ <*Rel''*> *Q* ∥ *T*
⟨*proof*⟩

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

assumes *PSimQ*: *P* ⇝^ <*Rel*> *Q*
and *PRelQ*: (*P*, *Q*) ∈ *Rel*
and *Par*: ⋀*P Q R*. (*P*, *Q*) ∈ *Rel* ⟹ (*P* ∥ *R*, *Q* ∥ *R*) ∈ *Rel'*
and *Res*: ⋀*P Q a*. (*P*, *Q*) ∈ *Rel'* ⟹ (<ν*a*>*P*, <ν*a*>*Q*) ∈ *Rel'*
and *EqvtRel*: *eqvt Rel*
and *EqvtRel'*: *eqvt Rel'*

shows *P* ∥ *R* ⇝^ <*Rel'*> *Q* ∥ *R*
⟨*proof*⟩

lemma *resPres*:
fixes *P* :: *pi*
and *Q* :: *pi*
and *Rel* :: (*pi* × *pi*) *set*
and *x* :: *name*
and *Rel'* :: (*pi* × *pi*) *set*

assumes *PSimQ*: *P* ⇝^ <*Rel*> *Q*
and *ResRel*: ⋀(*P*::*pi*) (*Q*::*pi*) (*x*::*name*). (*P*, *Q*) ∈ *Rel* ⟹ (<ν*x*>*P*, <ν*x*>*Q*) ∈ *Rel'*
and *RelRel'*: *Rel* ⊆ *Rel'*
and *EqvtRel*: *eqvt Rel*
and *EqvtRel'*: *eqvt Rel'*

shows <ν*x*>*P* ⇝^ <*Rel'*> <ν*x*>*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*: $\bigwedge P\ Q\ a.\ (P,\ Q) \in Rel \Longrightarrow (<\nu a>P,\ <\nu a>Q) \in Rel$
  and *PRelQ*: $P \leadsto\hat{}<Rel>\ Q$

  shows $(resChain\ lst)\ P \leadsto\hat{}<Rel>\ (resChain\ lst)\ Q$
⟨*proof*⟩

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

  assumes *PSimQ*: $P \leadsto\hat{}<Rel>\ Q$
  and *PRelQ*: $(P,\ Q) \in Rel$
  and *Sim*: $\bigwedge P\ Q.\ (P,\ Q) \in Rel \Longrightarrow P \leadsto\hat{}<Rel>\ Q$

  and *ParComp*: $\bigwedge P\ Q\ R\ T.\ [\![(P,\ Q) \in Rel;\ (R,\ T) \in Rel']\!] \Longrightarrow (P \parallel R,\ Q$
$\parallel T) \in Rel'$
  and *Res*: $\bigwedge P\ Q\ x.\ (P,\ Q) \in Rel' \Longrightarrow (<\nu x>P,\ <\nu x>Q) \in Rel'$

  and *RelStay*: $\bigwedge P\ Q.\ (P \parallel !P,\ Q) \in Rel' \Longrightarrow (!P,\ Q) \in Rel'$
  and *BangRelRel'*: $(bangRel\ Rel) \subseteq Rel'$
  and *eqvtRel'*: *eqvt Rel'*

  shows $!P \leadsto\hat{}<Rel'>\ !Q$
⟨*proof*⟩

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 $a{<}x{>}.P \approx a{<}x{>}.Q$
⟨*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 ${<}\nu x{>}P \approx {<}\nu x{>}Q$
⟨*proof*⟩

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 *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 $a{<}x{>}.P \simeq a{<}x{>}.Q$
$\langle proof \rangle$

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

assumes $P \simeq Q$

shows $[a \frown b]P \simeq [a \frown b]Q$
$\langle proof \rangle$

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

assumes $P \simeq Q$

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

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

assumes $P \simeq Q$

shows $P \oplus R \simeq Q \oplus R$
$\langle proof \rangle$

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 <νx>$P \simeq$ <νx>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* (-<-> [*110*, *110*] *110*)
| *OutputR name name* (-[-] [*110*, *110*] *110*)
| *TauR* (τ *110*)

nominal-datatype *residual* = *BoundOutputR name* «*name*» *pi* (-<ν-> $\prec$ - [*110*, *110*, *110*] *110*)
| *FreeR freeRes pi*

lemma *alphaBoundOutput*:
fixes a :: *name*
and x :: *name*
and P :: *pi*
and x' :: *name*

assumes *A1*: $x' \sharp P$

shows a<νx> $\prec P = a$<$\nu x'$> $\prec ([(x, x')] \cdot P)$

⟨proof⟩

declare *name-fresh*[*simp*]

abbreviation *Transitions-Freejudge* (- ≺ - [80, 80] 80) **where** $\alpha \prec P' \equiv (FreeR\ \alpha\ P')$

inductive *TransitionsEarly* :: *pi* ⇒ *residual* ⇒ *bool* (- ⟼ - [80, 80] 80)
where
Tau: $\tau.(P) \longmapsto \tau \prec P$
| *Input*: $[\![x \neq a; x \neq u]\!] \Longrightarrow a{<}x{>}.P \longmapsto a{<}u{>} \prec (P[x{::=}u])$
| *Output*: $a\{b\}.P \longmapsto a[b] \prec P$

| *Match*: $[\![P \longmapsto V]\!] \Longrightarrow [b{\frown}b]P \longmapsto V$
| *Mismatch*: $[\![P \longmapsto V; a \neq b]\!] \Longrightarrow [a{\neq}b]P \longmapsto V$

| *Open*: $[\![P \longmapsto a[b] \prec P'; a \neq b]\!] \Longrightarrow {<}\nu b{>}P \longmapsto a{<}\nu b{>} \prec P'$
| *Sum1*: $[\![P \longmapsto V]\!] \Longrightarrow (P \oplus Q) \longmapsto V$
| *Sum2*: $[\![Q \longmapsto V]\!] \Longrightarrow (P \oplus Q) \longmapsto V$

| *Par1B*: $[\![P \longmapsto a{<}\nu x{>} \prec P'; x \sharp P; x \sharp Q; x \neq a]\!] \Longrightarrow P \parallel Q \longmapsto a{<}\nu x{>} \prec (P' \parallel Q)$
| *Par1F*: $[\![P \longmapsto \alpha \prec P']\!] \Longrightarrow P \parallel Q \longmapsto \alpha \prec (P' \parallel Q)$
| *Par2B*: $[\![Q \longmapsto a{<}\nu x{>} \prec Q'; x \sharp P; x \sharp Q; x \neq a]\!] \Longrightarrow P \parallel Q \longmapsto a{<}\nu x{>} \prec (P \parallel Q')$
| *Par2F*: $[\![Q \longmapsto \alpha \prec Q']\!] \Longrightarrow P \parallel Q \longmapsto \alpha \prec (P \parallel Q')$

| *Comm1*: $[\![P \longmapsto a{<}b{>} \prec P'; Q \longmapsto a[b] \prec Q']\!] \Longrightarrow P \parallel Q \longmapsto \tau \prec P' \parallel Q'$
| *Comm2*: $[\![P \longmapsto a[b] \prec P'; Q \longmapsto a{<}b{>} \prec Q']\!] \Longrightarrow P \parallel Q \longmapsto \tau \prec P' \parallel Q'$
| *Close1*: $[\![P \longmapsto a{<}x{>} \prec P'; Q \longmapsto a{<}\nu x{>} \prec Q'; x \sharp P; x \sharp Q; x \neq a]\!] \Longrightarrow P \parallel Q \longmapsto \tau \prec {<}\nu x{>}(P' \parallel Q')$
| *Close2*: $[\![P \longmapsto a{<}\nu x{>} \prec P'; Q \longmapsto a{<}x{>} \prec Q'; x \sharp P; x \sharp Q; x \neq a]\!] \Longrightarrow P \parallel Q \longmapsto \tau \prec {<}\nu x{>}(P' \parallel Q')$

| *ResB*: $[\![P \longmapsto a{<}\nu x{>} \prec P'; y \neq a; y \neq x; x \sharp P; x \neq a]\!] \Longrightarrow {<}\nu y{>}P \longmapsto a{<}\nu x{>} \prec ({<}\nu y{>}P')$
| *ResF*: $[\![P \longmapsto \alpha \prec P'; y \sharp \alpha]\!] \Longrightarrow {<}\nu y{>}P \longmapsto \alpha \prec {<}\nu y{>}P'$

| *Bang*: $[\![P \parallel !P \longmapsto V]\!] \Longrightarrow !P \longmapsto V$

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 $P \longmapsto a[b] \prec P'$
and $c \sharp P$

shows $c \neq a$ **and** $c \neq b$ **and** $c \sharp P'$
⟨*proof*⟩

lemma *freshInputAction*:
fixes P :: *pi*
and a :: *name*
and b :: *name*
and P' :: *pi*
and c :: *name*

assumes $P \longmapsto a{<}b{>} \prec P'$
and $c \sharp P$

shows $c \neq a$
⟨*proof*⟩

lemma *freshBoundOutputAction*:
fixes P :: *pi*
and a :: *name*
and x :: *name*
and P' :: *pi*
and c :: *name*

assumes $P \longmapsto a{<}\nu x{>} \prec P'$
and $c \sharp P$

shows $c \neq a$
⟨*proof*⟩

lemmas *freshAction* = *freshOutputAction freshInputAction freshBoundOutputAction*

lemma *freshInputTransition*:
fixes P :: *pi*
and a :: *name*
and u :: *name*
and P' :: *pi*
and c :: *name*

assumes $P \longmapsto a{<}u{>} \prec P'$

and $c \sharp P$
and $c \neq u$

shows $c \sharp P'$
⟨*proof*⟩

lemma *freshBoundOutputTransition*:
fixes P :: *pi*
and a :: *name*
and x :: *name*
and P' :: *pi*
and c :: *name*

assumes $P \longmapsto a{<}\nu x{>} \prec P'$
and $c \sharp P$
and $c \neq x$

shows $c \sharp P'$
⟨*proof*⟩

lemma *freshTauTransition*:
fixes P :: *pi*
and P' :: *pi*
and c :: *name*

assumes $P \longmapsto \tau \prec P'$
and $c \sharp P$

shows $c \sharp P'$
⟨*proof*⟩

lemma *freshFreeTransition*:
fixes P :: *pi*
and α :: *freeRes*
and P' :: *pi*
and c :: *name*

assumes $P \longmapsto \alpha \prec P'$
and $c \sharp P$
and $c \sharp \alpha$

shows $c \sharp P'$
⟨*proof*⟩

lemmas *freshTransition* = *freshInputTransition freshOutputAction freshFreeTransition*
freshBoundOutputTransition freshTauTransition

lemma *substTrans*[*simp*]: $b \sharp P \Longrightarrow ((P{::}pi)[a{::=}b])[b{::=}c] = P[a{::=}c]$

⟨*proof*⟩

lemma *Input*:
fixes $a :: \mathit{name}$
and $x :: \mathit{name}$
and $u :: \mathit{name}$
and $P :: \mathit{pi}$

shows $a{<}x{>}.P \longmapsto a{<}u{>} \prec P[x{::=}u]$
⟨*proof*⟩

lemma *Par1B*:
fixes $P :: \mathit{pi}$
and $a :: \mathit{name}$
and $x :: \mathit{name}$
and $P' :: \mathit{pi}$
and $Q :: \mathit{pi}$

assumes $P \longmapsto a{<}\nu x{>} \prec P'$
and $x \sharp Q$

shows $P \parallel Q \longmapsto a{<}\nu x{>} \prec (P' \parallel Q)$
⟨*proof*⟩

lemma *Par2B*:
fixes $Q :: \mathit{pi}$
and $a :: \mathit{name}$
and $x :: \mathit{name}$
and $Q' :: \mathit{pi}$
and $P :: \mathit{pi}$

assumes $Q \longmapsto a{<}\nu x{>} \prec Q'$
and $x \sharp P$

shows $P \parallel Q \longmapsto a{<}\nu x{>} \prec (P \parallel Q')$
⟨*proof*⟩

lemma *inputInduct*[*consumes 1*, *case-names cInput cMatch cMismatch cSum1 cSum2 cPar1 cPar2 cRes cBang*]:
fixes $P :: \mathit{pi}$
and $a :: \mathit{name}$
and $u :: \mathit{name}$
and $P' :: \mathit{pi}$
and $F :: {'a}{::}\mathit{fs\text{-}name} \Rightarrow \mathit{pi} \Rightarrow \mathit{name} \Rightarrow \mathit{name} \Rightarrow \mathit{pi} \Rightarrow \mathit{bool}$
and $C :: {'a}{::}\mathit{fs\text{-}name}$

assumes *Trans*: $P \longmapsto a{<}u{>} \prec P'$
and $\bigwedge a\ x\ P\ u\ C.\ [\![x \sharp C;\ x \neq u;\ x \neq a]\!] \Longrightarrow F\ C\ (a{<}x{>}.P)\ a\ u\ (P[x{::=}u])$
and $\bigwedge P\ a\ u\ P'\ b\ C.\ [\![P \longmapsto a{<}u{>} \prec P';\ \bigwedge C.\ F\ C\ P\ a\ u\ P']\!] \Longrightarrow F\ C\ ([b{\frown}b]P)$

$a\ u\ P'$
and $\bigwedge P\ a\ u\ P'\ b\ c\ C.\ [\![P \longmapsto a{<}u{>} \prec P';\ \bigwedge C.\ F\ C\ P\ a\ u\ P';\ b{\neq}c]\!] \Longrightarrow F\ C\ ([b{\neq}c]P)\ a\ u\ P'$
and $\bigwedge P\ a\ u\ P'\ Q\ C.\ [\![P \longmapsto a{<}u{>} \prec P';\ \bigwedge C.\ F\ C\ P\ a\ u\ P']\!] \Longrightarrow F\ C\ (P \oplus Q)\ a\ u\ P'$
and $\bigwedge Q\ a\ u\ Q'\ P\ C.\ [\![Q \longmapsto a{<}u{>} \prec Q';\ \bigwedge C.\ F\ C\ Q\ a\ u\ Q']\!] \Longrightarrow F\ C\ (P \oplus Q)\ a\ u\ Q'$
and $\bigwedge P\ a\ u\ P'\ Q\ C.\ [\![P \longmapsto a{<}u{>} \prec P';\ \bigwedge C.\ F\ C\ P\ a\ u\ P']\!] \Longrightarrow F\ C\ (P \parallel Q)\ a\ u\ (P' \parallel Q)$
and $\bigwedge Q\ a\ u\ Q'\ P\ C.\ [\![Q \longmapsto a{<}u{>} \prec Q';\ \bigwedge C.\ F\ C\ Q\ a\ u\ Q']\!] \Longrightarrow F\ C\ (P \parallel Q)\ a\ u\ (P \parallel Q')$
and $\bigwedge P\ a\ u\ P'\ x\ C.\ [\![P \longmapsto a{<}u{>} \prec P';\ x \neq a;\ x \neq u;\ x \sharp C;\ \bigwedge C.\ F\ C\ P\ a\ u\ P']\!] \Longrightarrow F\ C\ ({<}\nu x{>}P)\ a\ u\ ({<}\nu x{>}P')$
and $\bigwedge P\ a\ u\ P'\ C.\ [\![P \parallel !P \longmapsto a{<}u{>} \prec P';\ \bigwedge C.\ F\ C\ (P \parallel !P)\ a\ u\ P']\!] \Longrightarrow F\ C\ (!P)\ a\ u\ P'$

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

lemma *inputAlpha*:
assumes $P \longmapsto a{<}u{>} \prec P'$
and $u \sharp P$
and $r \sharp P'$

shows $P \longmapsto a{<}r{>} \prec ([(u, r)] \cdot P')$
$\langle proof \rangle$

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

assumes $P \longmapsto a{<}x{>} \prec P'$
and $Q \longmapsto a{<}\nu x{>} \prec Q'$
and $x \sharp P$

shows $P \parallel Q \longmapsto \tau \prec {<}\nu x{>}(P' \parallel Q')$
$\langle proof \rangle$

lemma *Close2*:
fixes P :: *pi*
and a :: *name*
and x :: *name*
and P' :: *pi*
and Q :: *pi*
and Q' :: *pi*

assumes $P \longmapsto a{<}\nu x{>} \prec P'$
and $Q \longmapsto a{<}x{>} \prec Q'$
and $x \sharp Q$

shows $P \parallel Q \longmapsto \tau \prec {<}\nu x{>}(P' \parallel Q')$
⟨*proof*⟩

lemma *ResB*:
fixes P :: *pi*
and a :: *name*
and x :: *name*
and P' :: *pi*
and y :: *name*

assumes $P \longmapsto a{<}\nu x{>} \prec P'$
and $y \neq a$
and $y \neq x$

shows ${<}\nu y{>}P \longmapsto a{<}\nu x{>} \prec ({<}\nu y{>}P')$
⟨*proof*⟩

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* ⇒ *pi* ⇒ *name* ⇒ *name* ⇒ *pi* ⇒ *bool*
and C :: *'a::fs-name*

assumes *Trans*: $P \longmapsto a[b] \prec P'$
and $\bigwedge a\ b\ P\ C.\ F\ C\ (a\{b\}.P)\ a\ b\ P$
and $\bigwedge P\ a\ b\ P'\ c\ C.\ [\![P \longmapsto OutputR\ a\ b \prec P';\ \bigwedge C.\ F\ C\ P\ a\ b\ P']\!] \Longrightarrow F\ C\ ([c{\frown}c]P)\ a\ b\ P'$
and $\bigwedge P\ a\ b\ P'\ c\ d\ C.\ [\![P \longmapsto OutputR\ a\ b \prec P';\ \bigwedge C.\ F\ C\ P\ a\ b\ P';\ c{\neq}d]\!] \Longrightarrow F\ C\ ([c{\neq}d]P)\ a\ b\ P'$
and $\bigwedge P\ a\ b\ P'\ Q\ C.\ [\![P \longmapsto OutputR\ a\ b \prec P';\ \bigwedge C.\ F\ C\ P\ a\ b\ P']\!] \Longrightarrow F\ C\ (P \oplus Q)\ a\ b\ P'$
and $\bigwedge Q\ a\ b\ Q'\ P\ C.\ [\![Q \longmapsto OutputR\ a\ b \prec Q';\ \bigwedge C.\ F\ C\ Q\ a\ b\ Q']\!] \Longrightarrow F\ C\ (P \oplus Q)\ a\ b\ Q'$
and $\bigwedge P\ a\ b\ P'\ Q\ C.\ [\![P \longmapsto OutputR\ a\ b \prec P';\ \bigwedge C.\ F\ C\ P\ a\ b\ P']\!] \Longrightarrow F\ C\ (P \parallel Q)\ a\ b\ (P' \parallel Q)$
and $\bigwedge Q\ a\ b\ Q'\ P\ C.\ [\![Q \longmapsto OutputR\ a\ b \prec Q';\ \bigwedge C.\ F\ C\ Q\ a\ b\ Q']\!] \Longrightarrow F\ C\ (P \parallel Q)\ a\ b\ (P \parallel Q')$
and $\bigwedge P\ a\ b\ P'\ x\ C.\ [\![P \longmapsto OutputR\ a\ b \prec P';\ x \neq a;\ x \neq b;\ x \sharp C;\ \bigwedge C.\ F\ C\ P\ a\ b\ P']\!] \Longrightarrow$
$F\ C\ ({<}\nu x{>}P)\ a\ b\ ({<}\nu x{>}P')$
and $\bigwedge P\ a\ b\ P'\ C.\ [\![P \parallel !P \longmapsto OutputR\ a\ b \prec P';\ \bigwedge C.\ F\ C\ (P \parallel !P)\ a\ b\ P']\!]$

$\Longrightarrow F\ C\ (!P)\ a\ b\ P'$

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

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

assumes *a*: $P \longmapsto a{<}\nu x{>} \prec P'$
and *xFreshP*: $x \sharp P$
and *cMatch*: $\bigwedge P\ a\ x\ P'\ b\ C.\ [\![P \longmapsto a{<}\nu x{>} \prec P';\ \bigwedge C.\ F\ C\ P\ a\ x\ P']\!] \Longrightarrow F\ C\ ([b{\frown}b]P)\ a\ x\ P'$
and *cMismatch*: $\bigwedge P\ a\ x\ P'\ b\ c\ C.\ [\![P \longmapsto a{<}\nu x{>} \prec P';\ \bigwedge C.\ F\ C\ P\ a\ x\ P';\ b \neq c]\!] \Longrightarrow F\ C\ ([b{\neq}c]P)\ a\ x\ P'$
and *cOpen*: $\bigwedge P\ a\ x\ P'\ C.\ [\![P \longmapsto (OutputR\ a\ x) \prec P';\ a \neq x]\!] \Longrightarrow F\ C\ ({<}\nu x{>}P)\ a\ x\ P'$
and *cSum1*: $\bigwedge P\ Q\ a\ x\ P'\ C.\ [\![P \longmapsto a{<}\nu x{>} \prec P';\ \bigwedge C.\ F\ C\ P\ a\ x\ P']\!] \Longrightarrow F\ C\ (P \oplus Q)\ a\ x\ P'$
and *cSum2*: $\bigwedge P\ Q\ a\ x\ Q'\ C.\ [\![Q \longmapsto a{<}\nu x{>} \prec Q';\ \bigwedge C.\ F\ C\ Q\ a\ x\ Q']\!] \Longrightarrow F\ C\ (P \oplus Q)\ a\ x\ Q'$
and *cPar1B*: $\bigwedge P\ P'\ Q\ a\ x\ C.\ [\![P \longmapsto a{<}\nu x{>} \prec P';\ x \sharp Q;\ \bigwedge C.\ F\ C\ P\ a\ x\ P']\!] \Longrightarrow$
$F\ C\ (P \parallel Q)\ a\ x\ (P' \parallel Q)$
and *cPar2B*: $\bigwedge P\ Q\ Q'\ a\ x\ C.\ [\![Q \longmapsto a{<}\nu x{>} \prec Q';\ x \sharp P;\ \bigwedge C.\ F\ C\ Q\ a\ x\ Q']\!] \Longrightarrow$
$F\ C\ (P \parallel Q)\ a\ x\ (P \parallel Q')$
and *cResB*: $\bigwedge P\ P'\ a\ x\ y\ C.\ [\![P \longmapsto a{<}\nu x{>} \prec P';\ y \neq a;\ y \neq x;\ y \sharp C;$
$\bigwedge C.\ F\ C\ P\ a\ x\ P']\!] \Longrightarrow F\ C\ ({<}\nu y{>}P)\ a\ x\ ({<}\nu y{>}P')$
and *cBang*: $\bigwedge P\ a\ x\ P'\ C.\ [\![P \parallel !P \longmapsto a{<}\nu x{>} \prec P';\ \bigwedge C.\ F\ C\ (P \parallel !P)\ a\ x\ P']\!] \Longrightarrow$
$F\ C\ (!P)\ a\ x\ P'$
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*: $P \longmapsto \tau \prec P'$
and $\bigwedge P\ C.\ F\ C\ (\tau.(P))\ P$

and $\bigwedge P\ P'\ a\ C.\ [\![P \longmapsto\tau \prec P';\ \bigwedge C.\ F\ C\ P\ P']\!] \Longrightarrow F\ C\ ([a\frown a]P)\ P'$
and $\bigwedge P\ P'\ a\ b\ C.\ [\![P \longmapsto\tau \prec P';\ \bigwedge C.\ F\ C\ P\ P';\ a \neq b]\!] \Longrightarrow F\ C\ ([a{\neq}b]P)\ P'$
and $\bigwedge P\ P'\ Q\ C.\ [\![P \longmapsto\tau \prec P';\ \bigwedge C.\ F\ C\ P\ P']\!] \Longrightarrow F\ C\ (P \oplus Q)\ P'$
and $\bigwedge Q\ Q'\ P\ C.\ [\![Q \longmapsto\tau \prec Q';\ \bigwedge C.\ F\ C\ Q\ Q']\!] \Longrightarrow F\ C\ (P \oplus Q)\ Q'$
and $\bigwedge P\ P'\ Q\ C.\ [\![P \longmapsto\tau \prec P';\ \bigwedge C.\ F\ C\ P\ P']\!] \Longrightarrow F\ C\ (P \parallel Q)\ (P' \parallel Q)$
and $\bigwedge Q\ Q'\ P\ C.\ [\![Q \longmapsto\tau \prec Q';\ \bigwedge C.\ F\ C\ Q\ Q']\!] \Longrightarrow F\ C\ (P \parallel Q)\ (P \parallel Q')$
and $\bigwedge P\ a\ b\ P'\ Q\ Q'\ C.\ [\![P \longmapsto a{<}b{>} \prec P';\ Q \longmapsto \mathit{OutputR}\ a\ b \prec Q']\!] \Longrightarrow F\ C\ (P \parallel Q)\ (P' \parallel Q')$
and $\bigwedge P\ a\ b\ P'\ Q\ Q'\ C.\ [\![P \longmapsto \mathit{OutputR}\ a\ b \prec P';\ Q \longmapsto a{<}b{>} \prec Q']\!] \Longrightarrow F\ C\ (P \parallel Q)\ (P' \parallel Q')$
and $\bigwedge P\ a\ x\ P'\ Q\ Q'\ C.\ [\![P \longmapsto a{<}x{>} \prec P';\ Q \longmapsto a{<}\nu x{>} \prec Q';\ x \sharp P;\ x \sharp Q;\ x \neq a;\ x \sharp C]\!] \Longrightarrow F\ C\ (P \parallel Q)\ ({<}\nu x{>}(P' \parallel Q'))$
and $\bigwedge P\ a\ x\ P'\ Q\ Q'\ C.\ [\![P \longmapsto a{<}\nu x{>} \prec P';\ Q \longmapsto a{<}x{>} \prec Q';\ x \sharp P;\ x \sharp Q;\ x \neq a;\ x \sharp C]\!] \Longrightarrow F\ C\ (P \parallel Q)\ ({<}\nu x{>}(P' \parallel Q'))$
and $\bigwedge P\ P'\ x\ C.\ [\![P \longmapsto\tau \prec P';\ x \sharp C;\ \bigwedge C.\ F\ C\ P\ P']\!] \Longrightarrow$
$F\ C\ ({<}\nu x{>}P)\ ({<}\nu x{>}P')$
and $\bigwedge P\ P'\ C.\ [\![P \parallel !P \longmapsto\tau \prec P';\ \bigwedge C.\ F\ C\ (P \parallel !P)\ P']\!] \Longrightarrow F\ C\ (!P)\ P'$

shows $F\ C\ P\ P'$
⟨*proof*⟩

inductive *bangPred* :: *pi* ⇒ *pi* ⇒ *bool*
where
aux1: *bangPred* $P\ (!P)$
| *aux2*: *bangPred* $P\ (P \parallel !P)$

inductive-cases *tauCases'*[*simplified pi.distinct residual.distinct*]: $\tau.(P) \longmapsto Rs$
inductive-cases *inputCases'*[*simplified pi.inject residual.inject*]: $a{<}b{>}.P \longmapsto Rs$
inductive-cases *outputCases'*[*simplified pi.inject residual.inject*]: $a\{b\}.P \longmapsto Rs$
inductive-cases *matchCases'*[*simplified pi.inject residual.inject*]: $[a\frown b]P \longmapsto Rs$
inductive-cases *mismatchCases'*[*simplified pi.inject residual.inject*]: $[a{\neq}b]P \longmapsto Rs$
inductive-cases *sumCases'*[*simplified pi.inject residual.inject*]: $P \oplus Q \longmapsto Rs$
inductive-cases *parCasesB'*[*simplified pi.distinct residual.distinct*]: $A \parallel B \longmapsto b{<}\nu y{>} \prec A'$
inductive-cases *parCasesF'*[*simplified pi.distinct residual.distinct*]: $P \parallel Q \longmapsto \alpha \prec P'$
inductive-cases *resCasesB'*[*simplified pi.distinct residual.distinct*]: ${<}\nu x'{>}A \longmapsto a{<}\nu y'{>} \prec A'$
inductive-cases *resCasesF'*[*simplified pi.distinct residual.distinct*]: ${<}\nu x{>}A \longmapsto \alpha \prec A'$

lemma *tauCases*:
fixes P :: *pi*
and α :: *freeRes*
and P' :: *pi*

assumes $\tau.(P) \longmapsto \alpha \prec P'$

and *Prop* (τ) *P*

shows *Prop* α P'
⟨*proof*⟩

lemma *inputCases*[*consumes 1*, *case-names cInput*]:
fixes *a* :: *name*
and *x* :: *name*
and *P* :: *pi*
and P' :: *pi*

assumes *Input*: *a*<*x*>.*P* $\longmapsto\alpha \prec P'$
and *A*: $\bigwedge u$. *Prop* (*a*<*u*>) (*P*[*x*::=*u*])

shows *Prop* α P'
⟨*proof*⟩

lemma *outputCases*:
fixes *P* :: *pi*
and α :: *freeRes*
and P' :: *pi*

assumes *a*{*b*}.*P* $\longmapsto\alpha \prec P'$
and *Prop* (*OutputR a b*) *P*

shows *Prop* α P'
⟨*proof*⟩

lemma *zeroTrans*[*dest*]:
fixes *Rs* :: *residual*

assumes **0** $\longmapsto$ *e Rs*

shows *False*
⟨*proof*⟩

lemma *mismatchTrans*[*dest*]:
fixes *a* :: *name*
and *P* :: *pi*
and *Rs* :: *residual*

assumes [*a*≠*a*]*P* $\longmapsto$ *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* ⇒ *name* ⇒ *bool*

assumes *Trans*: $[a \frown b]P \longmapsto Rs$
and *cMatch*: $P \longmapsto Rs \Longrightarrow F\ a\ a$

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* ⇒ *name* ⇒ *bool*

assumes *Trans*: $[a \neq b]P \longmapsto Rs$
and *cMatch*: $[\![P \longmapsto Rs;\ a \neq b]\!] \Longrightarrow F\ a\ b$

shows $F\ a\ b$
⟨*proof*⟩

lemma *sumCases*[*consumes 1*, *case-names Sum1 Sum2*]:
fixes P :: *pi*
and Q :: *pi*
and Rs :: *residual*

assumes *Trans*: $P \oplus Q \longmapsto Rs$
and *cSum1*: $P \longmapsto Rs \Longrightarrow F$
and *cSum2*: $Q \longmapsto Rs \Longrightarrow F$

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*: $P \parallel Q \longmapsto a{<}\nu x{>} \prec PQ'$
and *icPar1B*: $\bigwedge P'.\ [\![P \longmapsto a{<}\nu x{>} \prec P';\ x \sharp Q]\!] \Longrightarrow F\ (P' \parallel Q)$
and *icPar2B*: $\bigwedge Q'.\ [\![Q \longmapsto a{<}\nu x{>} \prec Q';\ x \sharp P]\!] \Longrightarrow F\ (P \parallel Q')$

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 $P \parallel Q \longmapsto a[b] \prec PQ'$
and $\bigwedge P'. \llbracket P \longmapsto a[b] \prec P' \rrbracket \Longrightarrow F\ (P' \parallel Q)$
and $\bigwedge Q'. \llbracket Q \longmapsto a[b] \prec Q' \rrbracket \Longrightarrow F\ (P \parallel Q')$

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*: $P \parallel Q \longmapsto a{<}b{>} \prec PQ'$
and *icPar1F*: $\bigwedge P'. \llbracket P \longmapsto a{<}b{>} \prec P' \rrbracket \Longrightarrow F\ (P' \parallel Q)$
and *icPar2F*: $\bigwedge Q'. \llbracket Q \longmapsto a{<}b{>} \prec Q' \rrbracket \Longrightarrow F\ (P \parallel Q')$

shows $F\ PQ'$
⟨*proof*⟩

lemma *parCasesF*[*consumes 1*, *case-names cPar1 cPar2 cComm1 cComm2 cClose1 cClose2*]:
fixes P :: *pi*
and Q :: *pi*
and α :: *freeRes*
and P' :: *pi*
and C :: *'a::fs-name*

assumes *Trans*: $P \parallel Q \longmapsto \alpha \prec PQ'$
and *icPar1F*: $\bigwedge P'. \llbracket P \longmapsto \alpha \prec P' \rrbracket \Longrightarrow F\ \alpha\ (P' \parallel Q)$
and *icPar2F*: $\bigwedge Q'. \llbracket Q \longmapsto \alpha \prec Q' \rrbracket \Longrightarrow F\ \alpha\ (P \parallel Q')$
and *icComm1*: $\bigwedge P'\ Q'\ a\ b. \llbracket P \longmapsto a{<}b{>} \prec P';\ Q \longmapsto a[b] \prec Q' \rrbracket \Longrightarrow F\ (\tau)\ (P' \parallel Q')$
and *icComm2*: $\bigwedge P'\ Q'\ a\ b. \llbracket P \longmapsto a[b] \prec P';\ Q \longmapsto a{<}b{>} \prec Q' \rrbracket \Longrightarrow F\ (\tau)\ (P' \parallel Q')$
and *icClose1*: $\bigwedge P'\ Q'\ a\ x. \llbracket P \longmapsto a{<}x{>} \prec P';\ Q \longmapsto a{<}\nu x{>} \prec Q';\ x \sharp P;\ x \sharp C \rrbracket \Longrightarrow F\ (\tau)\ ({<}\nu x{>}(P' \parallel Q'))$
and *icClose2*: $\bigwedge P'\ Q'\ a\ x. \llbracket P \longmapsto a{<}\nu x{>} \prec P';\ Q \longmapsto a{<}x{>} \prec Q';\ x \sharp Q;\ x \sharp C \rrbracket \Longrightarrow F\ (\tau)\ ({<}\nu x{>}(P' \parallel Q'))$

shows $F\ \alpha\ PQ'$

⟨*proof*⟩

lemma *resCasesF*[*consumes 2*, *case-names Res*]:
fixes x :: *name*
and P :: *pi*
and α :: *freeRes*
and P' :: *pi*

assumes *Trans*: $<\nu x>P \longmapsto \alpha \prec RP'$
and *xFreshAlpha*: $x \sharp \alpha$
and *rcResF*: $\bigwedge P'.\ P \longmapsto \alpha \prec P' \Longrightarrow F\ (<\nu x>P')$

shows $F\ RP'$
⟨*proof*⟩

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*: $<\nu y>P \longmapsto a<\nu x> \prec RP'$
and *xineqy*: $x \neq y$
and *rcOpen*: $\bigwedge P'.\ [\![P \longmapsto (OutputR\ a\ y) \prec P';\ a \neq y]\!] \Longrightarrow F\ ([(x, y)] \cdot P')$
and *rcResB*: $\bigwedge P'.\ [\![P \longmapsto a<\nu x> \prec P';\ y \neq a]\!] \Longrightarrow F\ (<\nu y>P')$

shows $F\ RP'$
⟨*proof*⟩

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*: $!P \longmapsto Rs$
and *cPar1B*: $\bigwedge a\ x\ P'\ C.\ [\![P \longmapsto a<\nu x> \prec P';\ x \sharp P;\ x \sharp C]\!] \Longrightarrow F\ C\ (P \parallel !P)\ (a<\nu x> \prec (P' \parallel !P))$
and *cPar1F*: $\bigwedge(\alpha::freeRes)\ (P'::pi)\ C.\ [\![P \longmapsto \alpha \prec P']\!] \Longrightarrow F\ C\ (P \parallel !P)\ (\alpha \prec P' \parallel !P)$
and *cPar2B*: $\bigwedge a\ x\ P'\ C.\ [\![!P \longmapsto a<\nu x> \prec P';\ x \sharp P;\ x \sharp C;\ \bigwedge C.\ F\ C\ (!P)\ (a<\nu x> \prec P')]\!] \Longrightarrow F\ C\ (P \parallel !P)\ (a<\nu x> \prec (P \parallel P'))$
and *cPar2F*: $\bigwedge \alpha\ P'\ C.\ [\![!P \longmapsto \alpha \prec P';\ \bigwedge C.\ F\ C\ (!P)\ (\alpha \prec P')]\!] \Longrightarrow F\ C\ (P \parallel !P)\ (\alpha \prec P \parallel P')$
and *cComm1*: $\bigwedge a\ P'\ b\ P''\ C.\ [\![P \longmapsto a<b> \prec P';\ !P \longmapsto (OutputR\ a\ b) \prec P'';\ \bigwedge C.\ F\ C\ (!P)\ ((OutputR\ a\ b) \prec P'')]\!] \Longrightarrow$
$F\ C\ (P \parallel !P)\ (\tau \prec P' \parallel P'')$

and *cComm2*: $\bigwedge a\ b\ P'\ P''\ C.\ [\![P \longmapsto (OutputR\ a\ b) \prec P';\ !P \longmapsto a{<}b{>} \prec P'';\ \bigwedge C.\ F\ C\ (!P)\ (a{<}b{>} \prec P'')]\!] \Longrightarrow$
$F\ C\ (P \parallel !P)\ (\tau \prec P' \parallel P'')$
and *cClose1*: $\bigwedge a\ x\ P'\ P''\ C.\ [\![P \longmapsto a{<}x{>} \prec P';\ !P \longmapsto a{<}\nu x{>} \prec P'';\ x \sharp P;\ x \sharp C;\ \bigwedge C.\ F\ C\ (!P)\ (a{<}\nu x{>} \prec P'')]\!] \Longrightarrow$
$F\ C\ (P \parallel !P)\ (\tau \prec {<}\nu x{>}(P' \parallel P''))$
and *cClose2*: $\bigwedge a\ x\ P'\ P''\ C.\ [\![P \longmapsto a{<}\nu x{>} \prec P';\ !P \longmapsto a{<}x{>} \prec P'';\ x \sharp P;\ x \sharp C;\ \bigwedge C.\ F\ C\ (!P)\ (a{<}x{>} \prec P'')]\!] \Longrightarrow$
$F\ C\ (P \parallel !P)\ (\tau \prec {<}\nu x{>}(P' \parallel P''))$
and *cBang*: $\bigwedge Rs\ C.\ [\![P \parallel !P \longmapsto Rs;\ \bigwedge C.\ F\ C\ (P \parallel !P)\ Rs]\!] \Longrightarrow F\ C\ (!P)\ Rs$

shows $F\ C\ (!P)\ Rs$
⟨*proof*⟩

end

theory *Strong-Early-Sim*
imports *Early-Semantics Rel*
begin

definition *strongSimEarly* :: $pi \Rightarrow (pi \times pi)\ set \Rightarrow pi \Rightarrow bool$ (- ⇝[-] - [*80*, *80*, *80*] *80*) **where**
$P \leadsto[Rel]\ Q \equiv (\forall a\ y\ Q'.\ Q \longmapsto a{<}\nu y{>} \prec Q' \longrightarrow y \sharp P \longrightarrow (\exists P'.\ P \longmapsto a{<}\nu y{>} \prec P' \wedge (P', Q') \in Rel)) \wedge$
$(\forall \alpha\ Q'.\ Q \longmapsto \alpha \prec Q' \longrightarrow (\exists P'.\ P \longmapsto \alpha \prec P' \wedge (P', Q') \in Rel))$

lemma *monotonic*:
fixes A :: $(pi \times pi)\ set$
and B :: $(pi \times pi)\ set$
and P :: pi
and P' :: pi

assumes $P \leadsto[A]\ P'$
and $A \subseteq B$

shows $P \leadsto[B]\ P'$
⟨*proof*⟩

lemma *freshUnit*[*simp*]:
fixes y :: *name*

shows $y \sharp ()$
⟨*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*: $\bigwedge a\ y\ Q'.\ [\![Q \longmapsto a{<}\nu y{>} \prec Q';\ y \sharp P;\ y \sharp Q;\ y \sharp C]\!] \Longrightarrow \exists P'.\ P \longmapsto a{<}\nu y{>} \prec P' \wedge (P',\ Q') \in Rel$
and *Free*: $\bigwedge\alpha\ Q'.\ Q \longmapsto \alpha \prec Q' \Longrightarrow \exists P'.\ P \longmapsto \alpha \prec P' \wedge (P',\ Q') \in Rel$

shows $P \leadsto[Rel]\ Q$
⟨*proof*⟩

lemma *simCases*[*consumes 0*, *case-names Bound Free*]:
fixes P :: *pi*
and Q :: *pi*
and *Rel* :: (*pi* × *pi*) *set*
and C :: $'a$::*fs-name*

assumes *Bound*: $\bigwedge a\ y\ Q'.\ [\![Q \longmapsto a{<}\nu y{>} \prec Q';\ y \sharp P]\!] \Longrightarrow \exists P'.\ P \longmapsto a{<}\nu y{>} \prec P' \wedge (P',\ Q') \in Rel$
and *Free*: $\bigwedge\alpha\ Q'.\ Q \longmapsto \alpha \prec Q' \Longrightarrow \exists P'.\ P \longmapsto \alpha \prec P' \wedge (P',\ Q') \in Rel$

shows $P \leadsto[Rel]\ Q$
⟨*proof*⟩

lemma *elim*:
fixes P :: *pi*
and *Rel* :: (*pi* × *pi*) *set*
and Q :: *pi*
and a :: *name*
and x :: *name*
and Q' :: *pi*

assumes $P \leadsto[Rel]\ Q$

shows $Q \longmapsto a{<}\nu x{>} \prec Q' \Longrightarrow x \sharp P \Longrightarrow \exists P'.\ P \longmapsto a{<}\nu x{>} \prec P' \wedge (P',\ Q') \in Rel$
and $Q \longmapsto \alpha \prec Q' \Longrightarrow \exists P'.\ P \longmapsto \alpha \prec P' \wedge (P',\ Q') \in Rel$
⟨*proof*⟩

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

assumes *Sim*: $P \leadsto[Rel]\ Q$
and *RelRel'*: $Rel \subseteq Rel'$
and *EqvtRel'*: *eqvt Rel'*

shows $(perm \cdot P) \leadsto[Rel']\ (perm \cdot Q)$

⟨*proof*⟩

lemma *reflexive*:
fixes P :: *pi*
and *Rel* :: (*pi* × *pi*) *set*

assumes *Id* ⊆ *Rel*

shows $P \rightsquigarrow[Rel]\ P$
⟨*proof*⟩

lemmas *fresh-prod*[*simp*]

lemma *transitive*:
fixes P :: *pi*
and Q :: *pi*
and R :: *pi*
and *Rel* :: (*pi* × *pi*) *set*
and *Rel′* :: (*pi* × *pi*) *set*
and *Rel″* :: (*pi* × *pi*) *set*

assumes *PSimQ*: $P \rightsquigarrow[Rel]\ Q$
and *QSimR*: $Q \rightsquigarrow[Rel']\ R$
and *Eqvt′*: *eqvt Rel″*
and *Trans*: *Rel O Rel′* ⊆ *Rel″*

shows $P \rightsquigarrow[Rel'']\ R$
⟨*proof*⟩

end

theory *Strong-Early-Bisim*
imports *Strong-Early-Sim*
begin

lemma *monoAux*: $A \subseteq B \Longrightarrow P \rightsquigarrow[A]\ Q \longrightarrow P \rightsquigarrow[B]\ Q$
⟨*proof*⟩

coinductive-set *bisim* :: (*pi* × *pi*) *set*
where
step: $[\![P \rightsquigarrow[bisim]\ Q;\ (Q,\ P) \in bisim]\!] \Longrightarrow (P,\ Q) \in bisim$
monos *monoAux*

abbreviation *strongBisimJudge* (**infixr** ∼ *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*: $\bigwedge P\ Q.\ (P, Q) \in X \Longrightarrow P \rightsquigarrow[(X \cup bisim)]\ Q \wedge (Q, P) \in bisim \cup X$

  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 $\bigwedge R\ S.\ (R, S) \in X \Longrightarrow R \rightsquigarrow[(X \cup bisim)]\ S$
  and $\bigwedge R\ S.\ (R, S) \in X \Longrightarrow (S, R) \in X$

  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*: $\bigwedge P\ Q.\ (P, Q) \in X \Longrightarrow P \rightsquigarrow[X]\ Q \wedge (Q, P) \in X$

  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 $\bigwedge P\ Q.\ (P, Q) \in X \Longrightarrow P \rightsquigarrow[X]\ Q$
  and $\bigwedge P\ Q.\ (P, Q) \in X \Longrightarrow (Q, P) \in X$

  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** $\sim^s$ *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 $P \sim^s Q$

shows $P[<s>] \sim^s Q[<s>]$
⟨*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*: $(P, Q) \in Rel$

shows $\tau.(P) \leadsto[Rel]\ \tau.(Q)$
⟨*proof*⟩

lemma *inputPres*:
fixes P :: *pi*
and x :: *name*
and Q :: *pi*
and a :: *name*
and *Rel* :: (*pi* × *pi*) *set*

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

shows $a<x>.P \leadsto[Rel]\ a<x>.Q$
⟨*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*: $(P, Q) \in Rel$

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

lemma *matchPres*:

fixes P :: *pi*
and Q :: *pi*
and a :: *name*
and b :: *name*
and *Rel* :: (*pi* × *pi*) *set*
and *Rel'* :: (*pi* × *pi*) *set*

assumes *PSimQ*: $P \rightsquigarrow[Rel]\ Q$
and *RelRel'*: $Rel \subseteq Rel'$
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* × *pi*) *set*
and *Rel'* :: (*pi* × *pi*) *set*

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

shows $[a \neq b]P \rightsquigarrow[Rel']\ [a \neq b]Q$
⟨*proof*⟩

lemma *sumPres*:
fixes P :: *pi*
and Q :: *pi*
and R :: *pi*
and *Rel* :: (*pi* × *pi*) *set*
and *Rel'* :: (*pi* × *pi*) *set*

assumes $P \rightsquigarrow[Rel]\ Q$
and *C1*: $Id \subseteq Rel'$
and $Rel \subseteq Rel'$

shows $P \oplus R \rightsquigarrow[Rel']\ Q \oplus R$
⟨*proof*⟩

lemma *parCompose*:
fixes P :: *pi*
and Q :: *pi*
and R :: *pi*
and T :: *pi*
and *Rel* :: (*pi* × *pi*) *set*
and *Rel'* :: (*pi* × *pi*) *set*
and *Rel''* :: (*pi* × *pi*) *set*

assumes *PSimQ*: $P \leadsto[Rel]\ Q$
and *RSimT*: $R \leadsto[Rel']\ S$
and *PRelQ*: $(P,\ Q) \in Rel$
and *RRel'T*: $(R,\ S) \in Rel'$
and *Par*: $\bigwedge P'\ Q'\ R'\ S'.\ [\![(P',\ Q') \in Rel;\ (R',\ S') \in Rel']\!] \Longrightarrow (P' \parallel R',$
$Q' \parallel S') \in Rel''$
and *Res*: $\bigwedge S\ T\ x.\ (S,\ T) \in Rel'' \Longrightarrow (<\nu x>S,\ <\nu x>T) \in Rel''$

shows $P \parallel R \leadsto[Rel'']\ Q \parallel S$
⟨*proof*⟩

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

assumes *PSimQ*: $P \leadsto[Rel]\ Q$
and *PRelQ*: $(P,\ Q) \in Rel$
and *Par*: $\bigwedge S\ T\ U.\ (S,\ T) \in Rel \Longrightarrow (S \parallel U,\ T \parallel U) \in Rel'$
and *Res*: $\bigwedge S\ T\ x.\ (S,\ T) \in Rel' \Longrightarrow (<\nu x>S,\ <\nu x>T) \in Rel'$

shows $P \parallel R \leadsto[Rel']\ Q \parallel R$
⟨*proof*⟩

lemma *resPres*:
fixes P :: *pi*
and Q :: *pi*
and *Rel* :: (*pi* × *pi*) *set*
and x :: *name*
and *Rel'* :: (*pi* × *pi*) *set*

assumes *PSimQ*: $P \leadsto[Rel]\ Q$
and *ResSet*: $\bigwedge(R{::}pi)\ (S{::}pi)\ (y{::}name).\ (R,\ S) \in Rel \Longrightarrow (<\nu y>R,\ <\nu y>S)$
$\in Rel'$
and *RelRel'*: $Rel \subseteq Rel'$
and *EqvtRel*: *eqvt Rel*
and *EqvtRel'*: *eqvt Rel'*

shows $<\nu x>P \leadsto[Rel']\ <\nu x>Q$
⟨*proof*⟩

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

and *lst* :: *name list*

assumes *eqvtRel*: *eqvt Rel*
and *Res*: $\bigwedge R\ S\ x.\ (R,\ S) \in Rel \Longrightarrow (<\nu x>R,\ <\nu x>S) \in Rel$
and *PRelQ*: $P \leadsto[Rel]\ Q$

shows $(resChain\ lst)\ P \leadsto[Rel]\ (resChain\ lst)\ Q$
⟨*proof*⟩

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

assumes *PRelQ*: $(P,\ Q) \in Rel$
and *Sim*: $\bigwedge R\ S.\ (R,\ S) \in Rel \Longrightarrow R \leadsto[Rel]\ S$
and *eqvtRel*: *eqvt Rel*

shows $!P \leadsto[bangRel\ Rel]\ !Q$
⟨*proof*⟩

end

theory *Strong-Early-Bisim-Pres*
imports *Strong-Early-Bisim Strong-Early-Sim-Pres*
begin

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

assumes $P \sim Q$

shows $\tau.(P) \sim \tau.(Q)$
⟨*proof*⟩

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 $a<x>.P \sim a<x>.Q$
⟨*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 $[a\frown b]P \sim [a\frown b]Q$
⟨*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 $<\nu x>P \sim <\nu x>Q$
⟨*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 $Rel :: (pi \times pi)\ set$

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

shows $(Q, P) \in bangRel\ Rel$
⟨*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 $a{<}x{>}.P \sim^s a{<}x{>}.Q$
⟨*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 $P \oplus R \sim^s Q \oplus R$
$\langle proof \rangle$

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

lemma *bangPres*:
fixes P :: *pi*
and Q :: *pi*

assumes $P \sim^s Q$

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

end

theory *Early-Tau-Chain*
imports *Early-Semantics*
begin

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

lemma *tauActTauChain*:
fixes P :: *pi*
and P' :: *pi*

assumes $P \longmapsto \tau \prec P'$

shows $P \Longrightarrow_\tau P'$

⟨*proof*⟩

lemma *tauChainAddTau*[*intro*]:
fixes P :: *pi*
and P' :: *pi*
and P'' :: *pi*

shows $P \Longrightarrow_\tau P' \Longrightarrow P' \longmapsto \tau \prec P'' \Longrightarrow P \Longrightarrow_\tau P''$
and $P \longmapsto \tau \prec P' \Longrightarrow P' \Longrightarrow_\tau P'' \Longrightarrow P \Longrightarrow_\tau P''$
⟨*proof*⟩

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

assumes $P \Longrightarrow_\tau P'$
and $F\ P$
and $\bigwedge P''\ P'''.\ [\![P \Longrightarrow_\tau P''; P'' \longmapsto \tau \prec P'''; F\ P'']\!] \Longrightarrow F\ P'''$

shows $F\ P'$
⟨*proof*⟩

lemma *eqvtChainI*:
fixes P :: *pi*
and P' :: *pi*
and *perm* :: *name prm*

assumes $P \Longrightarrow_\tau P'$

shows $(perm \cdot P) \Longrightarrow_\tau (perm \cdot P')$
⟨*proof*⟩

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

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

shows $P \Longrightarrow_\tau P'$
⟨*proof*⟩

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

shows $P \Longrightarrow_\tau P' = (perm \cdot P) \Longrightarrow_\tau (perm \cdot P')$
⟨*proof*⟩

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

assumes $P \Longrightarrow_\tau P'$
and $x \sharp P$

shows $x \sharp P'$
⟨*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 *aineqb*: $a \neq b$
and *PineqP'*: $P \neq P'$

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 $<\nu a>P \Longrightarrow_\tau <\nu a>P'$

⟨*proof*⟩

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-Early-Step-Semantics*
  imports *Early-Tau-Chain*
begin

lemma *inputSupportDerivative*:
  assumes $P \longmapsto a{<}x{>} \prec P'$

  shows $(supp\ P') - \{x\} \subseteq supp\ P$
⟨*proof*⟩

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

  assumes $P \longmapsto a[b] \prec P'$

  shows $(supp\ P') \subseteq ((supp\ P)::name\ set)$
⟨*proof*⟩

lemma *boundOutputSupportDerivative*:
  assumes $P \longmapsto a{<}\nu x{>} \prec P'$
  and $x \sharp P$

shows $(supp\ P') - \{x\} \subseteq supp\ P$
$\langle proof \rangle$

lemma *tauSupportDerivative*:

assumes $P \longmapsto \tau \prec P'$

shows $((supp\ P')::name\ set) \subseteq supp\ P$
$\langle proof \rangle$

lemma *tauChainSupportDerivative*:
fixes P :: *pi*
and P' :: *pi*

assumes $P \Longrightarrow_\tau P'$

shows $((supp\ P')::name\ set) \subseteq (supp\ P)$
$\langle proof \rangle$

definition *outputTransition* :: $pi \Rightarrow name \Rightarrow name \Rightarrow pi \Rightarrow bool$ (- $\Longrightarrow$-<ν-> $\prec$ - [80, 80, 80, 80] 80)
where $P \Longrightarrow a{<}\nu x{>} \prec P' \equiv \exists P''' P''.\ P \Longrightarrow_\tau P''' \wedge P''' \longmapsto a{<}\nu x{>} \prec P'' \wedge P'' \Longrightarrow_\tau P'$

definition *freeTransition* :: $pi \Rightarrow freeRes \Rightarrow pi \Rightarrow bool$ (- $\Longrightarrow$- $\prec$ - [80, 80, 80] 80)
where $P \Longrightarrow \alpha \prec P' \equiv \exists P''' P''.\ P \Longrightarrow_\tau P''' \wedge P''' \longmapsto \alpha \prec P'' \wedge P'' \Longrightarrow_\tau P'$

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

shows $[\![P \Longrightarrow_\tau P'''; P''' \longmapsto \alpha \prec P''; P'' \Longrightarrow_\tau P']\!] \Longrightarrow P \Longrightarrow \alpha \prec P'$
and $[\![P \Longrightarrow_\tau P'''; P''' \longmapsto a{<}\nu x{>} \prec P''; P'' \Longrightarrow_\tau P']\!] \Longrightarrow P \Longrightarrow a{<}\nu x{>} \prec P'$
$\langle proof \rangle$

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

shows $P \Longrightarrow \alpha \prec P' \Longrightarrow (\exists P''\ P'''.\ P \Longrightarrow_\tau P'' \wedge P'' \longmapsto \alpha \prec P''' \wedge P''' \Longrightarrow_\tau$

P')
and $P \Longrightarrow a{<}\nu x{>} \prec P' \Longrightarrow \exists P''\, P'''.\ P \Longrightarrow_\tau P''' \wedge P''' \longmapsto a{<}\nu x{>} \prec P'' \wedge P'' \Longrightarrow_\tau P'$
⟨*proof*⟩

lemma *weakTransitionAlpha*:
fixes P :: *pi*
and a :: *name*
and x :: *name*
and P' :: *pi*
and y :: *name*

assumes *PTrans*: $P \Longrightarrow a{<}\nu x{>} \prec P'$
and $y \sharp P$

shows $P \Longrightarrow a{<}\nu y{>} \prec ([(x, y)] \cdot P')$
⟨*proof*⟩

lemma *singleActionChain*:
fixes P :: *pi*
and Rs :: *residual*

shows $P \longmapsto a{<}\nu x{>} \prec P' \Longrightarrow P \Longrightarrow a{<}\nu x{>} \prec P'$
and $P \longmapsto \alpha \prec P' \Longrightarrow P \Longrightarrow \alpha \prec P'$
⟨*proof*⟩

lemma *Tau*:
fixes P :: *pi*

shows $\tau.(P) \Longrightarrow \tau \prec P$
⟨*proof*⟩

lemma *Input*:
fixes a :: *name*
and x :: *name*
and u :: *name*
and P :: *pi*

shows $a{<}x{>}.P \Longrightarrow a{<}u{>} \prec P[x{::=}u]$
⟨*proof*⟩

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 α :: *freeRes*

shows $P \Longrightarrow b{<}\nu x{>} \prec P' \Longrightarrow [a\frown a]P \Longrightarrow b{<}\nu x{>} \prec P'$
and $P \Longrightarrow \alpha \prec P' \Longrightarrow [a\frown a]P \Longrightarrow \alpha \prec P'$
⟨*proof*⟩

lemma *Mismatch*:
fixes P :: *pi*
and c :: *name*
and x :: *name*
and a :: *name*
and b :: *name*
and P' :: *pi*
and α :: *freeRes*

shows $[\![P \Longrightarrow c{<}\nu x{>} \prec P'; a \neq b]\!] \Longrightarrow [a{\neq}b]P \Longrightarrow c{<}\nu x{>} \prec P'$
and $[\![P \Longrightarrow \alpha \prec P'; a \neq b]\!] \Longrightarrow [a{\neq}b]P \Longrightarrow \alpha \prec P'$
⟨*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 ${<}\nu b{>}P \Longrightarrow a{<}\nu b{>} \prec P'$
⟨*proof*⟩

lemma *Sum1*:
fixes P :: *pi*
and a :: *name*
and x :: *name*
and P' :: *pi*
and Q :: *pi*
and α :: *freeRes*

shows $P \Longrightarrow a{<}\nu x{>} \prec P' \Longrightarrow P \oplus Q \Longrightarrow a{<}\nu x{>} \prec P'$
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 α :: *freeRes*

shows $Q \Longrightarrow a{<}\nu x{>} \prec Q' \Longrightarrow P \oplus Q \Longrightarrow a{<}\nu x{>} \prec Q'$
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*: $P \Longrightarrow a{<}\nu x{>} \prec P'$
and $x \sharp Q$

shows $P \parallel Q \Longrightarrow a{<}\nu x{>} \prec (P' \parallel Q)$
⟨*proof*⟩

lemma *Par1F*:
fixes P :: *pi*
and α :: *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*: $Q \Longrightarrow a{<}\nu x{>} \prec Q'$
and $x \sharp P$

shows $P \parallel Q \Longrightarrow a{<}\nu x{>} \prec (P \parallel Q')$
⟨*proof*⟩

lemma *Par2F*:
fixes Q :: *pi*
and α :: *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*: $P \Longrightarrow a<b> \prec P'$
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*: $Q \Longrightarrow a<b> \prec Q'$

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*: $P \Longrightarrow a<x> \prec P'$
and *QTrans*: $Q \Longrightarrow a<\nu x> \prec Q'$

and $x \sharp P$

shows $P \parallel Q \Longrightarrow\tau \prec <\nu x>(P' \parallel Q')$
$\langle proof \rangle$

lemma *Close2*:
fixes P :: *pi*
and a :: *name*
and x :: *name*
and P' :: *pi*
and Q :: *pi*
and Q' :: *pi*

assumes *PTrans*: $P \Longrightarrow a<\nu x> \prec P'$
and *QTrans*: $Q \Longrightarrow a<x> \prec Q'$
and *xFreshQ*: $x \sharp Q$

shows $P \parallel Q \Longrightarrow\tau \prec <\nu x>(P' \parallel Q')$
$\langle proof \rangle$

lemma *ResF*:
fixes P :: *pi*
and α :: *freeRes*
and P' :: *pi*
and x :: *name*

assumes *PTrans*: $P \Longrightarrow\alpha \prec P'$
and $x \sharp \alpha$

shows $<\nu x>P \Longrightarrow\alpha \prec <\nu x>P'$
$\langle proof \rangle$

lemma *ResB*:
fixes P :: *pi*
and a :: *name*
and x :: *name*
and P' :: *pi*
and y :: *name*

assumes *PTrans*: $P \Longrightarrow a<\nu x> \prec P'$
and $y \neq a$
and $y \neq x$

shows $<\nu y>P \Longrightarrow a<\nu x> \prec (<\nu y>P')$
$\langle proof \rangle$

lemma *Bang*:
fixes P :: *pi*
and Rs :: *residual*

shows $P \parallel !P \Longrightarrow a{<}\nu x{>} \prec P' \Longrightarrow !P \Longrightarrow a{<}\nu x{>} \prec P'$
and $P \parallel !P \Longrightarrow \alpha \prec P' \Longrightarrow !P \Longrightarrow \alpha \prec P'$
⟨*proof*⟩

lemma *tauTransitionChain*:
fixes P :: *pi*
and P' :: *pi*

assumes $P \Longrightarrow \tau \prec P'$

shows $P \Longrightarrow_\tau P'$
⟨*proof*⟩

lemma *chainTransitionAppend*:
fixes P :: *pi*
and P' :: *pi*
and Rs :: *residual*
and a :: *name*
and x :: *name*
and P'' :: *pi*
and α :: *freeRes*

shows $P \Longrightarrow a{<}\nu x{>} \prec P'' \Longrightarrow P'' \Longrightarrow_\tau P' \Longrightarrow P \Longrightarrow a{<}\nu x{>} \prec P'$
and $P \Longrightarrow \alpha \prec P'' \Longrightarrow P'' \Longrightarrow_\tau P' \Longrightarrow P \Longrightarrow \alpha \prec P'$
and $P \Longrightarrow_\tau P'' \Longrightarrow P'' \Longrightarrow a{<}\nu x{>} \prec P' \Longrightarrow P \Longrightarrow a{<}\nu x{>} \prec P'$
and $P \Longrightarrow_\tau P'' \Longrightarrow P'' \Longrightarrow \alpha \prec P' \Longrightarrow P \Longrightarrow \alpha \prec P'$
⟨*proof*⟩

lemma *freshBoundOutputTransition*:
fixes P :: *pi*
and a :: *name*
and x :: *name*
and P' :: *pi*
and c :: *name*

assumes *PTrans*: $P \Longrightarrow a{<}\nu x{>} \prec P'$
and $c \sharp P$
and $c \neq x$

shows $c \sharp P'$
⟨*proof*⟩

lemma *freshTauTransition*:
fixes P :: *pi*
and c :: *name*

assumes *PTrans*: $P \Longrightarrow \tau \prec P'$

and $c \sharp P$

shows $c \sharp P'$
⟨*proof*⟩

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 $c \sharp P$

shows $c \sharp P'$
⟨*proof*⟩

lemma *eqvtI*[*eqvt*]:
fixes P :: *pi*
and a :: *name*
and x :: *name*
and P' :: *pi*
and p :: *name prm*
and α :: *freeRes*

shows $P \Longrightarrow a{<}\nu x{>} \prec P' \Longrightarrow (p \cdot P) \Longrightarrow (p \cdot a){<}\nu(p \cdot x){>} \prec (p \cdot P')$
and $P \Longrightarrow \alpha \prec P' \Longrightarrow (p \cdot P) \Longrightarrow (p \cdot \alpha) \prec (p \cdot P')$
⟨*proof*⟩

lemma *freshInputTransition*:
fixes P :: *pi*
and a :: *name*
and b :: *name*
and P' :: *pi*
and c :: *name*

assumes *PTrans*: $P \Longrightarrow a{<}b{>} \prec P'$
and $c \sharp P$
and $c \neq b$

shows $c \sharp P'$
⟨*proof*⟩

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* (- $\Longrightarrow$ˆ - $\prec$ - [*80*, *80*, *80*] *80*)
where $P \Longrightarrow\hat{}\ \alpha \prec P' \equiv P \Longrightarrow\alpha \prec P' \vee (\alpha = \tau \wedge P = P')$

lemma *weakTransitionI*:
fixes P :: *pi*
and α :: *freeRes*
and P' :: *pi*

shows $P \Longrightarrow\alpha \prec P' \Longrightarrow P \Longrightarrow\hat{}\ \alpha \prec P'$
and $P \Longrightarrow\hat{}\ \tau \prec P$
⟨*proof*⟩

lemma *transitionCases*[*consumes 1*, *case-names Step Stay*]:
fixes P :: *pi*
and α :: *freeRes*
and P' :: *pi*

assumes $P \Longrightarrow\hat{}\ \alpha \prec P'$
and $P \Longrightarrow\alpha \prec P' \Longrightarrow F\ \alpha\ P'$
and $F\ (\tau)\ P$

shows $F\ \alpha\ P'$
⟨*proof*⟩

lemma *singleActionChain*:
fixes P :: *pi*
and α :: *freeRes*
and P' :: *pi*

assumes $P \longmapsto\alpha \prec P'$

shows $P \Longrightarrow\hat{}\ \alpha \prec P'$
⟨*proof*⟩

lemma *Tau*:
fixes P :: *pi*

shows $\tau.(P) \Longrightarrow\hat{}\ \tau \prec P$
⟨*proof*⟩

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

and $P :: pi$

shows $a{<}x{>}.P \Longrightarrow^{\hat{}}\ a{<}u{>} \prec P[x{::=}u]$
⟨*proof*⟩

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

shows $a\{b\}.P \Longrightarrow^{\hat{}} a[b] \prec P$
⟨*proof*⟩

lemma *Par1F*:
fixes $P :: pi$
and $\alpha :: freeRes$
and $P' :: pi$
and $Q :: pi$

assumes $P \Longrightarrow^{\hat{}} \alpha \prec P'$

shows $P \parallel Q \Longrightarrow^{\hat{}} \alpha \prec (P' \parallel Q)$
⟨*proof*⟩

lemma *Par2F*:
fixes $Q :: pi$
and $\alpha :: freeRes$
and $Q' :: pi$
and $P :: pi$

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

shows $P \parallel Q \Longrightarrow^{\hat{}} \alpha \prec (P \parallel Q')$
⟨*proof*⟩

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

assumes $P \Longrightarrow^{\hat{}} \alpha \prec P'$
and $x \sharp \alpha$

shows ${<}\nu x{>}P \Longrightarrow^{\hat{}} \alpha \prec {<}\nu x{>}P'$
⟨*proof*⟩

lemma *Bang*:

fixes P :: *pi*
and Rs :: *residual*

assumes $P \parallel !P \Longrightarrow\hat{}\ \alpha \prec P'$
and $P' \neq P \parallel !P$

shows $!P \Longrightarrow\hat{}\ \alpha \prec P'$
⟨*proof*⟩

lemma *tauTransitionChain*[*simp*]:
fixes P :: *pi*
and P' :: *pi*

shows $P \Longrightarrow\hat{}\ \tau \prec P' = P \Longrightarrow_\tau P'$
⟨*proof*⟩

lemma *tauStepTransitionChain*[*simp*]:
fixes P :: *pi*
and P' :: *pi*

assumes $P \neq P'$

shows $P \Longrightarrow \tau \prec P' = P \Longrightarrow_\tau P'$
⟨*proof*⟩

lemma *chainTransitionAppend*:
fixes P :: *pi*
and P' :: *pi*
and Rs :: *residual*
and a :: *name*
and x :: *name*
and P'' :: *pi*
and α :: *freeRes*

shows $P \Longrightarrow_\tau P'' \Longrightarrow P'' \Longrightarrow\hat{}\ \alpha \prec P' \Longrightarrow P \Longrightarrow\hat{}\ \alpha \prec P'$
and $P \Longrightarrow\hat{}\ \alpha \prec P'' \Longrightarrow P'' \Longrightarrow_\tau P' \Longrightarrow P \Longrightarrow\hat{}\ \alpha \prec P'$
⟨*proof*⟩

lemma *freshTauTransition*:
fixes P :: *pi*
and c :: *name*

assumes $P \Longrightarrow\hat{}\ \tau \prec P'$
and $c \sharp P$

shows $c \sharp P'$
⟨*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 $c \sharp P$

shows $c \sharp P'$
⟨*proof*⟩

lemma *eqvtI*:
fixes P :: *pi*
and α :: *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 $P \Longrightarrow^{\hat{}} a<b> \prec P'$
and $c \sharp P$
and $c \neq b$

shows $c \sharp P'$
⟨*proof*⟩

lemmas *freshTransition* = *freshBoundOutputTransition freshOutputTransition freshInputTransition freshTauTransition*

end

theory *Weak-Early-Sim*
imports *Weak-Early-Semantics Strong-Early-Sim-Pres*
begin

definition *weakSimulation* :: $pi \Rightarrow (pi \times pi)\ set \Rightarrow pi \Rightarrow bool$ (- ⇝<-> - [*80*, *80*, *80*] *80*)
where $P \leadsto<Rel>\ Q \equiv (\forall\, a\ x\ Q'.\ Q \longmapsto a<\nu x> \prec Q' \wedge x \sharp P \longrightarrow (\exists P'.\ P$

$\Longrightarrow a{<}\nu x{>} \prec P' \wedge (P', Q') \in Rel)) \wedge$
$(\forall \alpha\ Q'.\ Q \longmapsto \alpha \prec Q' \longrightarrow (\exists P'.\ P \Longrightarrow^{\wedge} \alpha \prec P' \wedge (P', Q') \in$
$Rel))$

lemma *monotonic*:
fixes A :: $(pi \times pi)$ *set*
and B :: $(pi \times pi)$ *set*
and P :: *pi*
and P' :: *pi*

assumes $P \leadsto{<}A{>}\ P'$
and $A \subseteq B$

shows $P \leadsto{<}B{>}\ P'$
⟨*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*: $\bigwedge a\ x\ Q'.\ [\![Q \longmapsto a{<}\nu x{>} \prec Q';\ x \sharp P;\ x \sharp Q;\ x \neq a;\ x \sharp C]\!] \Longrightarrow$
$\exists P'.\ P \Longrightarrow a{<}\nu x{>} \prec P' \wedge (P', Q') \in Rel$
and *Free*: $\bigwedge \alpha\ Q'.\ Q \longmapsto \alpha \prec Q' \Longrightarrow \exists P'.\ P \Longrightarrow^{\wedge} \alpha \prec P' \wedge (P', Q') \in Rel$

shows $P \leadsto{<}Rel{>}\ Q$
⟨*proof*⟩

lemma *simCases*[*case-names Bound Free*]:
fixes P :: *pi*
and Q :: *pi*
and *Rel* :: $(pi \times pi)$ *set*
and C :: *'a::fs-name*

assumes $\bigwedge Q'\ a\ x.\ [\![Q \longmapsto a{<}\nu x{>} \prec Q';\ x \sharp P]\!] \Longrightarrow \exists P'.\ P \Longrightarrow a{<}\nu x{>} \prec P' \wedge$
$(P', Q') \in Rel$
and $\bigwedge Q'\ \alpha.\ Q \longmapsto \alpha \prec Q' \Longrightarrow \exists P'.\ P \Longrightarrow^{\wedge} \alpha \prec P' \wedge (P', Q') \in Rel$

shows $P \leadsto{<}Rel{>}\ Q$
⟨*proof*⟩

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

and Q' :: *pi*

assumes $P \leadsto <Rel> Q$

shows $Q \longmapsto a<\nu x> \prec Q' \Longrightarrow x \sharp P \Longrightarrow \exists P'.\ P \Longrightarrow a<\nu x> \prec P' \wedge (P', Q') \in Rel$
and $Q \longmapsto \alpha \prec Q' \Longrightarrow \exists P'.\ P \Longrightarrow\hat{}\ \alpha \prec P' \wedge (P', Q') \in Rel$
⟨*proof*⟩

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

assumes *QChain*: $Q \Longrightarrow_\tau Q'$
and *PRelQ*: $(P, Q) \in Rel$
and *PSimQ*: $\bigwedge R\ S.\ (R, S) \in Rel \Longrightarrow R \leadsto <Rel> S$

shows $\exists P'.\ P \Longrightarrow_\tau P' \wedge (P', Q') \in Rel$
⟨*proof*⟩

lemma *simE2*:
fixes P :: *pi*
and Rel :: $(pi \times pi)$ *set*
and Q :: *pi*
and a :: *name*
and x :: *name*
and Q' :: *pi*

assumes *Sim*: $\bigwedge R\ S.\ (R, S) \in Rel \Longrightarrow R \leadsto <Rel> S$
and *Eqvt*: *eqvt Rel*
and *PRelQ*: $(P, Q) \in Rel$

shows $Q \Longrightarrow a<\nu x> \prec Q' \Longrightarrow x \sharp P \Longrightarrow \exists P'.\ P \Longrightarrow a<\nu x> \prec P' \wedge (P', Q') \in Rel$
and $Q \Longrightarrow\hat{}\ \alpha \prec Q' \Longrightarrow \exists P'.\ P \Longrightarrow\hat{}\ \alpha \prec P' \wedge (P', Q') \in Rel$
⟨*proof*⟩

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

assumes *PSimQ*: $P \leadsto <Rel> Q$
and *RelRel'*: $Rel \subseteq Rel'$
and *EqvtRel'*: *eqvt* Rel'

shows (*perm* · *P*) ⇝<*Rel′*> (*perm* · *Q*)
⟨*proof*⟩

lemma *reflexive*:
fixes *P* :: *pi*
and *Rel* :: (*pi* × *pi*) *set*

assumes *Id* ⊆ *Rel*

shows *P* ⇝<*Rel*> *P*
⟨*proof*⟩

lemma *transitive*:
fixes *P* :: *pi*
and *Q* :: *pi*
and *R* :: *pi*
and *Rel* :: (*pi* × *pi*) *set*
and *Rel′* :: (*pi* × *pi*) *set*
and *Rel″* :: (*pi* × *pi*) *set*

assumes *QSimR*: *Q* ⇝<*Rel′*> *R*
and *Eqvt*: *eqvt Rel*
and *Eqvt″*: *eqvt Rel″*
and *Trans*: *Rel O Rel′* ⊆ *Rel″*
and *Sim*: ⋀*S T*. (*S*, *T*) ∈ *Rel* ⟹ *S* ⇝<*Rel*> *T*
and *PRelQ*: (*P*, *Q*) ∈ *Rel*

shows *P* ⇝<*Rel″*> *R*
⟨*proof*⟩

lemma *strongAppend*:
fixes *P* :: *pi*
and *Q* :: *pi*
and *R* :: *pi*
and *Rel* :: (*pi* × *pi*) *set*
and *Rel′* :: (*pi* × *pi*) *set*
and *Rel″* :: (*pi* × *pi*) *set*

assumes *PSimQ*: *P* ⇝<*Rel*> *Q*
and *QSimR*: *Q* ⇝[*Rel′*] *R*
and *Eqvt″*: *eqvt Rel″*
and *Trans*: *Rel O Rel′* ⊆ *Rel″*

shows *P* ⇝<*Rel″*> *R*
⟨*proof*⟩

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

assumes *PSimQ*: $P \leadsto[Rel]\ Q$

shows $P \leadsto$<*Rel*> Q
⟨*proof*⟩

end

theory *Weak-Early-Bisim*
imports *Weak-Early-Sim Strong-Early-Bisim*
begin

lemma *monoAux*: $A \subseteq B \Longrightarrow P \leadsto$<A> $Q \longrightarrow P \leadsto$<B> Q
⟨*proof*⟩

coinductive-set *weakBisim* :: (*pi* × *pi*) *set*
where
step: ⟦$P \leadsto$<*weakBisim*> Q; $(Q, P) \in$ *weakBisim*⟧ $\Longrightarrow (P, Q) \in$ *weakBisim*
monos *monoAux*

abbreviation *weakEarlyBisimJudge* (**infixr** ≈ *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*: $\bigwedge P\ Q.\ (P, Q) \in X \Longrightarrow P \leadsto$<($X \cup$ *weakBisim*)> $Q \wedge (Q, P) \in X \cup$ *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*: $\bigwedge P\ Q.\ (P, Q) \in X \Longrightarrow P \leadsto$<X> $Q \wedge (Q, P) \in X$

shows $P \approx Q$
⟨*proof*⟩

lemma *weakBisimCoinduct*[*consumes 1*, *case-names cSim cSym*]:
fixes P :: *pi*
and Q :: *pi*
and X :: (*pi* × *pi*) *set*

assumes $(P, Q) \in X$
and $\bigwedge R\ S.\ (R, S) \in X \Longrightarrow R \rightsquigarrow\!<\!(X \cup \mathit{weakBisim})\!>\ S$
and $\bigwedge R\ S.\ (R, S) \in X \Longrightarrow (S, R) \in X$

shows $P \approx Q$
⟨*proof*⟩

lemma *weakBisimWeakCoinduct*[*consumes 1*, *case-names cSim cSym*]:
fixes $P :: \mathit{pi}$
and $Q :: \mathit{pi}$
and $X :: (\mathit{pi} \times \mathit{pi})\ \mathit{set}$

assumes $(P, Q) \in X$
and $\bigwedge P\ Q.\ (P, Q) \in X \Longrightarrow P \rightsquigarrow\!<\!X\!>\ Q$
and $\bigwedge P\ Q.\ (P, Q) \in X \Longrightarrow (Q, P) \in X$

shows $P \approx Q$
⟨*proof*⟩

lemma *weakBisimE*:
fixes $P :: \mathit{pi}$
and $Q :: \mathit{pi}$

assumes $P \approx Q$

shows $P \rightsquigarrow\!<\!\mathit{weakBisim}\!>\ Q$
and $Q \approx P$
⟨*proof*⟩

lemma *weakBisimI*:
fixes $P :: \mathit{pi}$
and $Q :: \mathit{pi}$

assumes $P \rightsquigarrow\!<\!\mathit{weakBisim}\!>\ Q$
and $Q \approx P$

shows $P \approx Q$
⟨*proof*⟩

lemma *eqvt*[*simp*]:
shows *eqvt weakBisim*
⟨*proof*⟩

lemma *eqvtI*[*eqvt*]:
fixes $P :: \mathit{pi}$
and $Q :: \mathit{pi}$
and $\mathit{perm} :: \mathit{name\ prm}$

assumes $P \approx Q$

shows $(perm \cdot P) \approx (perm \cdot Q)$
⟨*proof*⟩

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

assumes $P \sim Q$

shows $P \approx Q$
⟨*proof*⟩

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

shows $P \approx P$
⟨*proof*⟩

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

assumes $P \approx Q$

shows $Q \approx P$
⟨*proof*⟩

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

assumes $P \approx Q$
and $Q \approx R$

shows $P \approx R$
⟨*proof*⟩

lemma *weakBisimWeakUpto*[*case-names cSim cSym*, *consumes 1*]:
assumes *p*: $(P, Q) \in X$
and *Eqvt*: *eqvt X*
and *rSim*: $\bigwedge P\ Q.\ (P, Q) \in X \Longrightarrow P \leadsto<(weakBisim\ O\ X\ O\ bisim)> Q$
and *rSym*: $\bigwedge\ P\ Q.\ (P, Q) \in X \Longrightarrow (Q, P) \in X$

shows $P \approx Q$
⟨*proof*⟩

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

assumes p: $(P,\ Q) \in X$
and *Eqvt*: *eqvt* X
and *rSim*: $\bigwedge R\ S.\ (R,\ S) \in X \Longrightarrow R \leadsto<(weakBisim\ O\ (X \cup weakBisim)\ O\ bisim)> S$
and *rSym*: $\bigwedge R\ S.\ (R,\ S) \in X \Longrightarrow (S,\ R) \in X$

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*: $\bigwedge P\ Q.\ (P,\ Q) \in X \Longrightarrow P \leadsto<(bisim\ O\ X\ O\ bisim)> Q$
and *rSym*: $\bigwedge\ P\ Q.\ (P,\ Q) \in X \Longrightarrow (Q,\ P) \in bisim\ O\ X\ O\ bisim$

shows $P \approx Q$
⟨*proof*⟩

end

theory *Weak-Early-Step-Sim*
imports *Weak-Early-Sim Strong-Early-Sim*
begin

definition *weakStepSimulation* :: $pi \Rightarrow (pi \times pi)\ set \Rightarrow pi \Rightarrow bool$ (- ⇝«-» - [*80*, *80*, *80*] *80*) **where**
$P \leadsto«Rel» Q \equiv (\forall Q'\ a\ x.\ Q \longmapsto a<\nu x> \prec Q' \longrightarrow x \sharp P \longrightarrow (\exists P'.\ P \Longrightarrow a<\nu x> \prec P' \wedge (P',\ Q') \in Rel)) \wedge$
$(\forall Q'\ \alpha.\ Q \longmapsto \alpha \prec Q' \longrightarrow (\exists P'.\ P \Longrightarrow \alpha \prec P' \wedge (P',\ Q') \in Rel))$

lemma *monotonic*:
fixes A :: $(pi \times pi)\ set$
and B :: $(pi \times pi)\ set$
and P :: pi
and P' :: pi

assumes $P \leadsto«A» P'$
and $A \subseteq B$

shows $P \leadsto«B» P'$
⟨*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*: $\bigwedge a\ x\ Q'.\ [\![x \sharp C;\ Q \longmapsto a{<}\nu x{>} \prec Q']\!] \Longrightarrow \exists P'.\ P \Longrightarrow a{<}\nu x{>} \prec P' \wedge (P', Q') \in Rel$
and *Free*: $\bigwedge \alpha\ Q'.\ Q \longmapsto \alpha \prec Q' \Longrightarrow \exists P'.\ P \Longrightarrow \alpha \prec P' \wedge (P', Q') \in Rel$

shows $P \leadsto«Rel»\ Q$
⟨*proof*⟩

lemma *simCases*[*consumes 0*, *case-names Bound Free*]:
fixes P :: *pi*
and Q :: *pi*
and *Rel* :: (*pi* × *pi*) *set*
and C :: *'a::fs-name*

assumes $\bigwedge a\ x\ Q'.\ [\![Q \longmapsto a{<}\nu x{>} \prec Q';\ x \sharp P]\!] \Longrightarrow \exists P'.\ P \Longrightarrow a{<}\nu x{>} \prec P' \wedge (P', Q') \in Rel$
and $\bigwedge \alpha\ Q'.\ Q \longmapsto \alpha \prec Q' \Longrightarrow \exists P'.\ P \Longrightarrow \alpha \prec P' \wedge (P', Q') \in Rel$

shows $P \leadsto«Rel»\ Q$
⟨*proof*⟩

lemma *simE*:
fixes P :: *pi*
and *Rel* :: (*pi* × *pi*) *set*
and Q :: *pi*
and a :: *name*
and x :: *name*
and Q' :: *pi*

assumes $P \leadsto«Rel»\ Q$

shows $Q \longmapsto a{<}\nu x{>} \prec Q' \Longrightarrow x \sharp P \Longrightarrow \exists P'.\ P \Longrightarrow a{<}\nu x{>} \prec P' \wedge (P', Q') \in Rel$
and $Q \longmapsto \alpha \prec Q' \Longrightarrow \exists P'.\ P \Longrightarrow \alpha \prec P' \wedge (P', Q') \in Rel$
⟨*proof*⟩

lemma *simE2*:
fixes P :: *pi*
and *Rel* :: (*pi* × *pi*) *set*
and Q :: *pi*
and a :: *name*
and x :: *name*
and Q' :: *pi*

assumes *PSimQ*: $P \leadsto«Rel»\ Q$
and *Sim*: $\bigwedge R\ S.\ (R, S) \in Rel \Longrightarrow R \leadsto{<}Rel{>}\ S$
and *Eqvt*: *eqvt Rel*
and *PRelQ*: $(P, Q) \in Rel$

shows $Q \Longrightarrow a{<}\nu x{>} \prec Q' \Longrightarrow x \sharp P \Longrightarrow \exists P'. P \Longrightarrow a{<}\nu x{>} \prec P' \wedge (P', Q') \in$ *Rel*
and $Q \Longrightarrow \alpha \prec Q' \Longrightarrow \exists P'. P \Longrightarrow \alpha \prec P' \wedge (P', Q') \in Rel$
⟨*proof*⟩

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

assumes *PSimQ*: $P \leadsto$«*Rel*» Q
and *RelRel'*: $Rel \subseteq Rel'$
and *EqvtRel'*: *eqvt Rel'*

shows $(perm \cdot P) \leadsto$«*Rel'*» $(perm \cdot Q)$
⟨*proof*⟩

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

assumes $Id \subseteq Rel$

shows $P \leadsto$«*Rel*» 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*: $P \leadsto$«*Rel*» Q
and *QSimR*: $Q \leadsto$«*Rel'*» R
and *Eqvt*: *eqvt Rel*
and *Eqvt''*: *eqvt Rel''*
and *Trans*: $Rel\ O\ Rel' \subseteq Rel''$
and *Sim*: $\bigwedge S\ T. (S, T) \in Rel \Longrightarrow S \leadsto{<}Rel{>}\ T$
and *PRelQ*: $(P, Q) \in Rel$

shows $P \leadsto$«*Rel''*» R
⟨*proof*⟩

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

assumes *PSimQ*: *P* ⇝[*Rel*] *Q*

shows *P* ⇝«*Rel*» *Q*
⟨*proof*⟩

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

assumes *P* ⇝«*Rel*» *Q*

shows *P* ⇝<*Rel*> *Q*
⟨*proof*⟩

end

theory *Weak-Early-Cong*
imports *Weak-Early-Bisim Weak-Early-Step-Sim Strong-Early-Bisim*
begin

definition *weakCongruence* :: *pi* ⇒ *pi* ⇒ *bool* (**infixr** ≃ *65*)
where *P* ≃ *Q* ≡ *P* ⇝«*weakBisim*» *Q* ∧ *Q* ⇝«*weakBisim*» *P*

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

assumes *Prop P Q*
and ⋀*R S*. *Prop R S* ⟹ *Prop S R*
and ⋀*R S*. *Prop R S* ⟹ (*F R*) ⇝«*weakBisim*» (*F S*)

shows *F P* ≃ *F Q*
⟨*proof*⟩

lemma *weakCongISym2*[*consumes 1*, *case-names cSim*]:
fixes *P* :: *pi*
and *Q* :: *pi*

assumes *P* ≃ *Q*
and ⋀*R S*. *R* ≃ *S* ⟹ (*F R*) ⇝«*weakBisim*» (*F S*)

shows *F P* ≃ *F Q*
⟨*proof*⟩

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

assumes $P \simeq Q$

shows $P \leadsto«weakBisim» Q$
and $Q \leadsto«weakBisim» P$
⟨*proof*⟩

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

assumes $P \leadsto«weakBisim» Q$
and $Q \leadsto«weakBisim» P$

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 $P \approx Q$
⟨*proof*⟩

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* (**infixr** $\approx^s$ *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 $P \approx^s Q$
⟨*proof*⟩

lemma *eqvt*:
shows *eqvt* (*substClosed weakBisim*)
⟨*proof*⟩

lemma *eqvtI*[*eqvt*]:
fixes P :: *pi*
and Q :: *pi*
and *perm* :: *name prm*

assumes $P \approx^s Q$

shows $(perm \cdot P) \approx^s (perm \cdot Q)$
⟨*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-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** $\simeq^s$ *65*) **where** $P \simeq^s Q \equiv \forall \sigma. P[<\sigma>] \simeq Q[<\sigma>]$

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

assumes $P \simeq^s Q$

shows $P[<s>] \rightsquigarrow$«*weakBisim*» $Q[<s>]$
and $Q[<s>] \rightsquigarrow$«*weakBisim*» $P[<s>]$
⟨*proof*⟩

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

assumes $\bigwedge s. P[<s>] \rightsquigarrow$«*weakBisim*» $Q[<s>]$
and $\bigwedge s. Q[<s>] \rightsquigarrow$«*weakBisim*» $P[<s>]$

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 $\tau.(P) \rightsquigarrow«Rel» \tau.(Q)$
⟨*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 $a<x>.P \rightsquigarrow«Rel»\ a<x>.Q$
⟨*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* × *pi*) *set*
and *Rel′* :: (*pi* × *pi*) *set*

assumes *PSimQ*: *P* ⇝«*Rel*» *Q*
and *RelRel′*: *Rel* ⊆ *Rel′*

shows [*a*⌢*b*]*P* ⇝«*Rel′*» [*a*⌢*b*]*Q*
⟨*proof*⟩

lemma *mismatchPres*:
fixes *P* :: *pi*
and *Q* :: *pi*
and *a* :: *name*
and *b* :: *name*
and *Rel* :: (*pi* × *pi*) *set*
and *Rel′* :: (*pi* × *pi*) *set*

assumes *PSimQ*: *P* ⇝«*Rel*» *Q*
and *RelRel′*: *Rel* ⊆ *Rel′*

shows [*a*≠*b*]*P* ⇝«*Rel′*» [*a*≠*b*]*Q*
⟨*proof*⟩

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

assumes *PSimQ*: *P* ⇝«*Rel*» *Q*
and *RelRel′*: *Rel* ⊆ *Rel′*
and *C*: *Id* ⊆ *Rel′*

shows *P* ⊕ *R* ⇝«*Rel′*» *Q* ⊕ *R*
⟨*proof*⟩

lemma *parPres*:

fixes P :: *pi*
and Q :: *pi*
and R :: *pi*
and T :: *pi*
and *Rel* :: (*pi* × *pi*) *set*
and *Rel'* :: (*pi* × *pi*) *set*
and *Rel''* :: (*pi* × *pi*) *set*

assumes *PSimQ*: $P \leadsto$«*Rel*» Q
and *PRelQ*: $(P, Q) \in Rel$
and *Par*: $\bigwedge S\ T\ U.\ (S, T) \in Rel \Longrightarrow (S \parallel U, T \parallel U) \in Rel'$
and *Res*: $\bigwedge S\ T\ x.\ (S, T) \in Rel' \Longrightarrow (<\nu x>S, <\nu x>T) \in Rel'$

shows $P \parallel R \leadsto$«*Rel'*» $Q \parallel R$
⟨*proof*⟩

lemma *resPres*:
fixes P :: *pi*
and Q :: *pi*
and *Rel* :: (*pi* × *pi*) *set*
and x :: *name*
and *Rel'* :: (*pi* × *pi*) *set*

assumes *PSimQ*: $P \leadsto$«*Rel*» Q
and *C1*: $\bigwedge R\ S\ x.\ (R, S) \in Rel \Longrightarrow (<\nu x>R, <\nu x>S) \in Rel'$
and *RelRel'*: $Rel \subseteq Rel'$
and *EqvtRel*: *eqvt Rel*
and *EqvtRel'*: *eqvt Rel'*

shows $<\nu x>P \leadsto$«*Rel'*» $<\nu x>Q$
⟨*proof*⟩

lemma *resChainI*:
fixes P :: *pi*
and Q :: *pi*
and *Rel* :: (*pi* × *pi*) *set*
and *lst* :: *name list*

assumes *eqvtRel*: *eqvt Rel*
and *Res*: $\bigwedge R\ S\ x.\ (R, S) \in Rel \Longrightarrow (<\nu x>R, <\nu x>S) \in Rel$
and *PRelQ*: $P \leadsto$«*Rel*» Q

shows (*resChain lst*) $P \leadsto$«*Rel*» (*resChain lst*) Q
⟨*proof*⟩

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

assumes *PRelQ*: (*P*, *Q*) ∈ *Rel*
and *Sim*: ⋀*R S*. (*R*, *S*) ∈ *Rel* ⟹ *R* ⇝«*Rel'*» *S*
and *C1*: *Rel* ⊆ *Rel'*
and *eqvtRel*: *eqvt Rel'*

shows !*P* ⇝«*bangRel Rel'*» !*Q*
⟨*proof*⟩

end

theory *Weak-Early-Sim-Pres*
imports *Weak-Early-Sim*
begin

lemma *tauPres*:
fixes *P* :: *pi*
and *Q* :: *pi*
and *Rel* :: (*pi* × *pi*) *set*
and *Rel'* :: (*pi* × *pi*) *set*

assumes *PRelQ*: (*P*, *Q*) ∈ *Rel*

shows τ.(*P*) ⇝<*Rel*> τ.(*Q*)
⟨*proof*⟩

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*

assumes *PRelQ*: (*P*, *Q*) ∈ *Rel*

shows *a*{*b*}.*P* ⇝<*Rel*> *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*: $P \leadsto<Rel> Q$
and *RelRel'*: $Rel \subseteq Rel'$
and *RelStay*: $\bigwedge R\ S\ c.\ (R, S) \in Rel \Longrightarrow ([c \frown c]R, S) \in Rel$

shows $[a \frown b]P \leadsto<Rel'> [a \frown b]Q$
$\langle proof \rangle$

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*: $P \leadsto<Rel> Q$
and *RelRel'*: $Rel \subseteq Rel'$
and *RelStay*: $\bigwedge R\ S\ c\ d.\ [\![(R, S) \in Rel;\ c \neq d]\!] \Longrightarrow ([c \neq d]R, S) \in Rel$

shows $[a \neq b]P \leadsto<Rel'> [a \neq b]Q$
$\langle proof \rangle$

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

assumes *PSimQ*: $P \leadsto<Rel> Q$
and *RSimT*: $R \leadsto<Rel'> S$
and *PRelQ*: $(P, Q) \in Rel$
and *RRel'T*: $(R, S) \in Rel'$
and *Par*: $\bigwedge P'\ Q'\ R'\ S'.\ [\![(P', Q') \in Rel;\ (R', S') \in Rel']\!] \Longrightarrow (P' \parallel R',$
$Q' \parallel S') \in Rel''$
and *Res*: $\bigwedge T\ U\ x.\ (T, U) \in Rel'' \Longrightarrow (<\nu x>T, <\nu x>U) \in Rel''$

shows $P \parallel R \leadsto\!<\!Rel''\!> Q \parallel S$
⟨*proof*⟩

lemma *parPres*:
fixes P :: *pi*
and Q :: *pi*
and R :: *pi*
and a :: *name*
and *Rel* :: (*pi* × *pi*) *set*
and *Rel′* :: (*pi* × *pi*) *set*

assumes *PSimQ*: $P \leadsto\!<\!Rel\!> Q$
and *PRelQ*: $(P, Q) \in Rel$
and *Par*: $\bigwedge S\ T\ U.\ (S, T) \in Rel \Longrightarrow (S \parallel U, T \parallel U) \in Rel'$
and *Res*: $\bigwedge S\ T\ x.\ (S, T) \in Rel' \Longrightarrow (<\!\nu x\!>S, <\!\nu x\!>T) \in Rel'$

shows $P \parallel R \leadsto\!<\!Rel'\!> Q \parallel R$
⟨*proof*⟩

lemma *resPres*:
fixes P :: *pi*
and Q :: *pi*
and *Rel* :: (*pi* × *pi*) *set*
and x :: *name*
and *Rel′* :: (*pi* × *pi*) *set*

assumes *PSimQ*: $P \leadsto\!<\!Rel\!> Q$
and *ResRel*: $\bigwedge R\ S\ y.\ (R, S) \in Rel \Longrightarrow (<\!\nu y\!>R, <\!\nu y\!>S) \in Rel'$
and *RelRel′*: $Rel \subseteq Rel'$
and *EqvtRel*: *eqvt Rel*
and *EqvtRel′*: *eqvt Rel′*

shows $<\!\nu x\!>P \leadsto\!<\!Rel'\!> <\!\nu x\!>Q$
⟨*proof*⟩

lemma *resChainI*:
fixes P :: *pi*
and Q :: *pi*
and *Rel* :: (*pi* × *pi*) *set*
and *lst* :: *name list*

assumes *eqvtRel*: *eqvt Rel*
and *Res*: $\bigwedge R\ S\ y.\ (R, S) \in Rel \Longrightarrow (<\!\nu y\!>R, <\!\nu y\!>S) \in Rel$
and *PRelQ*: $P \leadsto\!<\!Rel\!> Q$

shows (*resChain lst*) $P \leadsto\!<\!Rel\!>$ (*resChain lst*) Q
⟨*proof*⟩

lemma *bangPres*:

fixes *P* :: *pi*
and *Q* :: *pi*
and *Rel* :: (*pi* × *pi*) *set*

assumes *PRelQ*: (*P*, *Q*) ∈ *Rel*
and *Sim*: ⋀*R S*. (*R*, *S*) ∈ *Rel* ⟹ *R* ⇝<*Rel*> *S*

and *ParComp*: ⋀*R S T U*. ⟦(*R*, *S*) ∈ *Rel*; (*T*, *U*) ∈ *Rel′*⟧ ⟹ (*R* ∥ *T*, *S* ∥ *U*) ∈ *Rel′*
and *Res*: ⋀*R S x*. (*R*, *S*) ∈ *Rel′* ⟹ (<ν*x*>*R*, <ν*x*>*S*) ∈ *Rel′*

and *RelStay*: ⋀*R S*. (*R* ∥ !*R*, *S*) ∈ *Rel′* ⟹ (!*R*, *S*) ∈ *Rel′*
and *BangRelRel′*: (*bangRel Rel*) ⊆ *Rel′*
and *eqvtRel′*: *eqvt Rel′*

shows !*P* ⇝<*Rel′*> !*Q*
⟨*proof*⟩

lemma *bangRelSim*:
fixes *P* :: *pi*
and *Q* :: *pi*
and *Rel* :: (*pi* × *pi*) *set*
and *Rel′l* :: (*pi* × *pi*) *set*

assumes *PBangRelQ*: (*P*, *Q*) ∈ *bangRel Rel*
and *Sim*: ⋀*R S*. (*R*, *S*) ∈ *Rel* ⟹ *R* ⇝<*Rel*> *S*

and *ParComp*: ⋀*R S T U*. ⟦(*R*, *S*) ∈ *Rel*; (*T*, *U*) ∈ *Rel′*⟧ ⟹ (*R* ∥ *T*, *S* ∥ *U*) ∈ *Rel′*
and *Res*: ⋀*R S x*. (*R*, *S*) ∈ *Rel′* ⟹ (<ν*x*>*R*, <ν*x*>*S*) ∈ *Rel′*

and *RelStay*: ⋀*R S*. (*R* ∥ !*R*, *S*) ∈ *Rel′* ⟹ (!*R*, *S*) ∈ *Rel′*
and *BangRelRel′*: (*bangRel Rel*) ⊆ *Rel′*
and *eqvtRel′*: *eqvt Rel′*
and *Eqvt*: *eqvt Rel*

shows *P* ⇝<*Rel′*> *Q*
⟨*proof*⟩

end

theory *Strong-Early-Late-Comp*
imports *Strong-Late-Bisim-Subst-SC Strong-Early-Bisim-Subst*
begin

abbreviation *TransitionsLate-judge* (- $\longmapsto_l$ - [*80*, *80*] *80*) **where** *P* $\longmapsto_l$ *Rs* ≡ *transitions P Rs*
abbreviation *TransitionsEarly-judge* (- $\longmapsto_e$ - [*80*, *80*] *80*) **where** *P* $\longmapsto_e$ *Rs* ≡ *TransitionsEarly P Rs*

abbreviation *Transitions-InputjudgeLate* (-<-> $\prec_l$ - [*80*, *80*] *80*) **where** $a{<}x{>}$ $\prec_l$ $P' \equiv$ (*Late-Semantics.BoundR* (*Late-Semantics.InputS a*) *x* P')
abbreviation *Transitions-OutputjudgeLate* (-[-] $\prec_l$ - [*80*, *80*] *80*) **where** $a[b]$ $\prec_l$ $P' \equiv$ (*Late-Semantics.FreeR* (*Late-Semantics.OutputR a b*) P')
abbreviation *Transitions-BoundOutputjudgeLate* (-<ν-> $\prec_l$ - [*80*, *80*] *80*) **where** $a{<}\nu x{>}$ $\prec_l$ $P' \equiv$ (*Late-Semantics.BoundR* (*Late-Semantics.BoundOutputS a*) *x* P')
abbreviation *Transitions-TaujudgeLate* ($\tau \prec_l$ - *80*) **where** $\tau \prec_l P' \equiv$ (*Late-Semantics.FreeR* *Late-Semantics.TauR* P')

abbreviation *Transitions-InputjudgeEarly* (-<-> $\prec_e$ - [*80*, *80*] *80*) **where** $a{<}x{>}$ $\prec_e$ $P' \equiv$ (*Early-Semantics.FreeR* (*Early-Semantics.InputR a x*) P')
abbreviation *Transitions-OutputjudgeEarly* (-[-] $\prec_e$ - [*80*, *80*] *80*) **where** $a[b]$ $\prec_e$ $P' \equiv$ (*Early-Semantics.FreeR* (*Early-Semantics.OutputR a b*) P')
abbreviation *Transitions-BoundOutputjudgeEarly* (-<ν-> $\prec_e$ - [*80*, *80*] *80*) **where** $a{<}\nu x{>}$ $\prec_e$ $P' \equiv$(*Early-Semantics.BoundOutputR a x* P')
abbreviation *Transitions-TaujudgeEarly* ($\tau \prec_e$ - *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 $P \longmapsto_e a[b] \prec_e P'$

 shows $P \longmapsto_l a[b] \prec_l P'$
$\langle proof \rangle$

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

 assumes $P \longmapsto_l a[b] \prec_l P'$

 shows $P \longmapsto_e a[b] \prec_e P'$
$\langle proof \rangle$

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

 shows $P \longmapsto_e a[b] \prec_e P' = P \longmapsto_l a[b] \prec_l P'$
$\langle proof \rangle$

lemma *lateEarlyBoundOutput*:
fixes P :: *pi*
and a :: *name*
and x :: *name*
and P' :: *pi*

assumes $P \longmapsto_l a{<}\nu x{>} \prec_l P'$

shows $P \longmapsto_e a{<}\nu x{>} \prec_e P'$
⟨*proof*⟩

lemma *earlyLateBoundOutput*:
fixes P :: *pi*
and a :: *name*
and x :: *name*
and P' :: *pi*

assumes $P \longmapsto_e a{<}\nu x{>} \prec_e P'$

shows $P \longmapsto_l a{<}\nu x{>} \prec_l P'$
⟨*proof*⟩

lemma *BoundOutputEq*:
fixes P :: *pi*
and a :: *name*
and x :: *name*
and P' :: *pi*

shows $P \longmapsto_e a{<}\nu x{>} \prec_e P' = P \longmapsto_l a{<}\nu x{>} \prec_l P'$
⟨*proof*⟩

lemma *lateEarlyInput*:
fixes P :: *pi*
and a :: *name*
and x :: *name*
and P' :: *pi*
and u :: *name*

assumes *PTrans*: $P \longmapsto_l a{<}x{>} \prec_l P'$

shows $P \longmapsto_e a{<}u{>} \prec_e (P'[x{::=}u])$
⟨*proof*⟩

lemma *earlyLateInput*:
fixes P :: *pi*
and a :: *name*
and x :: *name*
and P' :: *pi*

and u :: *name*
and C :: *'a::fs-name*

assumes $P \longmapsto_e a{<}u{>} \prec_e P'$
and $x \sharp P$

shows $\exists P''. P \longmapsto_l a{<}x{>} \prec_l P'' \wedge P' = P''[x::=u]$
⟨*proof*⟩

lemma *lateEarlyTau*:
fixes P :: *pi*
and P' :: *pi*

assumes $P \longmapsto_l \tau \prec_l P'$

shows $P \longmapsto_e \tau \prec_e P'$
⟨*proof*⟩

lemma *earlyLateTau*:
fixes P :: *pi*
and P' :: *pi*

assumes $P \longmapsto_e \tau \prec_e P'$

shows $P \longmapsto_l \tau \prec_l P'$
⟨*proof*⟩

lemma *tauEq*:
fixes P :: *pi*
and P' :: *pi*

shows $P \longmapsto_e (\textit{Early-Semantics.FreeR Early-Semantics.TauR } P') = P \longmapsto \tau \prec_l P'$
⟨*proof*⟩

abbreviation *simLate-judge* (- $\leadsto_l$[-] - [80, 80, 80] 80) **where** $P \leadsto_l [Rel]\ Q \equiv$ *Strong-Late-Sim.simulation P Rel Q*
abbreviation *simEarly-judge* (- $\leadsto_e$[-] - [80, 80, 80] 80) **where** $P \leadsto_e [Rel]\ Q \equiv$ *Strong-Early-Sim.strongSimEarly P Rel Q*

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

assumes *PSimQ*: $P \leadsto_l [Rel]\ Q$

shows $P \rightsquigarrow_e[Rel]\ Q$
⟨*proof*⟩

abbreviation *bisimLate-judge* (- $\sim_l$ - [*80*, *80*] *80*) **where** $P \sim_l Q \equiv (P, Q) \in$ *Strong-Late-Bisim.bisim*
abbreviation *bisimEarly-judge* (- $\sim_e$ - [*80*, *80*] *80*) **where** $P \sim_e Q \equiv (P, Q) \in$ *Strong-Early-Bisim.bisim*

lemma *lateEarlyBisim*:
fixes P :: *pi*
and Q :: *pi*

assumes $P \sim_l Q$

shows $P \sim_e Q$
⟨*proof*⟩

abbreviation *congLate-judge* (- $\sim^s{}_l$ - [*80*, *80*] *80*) **where** $P \sim^s{}_l Q \equiv (P, Q) \in$ (*substClosed Strong-Late-Bisim.bisim*)
abbreviation *congEarly-judge* (- $\sim^s{}_e$ - [*80*, *80*] *80*) **where** $P \sim^s{}_e Q \equiv (P, Q) \in$ (*substClosed Strong-Early-Bisim.bisim*)

lemma *lateEarlyCong*:
fixes P :: *pi*
and Q :: *pi*

assumes $P \sim^s{}_l Q$

shows $P \sim^s{}_e Q$
⟨*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 $<\nu a><\nu b>P \sim_e <\nu b><\nu a>P$
⟨*proof*⟩

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

shows $[a\frown a]P \sim_e P$
⟨*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 $x \sharp P$

  shows $<\nu x>(P \parallel Q) \sim_e P \parallel <\nu x>Q$
⟨*proof*⟩

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

assumes *xFreshQ*: $x \sharp Q$

shows $<\nu x>(P \parallel Q) \sim_e (<\nu x>P) \parallel Q$
⟨*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*: $a \sharp P$

shows $<\nu a>P \sim_e P$
⟨*proof*⟩

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

assumes $x \sharp P$

shows $<\nu x>(P \oplus Q) \sim_e P \oplus <\nu x>Q$
⟨*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 $P \approx Q$
$\langle proof \rangle$

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

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

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

assumes $a \neq b$

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

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

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

lemma *resComm*:
fixes P :: *pi*

shows $<\nu a><\nu b>P \approx <\nu b><\nu a>P$
$\langle proof \rangle$

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 $x \sharp P$

shows $<\nu x>(P \parallel Q) \approx P \parallel <\nu x>Q$
⟨*proof*⟩

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

assumes $x \sharp Q$

shows $<\nu x>(P \parallel Q) \approx (<\nu x>P) \parallel Q$
⟨*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 $a \sharp P$

shows $<\nu a>P \approx P$
⟨*proof*⟩

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

assumes $x \sharp P$

shows $<\nu x>(P \oplus Q) \approx P \oplus <\nu x>Q$
⟨*proof*⟩

lemma *bangSC*:
fixes P

shows $!P \approx P \parallel !P$
⟨*proof*⟩

end

theory *Weak-Early-Bisim-Pres*
imports *Strong-Early-Bisim-Pres Weak-Early-Sim-Pres Weak-Early-Bisim-SC Weak-Early-Bisim*
begin

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

assumes $P \approx Q$

shows $\tau.(P) \approx \tau.(Q)$
⟨*proof*⟩

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 *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 $a{<}x{>}.P \approx a{<}x{>}.Q$
⟨*proof*⟩

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

assumes $P \approx Q$

shows ${<}\nu x{>}P \approx {<}\nu x{>}Q$
⟨*proof*⟩

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 *PBisimQ*: $P \approx Q$

shows $!P \approx !Q$
⟨*proof*⟩

lemma *bangRelSubWeakBisim*:
shows *bangRel weakBisim* $\subseteq$ *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 $[a\frown b]P \simeq [a\frown b]Q$
⟨*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 $<\nu x>P \simeq <\nu x>Q$
⟨*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 *PeqQ*: $P \simeq^s Q$

shows $a<x>.P \simeq^s a<x>.Q$
⟨*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 $P \oplus R \simeq^s Q \oplus R$
⟨*proof*⟩

lemma *parPres*:

fixes P :: *pi*
and Q :: *pi*
and R :: *pi*

assumes $P \simeq^s Q$

shows $P \parallel R \simeq^s Q \parallel R$
⟨*proof*⟩

lemma *resPres*:
fixes P :: *pi*
and Q :: *pi*
and x :: *name*

assumes *PeqQ*: $P \simeq^s Q$

shows $<\nu x>P \simeq^s <\nu x>Q$
⟨*proof*⟩

lemma *bangPres*:
fixes P :: *pi*
and Q :: *pi*

assumes $P \simeq^s Q$

shows $!P \simeq^s !Q$
⟨*proof*⟩

end

theory *Strong-Late-Expansion-Law*
imports *Strong-Late-Bisim-SC*
begin

nominal-primrec *summands* :: $pi \Rightarrow pi\ set$ **where**
$summands\ \mathbf{0} = \{\}$
$|\ summands\ (\tau.(P)) = \{\tau.(P)\}$
$|\ x \sharp a \Longrightarrow summands\ (a<x>.P) = \{a<x>.P\}$
$|\ summands\ (a\{b\}.P) = \{a\{b\}.P\}$
$|\ summands\ ([a \frown b]P) = \{\}$
$|\ summands\ ([a \neq b]P) = \{\}$
$|\ summands\ (P \oplus Q) = (summands\ P) \cup (summands\ Q)$
$|\ summands\ (P \parallel Q) = \{\}$
$|\ summands\ (<\nu x>P) = (if\ (\exists\ a\ P'.\ a \neq x \wedge P = a\{x\}.P')\ then\ (\{<\nu x>P\})\ else\ \{\})$
$|\ summands\ (!P) = \{\}$
⟨*proof*⟩

lemma *summandsInput*[*simp*]:

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

shows $summands\ (a{<}x{>}.P) = \{a{<}x{>}.P\}$
⟨*proof*⟩

lemma *finiteSummands*:
fixes $P :: pi$

shows $finite(summands\ P)$
⟨*proof*⟩

lemma *boundSummandDest*[*dest*]:
fixes $x :: name$
and $y :: name$
and $P' :: pi$
and $P :: pi$

assumes $<\nu x>x\{y\}.P' \in summands\ P$

shows *False*
⟨*proof*⟩

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

assumes $P \in summands\ Q$
and $x \sharp Q$

shows $x \sharp P$
⟨*proof*⟩

nominal-primrec $hnf :: pi \Rightarrow bool$ **where**
$hnf\ \mathbf{0} = True$
$|\ hnf\ (\tau.(P)) = True$
$|\ x \sharp a \Longrightarrow hnf\ (a{<}x{>}.P) = True$
$|\ hnf\ (a\{b\}.P) = True$
$|\ hnf\ ([a\frown b]P) = False$
$|\ hnf\ ([a{\neq}b]P) = False$
$|\ hnf\ (P \oplus Q) = ((hnf\ P) \wedge (hnf\ Q) \wedge P \neq \mathbf{0} \wedge Q \neq \mathbf{0})$
$|\ hnf\ (P \parallel Q) = False$
$|\ hnf\ (<\nu x>P) = (\exists a\ P'.\ a \neq x \wedge P = a\{x\}.P')$
$|\ hnf\ (!P) = False$
⟨*proof*⟩

lemma *hnfInput*[*simp*]:

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

shows $hnf\ (a{<}x{>}.P)$
$\langle proof \rangle$

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

assumes $hnf\ P$

shows $P \longmapsto \tau \prec P' = (\tau.(P') \in summands\ P)$
and $P \longmapsto a{<}x{>} \prec P' = (a{<}x{>}.P' \in summands\ P)$
and $P \longmapsto a[b] \prec P' = (a\{b\}.P' \in summands\ P)$
and $a \neq x \Longrightarrow P \longmapsto a{<}\nu x{>} \prec P' = ({<}\nu x{>}a\{x\}.P' \in summands\ P)$
$\langle proof \rangle$

definition $expandSet :: pi \Rightarrow pi \Rightarrow pi\ set$ **where**
$expandSet\ P\ Q \equiv \{\tau.(P' \parallel Q) \mid P'.\ \tau.(P') \in summands\ P\} \cup$
$\{\tau.(P \parallel Q') \mid Q'.\ \tau.(Q') \in summands\ Q\} \cup$
$\{a\{b\}.(P' \parallel Q) \mid a\ b\ P'.\ a\{b\}.P' \in summands\ P\} \cup$
$\{a\{b\}.(P \parallel Q') \mid a\ b\ Q'.\ a\{b\}.Q' \in summands\ Q\} \cup$
$\{a{<}x{>}.(P' \parallel Q) \mid a\ x\ P'.\ a{<}x{>}.P' \in summands\ P \wedge x \sharp Q\}$
$\cup$
$\{a{<}x{>}.(P \parallel Q') \mid a\ x\ Q'.\ a{<}x{>}.Q' \in summands\ Q \wedge x \sharp$
$P\} \cup$
$\{{<}\nu x{>}a\{x\}.(P' \parallel Q) \mid a\ x\ P'.\ {<}\nu x{>}a\{x\}.P' \in summands\ P$
$\wedge x \sharp Q\} \cup$
$\{{<}\nu x{>}a\{x\}.(P \parallel Q') \mid a\ x\ Q'.\ {<}\nu x{>}a\{x\}.Q' \in summands$
$Q \wedge x \sharp P\} \cup$
$\{\tau.(P'[x{::=}b] \parallel Q') \mid x\ P'\ b\ Q'.\ \exists\, a.\ a{<}x{>}.P' \in summands\ P$
$\wedge a\{b\}.Q' \in summands\ Q\} \cup$
$\{\tau.(P' \parallel (Q'[x{::=}b])) \mid b\ P'\ x\ Q'.\ \exists\, a.\ a\{b\}.P' \in summands$
$P \wedge a{<}x{>}.Q' \in summands\ Q\} \cup$
$\{\tau.({<}\nu y{>}(P'[x{::=}y] \parallel Q')) \mid x\ P'\ y\ Q'.\ \exists\, a.\ a{<}x{>}.P' \in$
$summands\ P \wedge {<}\nu y{>}a\{y\}.Q' \in summands\ Q \wedge y \sharp P\} \cup$
$\{\tau.({<}\nu y{>}(P' \parallel (Q'[x{::=}y]))) \mid y\ P'\ x\ Q'.\ \exists\, a.\ {<}\nu y{>}a\{y\}.P'$
$\in summands\ P \wedge a{<}x{>}.Q' \in summands\ Q \wedge y \sharp Q\}$

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

shows $finite(expandSet\ P\ Q)$

$\langle proof \rangle$

lemma *expandHnf*:
fixes P :: *pi*
and Q :: *pi*

shows $\forall R \in (expandSet\ P\ Q).\ hnf\ R$
$\langle proof \rangle$

inductive-set *sumComposeSet* :: (*pi* × *pi set*) *set*
where
empty: $(\mathbf{0}, \{\}) \in sumComposeSet$
| *insert*: $[\![Q \in S; (P, S - \{Q\}) \in sumComposeSet]\!] \Longrightarrow (P \oplus Q, S) \in sumComposeSet$

lemma *expandAction*:
fixes P :: *pi*
and Q :: *pi*
and S :: *pi set*

assumes $(P, S) \in sumComposeSet$
and $Q \in S$
and $Q \longmapsto Rs$

shows $P \longmapsto Rs$
$\langle proof \rangle$

lemma *expandAction'*:
fixes P :: *pi*
and Q :: *pi*
and R :: *pi*

assumes $(R, S) \in sumComposeSet$
and $R \longmapsto Rs$

shows $\exists P \in S.\ P \longmapsto Rs$
$\langle proof \rangle$

lemma *expandTrans*:
fixes P :: *pi*
and Q :: *pi*
and R :: *pi*
and a :: *name*
and b :: *name*
and x :: *name*

assumes *Exp*: $(R, expandSet\ P\ Q) \in sumComposeSet$
and *Phnf*: *hnf* P
and *Qhnf*: *hnf* Q

shows $(P \parallel Q \longmapsto \tau \prec P') = (R \longmapsto \tau \prec P')$
and $(P \parallel Q \longmapsto a[b] \prec P') = (R \longmapsto a[b] \prec P')$
and $(P \parallel Q \longmapsto a{<}x{>} \prec P') = (R \longmapsto a{<}x{>} \prec P')$
and $(P \parallel Q \longmapsto a{<}\nu x{>} \prec P') = (R \longmapsto a{<}\nu x{>} \prec P')$
⟨*proof*⟩

lemma *expandLeft*:
fixes P :: *pi*
and Q :: *pi*
and R :: *pi*
and *Rel* :: (*pi* × *pi*) *set*

assumes *Exp*: (R, *expandSet* P Q) ∈ *sumComposeSet*
and *Phnf*: *hnf* P
and *Qhnf*: *hnf* Q
and *Id*: *Id* ⊆ *Rel*

shows $P \parallel Q \rightsquigarrow[Rel]\ R$
⟨*proof*⟩

lemma *expandRight*:
fixes P :: *pi*
and Q :: *pi*
and R :: *pi*
and *Rel* :: (*pi* × *pi*) *set*

assumes *Exp*: (R, *expandSet* P Q) ∈ *sumComposeSet*
and *Phnf*: *hnf* P
and *Qhnf*: *hnf* Q
and *Id*: *Id* ⊆ *Rel*

shows $R \rightsquigarrow[Rel]\ P \parallel Q$
⟨*proof*⟩

lemma *expandSC*:
fixes P :: *pi*
and Q :: *pi*
and R :: *pi*

assumes (R, *expandSet* P Q) ∈ *sumComposeSet*
and *hnf* P
and *hnf* Q

shows $P \parallel Q \sim R$
⟨*proof*⟩

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*: $\bigwedge y.\ y \in supp(P,\ Q,\ x) \Longrightarrow (P[x{::=}y],\ Q[x{::=}y]) \in Rel$
  and *Eqvt*: *eqvt Rel*

  shows $a{<}x{>}.P \leadsto[Rel]\ a{<}x{>}.Q$
⟨*proof*⟩

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

  assumes *PSimQ*: $\bigwedge y.\ y \in supp(P,\ Q,\ x) \Longrightarrow P[x{::=}y] \sim Q[x{::=}y]$

  shows $a{<}x{>}.P \sim a{<}x{>}.Q$
⟨*proof*⟩

inductive $equiv :: pi \Rightarrow pi \Rightarrow bool$ (**infixr** $\equiv_e$ *80*)
where
  Refl: $P \equiv_e P$
| *Sym*: $P \equiv_e Q \Longrightarrow Q \equiv_e P$
| *Trans*: $[\![P \equiv_e Q;\ Q \equiv_e R]\!] \Longrightarrow P \equiv_e R$

| *Match*: $[a{\frown}a]P \equiv_e P$
| *Match'*: $a \neq b \Longrightarrow [a{\frown}b]P \equiv_e \mathbf{0}$

| *Mismatch*: $a \neq b \Longrightarrow [a{\neq}b]P \equiv_e P$
| *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*: ${<}\nu x{>}(P \oplus Q) \equiv_e ({<}\nu x{>}P) \oplus ({<}\nu x{>}Q)$

| *ResNil*: ${<}\nu x{>}\mathbf{0} \equiv_e \mathbf{0}$
| *ResInput*: $[\![x \neq a;\ x \neq y]\!] \Longrightarrow {<}\nu x{>}a{<}y{>}.P \equiv_e a{<}y{>}.({<}\nu x{>}P)$
| *ResInput'*: ${<}\nu x{>}x{<}y{>}.P \equiv_e \mathbf{0}$
| *ResOutput*: $[\![x \neq a;\ x \neq b]\!] \Longrightarrow {<}\nu x{>}a\{b\}.P \equiv_e a\{b\}.({<}\nu x{>}P)$
| *ResOutput'*: ${<}\nu x{>}x\{b\}.P \equiv_e \mathbf{0}$

| *ResTau*: $<\nu x>\tau.(P) \equiv_e \tau.(<\nu x>P)$
| *ResComm*: $<\nu x><\nu y>P \equiv_e <\nu y><\nu x>P$
| *ResFresh*: $x \sharp P \Longrightarrow <\nu x>P \equiv_e P$

| *Expand*: $[\![(R,\ expandSet\ P\ Q) \in sumComposeSet;\ hnf\ P;\ hnf\ Q]\!] \Longrightarrow P \parallel Q \equiv_e R$

| *SumPres*: $P \equiv_e Q \Longrightarrow P \oplus R \equiv_e Q \oplus R$
| *ParPres*: $[\![P \equiv_e P';\ Q \equiv_e Q']\!] \Longrightarrow P \parallel Q \equiv_e P' \parallel Q'$
| *ResPres*: $P \equiv_e Q \Longrightarrow <\nu x>P \equiv_e <\nu x>Q$
| *TauPres*: $P \equiv_e Q \Longrightarrow \tau.(P) \equiv_e \tau.(Q)$
| *OutputPres*: $P \equiv_e Q \Longrightarrow a\{b\}.P \equiv_e a\{b\}.Q$
| *InputPres*: $[\![\forall\, y \in supp(P,\ Q,\ x).\ P[x::=y] \equiv_e Q[x::=y]]\!] \Longrightarrow a<x>.P \equiv_e a<x>.Q$

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 *PeqP'*: $P \equiv_e P'$
and *QeqQ'*: $Q \equiv_e Q'$

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 $(a\{b\}.P) \equiv_e \mathbf{0} \Longrightarrow False$
and $(a{<}x{>}.P) \equiv_e \mathbf{0} \Longrightarrow False$
and $(\tau.(P)) \equiv_e \mathbf{0} \Longrightarrow False$

and $\mathbf{0} \equiv_e a\{b\}.P \Longrightarrow False$
and $\mathbf{0} \equiv_e a{<}x{>}.P \Longrightarrow False$
and $\mathbf{0} \equiv_e \tau.(P) \Longrightarrow False$
⟨*proof*⟩

lemma *eq-eqvt*:
fixes $pi::name\ prm$
and $x::'a::pt\text{-}name$
shows $pi\cdot(x=y) = ((pi\cdot x)=(pi\cdot y))$
⟨*proof*⟩

nominal-primrec $depth :: pi \Rightarrow nat$ **where**
$depth\ \mathbf{0} = 0$
| $depth\ (\tau.(P)) = 1 + (depth\ P)$
| $a \sharp x \Longrightarrow depth\ (a{<}x{>}.P) = 1 + (depth\ P)$
| $depth\ (a\{b\}.P) = 1 + (depth\ P)$
| $depth\ ([a\frown b]P) = (depth\ P)$
| $depth\ ([a\neq b]P) = (depth\ P)$
| $depth\ (P \oplus Q) = max\ (depth\ P)\ (depth\ Q)$
| $depth\ (P \parallel Q) = ((depth\ P) + (depth\ Q))$
| $depth\ ({<}\nu x{>}P) = (depth\ P)$
| $depth\ (!P) = (depth\ P)$
⟨*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 $depth\ (a{<}x{>}.P) = 1 + (depth\ P)$
⟨*proof*⟩

nominal-primrec $valid :: pi \Rightarrow bool$ **where**
$valid\ \mathbf{0} = True$
| $valid\ (\tau.(P)) = valid\ P$
| $x \sharp a \Longrightarrow valid\ (a{<}x{>}.P) = valid\ P$

| $valid\ (a\{b\}.P) = valid\ P$
| $valid\ ([a\frown b]P) = valid\ P$
| $valid\ ([a\neq b]P) = valid\ P$
| $valid\ (P \oplus Q) = ((valid\ P) \wedge (valid\ Q))$
| $valid\ (P \parallel Q) = ((valid\ P) \wedge (valid\ Q))$
| $valid\ (<\nu x>P) = valid\ P$
| $valid\ (!P) = False$
$\langle proof \rangle$

lemma *validEqvt*[*simp*]:
fixes $P :: pi$
and $p :: name\ prm$

shows $valid(p \cdot P) = valid\ P$
$\langle proof \rangle$

lemma *validInput*[*simp*]:
fixes $a :: name$
and $x :: name$
and $P :: pi$

shows $valid\ (a<x>.P) = valid\ P$
$\langle proof \rangle$

lemma *depthMin*[*intro*]:
fixes P

shows $0 \leq depth\ P$
$\langle proof \rangle$

lemma *hnfTransition*:
fixes $P :: pi$

assumes $hnf\ P$
and $P \neq \mathbf{0}$

shows $\exists Rs.\ P \longmapsto Rs$
$\langle proof \rangle$

definition $uhnf :: pi \Rightarrow bool$ **where**
$uhnf\ P \equiv hnf\ P \wedge (\forall R \in summands\ P.\ \forall R' \in summands\ P.\ R \neq R' \longrightarrow \neg(R \equiv_e R'))$

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

assumes $Q \in summands\ P$
and $Q \equiv_e Q'$

shows $P \oplus Q' \equiv_e P$
⟨*proof*⟩

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

assumes *Phnf*: *uhnf P*
and *Qhnf*: *uhnf Q*
and *validP*: *valid P*
and *validQ*: *valid Q*

shows $\exists R.\ uhnf\ R \wedge valid\ R \wedge P \oplus Q \equiv_e R \wedge (depth\ R) \leq (depth\ (P \oplus Q))$
⟨*proof*⟩

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

assumes *Phnf*: *uhnf P*
and *validP*: *valid P*

shows $\exists P'.\ uhnf\ P' \wedge valid\ P' \wedge <\nu x>P \equiv_e P' \wedge depth\ P' \leq depth(<\nu x>P)$
⟨*proof*⟩

lemma *expandHnf*:
fixes $P :: pi$
and $S :: pi\ set$

assumes $(P, S) \in sumComposeSet$
and $\forall P \in S.\ uhnf\ P \wedge valid\ P$

shows $\exists P'.\ uhnf\ P' \wedge valid\ P' \wedge P \equiv_e P' \wedge depth\ P' \leq depth\ P$
⟨*proof*⟩

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

assumes $P \in summands\ Q$
and *uhnf Q*

shows $(summands\ Q) - \{P' \mid P'.\ P' \in summands\ Q \wedge P' \equiv_e P\} = (summands\ Q) - \{P\}$
⟨*proof*⟩

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 $P \longmapsto a[b] \prec P'$
and $valid\ P$

shows $valid\ P'$
⟨*proof*⟩

lemma *validInputTransition*:
fixes $P :: pi$
and $a :: name$
and $x :: name$
and $P' :: pi$

assumes *PTrans*: $P \longmapsto a{<}x{>} \prec P'$
and *validP*: $valid\ P$

shows $valid\ P'$
⟨*proof*⟩

lemma *validBoundOutputTransition*:
fixes $P :: pi$
and $a :: name$
and $x :: name$
and $P' :: pi$

assumes *PTrans*: $P \longmapsto a{<}\nu x{>} \prec P'$
and *validP*: $valid\ P$

shows $valid\ P'$
⟨*proof*⟩

lemma *validTauTransition*:
fixes $P :: pi$
and $P' :: pi$

assumes *PTrans*: $P \longmapsto \tau \prec P'$
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* $P \Longrightarrow$ *valid* P'
and $a\{b\}.P' \in$ *summands* $P \Longrightarrow$ *valid* P'
and a<x>$.P' \in$ *summands* $P \Longrightarrow$ *valid* P'
and $[\![a \neq x;$ <νx>$a\{x\}.P' \in$ *summands* $P]\!] \Longrightarrow$ *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 $\exists\, Q \in F.\ depth\ P \leq depth\ Q \wedge (\forall\, R \in F.\ depth\ R \leq depth\ Q)$
$\langle proof \rangle$

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

shows $depth(P[a{::=}b]) = depth\ P$
$\langle proof \rangle$

lemma *depthTransition*:
fixes $P :: pi$
and $a :: name$
and $b :: name$
and $P' :: pi$

assumes *Phnf*: *hnf P*

shows $P \longmapsto a[b] \prec P' \Longrightarrow depth\ P' < depth\ P$
and $P \longmapsto a{<}x{>} \prec P' \Longrightarrow depth\ P' < depth\ P$
and $P \longmapsto \tau \prec P' \Longrightarrow depth\ P' < depth\ P$
and $P \longmapsto a{<}\nu x{>} \prec P' \Longrightarrow depth\ P' < depth\ P$
$\langle proof \rangle$

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

assumes $R \in$ *expandSet P Q*
and *hnf P*
and *hnf Q*

shows $depth\ R \leq depth(P \parallel Q)$
$\langle proof \rangle$

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*: $P \sim Q$

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.