# The pi-calculus

Jesper Bengtson

March 17, 2025

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

**lemma** *pt-swap*:
  **fixes** $x$ :: $'a$
  **and** $a$ :: $'x$
  **and** $b$ :: $'x$

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

  **shows** $[(a,\ b)] \cdot x = [(b,\ a)] \cdot x$
**proof** −
  **show** *?thesis* **by**(*simp add*: *pt3*[*OF pt*] *at-ds5*[*OF at*])
**qed**

**atom-decl** *name*

**lemmas** *name-fresh-abs* = *fresh-abs-fun-iff*[*OF pt-name-inst*, *OF at-name-inst*, *OF fs-name1*]
**lemmas** *name-bij* = *at-bij*[*OF at-name-inst*]
**lemmas** *name-supp-abs* = *abs-fun-supp*[*OF pt-name-inst*, *OF at-name-inst*, *OF fs-name1*]
**lemmas** *name-abs-eq* = *abs-fun-eq*[*OF pt-name-inst*, *OF at-name-inst*]
**lemmas** *name-supp* = *at-supp*[*OF at-name-inst*]
**lemmas** *name-calc* = *at-calc*[*OF at-name-inst*]
**lemmas** *name-fresh-fresh* = *pt-fresh-fresh*[*OF pt-name-inst*, *OF at-name-inst*]
**lemmas** *name-fresh-left* = *pt-fresh-left*[*OF pt-name-inst*, *OF at-name-inst*]
**lemmas** *name-fresh-right* = *pt-fresh-right*[*OF pt-name-inst*, *OF at-name-inst*]
**lemmas** *name-id*[*simp*] = *pt-id*[*OF pt-name-inst*, *OF at-name-inst*]
**lemmas** *name-swap-bij*[*simp*] = *pt-swap-bij*[*OF pt-name-inst*, *OF at-name-inst*]
**lemmas** *name-swap* = *pt-swap*[*OF pt-name-inst*, *OF at-name-inst*]
**lemmas** *name-rev-per* = *pt-rev-pi*[*OF pt-name-inst*, *OF at-name-inst*]
**lemmas** *name-per-rev* = *pt-pi-rev*[*OF pt-name-inst*, *OF at-name-inst*]
**lemmas** *name-exists-fresh* = *at-exists-fresh*[*OF at-name-inst*, *OF fs-name1*]

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

nominal-datatype pi = PiNil                    (‹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(cases x = c)
  assume x=c
  thus ?thesis by(simp)
next
  assume x ≠ c
  with A1 show ?thesis
    by(simp add: pi.inject alpha name-fresh-left name-calc)
qed

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

  assumes A1: c ♯ P

  shows <νa>P = <νc>([(a, c)] • P)
proof(cases a=c)
  assume a=c
  thus ?thesis by simp
next
  assume a ≠ c
  with A1 show ?thesis
    by(simp add: pi.inject alpha fresh-left name-calc)
qed
```

**definition** *subst-name* :: *name* ⇒ *name* ⇒ *name* ⇒ *name* (‹-[-::=-]› [*110*, *110*, *110*] *110*)
**where**
$a[b{::=}c] \equiv$ *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** $p \bullet (a[b{::=}c]) = (p\bullet\ a)[(p \bullet b){::=}(p \bullet c)]$
**by**(*auto simp add*: *at-bij*[*OF at-name-inst*])

**nominal-primrec** (*freshness-context*: (*c*::*name*, *d*::*name*))
*subs* :: *pi* ⇒ *name* ⇒ *name* ⇒ *pi* (‹-[-::=-]› [*100*,*100*,*100*] *100*)
**where**
$\mathbf{0}[c{::=}d] = \mathbf{0}$
$\mid \tau.(P)[c{::=}d] = \tau.(P[c{::=}d])$
$\mid a\{b\}.P[c{::=}d] = (a[c{::=}d])\{(b[c{::=}d])\}.(P[c{::=}d])$
$\mid [\![x \neq a;\ x \neq c;\ x \neq d]\!] \Longrightarrow (a{<}x{>}.P)[c{::=}d] = (a[c{::=}d]){<}x{>}.(P[c{::=}d])$
$\mid [a\frown b]P[c{::=}d] = [(a[c{::=}d])\frown(b[c{::=}d])](P[c{::=}d])$
$\mid [a{\neq}b]P[c{::=}d] = [(a[c{::=}d]){\neq}(b[c{::=}d])](P[c{::=}d])$
$\mid (P \oplus Q)[c{::=}d] = (P[c{::=}d]) \oplus (Q[c{::=}d])$
$\mid (P \parallel Q)[c{::=}d] = (P[c{::=}d]) \parallel (Q[c{::=}d])$
$\mid [\![x \neq c;\ x \neq d]\!] \Longrightarrow ({<}\nu x{>}P)[c{::=}d] = {<}\nu x{>}(P[c{::=}d])$
$\mid\ !P[c{::=}d] =\ !(P[c{::=}d])$
**apply**(*simp-all add*: *abs-fresh*)
**apply**(*finite-guess*)+
**by**(*fresh-guess*)+

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

**assumes** $a \sharp P$

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

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

  **assumes** $a \neq b$

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

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

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

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

  **assumes** $c \sharp P$

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

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

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

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

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

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

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

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

**assumes** *x* ≠ *b*
**and** *x* ≠ *c*

**shows** (*a*<*x*>.*P*)[*b*::=*c*] = (*a*[*b*::=*c*])<*x*>.(*P*[*b*::=*c*])
**proof** −
**obtain** *y*::*name* **where***y* ≠ *a* **and** *y* ♯ *P* **and** *y* ≠ *b* **and** *y* ≠ *c*
**by**(*generate-fresh name*) (*auto simp add*: *fresh-prod*)
**from** ‹*y* ♯ *P*› **have** *a*<*x*>.*P* = *a*<*y*>.([(*x*, *y*)] • *P*) **by**(*simp add*: *alphaInput*)
**moreover have** (*a*[*b*::=*c*])<*x*>.(*P*[*b*::=*c*]) = (*a*[*b*::=*c*])<*y*>.(([(*x*, *y*)] • *P*)[*b*::=*c*])
(**is** *?LHS* = *?RHS*)
**proof** −
**from** ‹*y* ♯ *P*› ‹*y* ≠ *c*› **have** *y* ♯ *P*[*b*::=*c*] **by**(*rule fresh-fact1*)
**hence** *?LHS* = (*a*[*b*::=*c*])<*y*>.([(*x*, *y*)] • (*P*[*b*::=*c*])) **by**(*simp add*: *alphaInput*)
**moreover with** ‹*x* ≠ *b*› ‹*x* ≠ *c*› ‹*y* ≠ *b*› ‹*y* ≠ *c*› **have** ... = *?RHS*
**by**(*auto simp add*: *eqvt-subs name-calc*)
**ultimately show** *?thesis* **by** *simp*
**qed**
**ultimately show** *?thesis* **using** ‹*y* ≠ *a*› ‹*y* ≠ *b*› ‹*y* ≠ *c*› **by** *simp*
**qed**

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

**assumes** *b* ♯ *P*

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

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

  assumes b ♯ P

  shows <νa>P = <νb>(P[a::=b])
proof(case-tac a = b)
  assume a = b
  thus ?thesis by auto
next
  assume a ≠ b
  moreover with ‹b ♯ P› show ?thesis
    apply(simp add: pi.inject abs-fun-eq[OF pt-name-inst, OF at-name-inst])
    apply auto
    apply(simp add: renaming)
    apply(simp add: pt-swap[OF pt-name-inst, OF at-name-inst])
    apply(simp add: renaming)
    apply(simp add: pt-fresh-left[OF pt-name-inst, OF at-name-inst])
    by(force simp add: at-calc[OF at-name-inst])
qed

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

  shows a ♯ <νa>P
by(simp add: name-fresh-abs)

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

  assumes b ♯ P

  shows (<νa>P)[a::=b] = <νb>(P[a::=b])
proof −
  have (<νa>P)[a::=b] = <νa>P
    using freshRes by(simp add: forget)
  thus ?thesis using ‹b ♯ P› by(simp add: substRes2)
qed

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

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

**primrec** *seqSubs* :: *pi* ⇒ (*name* × *name*) *list* ⇒ *pi* (‹-[<->]› [*100*,*100*] *100*)
**where**
*P*[<[]>] = *P*
| *P*[<(*x*#σ)>] = (*P*[(*fst x*)::=(*snd x*)])[<σ>]

**primrec** *seq-subst-name* :: *name* ⇒ (*name* × *name*) *list* ⇒ *name* **where**
*seq-subst-name a* [] = *a*
| *seq-subst-name a* (*x*#σ) = *seq-subst-name* (*a*[(*fst x*)::=(*snd x*)]) σ

**lemma** *freshSeqSubstName*:
**fixes** *x* :: *name*
**and** *a* :: *name*
**and** *s* :: (*name* × *name*) *list*

**assumes** *x* ≠ *a*
**and** *x* ♯ *s*

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

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

**shows** **0**[<σ>] = **0**
**by**(*induct* σ, *auto*)

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

**shows** (τ.(*P*))[<σ>] = τ.(*P*[<σ>])
**by**(*induct* σ *arbitrary*: *P*, *auto*)

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

**and** $\sigma :: (name \times name)\ list$

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

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

**assumes** $x \sharp \sigma$

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

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

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

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

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

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

**shows** $(P \oplus Q)[<\sigma>] = (P[<\sigma>]) \oplus (Q[<\sigma>])$
**by**(*induct* $\sigma$ *arbitrary*: *P Q* , *auto*)

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

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

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

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

**assumes** $x \sharp \sigma$

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

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

**shows** $(!P)[<\sigma>] = !(P[<\sigma>])$
**by**(*induct* $\sigma$ *arbitrary*: $P$, *auto*)

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

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

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

**shows** $P[<(\sigma @ \sigma')>] = (P[<\sigma>])[<\sigma'>]$
**by**(*induct* $\sigma$ *arbitrary*: $P$, *auto*)

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

**assumes** $a \sharp P$
**and** $a \sharp \sigma$

**shows** $a \sharp P[<\sigma>]$
**using** *assms*
**by**(*induct* $\sigma$ *arbitrary*: $a$ $P$, *auto simp add*: *fresh-list-cons fresh-prod fresh-fact1*)

**end**

**theory** *Late-Semantics*
  **imports** *Agent*
**begin**

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

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

**nominal-datatype** *residual* = *BoundR subject* «*name*» *pi* (‹-«-» $\prec$ -› [*80*, *80*, *80*] *80*)
| *FreeR freeRes pi* (‹- $\prec$ -› [*80*, *80*] *80*)

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

**abbreviation** *Transitions-Inputjudge* :: *name* $\Rightarrow$ *name* $\Rightarrow$ *pi* $\Rightarrow$ *residual* (‹-$<$-$>$ $\prec$ -› [*80*, *80*, *80*] *80*)
**where** $a<x> \prec P' \equiv ((InputS\ a)«x» \prec P')$

**abbreviation** *Transitions-BoundOutputjudge* :: *name* $\Rightarrow$ *name* $\Rightarrow$ *pi* $\Rightarrow$ *residual* (‹-$<\nu$-$>$ $\prec$ -› [*80*, *80*, *80*] *80*)
**where** $a<\nu x> \prec P' \equiv (BoundR\ (BoundOutputS\ a)\ x\ P')$

**inductive** *transitions* :: *pi* $\Rightarrow$ *residual* $\Rightarrow$ *bool* (‹- $\longmapsto$ -› [*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*: $[\![P \longmapsto a[b] \prec P';\ Q \longmapsto a<x> \prec Q';\ x \sharp P;\ x \sharp Q;\ x \neq a;\ x$

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

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

| *Bang*: $\llbracket P \parallel !P \longmapsto Rs\rrbracket \Longrightarrow !P \longmapsto Rs$

**equivariance** *transitions*
**nominal-inductive** *transitions*
**by**(*auto simp add*: *abs-fresh fresh-fact2*)

**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**(*cases* $x{=}x'$)
  **assume** $x{=}x'$
  **thus** *?thesis* **by** *simp*
**next**
  **assume** $x \neq x'$
  **with** *A1* **show** *?thesis*
    **by**(*simp add*: *residualInject alpha name-fresh-left name-calc*)
**qed**

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

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

  **shows** $x \sharp Rs$
**using** *assms*
**by**(*nominal-induct rule*: *transitions.strong-induct*)
  (*auto simp add*: *abs-fresh fresh-fact2 fresh-fact1*)

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

  **shows** $y \sharp a$
  **and** $y \neq x \Longrightarrow y \sharp P'$
**apply** −
**using** *assms*
**by**(*fastforce dest*: *freshResidual simp add*: *abs-fresh*)+

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

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

  **shows** $y \sharp \alpha$
  **and** $y \sharp P'$
**apply** −
**using** *assms*
**by**(*fastforce dest*: *freshResidual*)+

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

  **assumes** $b \sharp P$

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

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

  **shows** $a{<}x{>}.P \longmapsto a{<}x{>} \prec P$
**proof** −
  **obtain** $y$::*name* **where** $y \neq a$ **and** $y \sharp P$
    **by**(*generate-fresh name*, *auto simp add*: *fresh-prod*)
  **from** ‹$y \sharp P$› **have** $a{<}x{>}.P = a{<}y{>}.([(x, y)] \cdot P)$ **and** $a{<}x{>} \prec P = a{<}y{>} \prec$

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

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

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

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

 **shows** $P \parallel Q \longmapsto a«x» \prec P' \parallel Q$
**proof** −
 **obtain** $y$::*name* **where** $y \sharp P$ **and** $y \sharp P'$ **and** $y \sharp Q$ **and** $y \sharp a$
  **by**(*generate-fresh name*, *auto*)
 **from** $\langle P \longmapsto a«x» \prec P'\rangle$ $\langle y \sharp P'\rangle$ **have** $P \longmapsto a«y» \prec ([(x,\ y)] \cdot P')$
  **by**(*simp add*: *alphaBoundResidual*)
 **hence** $P \parallel Q \longmapsto a«y» \prec ([(x,\ y)] \cdot P') \parallel Q$ **using** $\langle y \sharp P\rangle$ $\langle y \sharp Q\rangle$ $\langle y \sharp a\rangle$
  **by**(*rule Par1B*)
 **with** $\langle x \sharp Q\rangle$ $\langle y \sharp P'\rangle$ $\langle y \sharp Q\rangle$ **show** *?thesis*
  **by**(*subst alphaBoundResidual*[**where** $x'$=$y$]) *auto*
**qed**

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

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

 **shows** $P \parallel Q \longmapsto a«x» \prec P \parallel Q'$
**proof** −
 **obtain** $y$::*name* **where** $y \sharp Q$ **and** $y \sharp Q'$ **and** $y \sharp P$ **and** $y \sharp a$
  **by**(*generate-fresh name*, *auto simp add*: *fresh-prod*)
 **from** *QTrans* $\langle y \sharp Q'\rangle$ **have** $Q \longmapsto a«y» \prec ([(x,\ y)] \cdot Q')$
  **by**(*simp add*:*alphaBoundResidual*)
 **hence** $P \parallel Q \longmapsto a«y» \prec P \parallel ([(x,\ y)] \cdot Q')$ **using** $\langle y \sharp P\rangle$ $\langle y \sharp Q\rangle$ $\langle y \sharp a\rangle$
  **by**(*rule Par2B*)
 **moreover have** $a«y» \prec P \parallel ([(x,\ y)] \cdot Q') = a«x» \prec P \parallel Q'$
 **proof** −

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

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

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

**shows** $P \parallel Q \longmapsto \tau \prec P'[x{::=}b] \parallel Q'$
**proof** −
**obtain** $y$::*name* **where** $y \sharp P$ **and** $y \sharp P'$ **and** $y \sharp Q$ **and** $y \neq a$ **and** $y \neq b$ **and** $y \sharp Q'$
**by**(*generate-fresh name*, *auto simp add*: *fresh-prod*)
**from** *PTrans* ‹$y \sharp P'$› **have** $P \longmapsto a{<}y{>} \prec ([(x, y)] \cdot P')$
**by**(*simp add*: *alphaBoundResidual*)
**hence** $P \parallel Q \longmapsto \tau \prec ([(x, y)] \cdot P')[y{::=}b] \parallel Q'$
**using** *QTrans* ‹$y \sharp P$› ‹$y \sharp Q$› ‹$y \neq a$› ‹$y \neq b$› ‹$y \sharp Q'$›
**by**(*rule Comm1*)
**with** ‹$y \sharp P'$› **show** *?thesis* **by**(*simp add*: *renaming name-swap*)
**qed**

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

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

**shows** $P \parallel Q \longmapsto \tau \prec P' \parallel (Q'[x{::=}b])$
**proof** −
**obtain** $y$::*name* **where** $y \sharp P$ **and** $y \sharp P'$ **and** $y \sharp Q$ **and** $y \neq a$ **and** $y \neq b$ **and** $y \sharp Q'$

**by**(*generate-fresh name*, *auto simp add*: *fresh-prod*)
**from** *QTrans* ‹$y \sharp Q'$› **have** $Q \longmapsto a{<}y{>} \prec ([(x, y)] \cdot Q')$
**by**(*simp add*: *alphaBoundResidual*)
**with** *PTrans* **have** $P \parallel Q \longmapsto \tau \prec P' \parallel (([(x, y)] \cdot Q')[y::=b])$
**using** ‹$y \sharp P$› ‹$y \sharp Q$› ‹$y \neq a$› ‹$y \neq b$› ‹$y \sharp P'$›
**by**(*rule Comm2*)
**with** ‹$y \sharp Q'$› **show** *?thesis* **by**(*simp add*: *renaming name-swap*)
**qed**

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

**assumes** *PTrans*: $P \longmapsto a{<}x{>} \prec P'$
**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** −
**obtain** $x'$::*name* **where** $x' \sharp P$ **and** $x' \sharp P'$ **and** $x' \sharp Q$ **and** $x' \sharp Q'$ **and** $x' \neq a$
**by**(*generate-fresh name*, *auto simp add*: *fresh-prod*)
**obtain** $y'$::*name* **where** $y' \sharp P$ **and** $y' \sharp Q'$ **and** $y' \sharp Q$
**and** $y' \sharp P'$ **and** $y' \neq x'$ **and** $y' \neq y$ **and** $y' \neq a$
**by**(*generate-fresh name*, *auto simp add*: *fresh-prod*)
**from** *PTrans* ‹$x' \sharp P'$› **have** $P \longmapsto a{<}x'{>} \prec ([(x, x')] \cdot P')$
**by**(*simp add*: *alphaBoundResidual*)
**moreover from** *QTrans* ‹$y' \sharp Q'$› **have** $Q \longmapsto a{<}\nu y'{>} \prec ([(y, y')] \cdot Q')$
**by**(*simp add*: *alphaBoundResidual*)
**ultimately have** $P \parallel Q \longmapsto \tau \prec {<}\nu y'{>}(([(x, x')] \cdot P')[x'::=y'] \parallel ([(y, y')] \cdot Q'))$
**using** ‹$y' \sharp P$› ‹$y' \sharp Q$› ‹$x' \sharp P$› ‹$x' \sharp Q$› ‹$y' \neq x'$› ‹$y' \neq a$› ‹$x' \neq a$›
‹$y' \sharp P'$› ‹$y' \sharp Q'$› ‹$x' \sharp P'$› ‹$x' \sharp Q'$›
**apply**(*rule-tac Close1*)
**by** *assumption* (*auto simp add*: *fresh-left calc-atm*)
**moreover have** ${<}\nu y'{>}(([(x, x')] \cdot P')[x'::=y'] \parallel ([(y, y')] \cdot Q')) = {<}\nu y{>}(P'[x::=y] \parallel Q')$
**proof** −
**from** ‹$x' \sharp P'$› **have** $([(x, x')] \cdot P')[x'::=y'] = P'[x::=y']$ **by**(*simp add*: *renaming name-swap*)
**moreover have** $y \sharp (P'[x::=y'] \parallel ([(y, y')] \cdot Q'))$
**proof**(*case-tac* $y = x$)
**assume** $y = x$
**with** ‹$y' \sharp Q'$› ‹$y' \neq y$› **show** *?thesis* **by**(*auto simp add*: *fresh-fact2 fresh-left calc-atm*)
**next**

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

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

**assumes** *PTrans*: $P \longmapsto a{<}\nu y{>} \prec P'$
**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** −
**obtain** $x'$::*name* **where** $x' \sharp P$ **and** $x' \sharp Q'$ **and** $x' \sharp Q$ **and** $x' \sharp P'$ **and** $x' \neq a$
**by**(*generate-fresh name*, *auto simp add*: *fresh-prod*)
**obtain** $y'$::*name* **where** $y' \sharp P$ **and** $y' \sharp P'$ **and** $y' \sharp Q$
**and** $y' \sharp Q'$ **and** $y' \neq x'$ **and** $y' \neq y$ **and** $y' \neq a$
**by**(*generate-fresh name*, *auto simp add*: *fresh-prod*)
**from** *PTrans* ‹$y' \sharp P'$› **have** $P \longmapsto a{<}\nu y'{>} \prec ([(y, y')] \cdot P')$
**by**(*simp add*: *alphaBoundResidual*)
**moreover from** *QTrans* ‹$x' \sharp Q'$› **have** $Q \longmapsto a{<}x'{>} \prec ([(x, x')] \cdot Q')$
**by**(*simp add*: *alphaBoundResidual*)
**ultimately have** $P \parallel Q \longmapsto \tau \prec {<}\nu y'{>}(([(y, y')] \cdot P') \parallel (([(x, x')] \cdot Q')[x'::=y']))$
**using** ‹$y' \sharp P$› ‹$y' \sharp Q$› ‹$x' \sharp P$› ‹$x' \sharp Q$› ‹$y' \neq x'$› ‹$x' \neq a$› ‹$y' \neq a$›
‹$x' \sharp P'$› ‹$x' \sharp Q'$› ‹$y' \sharp P'$› ‹$y' \sharp Q'$›
**by**(*rule-tac Close2*) (*assumption* | *auto simp add*: *fresh-left calc-atm*)+
**moreover have** ${<}\nu y'{>}(([(y, y')] \cdot P') \parallel (([(x, x')] \cdot Q')[x'::=y'])) = {<}\nu y{>}(P' \parallel (Q'[x::=y]))$
**proof** −
**from** ‹$x' \sharp Q'$› **have** $([(x, x')] \cdot Q')[x'::=y'] = Q'[x::=y']$ **by**(*simp add*: *renaming name-swap*)
**moreover have** $y \sharp (([(y, y')] \cdot P') \parallel (Q'[x::=y']))$
**proof**(*case-tac* $y = x$)
**assume** $y = x$
**with** ‹$y' \sharp P'$› ‹$y' \neq y$› **show** *?thesis* **by**(*auto simp add*: *fresh-fact2 fresh-left calc-atm*)

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

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

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

**shows** $<\nu y>P \longmapsto a«x» \prec <\nu y>P'$
**proof** $-$
**obtain** $z$ **where** $z \sharp P$ **and** $z \sharp a$ **and** $z \neq y$ **and** $z \sharp P'$
**by**(*generate-fresh name*, *auto simp add*: *fresh-prod*)
**from** *PTrans* ‹$z \sharp P'$› **have** $P \longmapsto a«z» \prec ([(x, z)] \cdot P')$ **by**(*simp add*: *alphaBoundResidual*)
**with** ‹$z \sharp P$› ‹$z \sharp a$› ‹$z \neq y$› ‹$y \sharp a$› **have** $<\nu y>P \longmapsto a«z» \prec <\nu y>([(x, z)] \cdot P')$ **by**(*rule-tac ResB*) *auto*
**moreover have** $a«z» \prec <\nu y>([(x, z)] \cdot P') = a«x» \prec <\nu y>P'$
**proof** $-$
**from** ‹$z \sharp P'$› ‹$y \neq x$› **have** $x \sharp <\nu y>([(x, z)] \cdot P')$ **by**(*auto simp add*: *abs-fresh fresh-left calc-atm*)
**with** ‹$y \neq x$› ‹$z \neq y$› **show** *?thesis* **by**(*simp add*: *alphaBoundResidual name-swap calc-atm*)
**qed**
**ultimately show** *?thesis* **by** *simp*
**qed**

**lemma** *outputInduct*[*consumes 1*, *case-names Output Match Mismatch Sum1 Sum2 Par1 Par2 Res Bang*]:
**fixes** $P$ :: *pi*
**and** $a$ :: *name*
**and** $b$ :: *name*
**and** $P'$ :: *pi*
**and** $F$ :: *'a::fs-name* $\Rightarrow$ *pi* $\Rightarrow$ *name* $\Rightarrow$ *name* $\Rightarrow$ *pi* $\Rightarrow$ *bool*
**and** $C$ :: *'a::fs-name*

**assumes** *Trans*: $P \longmapsto a[b] \prec P'$
**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** −
**from** *Trans* **show** *?thesis*
**by**(*nominal-induct x2 == OutputR a b* $\prec$ *P' avoiding*: *C arbitrary*: *P' rule*: *transitions.strong-induct*,
*auto simp add*: *residualInject freeRes.inject intro*: *assms*)
**qed**

**lemma** *inputInduct*[*consumes 2*, *case-names Input Match Mismatch Sum1 Sum2 Par1 Par2 Res Bang*]:
**fixes** $P$ :: *pi*
**and** $a$ :: *name*
**and** $x$ :: *name*
**and** $P'$ :: *pi*
**and** $F$ :: $('a::fs\text{-}name) \Rightarrow pi \Rightarrow name \Rightarrow name \Rightarrow pi \Rightarrow bool$
**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** −
**from** $a$ ‹$x \sharp P$› **show** *?thesis*
**proof**(*nominal-induct* $x2 == a{<}x{>} \prec P'$ *avoiding*: $C\ a\ x\ P'$ *rule*: *transitions.strong-induct*)
**case**(*Tau P*)
**thus** *?case* **by**(*simp add*: *residualInject*)
**next**
**case**(*Input x a P C a' x' P'*)
**have** $x \sharp x'$ **by** *fact* **hence** $x \neq x'$ **by** *simp*
**moreover have** $a{<}x{>} \prec P = a'{<}x'{>} \prec P'$ **by** *fact*
**ultimately have** *aeqa'*: $a = a'$ **and** *PeqP'*: $P = [(x, x')] \cdot P'$
**by**(*simp add*: *residualInject freeRes.inject subject.inject name-abs-eq*)+

**have** $F\ C\ (a{<}x'{>}.([(x, x')] \cdot P))\ a\ x'\ ([(x, x')] \cdot P)$ **by**(*rule cInput*)
**moreover have** $x \sharp P'$ **by** *fact*
**ultimately show** *?case* **using** *PeqP' aeqa'* **by**(*simp add*: *alphaInput name-swap*)
**next**
**case**(*Output P a b*)
**thus** *?case* **by**(*simp add*: *residualInject*)
**next**
**case**(*Match P b Rs a x*)
**thus** *?case*
**by**(*force intro*: *cMatch simp add*: *residualInject*)
**next**
**case**(*Mismatch P Rs a b C a x*)
**thus** *?case*
**by**(*force intro*: *cMismatch simp add*: *residualInject*)
**next**
**case**(*Open P P' a b C a' x P'*)
**thus** *?case* **by**(*simp add*: *residualInject*)
**next**
**case**(*Sum1 P Q Rs C*)
**thus** *?case* **by**(*force intro*: *cSum1*)
**next**
**case**(*Sum2 P Q Rs C*)
**thus** *?case* **by**(*force intro*: *cSum2*)
**next**

**case**(*Par1B P a x P′ Q C a′ x′ P″*)
**have** $x \sharp x'$ **by** *fact* **hence** *xineqx′*: $x \neq x'$ **by** *simp*
**moreover have** *Eq*: $a«x» \prec (P' \parallel Q) = a'<x'> \prec P''$ **by** *fact*
**hence** *aeqa′*: $a = InputS\ a'$ **by**(*simp add*: *residualInject*)
**have** $x' \sharp P \parallel Q$ **by** *fact*
**hence** $x' \sharp P$ **and** $x' \sharp Q$ **by** *simp+*
**have** *P″eq*: $P'' = ([(x, x')] \cdot P') \parallel Q$
**proof** −
  **from** *Eq xineqx′* **have** $(P' \parallel Q) = [(x, x')] \cdot P''$
    **by**(*simp add*: *residualInject name-abs-eq*)
  **hence** $([(x, x')] \cdot (P' \parallel Q)) = P''$ **by** *simp*
  **with** ‹$x' \sharp Q$›‹$x \sharp Q$› **show** *?thesis* **by**(*simp add*: *name-fresh-fresh*)
**qed**

**have** $x \sharp P''$ **by** *fact*
**with** *P″eq* ‹$x \neq x'$› **have** $x' \sharp P'$ **by**(*simp add*: *name-fresh-left name-calc*)

**have** *PTrans*: $P \longmapsto a«x» \prec P'$ **by** *fact*
**with** ‹$x' \sharp P'$› *aeqa′* **have** $P \longmapsto a'<x'> \prec ([(x, x')] \cdot P')$
  **by**(*simp add*: *alphaBoundResidual*)
**moreover have** $\bigwedge C.\ F\ C\ P\ a'\ x'\ ([(x, x')] \cdot P')$
**proof** −
  **fix** *C*
  **have** $\bigwedge C\ a'\ x'\ P''.\ [\![a«x» \prec P' = a'<x'> \prec P''; x' \sharp P]\!] \Longrightarrow F\ C\ P\ a'\ x'\ P''$
**by** *fact*
  **moreover with** *aeqa′ xineqx′* ‹$x' \sharp P'$› **have** $a«x» \prec P' = a'<x'> \prec ([(x,$
$x')] \cdot P')$
    **by**(*simp add*: *residualInject name-abs-eq name-fresh-left name-calc*)
  **ultimately show** $F\ C\ P\ a'\ x'\ ([(x, x')] \cdot P')$ **using** ‹$x' \sharp P$› **by** *blast*
**qed**
**moreover from** *PTrans* ‹$x' \sharp P$› **have** $x' \sharp a$ **by**(*auto dest*: *freshBoundDeriva-*
*tive*)
**ultimately have** $F\ C\ (P \parallel Q)\ a'\ x'\ (([(x, x')] \cdot P') \parallel Q)$ **using** ‹$x' \sharp Q$›*aeqa′*
‹$x' \sharp P$›
  **by**(*rule-tac cPar1B*) *auto*
**with** *P″eq* **show** *?case* **by** *simp*
**next**
**case**(*Par1F P P′ Q α*)
**thus** *?case* **by**(*simp add*: *residualInject*)
**next**
**case**(*Par2B Q a x Q′ P C a′ x′ Q″*)
**have** $x \sharp x'$ **by** *fact* **hence** *xineqx′*: $x \neq x'$ **by** *simp*
**moreover have** *Eq*: $a«x» \prec (P \parallel Q') = a'<x'> \prec Q''$ **by** *fact*
**hence** *aeqa′*: $a = InputS\ a'$ **by**(*simp add*: *residualInject*)
**have** $x \sharp P$ **by** *fact*
**have** $x' \sharp P \parallel Q$ **by** *fact*
**hence** $x' \sharp P$ **and** $x' \sharp Q$ **by** *simp+*
**have** *Q″eq*: $Q'' = P \parallel ([(x, x')] \cdot Q')$
**proof** −

**from** *Eq xineqx'* **have** $(P \parallel Q') = [(x, x')] \cdot Q''$
**by**(*simp add*: *residualInject name-abs-eq*)
**hence** $([(x, x')] \cdot (P \parallel Q')) = Q''$ **by** *simp*
**with** ‹$x' \sharp P$› ‹$x \sharp P$› **show** *?thesis* **by**(*simp add*: *name-fresh-fresh*)
**qed**

**have** $x \sharp Q''$ **by** *fact*
**with** *Q''eq* ‹$x \neq x'$› **have** $x' \sharp Q'$ **by**(*simp add*: *name-fresh-left name-calc*)

**have** *QTrans*: $Q \longmapsto a«x» \prec Q'$ **by** *fact*
**with** ‹$x' \sharp Q'$› *aeqa'* **have** $Q \longmapsto a'<x'> \prec ([(x, x')] \cdot Q')$
**by**(*simp add*: *alphaBoundResidual*)
**moreover have** $\bigwedge C.\ F\ C\ Q\ a'\ x'\ ([(x, x')] \cdot Q')$
**proof** −
**fix** $C$
**have** $\bigwedge C\ a'\ x'\ Q''.\ [\![a«x» \prec Q' = a'<x'> \prec Q''; x' \sharp Q]\!] \Longrightarrow F\ C\ Q\ a'\ x'\ Q''$ **by** *fact*
**moreover with** *aeqa' xineqx'* ‹$x' \sharp Q'$› **have** $a«x» \prec Q' = a'<x'> \prec ([(x, x')] \cdot Q')$
**by**(*simp add*: *residualInject name-abs-eq name-fresh-left name-calc*)
**ultimately show** $F\ C\ Q\ a'\ x'\ ([(x, x')] \cdot Q')$ **using** ‹$x' \sharp Q$›*aeqa'* **by** *blast*
**qed**
**moreover from** *QTrans* ‹$x' \sharp Q$› **have** $x' \sharp a$ **by**(*force dest*: *freshBoundDerivative*)
**ultimately have** $F\ C\ (P \parallel Q)\ a'\ x'\ (P \parallel ([(x, x')] \cdot Q'))$ **using** ‹$x' \sharp P$› *aeqa'* ‹$x' \sharp Q$›
**by**(*rule-tac cPar2B*) *auto*
**with** *Q''eq* **show** *?case* **by** *simp*
**next**
**case**(*Par2F P P' Q* $\alpha$)
**thus** *?case* **by**(*simp add*: *residualInject*)
**next**
**case**(*Comm1 P P' Q Q' a b x*)
**thus** *?case* **by**(*simp add*: *residualInject*)
**next**
**case**(*Comm2 P P' Q Q' a b x*)
**thus** *?case* **by**(*simp add*: *residualInject*)
**next**
**case**(*Close1 P P' Q Q' a x y*)
**thus** *?case* **by**(*simp add*: *residualInject*)
**next**
**case**(*Close2 P P' Q Q' a x y*)
**thus** *?case* **by**(*simp add*: *residualInject*)
**next**
**case**(*ResB P a x P' y C a' x' P''*)
**have** $x \sharp x'$ **by** *fact* **hence** *xineqx'*: $x \neq x'$ **by** *simp*
**moreover have** *Eq*: $a«x» \prec (<\nu y>P') = a'<x'> \prec P''$ **by** *fact*
**hence** *aeqa'*: $a = InputS\ a'$ **by**(*simp add*: *residualInject*)
**have** $y \sharp x'$ **by** *fact* **hence** *yineqx'*: $y \neq x'$ **by** *simp*

**moreover have** $x' \sharp$ <$\nu y$>$P$ **by** *fact*
**ultimately have** $x' \sharp P$ **by**(*simp add*: *name-fresh-abs*)
**have** $y \neq x$ **and** *yineqa*: $y \sharp a$ **and** *yFreshC*: $y \sharp C$ **by** *fact+*

**have** $P''eq$: $P'' =$ <$\nu y$>($[(x, x')] \cdot P'$)
**proof** −
**from** *Eq xineqx'* **have** <$\nu y$>$P' = [(x, x')] \cdot P''$
**by**(*simp add*: *residualInject name-abs-eq*)
**hence** ($[(x, x')] \cdot$ (<$\nu y$>$P'$)) $= P''$ **by** *simp*
**with** *yineqx'* ‹$y \neq x$› **show** *?thesis* **by**(*simp add*: *name-fresh-fresh*)
**qed**

**have** $x \sharp P''$ **by** *fact*
**with** $P''eq$ ‹$y \neq x$› ‹$x \neq x'$› **have** $x' \sharp P'$ **by**(*simp add*: *name-fresh-left name-calc name-fresh-abs*)

**have** $P \longmapsto a$«$x$» $\prec P'$ **by** *fact*
**with** ‹$x' \sharp P'$› *aeqa'* **have** $P \longmapsto a'$<$x'$> $\prec$ ($[(x, x')] \cdot P'$)
**by**(*simp add*: *alphaBoundResidual*)
**moreover have** $\bigwedge C.\ F\ C\ P\ a'\ x'$ ($[(x, x')] \cdot P'$)
**proof** −
**fix** $C$
**have** $\bigwedge C\ a'\ x'\ P''.$ ⟦$a$«$x$» $\prec P' = a'$<$x'$> $\prec P''$; $x' \sharp P$⟧ $\Longrightarrow F\ C\ P\ a'\ x'\ P''$ **by** *fact*
**moreover with** *aeqa' xineqx'* ‹$x' \sharp P'$› **have** $a$«$x$» $\prec P' = a'$<$x'$> $\prec$ ($[(x, x')] \cdot P'$)
**by**(*simp add*: *residualInject name-abs-eq name-fresh-left name-calc*)
**ultimately show** $F\ C\ P\ a'\ x'$ ($[(x, x')] \cdot P'$) **using** ‹$x' \sharp P$› *aeqa'* **by** *blast*
**qed**
**ultimately have** $F\ C$ (<$\nu y$>$P$) $a'\ x'$ (<$\nu y$>($[(x, x')] \cdot P'$)) **using** *yineqx' yineqa yFreshC aeqa'*
**by**(*force intro*: *cResB*)
**with** $P''eq$ **show** *?case* **by** *simp*
**next**
**case**(*ResF P P' α y*)
**thus** *?case* **by**(*simp add*: *residualInject*)
**next**
**case**(*Bang P Rs*)
**thus** *?case* **by**(*force intro*: *cBang*)
**qed**
**qed**

**lemma** *boundOutputInduct*[*consumes 2*, *case-names Match Mismatch Open Sum1 Sum2 Par1 Par2 Res Bang*]:
**fixes** $P$ :: *pi*
**and** $a$ :: *name*
**and** $x$ :: *name*
**and** $P'$ :: *pi*
**and** $F$ :: (*'a::fs-name*) ⇒ *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** −
**from** $a$ ‹$x \sharp P$› **show** *?thesis*
**proof**(*nominal-induct x2* == $a{<}\nu x{>} \prec P'$ *avoiding*: $C\ a\ x\ P'$ *rule*: *transitions.strong-induct*)
**case**(*Tau P*)
**thus** *?case* **by**(*simp add*: *residualInject*)
**next**
**case**(*Input P a x*)
**thus** *?case* **by**(*simp add*: *residualInject*)
**next**
**case**(*Output P a b*)
**thus** *?case* **by**(*simp add*: *residualInject*)
**next**
**case**(*Match P Rs b C a x*)
**thus** *?case*
**by**(*force intro*: *cMatch simp add*: *residualInject*)
**next**
**case**(*Mismatch P Rs a b C c x*)
**thus** *?case*
**by**(*force intro*: *cMismatch simp add*: *residualInject*)
**next**
**case**(*Sum1 P Q Rs C*)
**thus** *?case* **by**(*force intro*: *cSum1*)

**next**
**case**(*Sum2 P Q Rs C*)
**thus** *?case* **by**(*force intro*: *cSum2*)
**next**
**case**(*Open P a b P′ C a′ x P′′*)
**have** $b \sharp x$ **by** *fact* **hence** *bineqx*: $b \neq x$ **by** *simp*
**moreover have** $a{<}\nu b{>} \prec P' = a'{<}\nu x{>} \prec P''$ **by** *fact*
**ultimately have** *aeqa′*: $a{=}a'$ **and** *P′eqP′′*: $P'' = [(b, x)] \cdot P'$
**by**(*simp add*: *residualInject name-abs-eq*)+
**have** $x \sharp {<}\nu b{>}P$ **by** *fact*
**with** *bineqx* **have** $x \sharp P$ **by**(*simp add*: *name-fresh-abs*)
**have** *aineqb*: $a \neq b$ **by** *fact*

**have** *PTrans*: $P \longmapsto a[b] \prec P'$ **by** *fact*
**with** ‹$x \sharp P$› **have** *xineqa*: $x \neq a$ **by**(*force dest*: *freshFreeDerivative*)
**from** *PTrans* **have** $([(b, x)] \cdot P) \longmapsto [(b, x)] \cdot (a[b] \prec P')$ **by**(*rule transitions.eqvt*)
**with** *P′eqP′′ xineqa aineqb* **have** *Trans*: $([(b, x)] \cdot P) \longmapsto a[x] \prec P''$
**by**(*auto simp add*: *name-calc*)
**hence** $F\ C\ ({<}\nu x{>}([(b, x)] \cdot P))\ a\ x\ P''$ **using** *xineqa* **by**(*blast intro*: *cOpen*)
**with** ‹$x \sharp P$› *aeqa′* **show** *?case* **by**(*simp add*: *alphaRes*)
**next**
**case**(*Par1B P a x P′ Q C a′ x′ P′′*)
**have** $x \sharp x'$ **by** *fact* **hence** *xineqx′*: $x \neq x'$ **by** *simp*
**moreover have** *Eq*: $a«x» \prec (P' \parallel Q) = a'{<}\nu x'{>} \prec P''$ **by** *fact*
**hence** *aeqa′*: $a = BoundOutputS\ a'$ **by**(*simp add*: *residualInject*)
**have** $x \sharp Q$ **by** *fact*
**have** $x' \sharp P \parallel Q$ **by** *fact*
**hence** $x' \sharp P$ **and** $x' \sharp Q$ **by** *simp*+
**have** *P′′eq*: $P'' = ([(x, x')] \cdot P') \parallel Q$
**proof** −
**from** *Eq xineqx′* **have** $(P' \parallel Q) = [(x, x')] \cdot P''$
**by**(*simp add*: *residualInject name-abs-eq*)
**hence** $([(x, x')] \cdot (P' \parallel Q)) = P''$ **by** *simp*
**with** ‹$x' \sharp Q$›‹$x \sharp Q$› **show** *?thesis* **by**(*simp add*: *name-fresh-fresh*)
**qed**

**have** $x \sharp P''$ **by** *fact*
**with** *P′′eq* ‹$x \neq x'$› **have** $x' \sharp P'$ **by**(*simp add*: *name-fresh-left name-calc*)

**have** $P \longmapsto a«x» \prec P'$ **by** *fact*
**with** ‹$x' \sharp P'$› *aeqa′* **have** $P \longmapsto a'{<}\nu x'{>} \prec ([(x, x')] \cdot P')$
**by**(*simp add*: *alphaBoundResidual*)
**moreover have** $\bigwedge C.\ F\ C\ P\ a'\ x'\ ([(x, x')] \cdot P')$
**proof** −
**fix** *C*
**have** $\bigwedge C\ a'\ x'\ P''.\ [\![a«x» \prec P' = a'{<}\nu x'{>} \prec P''; x' \sharp P]\!] \Longrightarrow F\ C\ P\ a'\ x'\ P''$ **by** *fact*
**moreover with** *aeqa′ xineqx′* ‹$x' \sharp P'$› **have** $a«x» \prec P' = a'{<}\nu x'{>} \prec ([(x,$

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

**have** $x \sharp Q''$ **by** *fact*
**with** *Q''eq* ‹$x \neq x'$› **have** $x' \sharp Q'$ **by**(*simp add*: *name-fresh-left name-calc*)

**have** $Q \longmapsto a«x» \prec Q'$ **by** *fact*
**with** ‹$x' \sharp Q'$› *aeqa'* **have** $Q \longmapsto a'<\nu x'> \prec ([(x, x')] \cdot Q')$
**by**(*simp add*: *alphaBoundResidual*)
**moreover have** $\bigwedge C.\ F\ C\ Q\ a'\ x'\ ([(x, x')] \cdot Q')$
**proof** −
**fix** $C$
**have** $\bigwedge C\ a'\ x'\ Q''.\ [\![a«x» \prec Q' = a'<\nu x'> \prec Q''; x' \sharp Q]\!] \Longrightarrow F\ C\ Q\ a'\ x'$
$Q''$ **by** *fact*
**moreover with** *aeqa' xineqx'* ‹$x' \sharp Q'$› **have** $a«x» \prec Q' = a'<\nu x'> \prec ([(x,$
$x')] \cdot Q')$
**by**(*simp add*: *residualInject name-abs-eq name-fresh-left name-calc*)
**ultimately show** $F\ C\ Q\ a'\ x'\ ([(x, x')] \cdot Q')$ **using** ‹$x' \sharp Q$›*aeqa'* **by** *blast*
**qed**
**ultimately have** $F\ C\ (P \parallel Q)\ a'\ x'\ (P \parallel ([(x, x')] \cdot Q'))$ **using** ‹$x' \sharp P$›
**by**(*blast intro*: *cPar2B*)
**with** *Q''eq* **show** *?case* **by** *simp*
**next**
**case**(*Par2F P P' Q* $\alpha$)
**thus** *?case* **by**(*simp add*: *residualInject*)
**next**

**case**(*Comm1 P P′ Q Q′ a b x*)
**thus** *?case* **by**(*simp add*: *residualInject*)
**next**
**case**(*Comm2 P P′ Q Q′ a b x*)
**thus** *?case* **by**(*simp add*: *residualInject*)
**next**
**case**(*Close1 P P′ Q Q′ a x y*)
**thus** *?case* **by**(*simp add*: *residualInject*)
**next**
**case**(*Close2 P P′ Q Q′ a x y*)
**thus** *?case* **by**(*simp add*: *residualInject*)
**next**
**case**(*ResB P a x P′ y C a′ x′ P″*)
**have** $x \sharp x'$ **by** *fact* **hence** *xineqx′*: $x \neq x'$ **by** *simp*
**moreover have** *Eq*: $a«x» \prec (<\nu y>P') = a'<\nu x'> \prec P''$ **by** *fact*
**hence** *aeqa′*: $a = BoundOutputS\ a'$ **by**(*simp add*: *residualInject*)
**have** $y \sharp x'$ **by** *fact* **hence** *yineqx′*: $y \neq x'$ **by** *simp*
**moreover have** $x' \sharp <\nu y>P$ **by** *fact*
**ultimately have** $x' \sharp P$ **by**(*simp add*: *name-fresh-abs*)
**have** $y \neq x$ **and** $y \sharp a$ **and** *yFreshC*: $y \sharp C$ **by** *fact+*

**have** *P″eq*: $P'' = <\nu y>([(x, x')] \cdot P')$
**proof** −
**from** *Eq xineqx′* **have** $<\nu y>P' = [(x, x')] \cdot P''$
**by**(*simp add*: *residualInject name-abs-eq*)
**hence** $([(x, x')] \cdot (<\nu y>P')) = P''$ **by** *simp*
**with** *yineqx′* ‹$y \neq x$› **show** *?thesis* **by**(*simp add*: *name-fresh-fresh*)
**qed**

**have** $x \sharp P''$ **by** *fact*
**with** *P″eq* ‹$y \neq x$› ‹$x \neq x'$› **have** $x' \sharp P'$ **by**(*simp add*: *name-fresh-left name-calc name-fresh-abs*)

**have** $P \longmapsto a«x» \prec P'$ **by** *fact*
**with** ‹$x' \sharp P'$› *aeqa′* **have** $P \longmapsto a'<\nu x'> \prec ([(x, x')] \cdot P')$
**by**(*simp add*: *alphaBoundResidual*)
**moreover have** $\bigwedge C.\ F\ C\ P\ a'\ x'\ ([(x, x')] \cdot P')$
**proof** −
**fix** $C$
**have** $\bigwedge C\ a'\ x'\ P''.\ [\![a«x» \prec P' = a'<\nu x'> \prec P''; x' \sharp P]\!] \Longrightarrow F\ C\ P\ a'\ x'\ P''$ **by** *fact*
**moreover with** *aeqa′ xineqx′* ‹$x' \sharp P'$› **have** $a«x» \prec P' = a'<\nu x'> \prec ([(x, x')] \cdot P')$
**by**(*simp add*: *residualInject name-abs-eq name-fresh-left name-calc*)
**ultimately show** $F\ C\ P\ a'\ x'\ ([(x, x')] \cdot P')$ **using** ‹$x' \sharp P$› *aeqa′* **by** *blast*
**qed**
**ultimately have** $F\ C\ (<\nu y>P)\ a'\ x'\ (<\nu y>([(x, x')] \cdot P'))$ **using** *yineqx′* ‹$y \sharp a$› *yFreshC aeqa′*
**by**(*force intro*: *cResB*)

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

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

**assumes** *Trans*: $P \longmapsto \tau \prec P'$
**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** $\bigwedge P\ a\ x\ P'\ Q\ b\ Q'\ C.\ [\![P \longmapsto (BoundR\ (InputS\ a)\ x\ P');\ Q \longmapsto OutputR\ a\ b \prec Q';\ x \sharp P;\ x \sharp Q;\ x \sharp C]\!] \Longrightarrow F\ C\ (P \parallel Q)\ (P'[x::=b] \parallel Q')$
**and** $\bigwedge P\ a\ b\ P'\ Q\ x\ Q'\ C.\ [\![P \longmapsto OutputR\ a\ b \prec P';\ Q \longmapsto (BoundR\ (InputS\ a)\ x\ Q');\ x \sharp P;\ x \sharp Q;\ x \sharp C]\!] \Longrightarrow F\ C\ (P \parallel Q)\ (P' \parallel Q'[x::=b])$
**and** $\bigwedge P\ a\ x\ P'\ Q\ y\ Q'\ C.\ [\![P \longmapsto (BoundR\ (InputS\ a)\ x\ P');\ Q \longmapsto a{<}\nu y{>} \prec Q';\ x \sharp P;\ x \sharp Q;\ x \sharp C;\ y \sharp P;\ y \sharp Q;\ y \sharp C;\ x \neq y]\!] \Longrightarrow F\ C\ (P \parallel Q)\ ({<}\nu y{>}(P'[x::=y] \parallel Q'))$
**and** $\bigwedge P\ a\ y\ P'\ Q\ x\ Q'\ C.\ [\![P \longmapsto a{<}\nu y{>} \prec P';\ Q \longmapsto (BoundR\ (InputS\ a)\ x\ Q');\ x \sharp P;\ x \sharp Q;\ x \sharp C;\ y \sharp P;\ y \sharp Q;\ y \sharp C;\ x \neq y]\!] \Longrightarrow F\ C\ (P \parallel Q)\ ({<}\nu y{>}(P' \parallel Q'[x::=y]))$
**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** −
**from** *Trans* **show** *?thesis*
**by**(*nominal-induct x2*$==\tau \prec P'$ *avoiding*: $C$ *arbitrary*: $P'$ *rule*: *transitions.strong-induct*, *auto simp add*: *residualInject intro*: *assms*)
**qed**

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

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

**shows** $Prop\ \alpha\ P'$
**using** *assms*
**by**(*erule-tac tauCases′*, *auto simp add*: *pi.inject residualInject*)

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

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

**shows** $Prop\ \alpha\ P'$
**using** *assms*
**by**(*erule-tac outputCases′*, *auto simp add*: *residualInject*)

**lemma** *zeroTrans*[*dest*]:

**fixes** *Rs* :: *residual*

**assumes** $\mathbf{0} \longmapsto Rs$

**shows** *False*
**using** *assms*
**by**(*induct rule*: *nilCases′*, *auto*)

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

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

**shows** *False*
**using** *assms*
**by**(*induct rule*: *resCases′*, *auto simp add*: *pi.inject alpha′*)

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

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

**shows** *False*
**using** *assms*
**by**(*induct rule*: *matchCases′*, *auto*)

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

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

**shows** *False*
**using** *assms*
**by**(*induct rule*: *mismatchCases′*, *auto*)

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

**assumes** *Input*: $a<x>.P \longmapsto b«y» \prec yP'$

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

**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*
**using** *assms*
**by** − (*ind-cases* $\tau.(P) \longmapsto a«x» \prec P'$)

**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*
**using** *assms*
**by** − (*ind-cases* $\tau.(P) \longmapsto a[b] \prec P'$, *auto simp add*: *residualInject*)

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

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

  **shows** *False*
**using** *assms*
**by** − (*ind-cases* $a{<}x{>}.P \longmapsto \alpha \prec P'$)

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

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

  **shows** *False*
**using** *assms*
**by** − (*ind-cases* $a{<}x{>}.P \longmapsto b{<}\nu y{>} \prec P'$, *auto simp add*: *residualInject*)

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

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

  **shows** *False*
**using** *assms*
**by** − (*ind-cases* $a\{b\}.P \longmapsto \tau \prec P'$, *auto simp add*: *residualInject*)

**lemma** *outputBoundTrans*[*dest*]:
  **fixes** $a$ :: *name*
  **and** $b$ :: *name*
  **and** $P$ :: *pi*
  **and** $c$ :: *subject*
  **and** $x$ :: *name*
  **and** $P'$ :: *pi*

  **assumes** $a\{b\}.P \longmapsto c«x» \prec P'$

**shows** *False*
**using** *assms*
**by** − (*ind-cases* $a\{b\}.P \longmapsto c«x» \prec P'$)

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

**lemma** *outputFreshTrans*[*dest*]:
**fixes** $a$ :: *name*
**and** $b$ :: *name*
**and** $P$ :: *pi*
**and** $\alpha$ :: *freeRes*
**and** $P'$ :: *pi*

**assumes** $a\{b\}.P \longmapsto \alpha \prec P'$
**and** $a \sharp \alpha \vee b \sharp \alpha$

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

**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*
**using** *assms*
**by** − (*ind-cases* $a<x>.P \longmapsto b«y» \prec P'$, *auto simp add*: *residualInject pi.inject*)

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

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

**shows** *False*
**using** *assms*
**by** − (*ind-cases* $<\nu x>\tau.(P) \longmapsto a«y» \prec P'$, *auto simp add*: *residualInject pi.inject alpha'*)

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

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

**shows** *False*
**using** *assms*
**by** − (*ind-cases* $<\nu x>\tau.(P) \longmapsto a[b] \prec P'$, *auto simp add*: *residualInject pi.inject alpha'*)

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

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

**shows** *False*
**using** *assms*
**by** − (*ind-cases* $<\nu x>a<y>.P \longmapsto \alpha \prec P'$, *auto simp add*: *pi.inject residualInject alpha'*)

**lemma** *resInputBoundOutputTrans*[*dest*]:
**fixes** $x$ :: *name*
**and** $a$ :: *name*
**and** $y$ :: *name*
**and** $P$ :: *pi*

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

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

**shows** *False*
**using** *assms*
**by** − (*ind-cases* $<\nu x>a<y>.P \longmapsto b<\nu z> \prec P'$, *auto simp add*: *pi.inject residualInject alpha'*)

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

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

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

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

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

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

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

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

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

**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** $<\nu x>x<y>.P \longmapsto Rs \Longrightarrow False$
**apply**(*ind-cases* $<\nu x>x\{b\}.P \longmapsto Rs$, *auto simp add*: *pi.inject alpha′ calc-atm*)
**by**(*ind-cases* $<\nu x>x<y>.P \longmapsto Rs$, *auto simp add*: *pi.inject alpha′ calc-atm abs-fresh fresh-left*)

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

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

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

**lemma** *mismatchCases*[*consumes 1*, *case-names cMismatch*]:
**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** $F\ a\ b$
**using** *assms*
**by**(*induct rule*: *mismatchCases′*, *auto*)

**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*
**using** *assms*
**by**(*induct rule*: *sumCases'*, *auto*)

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

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

**assumes** $P \parallel Q \longmapsto a«x» \prec PQ'$
**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** −
**note** *assms*
**moreover from** ‹$P \parallel Q \longmapsto a«x» \prec PQ'$› ‹$x \sharp P$› ‹$x \sharp Q$› **have** $x \sharp a$
**by**(*force dest*: *freshBoundDerivative*)
**moreover obtain** $y$::*name* **where** $y \neq x$ **and** $y \sharp P$ **and** $y \sharp Q$ **and** $y \sharp a$ **and** $y \sharp PQ'$
**by**(*generate-fresh name*, *auto simp add*: *fresh-prod*)
**moreover obtain** $z$::*name* **where** $z \neq y$ **and** $z \neq x$ **and** $z \sharp P$ **and** $z \sharp Q$ **and**

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

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

**assumes** *Trans*: $P \parallel Q \longmapsto \alpha \prec PQ'$
**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*: $\bigwedge P'\ Q'\ a\ b\ x.\ [\![P \longmapsto a[b] \prec P';\ Q \longmapsto a{<}x{>} \prec Q';\ x \sharp P;\ x \sharp Q;\ x \neq a;\ x \neq b;\ x \sharp P';\ x \sharp C;\ \alpha = \tau]\!] \Longrightarrow F\ (\tau)\ (P' \parallel Q'[x{::=}b])$
**and** *icClose1*: $\bigwedge P'\ Q'\ a\ x\ y.\ [\![P \longmapsto a{<}x{>} \prec P';\ Q \longmapsto a{<}\nu y{>} \prec Q';\ x \sharp P;\ x \sharp Q;\ x \neq a;\ x \neq y;\ x \sharp Q';\ y \sharp P;\ y \sharp Q;\ y \neq a;\ y \sharp P';\ x \sharp C;\ y \sharp C;\ \alpha = \tau]\!] \Longrightarrow$
$F\ (\tau)\ ({<}\nu y{>}(P'[x{::=}y] \parallel Q'))$
**and** *icClose2*: $\bigwedge P'\ Q'\ a\ x\ y.\ [\![P \longmapsto a{<}\nu y{>} \prec P';\ Q \longmapsto a{<}x{>} \prec Q';\ x \sharp P;\ x \sharp Q;\ x \neq a;\ x \neq y;\ x \sharp P';\ y \sharp P;\ y \sharp Q;\ y \neq a;\ y \sharp Q';\ x \sharp C;\ y \sharp C;\ \alpha = \tau]\!] \Longrightarrow$
$F\ (\tau)\ ({<}\nu y{>}(P' \parallel Q'[x{::=}y]))$

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

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

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

 **assumes** $<\nu x>P \longmapsto \alpha \prec xP'$
 **and** $\bigwedge P'. [\![P \longmapsto \alpha \prec P'; x \sharp \alpha]\!] \Longrightarrow F\ (<\nu x>P')$

 **shows** $F\ xP'$
**proof** −
 **note** *assms*
 **moreover from** ‹$<\nu x>P \longmapsto \alpha \prec xP'$› **have** $x \sharp \alpha$ **and** $x \sharp xP'$
  **by**(*force dest*: *freshFreeDerivative simp add*: *abs-fresh*)+
 **moreover obtain** $y$::*name* **where** $y \neq x$ **and** $y \sharp P$ **and** $y \sharp \alpha$ **and** $y \sharp xP'$
  **by**(*generate-fresh name*, *auto simp add*: *fresh-prod*)
 **moreover obtain** $z$::*name* **where** $z \neq y$ **and** $z \neq x$ **and** $z \sharp P$ **and** $z \sharp \alpha$ **and** $z \sharp xP'$
  **by**(*generate-fresh name*, *auto simp add*: *fresh-prod*)
 **ultimately show** *?thesis*
  **by**(*cases rule*: *transitions.strong-cases*[**where** $x$=$y$ **and** $b$=$y$ **and** $xa$=$y$ **and** $xb$=$y$ **and** $xc$=$y$ **and** $xd$=$y$ **and** $xe$=$y$
   **and** $xf$=$y$ **and** $xg$=$y$ **and** $y$=$z$ **and** $ya$=$z$ **and** $yb$=$z$ **and** $yc$=$x$])
  (*auto simp add*: *pi.inject residualInject alpha abs-fresh fresh-prod*)+
**qed**

**lemma** *resCasesB*[*consumes 3*, *case-names cOpen cRes*]:
 **fixes** $x$ :: *name*
 **and** $P$ :: *pi*
 **and** $a$ :: *subject*
 **and** $y$ :: *name*
 **and** $yP'$ :: *pi*
 **and** $C$ :: *'a::fs-name*

 **assumes** *Trans*: $<\nu y>P \longmapsto a«x» \prec yP'$
 **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** −

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

**lemma** *bangInduct*[*consumes 1*, *case-names cPar1B cPar1F cPar2B cPar2F cComm1 cComm2 cClose1 cClose2 cBang*]:
**fixes** $F$ :: $'a$::*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*: $\bigwedge a\ y\ P'\ x\ P''\ C.\ [\![P \longmapsto a<\nu y> \prec P';\ !P \longmapsto a<x> \prec P'';\ y$

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

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

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

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

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

end

theory Late-Semantics1
  imports Late-Semantics
begin

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

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

end

theory Rel
  imports Agent
begin

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

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

**assumes** *eqvt Rel*
**and** (*P*, *Q*) ∈ *Rel*

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

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

**assumes** *eqvt Rel*
**and** (*perm* • *P*, *perm* • *Q*) ∈ *Rel*

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

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

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

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

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

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

**shows** *eqvt* (*Rel* ∪ *Rel'*)

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

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

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

  **assumes** *eqvtRel*: *eqvt Rel*

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

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

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

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

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

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

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

```
  thus ((P[<σ>])[<σ'>], (Q[<σ>])[<σ'>]) ∈ Rel
    by simp
qed

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

    from eqvtRel R have (perm • R, perm • T) ∈ Rel
      by(rule eqvtRelI)

    with BR show (perm • (R ∥ P), perm • (T ∥ Q)) ∈ bangRel Rel
      by(force intro: BRPar)
  next
    fix P Q a
    assume (perm • P, perm • Q) ∈ bangRel Rel
    thus (perm • <νa>P, perm • <νa>Q) ∈ bangRel Rel
      by(force intro: BRRes)
  qed
```

**qed**

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

**lemma** *BRParCases*[*consumes 1*, *case-names BRPar*]:
**fixes** $P$ :: *pi*
**and** $Q$ :: *pi*
**and** *Rel* :: (*pi* × *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** $F\ P$
**using** *assms*
**by**(*induct rule*: *BRParCases'*, *auto simp add*: *pi.inject*)

**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'*
**using** *assms*
**by**(*induct rule*: *bangRel.induct*) (*auto intro*: *BRBang BRPar BRRes*)

**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** –

```
  from A show ?thesis
  proof(induct)
    fix P Q
    assume (P, Q) ∈ Rel
    hence (Q, P) ∈ Rel by(rule Sym)
    thus (!Q, !P) ∈ bangRel Rel by(rule BRBang)
  next
    fix P Q R T
    assume RRelT: (R, T) ∈ Rel
    assume IH: (Q, P) ∈ bangRel Rel
    from RRelT have (T, R) ∈ Rel by(rule Sym)
    thus (T ‖ Q, R ‖ P) ∈ bangRel Rel using IH by(rule BRPar)
  next
    fix P Q a
    assume (Q, P) ∈ bangRel Rel
    thus (<νa>Q, <νa>P) ∈ bangRel Rel by(rule BRRes)
  qed
qed

primrec resChain :: name list ⇒ pi ⇒ pi where
   base: resChain [] P = P
 | step: resChain (x#xs) P = <νx>(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)
by(induct-tac lst, auto)

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

  assumes a ♯ (lst, P)

  shows a ♯ (resChain lst P)
using assms apply(induct-tac lst)
apply(simp add: fresh-prod)
by(simp add: fresh-prod name-fresh-abs)

end

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

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

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

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

**assumes** *P* ⇝[*A*] *P*′
**and** *A* ⊆ *B*

**shows** *P* ⇝[*B*] *P*′
**using** *assms*
**apply**(*auto simp add*: *simulation-def derivative-def*)
**by**(*case-tac a*) *fastforce*+

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

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

**shows** *derivative P Q a x B*
**using** *assms*
**by**(*case-tac a*, *auto simp add*: *derivative-def*)

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

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

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

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

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

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

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

**shows** $y \sharp ()$
**by**(*auto simp add*: *fresh-def supp-unit*)

**lemma** *simCasesCont*[*consumes 1*, *case-names Bound Free*]:
**fixes** *P* :: *pi*
**and** *Q* :: *pi*
**and** *Rel* :: (*pi* × *pi*) *set*
**and** *C* :: *'a::fs-name*

**assumes** *Eqvt*: *eqvt Rel*
**and** *Bound*: $\bigwedge a\ x\ Q'.\ [\![Q \longmapsto a«x» \prec Q';\ x \sharp P;\ x \sharp Q;\ x \sharp a;\ x \sharp C]\!] \Longrightarrow \exists P'.\ P \longmapsto a«x» \prec P' \wedge$ *derivative* $P'\ Q'\ a\ x$ *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** $P \leadsto[Rel]\ Q$
**using** *Free*

**proof**(*auto simp add*: *simulation-def*)
  **fix** $a\ x\ Q'$
  **assume** $(x::name) \sharp P$
  **assume** *Trans*: $Q \longmapsto a«x» \prec Q'$

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

  **from** *Trans* ‹$y \sharp Q'$› **have** $Q \longmapsto a«y» \prec [(x, y)] \bullet Q'$ **by**(*simp add*: *alphaBoundResidual*)
  **hence** $\exists P'.\ P \longmapsto a«y» \prec P' \wedge$ *derivative* $P'$ $([(x, y)] \bullet Q')$ $a\ y$ *Rel*
    **using** ‹$y \sharp P$› ‹$y \sharp Q$› ‹$y \sharp a$› ‹$y \sharp C$›
    **by**(*rule Bound*)
  **then obtain** $P'$ **where** *PTrans*: $P \longmapsto a«y» \prec P'$ **and** *PDer*: *derivative* $P'$ $([(x, y)] \bullet Q')$ $a\ y$ *Rel*
    **by** *blast*

  **from** *PTrans* ‹$x \sharp P$› ‹$y \neq x$› **have** $x \sharp P'$ **by**(*force intro*: *freshBoundDerivative*)
  **with** *PTrans* **have** $P \longmapsto a«x» \prec [(x, y)] \bullet P'$ **by**(*simp add*: *alphaBoundResidual name-swap*)
  **moreover have** *derivative* $([(x, y)] \bullet P')$ $Q'$ $a\ x$ *Rel*
  **proof** −
    **from** *PDer Eqvt* **have** *derivative* $([(x, y)] \bullet P')$ $([(x, y)] \bullet [(x, y)] \bullet Q')$ $a$ $([(x, y)] \bullet y)$ *Rel*
      **by**(*rule derivativeEqvtI2*)
    **with** ‹$y \neq x$› **show** *?thesis* **by**(*simp add*: *name-calc*)
  **qed**
  **ultimately show** $\exists P'.\ P \longmapsto a«x» \prec P' \wedge$ *derivative* $P'$ $Q'$ $a\ x$ *Rel* **by** *blast*
**qed**

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

  **assumes** *Bound*: $\bigwedge a\ y\ Q'.\ [\![Q \longmapsto a«y» \prec Q';\ y \sharp P]\!] \Longrightarrow \exists P'.\ P \longmapsto a«y» \prec P' \wedge$ *derivative* $P'$ $Q'$ $a\ y$ *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** $P \leadsto[Rel]\ Q$
**using** *assms*
**by**(*auto simp add*: *simulation-def*)

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

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

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

**shows** $P \rightsquigarrow[Rel]\ {<}\nu x{>}Q$
**using** *Eqvt*
**proof**(*induct rule*: *simCasesCont*[**where** $C{=}(C,\ x,\ Q)$])
**case**(*Bound a y Q'*)
**have** $y \sharp (C,\ x,\ Q)$ **by** *fact*
**hence** *yFreshC*: $y \sharp C$ **and** *yineqx*: $y \neq x$ **and** $y \sharp Q$
**by**(*simp add*: *fresh-prod*)+
**have** ${<}\nu x{>}Q \longmapsto a«y» \prec Q'$ **by** *fact*
**thus** *?case* **using** *yineqx* ‹$y \sharp Q$›
**proof**(*induct rule*: *resCasesB*)
**case**(*cOpen a' Q'*)
**have** $Q \longmapsto a'[x] \prec Q'$ **and** $a' \neq x$ **by** *fact*+
**then obtain** $P'$ **where** *PTrans*: $P \longmapsto a'{<}\nu x{>} \prec P'$ **and** *P'RelQ'*: $(P',\ Q') \in Rel$ **by**(*force dest*: *BoundO*)

**from** *PTrans* ‹$y \sharp P$› *yineqx* **have** $y \sharp P'$ **by**(*force dest*: *freshBoundDerivative*)
**with** *PTrans* **have** $P \longmapsto a'{<}\nu y{>} \prec ([(x,\ y)] \cdot P')$ **by**(*simp add*: *alphaBoundResidual*)
**moreover from** *P'RelQ' Eqvt* **have** $([(x,\ y)] \cdot P',\ [(x,\ y)] \cdot Q') \in Rel$ **by**(*auto simp add*: *eqvt-def*)
**ultimately show** *?case* **by**(*force simp add*: *derivative-def name-swap*)
**next**
**case**(*cRes Q'*)
**have** $Q \longmapsto a«y» \prec Q'$ **and** $x \sharp a$ **by** *fact*+
**with** *yineqx yFreshC* **show** *?case* **by**(*force dest*: *BR*)
**qed**
**next**
**case**(*Free* $\alpha$ *Q'*)
**have** ${<}\nu x{>}Q \longmapsto \alpha \prec Q'$ **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *resCasesF*)
**case**(*cRes Q'*)
**have** $Q \longmapsto \alpha \prec Q'$ **and** $x \sharp \alpha$ **by** *fact*+
**thus** *?case* **by**(*rule BF*)
**qed**
**qed**

**lemma** *simE*:
**fixes** $P$ :: *pi*

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

**assumes** $P \rightsquigarrow[Rel]\ Q$

**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** $Q \longmapsto \alpha \prec Q' \Longrightarrow \exists P'.\ P \longmapsto \alpha \prec P' \wedge (P', Q') \in Rel$
**using** *assms* **by**(*simp add*: *simulation-def*)+

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

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

**shows** $(perm \cdot P) \rightsquigarrow[Rel']\ (perm \cdot Q)$
**proof**(*induct rule*: *simCases*)
**case**(*Bound a y* $Q'$)
**have** *QTrans*: $(perm \cdot Q) \longmapsto a«y» \prec Q'$ **and** *yFreshP*: $y \sharp perm \cdot P$ **by** *fact*+

**from** *QTrans* **have** $(rev\ perm \cdot (perm \cdot Q)) \longmapsto rev\ perm \cdot (a«y» \prec Q')$
**by**(*rule transitions.eqvt*)
**hence** $Q \longmapsto (rev\ perm \cdot a)«(rev\ perm \cdot y)» \prec (rev\ perm \cdot Q')$
**by**(*simp add*: *name-rev-per*)
**moreover from** *yFreshP* **have** $(rev\ perm \cdot y) \sharp P$ **by**(*simp add*: *name-fresh-left*)
**ultimately have** $\exists P'.\ P \longmapsto (rev\ perm \cdot a)«rev\ perm \cdot y» \prec P' \wedge derivative\ P'\ (rev\ perm \cdot Q')\ (rev\ perm \cdot a)\ (rev\ perm \cdot y)\ Rel$ **using** *Sim*
**by**(*force intro*: *simE*)
**then obtain** $P'$ **where** *PTrans*: $P \longmapsto (rev\ perm \cdot a)«rev\ perm \cdot y» \prec P'$ **and** *Pderivative*: $derivative\ P'\ (rev\ perm \cdot Q')\ (rev\ perm \cdot a)\ (rev\ perm \cdot y)\ Rel$ **by** *blast*

**from** *PTrans* **have** $(perm \cdot P) \longmapsto perm \cdot ((rev\ perm \cdot a)«rev\ perm \cdot y» \prec P')$ **by**(*rule transitions.eqvt*)
**hence** *L1*: $(perm \cdot P) \longmapsto a«y» \prec (perm \cdot P')$ **by**(*simp add*: *name-per-rev*)
**from** *Pderivative RelRel'* **have** $derivative\ P'\ (rev\ perm \cdot Q')\ (rev\ perm \cdot a)\ (rev\ perm \cdot y)\ Rel'$
**by**(*rule derivativeMonotonic*)
**hence** $derivative\ (perm \cdot P')\ (perm \cdot (rev\ perm \cdot Q'))\ (perm \cdot (rev\ perm \cdot a))\ (perm \cdot (rev\ perm \cdot y))\ Rel'$ **using** *EqvtRel'*
**by**(*rule derivativeEqvtI*)

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

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

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

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

**assumes** $Id \subseteq Rel$

**shows** *derivative P P a x Rel*
**using** *assms*
**apply**(*cases a*)
**by**(*auto simp add*: *derivative-def*)

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

**assumes** $Id \subseteq Rel$

**shows** $P \leadsto[Rel]\ P$
**using** *assms*
**by**(*auto simp add*: *simulation-def derivativeReflexive*)

```
lemma transitive:
  fixes P      :: pi
  and   Q      :: pi
  and   R      :: pi
  and   Rel    :: (pi × 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
using Eqvt'
proof(induct rule: simCasesCont[where C=Q])
  case(Bound a x R')
  have RTrans: R ⟼ a«x» ≺ R' by fact

  from ‹x ♯ Q› QSimR RTrans obtain Q' where QTrans: Q ⟼ a«x» ≺ Q'
                                      and QDer: derivative Q' R' a x Rel'
    by(blast dest: simE)
  with QTrans ‹x ♯ P› PSimQ obtain P' where PTrans: P ⟼ a«x» ≺ P'
                                      and PDer: derivative P' Q' a x Rel
    by(blast dest: simE)
  moreover from PDer QDer Trans have derivative P' R' a x Rel''
    by(cases a) (auto simp add: derivative-def)
  ultimately show ?case by blast
next
  case(Free α R')
  have RTrans: R ⟼ α ≺ R' by fact
  with QSimR obtain Q' where QTrans: Q ⟼ α ≺ Q'
                       and Q'RelR': (Q', R') ∈ Rel'
    by(blast dest: simE)
  from QTrans PSimQ obtain P' where PTrans: P ⟼ α ≺ P'
                              and P'RelQ': (P', Q') ∈ Rel
    by(blast dest: simE)
  from P'RelQ' Q'RelR' Trans have (P', R') ∈ Rel'' by blast
  with PTrans show ?case by blast
qed

end

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

lemma monoAux: A ⊆ B ⟹ P ⇝[A] Q ⟶ P ⇝[B] Q
```

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

coinductive-set bisim :: (pi × pi) set
where
  step: ⟦P ↝[bisim] Q; (Q, P) ∈ bisim⟧ ⟹ (P, Q) ∈ bisim
monos monoAux

abbreviation
  strongBisimJudge (infixr ‹∼› 65) where P ∼ Q ≡ (P, Q) ∈ bisim

lemma monotonic': mono(λS. {(P, Q) |P Q. P ↝[S] Q ∧ Q ↝[S] P})
apply(rule monoI)
by(auto dest: monoAux)

lemma monotonic: mono(λp x1 x2.
        ∃ P Q. x1 = P ∧
              x2 = Q ∧ P ↝[{(xa, x). p xa x}] Q ∧ Q ↝[{(xa, x). p xa x}] P)
apply(rule monoI)
by(auto intro: Strong-Late-Sim.monotonic)


lemma bisimCoinduct[case-names cSim cSym , consumes 1]:
  assumes p: (P, Q) ∈ X
  and     rSim: ⋀R S. (R, S) ∈ X ⟹ R ↝[(X ∪ bisim)] S
  and     rSym: ⋀R S. (R, S) ∈ X ⟹ (S, R) ∈ X

  shows P ∼ Q
proof −
  have aux: X ∪ bisim = {(P, Q). (P, Q) ∈ X ∨ P ∼ Q} by blast

  from p show ?thesis
    apply(coinduct, auto)
    apply(fastforce dest: rSim simp add: aux)
    by(fastforce dest: rSym)
qed

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

  assumes P ∼ Q

  shows P ↝[bisim] Q
using assms
by(auto intro: bisim.cases)

lemma bisimI:
  fixes P :: pi
  and   Q :: pi
```

```
  assumes P ⇝[bisim] Q
  and     Q ∼ P

  shows P ∼ Q
using assms
by(rule bisim.intros)

definition old-bisim :: (pi × pi) set ⇒ bool where
  old-bisim Rel ≡ ∀ (P, Q) ∈ Rel. P ⇝[Rel] Q ∧ (Q, P) ∈ Rel

lemma oldBisimBisimEq:
  shows (⋃{Rel. (old-bisim Rel)}) = bisim  (is ?LHS = ?RHS)
proof
  show ?LHS ⊆ ?RHS
  proof auto
    fix P Q Rel
    assume (P, Q) ∈ Rel and old-bisim Rel
    thus P ∼ Q
    proof(coinduct rule: bisimCoinduct)
      case(cSim P Q)
      with ‹old-bisim Rel› show ?case
        by(fastforce simp add: old-bisim-def intro: Strong-Late-Sim.monotonic)
    next
      case(cSym P Q)
      with ‹old-bisim Rel› show ?case
        by(auto simp add: old-bisim-def)
    qed
  qed
next
  show ?RHS ⊆ ?LHS
  proof auto
    fix P Q
    assume P ∼ Q
    moreover hence old-bisim bisim
      by(auto simp add: old-bisim-def dest: bisim.cases)
    ultimately show ∃ Rel. old-bisim Rel ∧ (P, Q) ∈ Rel
      by blast
  qed
qed

lemma reflexive:
  fixes P :: pi

  shows P ∼ P
proof -
  have (P, P) ∈ Id by simp
  thus ?thesis
    by(coinduct rule: bisimCoinduct, auto intro: Strong-Late-Sim.reflexive)
```

```
qed

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

  assumes P ∼ Q

  shows Q ∼ P
using assms
by(auto dest: bisim.cases)

lemma bisimClosed:
  fixes P :: pi
  and   Q :: pi
  and   p :: name prm

  assumes P ∼ Q

  shows (p • P) ∼ (p • Q)
proof −
  let ?X = {(p • P, p • Q) | P Q (p::name prm). P ∼ Q}
  from ‹P ∼ Q› have  (p • P, p • Q) ∈ ?X by blast
  thus ?thesis
  proof(coinduct rule: bisimCoinduct)
    case(cSim pP pQ)
   from ‹(pP, pQ) ∈ ?X› obtain P Q p where P ∼ Q and pP = (p::name prm)
• P and pQ = p • Q
      by auto
    from ‹P ∼ Q› have P ⇝[bisim] Q by(rule bisimE)
    moreover have bisim ⊆ ?X
    proof
      fix x
      assume x ∈ bisim
      moreover have x = (([]::name prm) • x) by auto
      ultimately show x ∈ ?X
        apply(case-tac x)
        by(clarify, simp only: eqvts) metis
    qed
    moreover have eqvt ?X
    proof(auto simp add: eqvt-def)
      fix P Q
      fix perm1::name prm
      fix perm2::name prm

      assume P ∼ Q
      moreover have perm1 • perm2 • P = (perm1 @ perm2) • P by(simp add:
pt2[OF pt-name-inst])
      moreover have perm1 • perm2 • Q = (perm1 @ perm2) • Q by(simp add:
```

*pt2*[*OF pt-name-inst*])

**ultimately show** $\exists P'\ Q'.\ (\exists(perm{::}name\ prm).\ perm1 \cdot perm2 \cdot P = perm \cdot P' \wedge$
$perm1 \cdot perm2 \cdot Q = perm \cdot Q') \wedge P' \sim Q'$
**by** *blast*
**qed**
**ultimately have** $(p \cdot P) \rightsquigarrow[?X]\ (p \cdot Q)$
**by**(*rule Strong-Late-Sim.eqvtI*)
**with** ‹$pP = p \cdot P$› ‹$pQ = p \cdot Q$› **show** *?case*
**by**(*force intro*: *Strong-Late-Sim.monotonic*)
**next**
**case**(*cSym P Q*)
**thus** *?case* **by**(*auto intro*: *symmetric*)
**qed**
**qed**

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

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

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

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

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

**and** *rSim*: $\bigwedge R\ S.\ (R,\ S) \in X \Longrightarrow R \rightsquigarrow[(bisim\ O\ (X \cup bisim)\ O\ bisim)]\ S$
**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** −
**let** $?X = bisim\ O\ (X \cup bisim)\ O\ bisim$
**from** ‹$(P,\ Q) \in X$› **have** $(P,\ Q) \in ?X$ **by**(*auto intro*: *reflexive*)
**thus** *?thesis*
**proof**(*coinduct rule*: *bisimCoinduct*)
**case**(*cSim P Q*)
**{**
**fix** $P\ P'\ Q'\ Q$

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

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

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

**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** $P \sim Q$
**using** *assms substClosedSubset* **by** *blast*

**lemma** *eqvt*:
**shows** *eqvt* (*substClosed bisim*)
**by**(*rule eqvtSubstClosed*[*OF Strong-Late-Bisim.bisimEqvt*])

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

**assumes** $P \sim^s Q$

**shows** $(perm \cdot P) \sim^s (perm \cdot Q)$
**using** *assms*
**by**(*rule eqvtRelI*[*OF eqvt*])

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

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

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

  **assumes** $P \sim^s Q$

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

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

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

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

**end**

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

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

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

  **shows** $\tau.(P) \rightsquigarrow[Rel]\ \tau.(Q)$
**proof** −
  **show** $\tau.(P) \rightsquigarrow[Rel]\ \tau.(Q)$
  **proof**(*induct rule*: *simCases*)
    **case**(*Bound a x Q'*)
    **have** $\tau.(Q) \longmapsto a«x» \prec Q'$ **by** *fact*
    **hence** *False* **by** *auto*

```
    thus ?case by simp
  next
    case(Free α Q′)
    have τ.(Q) ⟼ α ≺ Q′ by fact
    thus ?case
    proof(induct rule: tauCases)
      case cTau
      have τ.(P) ⟼ τ ≺ P by(rule Late-Semantics.Tau)
      with PRelQ show ?case by blast
    qed
  qed
qed

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

  assumes PRelQ: ∀ y. (P[x::=y], Q[x::=y]) ∈ Rel
  and     Eqvt: eqvt Rel

  shows a<x>.P ↝[Rel] a<x>.Q
using Eqvt
proof(induct rule: simCasesCont[where C=(x, a, P, Q)])
  case(Bound b y Q′)
  from ‹y ♯ (x, a, P, Q)› have y ≠ x y ≠ a y ♯ P y ♯ Q by simp+
  from ‹a<x>.Q ⟼b«y» ≺ Q′› ‹y ≠ a› ‹y ≠ x› ‹y ♯ Q› show ?case
  proof(induct rule: inputCases)
    case cInput

    have a<x>.P ⟼ a<x> ≺ P by(rule Input)
    hence a<x>.P ⟼ a<y> ≺ ([(x, y)] • P) using ‹y ♯ P›
      by(simp add: alphaBoundResidual)

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

```
      next
        assume yinequ: y ≠ u
        show ?thesis
        proof(cases x = u)
          assume x = u
          moreover have ((([(y, x)] • P)[y::=x], ([(y, x)] • Q)[y::=x]) ∈ Rel
          proof −
            from PRelQ have (P[x::=y], Q[x::=y]) ∈ Rel by blast
            with Eqvt have ([(y, x)] • (P[x::=y]), [(y, x)] • (Q[x::=y])) ∈ Rel
              by(rule eqvtRelI)
            with ‹y ≠ x› show ?thesis
              by(simp add: eqvt-subs name-calc)
          qed
          ultimately show ?thesis by simp
        next
          assume xinequ: x ≠ u
          hence ((([(y, x)] • P)[y::=u], ([(y, x)] • Q)[y::=u]) ∈ Rel
          proof −
            from PRelQ have (P[x::=u], Q[x::=u]) ∈ Rel by blast
            with Eqvt have ([(y, x)] • (P[x::=u]), [(y, x)] • (Q[x::=u])) ∈ Rel
              by(rule eqvtRelI)
            with ‹y ≠ x›  xinequ yinequ show ?thesis
              by(simp add: eqvt-subs name-calc)
          qed
          thus ?thesis by simp
        qed
      qed
    qed

    ultimately show ?case by blast
  qed
next
  case(Free α Q')
  have a<x>.Q ⟼ α ≺ Q' by fact
  hence False by auto
  thus ?case by blast
qed

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

  assumes PRelQ: (P, Q) ∈ Rel

  shows a{b}.P ⇝[Rel] a{b}.Q
```

```
proof –
  show ?thesis
  proof(induct rule: simCases)
    case(Bound c x Q')
    have a{b}.Q ⟼ c«x» ≺ Q' by fact
    hence False by auto
    thus ?case by simp
  next
    case(Free α Q')
    have a{b}.Q ⟼ α ≺ Q' by fact
    thus ?case
    proof(induct rule: outputCases)
      case cOutput
      have a{b}.P ⟼ a[b] ≺ P by(rule Late-Semantics.Output)
      with PRelQ show ?case by blast
    qed
  qed
qed

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

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

  shows [a⌢b]P ⇝[Rel'] [a⌢b]Q
proof –
  show ?thesis
  proof(induct rule: simCases)
    case(Bound c x Q')
    have x ♯ [a⌢b]P by fact
    hence xFreshP: x ♯ P by simp
    have [a⌢b]Q ⟼ c«x» ≺ Q' by fact
    thus ?case
    proof(induct rule: matchCases)
      case cMatch
      have Q ⟼c«x» ≺ Q' by fact
      with PSimQ xFreshP obtain P' where PTrans: P ⟼c«x» ≺ P'
                              and Pderivative: derivative P' Q' c x Rel
        by(blast dest: simE)

      from PTrans have [a⌢a]P ⟼ c«x» ≺ P' by(rule Late-Semantics.Match)
      moreover from Pderivative ‹Rel ⊆ Rel'› have derivative P' Q' c x Rel'
        by(cases c) (auto simp add: derivative-def)
```

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

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

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

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

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

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

**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** −
**show** *?thesis*
**proof**(*induct rule*: *simCases*)
**case**(*Bound a x QR*)
**have** $x \sharp P \oplus R$ **by** *fact*
**hence** *xFreshP*: $x \sharp P$ **and** *xFreshR*: $x \sharp R$ **by** *simp*+
**have** $Q \oplus R \longmapsto a«x» \prec QR$ **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *sumCases*)
**case** *cSum1*
**have** $Q \longmapsto a«x» \prec QR$ **by** *fact*
**with** *xFreshP PSimQ* **obtain** $P'$ **where** *PTrans*: $P \longmapsto a«x» \prec P'$
**and** *Pderivative*: *derivative* $P'$ *QR a x Rel*
**by**(*blast dest*: *simE*)

**from** *PTrans* **have** $P \oplus R \longmapsto a«x» \prec P'$ **by**(*rule Late-Semantics.Sum1*)
**moreover from** *Pderivative* ‹$Rel \subseteq Rel'$› **have** *derivative* $P'$ *QR a x* $Rel'$
**by**(*cases a*) (*auto simp add*: *derivative-def*)
**ultimately show** *?case* **by** *blast*
**next**
**case** *cSum2*
**from** ‹$R \longmapsto a«x» \prec QR$› **have** $P \oplus R \longmapsto a«x» \prec QR$ **by**(*rule Sum2*)
**thus** *?case* **using** ‹$Id \subseteq Rel'$› **by**(*blast intro*: *derivativeReflexive*)
**qed**
**next**
**case**(*Free* $\alpha$ *QR*)

```
    have Q ⊕ R ⟼α ≺ QR by fact
    thus ?case
    proof(induct rule: sumCases)
      case cSum1
      have Q ⟼α ≺ QR by fact
      with PSimQ obtain P' where PTrans: P ⟼α ≺ P' and PRel: (P', QR)
∈ Rel
        by(blast dest: simE)
      from PTrans have P ⊕ R ⟼α ≺ P' by(rule Late-Semantics.Sum1)
      with PRel ‹Rel ⊆ Rel'› show ?case by blast
    next
      case cSum2
      from ‹R ⟼α ≺ QR› have P ⊕ R ⟼α ≺ QR by(rule Sum2)
      thus ?case using ‹Id ⊆ Rel'› by(blast intro: derivativeReflexive)
    qed
  qed
qed

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

  assumes PSimQ:    P ↝[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'' ⟹ (<νa>P, <νa>Q) ∈ Rel''
  and     EqvtRel:  eqvt Rel
  and     EqvtRel': eqvt Rel'
  and     EqvtRel'': eqvt Rel''

  shows P ∥ R ↝[Rel''] Q ∥ T
using EqvtRel''
proof(induct rule: simCasesCont[where C=()])
  case(Bound a x Q')
  have x ♯ P ∥ R and x ♯ Q ∥ T by fact+
  hence xFreshP: x ♯ P and xFreshR: x ♯ R and x ♯ Q and x ♯ T by simp+
  have QTTrans: Q ∥ T ⟼ a«x» ≺ Q' by fact
  thus ?case using ‹x ♯ Q› ‹x ♯ T›
  proof(induct rule: parCasesB)
    case(cPar1 Q')
    have QTrans: Q ⟼ a«x» ≺ Q' and xFreshT: x ♯ T by fact+
```

**from** *xFreshP PSimQ QTrans* **obtain** $P'$ **where** *PTrans*:$P \longmapsto a«x» \prec P'$
**and** *Pderivative*: *derivative* $P'$ $Q'$ *a x Rel*
**by**(*blast dest*: *simE*)
**from** *PTrans xFreshR* **have** $P \parallel R \longmapsto a«x» \prec P' \parallel R$ **by**(*rule Late-Semantics.Par1B*)
**moreover from** *Pderivative xFreshR xFreshT RRel'T* **have** *derivative* $(P' \parallel R)$ $(Q' \parallel T)$ *a x* $Rel''$
**by**(*cases a*, *auto intro*: *Par simp add*: *derivative-def forget*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*cPar2* $T'$)
**have** *TTrans*: $T \longmapsto a«x» \prec T'$ **and** *xFreshQ*: $x \sharp Q$ **by** *fact+*

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

**from** ‹$x \sharp (P, R)$› **have** $x \sharp P$ **by** *simp*
**with** ‹$P \leadsto[Rel]\ Q$› ‹$Q \longmapsto a{<}x{>} \prec Q'$› ‹$x \sharp P$›
**obtain** $P'$ **where** *PTrans*: $P \longmapsto a{<}x{>} \prec P'$
**and** *Pderivative*: *derivative* $P'\ Q'$ (*InputS* $a$) $x$ *Rel*
**by**(*blast dest*: *simE*)
**from** *Pderivative* **have** *PRel*: $(P'[x::=b],\ Q'[x::=b]) \in Rel$ **by**(*simp add*: *derivative-def*)

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

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

**from** ‹$x \sharp (P, R)$› **have** $x \sharp R$ **by** *simp*
**with** ‹$R \leadsto[Rel']\ T$› ‹$T \longmapsto a{<}x{>} \prec T'$›
**obtain** $R'$ **where** *RTrans*: $R \longmapsto a{<}x{>} \prec R'$
**and** *Rderivative*: *derivative* $R'\ T'$ (*InputS* $a$) $x$ *Rel'*
**by**(*blast dest*: *simE*)
**from** *Rderivative* **have** *RRel*: $(R'[x::=b],\ T'[x::=b]) \in Rel'$ **by**(*simp add*: *derivative-def*)

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

**from** ‹$y \sharp (P, R)$› **have** $y \sharp R$ **and** $y \sharp P$ **by** *simp+*
**from** ‹$R \leadsto[Rel']\ T$› ‹$T \longmapsto a{<}\nu y{>} \prec T'$› ‹$y \sharp R$›
**obtain** $R'$ **where** *RTrans*: $R \longmapsto a{<}\nu y{>} \prec R'$
**and** *Rderivative*: *derivative* $R'\ T'$ (*BoundOutputS* $a$) $y$ *Rel'*

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

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

**from** ‹$x \sharp (P, R)$› **have** $x \sharp R$ **by** *simp+*
**with** ‹$R \leadsto[Rel']\ T$› ‹$T \longmapsto a<x> \prec T'$›
**obtain** $R'$ **where** *RTrans*: $R \longmapsto a<x> \prec R'$
**and** $R'Rel'T'$: $(R'[x::=y], T'[x::=y]) \in Rel'$
**by**(*force dest*: *simE simp add*: *derivative-def*)

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

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

**assumes** *PSimQ*: $P \leadsto[Rel]\ Q$
**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** −
  **note** *PSimQ*
  **moreover have** *RSimR*: $R \leadsto[Id]\ R$ **by**(*auto intro*: *reflexive*)
  **moreover note** *PRelQ* **moreover have** $(R,\ R) \in Id$ **by** *auto*
  **moreover from** *Par* **have** $\bigwedge P\ Q\ R\ T.\ [\![(P,\ Q) \in Rel;\ (R,\ T) \in Id]\!] \Longrightarrow (P \parallel R,\ Q \parallel T) \in Rel'$
    **by** *auto*
  **moreover note** *Res* ‹*eqvt Rel*›
  **moreover have** *eqvt Id* **by**(*auto simp add*: *eqvt-def*)
  **ultimately show** *?thesis* **using** *EqvtRel'* **by**(*rule parCompose*)
**qed**

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

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

  **shows** *derivative* $(<\nu y>P)\ (<\nu y>Q)\ a\ x\ Rel'$
**proof** −
  **from** *Der Rel* **show** *?thesis*
  **proof**(*cases a*, *auto simp add*: *derivative-def*)
    **fix** $u$
    **assume** *A1*: $\forall u.\ (P[x{::=}u],\ Q[x{::=}u]) \in Rel$
    **show** $((<\nu y>P)[x{::=}u],\ (<\nu y>Q)[x{::=}u]) \in Rel'$
    **proof**(*cases* $x=y$)
      **assume** *xeqy*: $x=y$

      **from** *A1* **have** $(P[x{::=}x],\ Q[x{::=}x]) \in Rel$ **by** *blast*
      **hence** *L1*: $(<\nu y>P,\ <\nu y>Q) \in Rel'$ **by**(*force intro*: *Rel*)

      **have** $y \sharp <\nu y>P$ **and** $y \sharp <\nu y>Q$ **by**(*simp only*: *freshRes*)+
      **hence** $(<\nu y>P)[y{::=}u] = <\nu y>P$ **and** $(<\nu y>Q)[y{::=}u] = <\nu y>Q$ **by**(*simp add*: *forget*)+

      **with** *L1 xeqy* **show** *?thesis* **by** *simp*
    **next**
      **assume** *xineqy*: $x \neq y$

      **show** *?thesis*
      **proof**(*cases* $y=u$)

**assume** *yequ*: $y=u$

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

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

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

**assumes** *PSimQ*: $P \leadsto[Rel]\ Q$
**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** $<\nu x>P \leadsto[Rel']\ <\nu x>Q$
**using** *EqvtRel'*
**proof**(*induct rule*: *resSimCases*[*of - - - -* $(P,\ x)$])
**case**(*BoundOutput* $Q'$ *a*)

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

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

**from** *PTrans aineqx* **have** $<\nu x>P \longmapsto a<\nu x> \prec P'$ **by**(*rule Late-Semantics.Open*)
**moreover from** $P'RelQ'$ $RelRel'$ **have** $(P', Q') \in Rel'$ **by** *force*
**ultimately show** *?case* **by** *blast*
**next**
**case**(*BoundR* $Q'$ *a y*)
**have** *QTrans*: $Q \longmapsto a«y» \prec Q'$ **and** *xFresha*: $x \sharp a$ **by** *fact+*
**have** $y \sharp (P, x)$ **by** *fact*
**hence** *yFreshP*: $y \sharp P$ **and** *yineqx*: $y \neq x$ **by**(*simp add*: *fresh-prod*)+

**from** *PSimQ yFreshP QTrans* **obtain** $P'$ **where** *PTrans*: $P \longmapsto a«y» \prec P'$
**and** *Pderivative*: *derivative* $P'$ $Q'$ *a y Rel*
**by**(*blast dest*: *simE*)
**from** *PTrans xFresha yineqx* **have** *ResTrans*: $<\nu x>P \longmapsto a«y» \prec <\nu x>P'$
**by**(*blast intro*: *Late-Semantics.ResB*)
**moreover from** *Pderivative ResRel EqvtRel* **have** *derivative* $(<\nu x>P')$ $(<\nu x>Q')$ *a y* $Rel'$
**by**(*rule resDerivative*)

**ultimately show** *?case* **by** *blast*
**next**
**case**(*FreeR* $Q'$ $\alpha$)
**have** *QTrans*: $Q \longmapsto \alpha \prec Q'$ **and** *xFreshAlpha*: $(x::name) \sharp \alpha$ **by** *fact+*

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

**from** *PTrans xFreshAlpha* **have** $<\nu x>P \longmapsto \alpha \prec <\nu x>P'$ **by**(*rule Late-Semantics.ResF*)
**moreover from** $P'RelQ'$ **have** $(<\nu x>P', <\nu x>Q') \in Rel'$ **by**(*rule ResRel*)
**ultimately show** *?case* **by** *blast*
**qed**

**lemma** *resChainI*:
**fixes** $P$ :: *pi*
**and** $Q$ :: *pi*
**and** *Rel* :: $(pi \times pi)$ *set*
**and** *xs* :: *name list*

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

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

**proof**(*induct xs*)
  **from** *PRelQ* **show** *resChain* [] *P* ⇝[*Rel*] *resChain* [] *Q* **by** *simp*
**next**
  **fix** *x xs*
  **assume** *IH*: (*resChain xs P*) ⇝[*Rel*] (*resChain xs Q*)
  **moreover note** *Res*
  **moreover have** *Rel* ⊆ *Rel* **by** *simp*
  **ultimately have** <νx>(*resChain xs P*) ⇝[*Rel*] <νx>(*resChain xs Q*) **using** *eqvtRel*
    **by**(*rule-tac resPres*)

  **thus** *resChain* (*x* # *xs*) *P* ⇝[*Rel*] *resChain* (*x* # *xs*) *Q*
    **by** *simp*
**qed**

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

  **assumes** *PRelQ*: (*P*, *Q*) ∈ *Rel*
  **and** *Sim*: ⋀*P Q*. (*P*, *Q*) ∈ *Rel* ⟹ *P* ⇝[*Rel*] *Q*
  **and** *eqvtRel*: *eqvt Rel*

  **shows** !*P* ⇝[*bangRel Rel*] !*Q*
**proof** −
  **let** *?Sim* = λ*P Rs*. (∀ *a x Q'*. *Rs* = *a*«*x*» ≺ *Q'* ⟶ *x* ♯ *P* ⟶ (∃ *P'*. *P* ⟼*a*«*x*» ≺ *P'* ∧ *derivative P' Q' a x* (*bangRel Rel*))) ∧
    (∀ α *Q'*. *Rs* = α ≺ *Q'* ⟶ (∃ *P'*. *P* ⟼α ≺ *P'* ∧ (*P'*, *Q'*) ∈ *bangRel Rel*))
  **from** *eqvtRel* **have** *EqvtBangRel*: *eqvt*(*bangRel Rel*) **by**(*rule eqvtBangRel*)

  **{**
    **fix** *Pa Rs*
    **assume** !*Q* ⟼ *Rs* **and** (*Pa*, !*Q*) ∈ *bangRel Rel*
    **hence** *?Sim Pa Rs* **using** *PRelQ*
    **proof**(*nominal-induct avoiding*: *Pa P rule*: *bangInduct*)
      **case**(*cPar1B a x Q' Pa P*)
      **have** *QTrans*: *Q* ⟼ *a*«*x*» ≺ *Q'* **by** *fact*
      **have** (*Pa*, *Q* ∥ !*Q*) ∈ *bangRel Rel* **and** *x* ♯ *Pa* **by** *fact+*
      **thus** *?Sim Pa* (*a*«*x*» ≺ (*Q'* ∥ !*Q*))
      **proof**(*induct rule*: *BRParCases*)
        **case**(*BRPar P R*)
        **have** *PRelQ*: (*P*, *Q*) ∈ *Rel* **by** *fact*
        **have** *PBRQ*: (*R*, !*Q*) ∈ *bangRel Rel* **by** *fact*
        **have** *x* ♯ *P* ∥ *R* **by** *fact*
        **hence** *xFreshP*: *x* ♯ *P* **and** *xFreshR*: *x* ♯ *R* **by** *simp+*
        **show** *?case*
        **proof**(*auto simp add*: *residual.inject alpha'*)

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

**with** *QTrans xFreshP* **obtain** $P'$ **where** *PTrans*: $P \longmapsto a«x» \prec P'$ **and** $P'RelQ'$: *derivative* $P'\ Q'\ a\ x\ Rel$
**by**(*blast dest*: *simE*)

**from** *PTrans xFreshR* **have** $P \parallel R \longmapsto a«x» \prec (P' \parallel R)$
**by**(*force intro*: *Late-Semantics.Par1B*)
**moreover from** $P'RelQ'$ *PBRQ* ‹$x \sharp Q$› ‹$x \sharp R$› **have** *derivative* $(P' \parallel R)\ (Q' \parallel !Q)\ a\ x$ (*bangRel Rel*)
**by**(*cases a*) (*auto simp add*: *derivative-def forget intro*: *Rel.BRPar*)
**ultimately show** $\exists P'.\ P \parallel R \longmapsto a«x» \prec P' \wedge$ *derivative* $P'\ (Q' \parallel !Q)\ a\ x$ (*bangRel Rel*) **by** *blast*
**next**
**fix** $y$
**assume** $(y::name) \sharp Q'$ **and** $y \sharp P$ **and** $y \sharp R$ **and** $y \sharp Q$
**from** *QTrans* ‹$y \sharp Q'$› **have** $Q \longmapsto a«y» \prec ([(x, y)] \cdot Q')$
**by**(*simp add*: *alphaBoundResidual*)
**moreover from** *PRelQ* **have** $P \rightsquigarrow[Rel]\ Q$ **by**(*rule Sim*)
**ultimately obtain** $P'$ **where** *PTrans*: $P \longmapsto a«y» \prec P'$ **and** $P'RelQ'$: *derivative* $P'\ ([(x, y)] \cdot Q')\ a\ y\ Rel$
**using** ‹$y \sharp P$›
**by**(*blast dest*: *simE*)
**from** *PTrans* ‹$y \sharp R$› **have** $P \parallel R \longmapsto a«y» \prec (P' \parallel R)$ **by**(*force intro*: *Late-Semantics.Par1B*)
**moreover from** $P'RelQ'$ *PBRQ* ‹$y \sharp Q$› ‹$y \sharp R$› **have** *derivative* $(P' \parallel R)\ (([(x, y)] \cdot Q') \parallel !Q)\ a\ y$ (*bangRel Rel*)
**by**(*cases a*) (*auto simp add*: *derivative-def forget intro*: *Rel.BRPar*)
**with** ‹$x \sharp Q$› ‹$y \sharp Q$› **have** *derivative* $(P' \parallel R)\ (([(y, x)] \cdot Q') \parallel !([(y, x)] \cdot Q))\ a\ y$ (*bangRel Rel*)
**by**(*simp add*: *name-fresh-fresh name-swap*)
**ultimately show** $\exists P'.\ P \parallel R \longmapsto a«y» \prec P' \wedge$ *derivative* $P'\ (([(y, x)] \cdot Q') \parallel !([(y, x)] \cdot Q))\ a\ y$ (*bangRel Rel*)
**by** *blast*
**qed**
**qed**
**next**
**case**(*cPar1F* $\alpha\ Q'$ *Pa P*)
**have** *QTrans*: $Q \longmapsto \alpha \prec Q'$ **by** *fact*
**have** $(Pa, Q \parallel !Q) \in$ *bangRel Rel* **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *BRParCases*)
**case**(*BRPar P R*)
**have** *PRelQ*: $(P, Q) \in Rel$ **and** *BR*: $(R, !Q) \in$ *bangRel Rel* **by** *fact+*
**show** *?case*
**proof**(*auto simp add*: *residual.inject*)
**from** *PRelQ* **have** $P \rightsquigarrow[Rel]\ Q$ **by**(*rule Sim*)
**with** *QTrans* **obtain** $P'$ **where** *PTrans*: $P \longmapsto \alpha \prec P'$ **and** *RRel*: $(P', Q') \in Rel$

**by**(*blast dest*: *simE*)

**from** *PTrans* **have** $P \parallel R \longmapsto \alpha \prec P' \parallel R$ **by**(*rule Par1F*)
**moreover from** *RRel BR* **have** $(P' \parallel R,\ Q' \parallel !Q) \in$ *bangRel Rel* **by**(*rule Rel.BRPar*)
**ultimately show** $\exists P'.\ P \parallel R \longmapsto \alpha \prec P' \wedge (P',\ Q' \parallel !Q) \in$ *bangRel Rel* **by** *blast*
**qed**
**qed**
**next**
**case**(*cPar2B a x Q' Pa P*)
**hence** *IH*: $\bigwedge Pa.\ (Pa,\ !Q) \in$ *bangRel Rel* $\Longrightarrow$ *?Sim Pa* $(a«x» \prec Q')$ **by** *simp*
**have** $(Pa,\ Q \parallel !Q) \in$ *bangRel Rel* **and** $x \sharp Pa$ **by** *fact+*
**thus** *?Sim Pa* $(a«x» \prec (Q \parallel Q'))$
**proof**(*induct rule*: *BRParCases*)
**case**(*BRPar P R*)
**have** *PRelQ*: $(P,\ Q) \in$ *Rel* **and** *RBRQ*: $(R,\ !Q) \in$ *bangRel Rel* **by** *fact+*
**have** $x \sharp P \parallel R$ **by** *fact*
**hence** *xFreshP*: $x \sharp P$ **and** *xFreshR*: $x \sharp R$ **by** *simp+*

**from** *EqvtBangRel* ‹$x \sharp Q$› **show** *?Sim* $(P \parallel R)$ $(a«x» \prec (Q \parallel Q'))$
**proof**(*auto simp add*: *residual.inject alpha' name-fresh-fresh*)
**from** *RBRQ* **have** *?Sim R* $(a«x» \prec Q')$ **by**(*rule IH*)
**with** *xFreshR* **obtain** $R'$ **where** *RTrans*: $R \longmapsto a«x» \prec R'$ **and** $R'BRQ'$: *derivative* $R'\ Q'\ a\ x$ (*bangRel Rel*)
**by**(*auto simp add*: *residual.inject*)
**from** *RTrans xFreshP* **have** $P \parallel R \longmapsto a«x» \prec (P \parallel R')$ **by**(*auto intro*: *Par2B*)
**moreover from** *PRelQ* $R'BRQ'$ ‹$x \sharp Q$› ‹$x \sharp P$› **have** *derivative* $(P \parallel R')$ $(Q \parallel Q')$ $a\ x$ (*bangRel Rel*)
**by**(*cases a*) (*auto simp add*: *derivative-def forget intro*: *Rel.BRPar*)
**ultimately show** $\exists P'.\ P \parallel R \longmapsto a«x» \prec P' \wedge$ *derivative* $P'$ $(Q \parallel Q')$ $a$ $x$ (*bangRel Rel*) **by** *blast*
**next**
**fix** $y$
**assume** $(y::name) \sharp Q$ **and** $y \sharp Q'$ **and** $y \sharp P$ **and** $y \sharp R$
**from** *RBRQ* **have** *?Sim R* $(a«x» \prec Q')$ **by**(*rule IH*)
**with** ‹$y \sharp Q'$› **have** *?Sim R* $(a«y» \prec ([(x,\ y)] \cdot Q'))$ **by**(*simp add*: *alphaBoundResidual*)
**with** ‹$y \sharp R$› **obtain** $R'$ **where** *RTrans*: $R \longmapsto a«y» \prec R'$ **and** $R'BRQ'$: *derivative* $R'$ $([(x,\ y)] \cdot Q')$ $a\ y$ (*bangRel Rel*)
**by**(*auto simp add*: *residual.inject*)
**from** *RTrans* ‹$y \sharp P$› **have** $P \parallel R \longmapsto a«y» \prec (P \parallel R')$ **by**(*auto intro*: *Par2B*)
**moreover from** *PRelQ* $R'BRQ'$ ‹$y \sharp P$› ‹$y \sharp Q$› **have** *derivative* $(P \parallel R')$ $(Q \parallel ([(x,\ y)] \cdot Q'))$ $a\ y$ (*bangRel Rel*)
**by**(*cases a*) (*auto simp add*: *derivative-def forget intro*: *Rel.BRPar*)
**hence** *derivative* $(P \parallel R')$ $(Q \parallel ([(y,\ x)] \cdot Q'))$ $a\ y$ (*bangRel Rel*)
**by**(*simp add*: *name-swap*)

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

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

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

**by** *blast*

**from** *PTrans RTrans* **have** $P \parallel R \longmapsto \tau \prec P'[x::=b] \parallel R'$ **by**(*rule Comm1*)
**moreover from** $P'RelQ'$ $R'RelQ''$ **have** $(P'[x::=b] \parallel R',\ Q'[x::=b] \parallel Q'') \in$ *bangRel Rel* **by**(*rule Rel.BRPar*)
**ultimately show** $\exists P'.\ P \parallel R \longmapsto \tau \prec P' \wedge (P',\ Q'[x::=b] \parallel Q'') \in$ *bangRel Rel* **by** *blast*
**qed**
**qed**
**next**
**case**(*cComm2 a b* $Q'$ *x* $Q''$ *Pa P*)
**hence** *IH*: $\bigwedge Pa.\ (Pa,\ !Q) \in$ *bangRel Rel* $\Longrightarrow$ *?Sim Pa* $(a<x> \prec Q'')$ **by** *simp*
**have** *QTrans*: $Q \longmapsto a[b] \prec Q'$ **by** *fact*
**have** $(Pa,\ Q \parallel !Q) \in$ *bangRel Rel* **by** *fact*
**thus** *?case* **using** ‹$x \sharp Pa$›
**proof**(*induct rule*: *BRParCases*)
**case**(*BRPar P R*)
**have** *PRelQ*: $(P,\ Q) \in$ *Rel* **and** *RBRQ*: $(R,\ !Q) \in$ *bangRel Rel* **by** *fact+*
**from** ‹$x \sharp P \parallel R$› **have** $x \sharp P$ **and** $x \sharp R$ **by** *simp+*
**show** *?case*
**proof**(*auto simp add*: *residual.inject*)
**from** *PRelQ* **have** $P \leadsto[Rel]\ Q$ **by**(*rule Sim*)
**with** *QTrans* **obtain** $P'$ **where** *PTrans*: $P \longmapsto a[b] \prec P'$ **and** $P'RelQ'$: $(P',\ Q') \in$ *Rel*
**by**(*blast dest*: *simE*)

**from** *IH RBRQ* ‹$x \sharp R$› **have** *RTrans*: $\exists R'.\ R \longmapsto a<x> \prec R' \wedge (R'[x::=b],\ Q''[x::=b]) \in$ *bangRel Rel*
**by**(*fastforce simp add*: *derivative-def residual.inject*)
**then obtain** $R'$ **where** *RTrans*: $R \longmapsto a<x> \prec R'$ **and** $R'RelQ''$: $(R'[x::=b],\ Q''[x::=b]) \in$ *bangRel Rel*
**by** *blast*

**from** *PTrans RTrans* **have** $P \parallel R \longmapsto \tau \prec P' \parallel R'[x::=b]$ **by**(*rule Comm2*)
**moreover from** $P'RelQ'$ $R'RelQ''$ **have** $(P' \parallel R'[x::=b],\ Q' \parallel Q''[x::=b]) \in$ *bangRel Rel* **by**(*rule Rel.BRPar*)
**ultimately show** $\exists P'.\ P \parallel R \longmapsto \tau \prec P' \wedge (P',\ Q' \parallel (Q''[x::=b])) \in$ *bangRel Rel* **by** *blast*
**qed**
**qed**
**next**
**case**(*cClose1 a x* $Q'$ *y* $Q''$ *Pa P*)
**hence** *IH*: $\bigwedge Pa.\ (Pa,\ !Q) \in$ *bangRel Rel* $\longrightarrow$ *?Sim Pa* $(a<\nu y> \prec Q'')$ **by** *simp*
**have** *QTrans*: $Q \longmapsto a<x> \prec Q'$ **by** *fact*
**have** $(Pa,\ Q \parallel !Q) \in$ *bangRel Rel* **by** *fact*
**moreover have** *xFreshPa*: $x \sharp Pa$ **by** *fact*
**ultimately show** *?case* **using** ‹$y \sharp Pa$›
**proof**(*induct rule*: *BRParCases*)

**case**(*BRPar P R*)
**have** *PRelQ*: $(P, Q) \in$ *Rel* **and** *RBRQ*: $(R, !Q) \in$ *bangRel Rel* **by** *fact+*
**have** $x \sharp P \parallel R$ **and** $y \sharp P \parallel R$ **by** *fact+*
**hence** *xFreshP*: $x \sharp P$ **and** *xFreshR*: $x \sharp R$ **and** $y \sharp R$ **and** $y \sharp P$ **by** *simp+*
**show** *?case*
**proof**(*auto simp add*: *residual.inject*)
**from** *PRelQ* **have** $P \leadsto$[*Rel*] $Q$ **by**(*rule Sim*)
**with** *QTrans xFreshP* **obtain** $P'$ **where** *PTrans*: $P \longmapsto a{<}x{>} \prec P'$ **and** $P'RelQ'$: $(P'[x::=y], Q'[x::=y]) \in$ *Rel*
**by**(*fastforce dest*: *simE simp add*: *derivative-def*)

**from** *RBRQ* ‹$y \sharp R$› *IH* **have** $\exists R'.\ R \longmapsto a{<}\nu y{>} \prec R' \wedge (R', Q'') \in$ *bangRel Rel*
**by**(*auto simp add*: *residual.inject derivative-def*)
**then obtain** $R'$ **where** *RTrans*: $R \longmapsto a{<}\nu y{>} \prec R'$ **and** $R'RelQ''$: $(R', Q'') \in$ *bangRel Rel*
**by** *blast*

**from** *PTrans RTrans* ‹$y \sharp P$› **have** $P \parallel R \longmapsto \tau \prec {<}\nu y{>}(P'[x::=y] \parallel R')$
**by**(*rule Close1*)
**moreover from** $P'RelQ'$ $R'RelQ''$ **have** $({<}\nu y{>}(P'[x::=y] \parallel R'), {<}\nu y{>}(Q'[x::=y] \parallel Q'')) \in$ *bangRel Rel*
**by**(*force intro*: *Rel.BRPar BRRes*)
**ultimately show** $\exists P'.\ P \parallel R \longmapsto \tau \prec P' \wedge (P', {<}\nu y{>}(Q'[x::=y] \parallel Q'')) \in$ *bangRel Rel* **by** *blast*
**qed**
**qed**
**next**
**case**(*cClose2 a x Q' y Q'' Pa P*)
**hence** *IH*: $\bigwedge Pa.\ (Pa, !Q) \in$ *bangRel Rel* $\Longrightarrow$ *?Sim Pa* $(a{<}y{>} \prec Q'')$ **by** *simp*
**have** *QTrans*: $Q \longmapsto a{<}\nu x{>} \prec Q'$ **by** *fact*
**have** $(Pa, Q \parallel !Q) \in$ *bangRel Rel* **and** $x \sharp Pa$ **and** $y \sharp Pa$ **by** *fact+*
**thus** *?case*
**proof**(*induct rule*: *BRParCases*)
**case**(*BRPar P R*)
**have** *PRelQ*: $(P, Q) \in$ *Rel* **and** *RBRQ*: $(R, !Q) \in$ *bangRel Rel* **by** *fact+*
**have** $x \sharp P \parallel R$ **and** $y \sharp P \parallel R$ **by** *fact+*
**hence** *xFreshP*: $x \sharp P$ **and** *xFreshR*: $x \sharp R$ **and** $y \sharp R$ **by** *simp+*
**show** *?case*
**proof**(*auto simp add*: *residual.inject*)
**from** *PRelQ* **have** $P \leadsto$[*Rel*] $Q$ **by**(*rule Sim*)
**with** *QTrans xFreshP* **obtain** $P'$ **where** *PTrans*: $P \longmapsto a{<}\nu x{>} \prec P'$ **and** $P'RelQ'$: $(P', Q') \in$ *Rel*
**by**(*fastforce dest*: *simE simp add*: *derivative-def*)

**from** *RBRQ IH* ‹$y \sharp R$› **have** $\exists R'.\ R \longmapsto a{<}y{>} \prec R' \wedge (R'[y::=x], Q''[y::=x]) \in$ *bangRel Rel*
**by**(*fastforce simp add*: *derivative-def residual.inject*)
**then obtain** $R'$ **where** *RTrans*: $R \longmapsto a{<}y{>} \prec R'$ **and** $R'RelQ''$: $(R'[y::=x],$

$Q''[y::=x]) \in$ *bangRel Rel*
**by** *blast*

**from** *PTrans RTrans xFreshR* **have** $P \parallel R \longmapsto \tau \prec$ <$\nu x$>$(P' \parallel R'[y::=x])$
**by**(*rule Close2*)
**moreover from** *P′RelQ′ R′RelQ′′* **have** (<$\nu x$>$(P' \parallel R'[y::=x])$, <$\nu x$>$(Q' \parallel Q''[y::=x])) \in$ *bangRel Rel*
**by**(*force intro*: *Rel.BRPar BRRes*)
**ultimately show** $\exists P'.\ P \parallel R \longmapsto \tau \prec P' \wedge (P'$, <$\nu x$>$(Q' \parallel Q''[y::=x])) \in$ *bangRel Rel* **by** *blast*
**qed**
**qed**
**next**
**case**(*cBang Rs Pa P*)
**hence** *IH*: $\bigwedge Pa.\ (Pa,\ Q \parallel !Q) \in$ *bangRel Rel* $\Longrightarrow$ *?Sim Pa Rs* **by** *simp*
**have** $(Pa, !Q) \in$ *bangRel Rel* **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *BRBangCases*)
**case**(*BRBang P*)
**have** *PRelQ*: $(P,\ Q) \in$ *Rel* **by** *fact*
**hence** $(!P, !Q) \in$ *bangRel Rel* **by**(*rule Rel.BRBang*)
**with** *PRelQ* **have** $(P \parallel !P,\ Q \parallel !Q) \in$ *bangRel Rel* **by**(*rule BRPar*)
**with** *IH* **have** *?Sim* $(P \parallel !P)$ *Rs* **by** *simp*
**thus** *?case* **by**(*force intro*: *Bang*)
**qed**
**qed**
}

**moreover from** *PRelQ* **have** $(!P, !Q) \in$ *bangRel Rel* **by**(*rule BRBang*)
**ultimately show** *?thesis* **by**(*auto simp add*: *simulation-def*)
**qed**

**end**

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

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

**assumes** $P \sim Q$

**shows** $\tau.(P) \sim \tau.(Q)$
**proof** −
**let** $?X = \{(\tau.(P), \tau.(Q)), (\tau.(Q), \tau.(P))\}$
**have** $(\tau.(P), \tau.(Q)) \in ?X$ **by** *auto*
**thus** *?thesis* **using** $\langle P \sim Q\rangle$

```
    by(coinduct rule: bisimCoinduct)
      (auto intro: Strong-Late-Sim-Pres.tauPres dest: symmetric)
qed

lemma inputPres:
  fixes P :: pi
  and   Q :: pi
  and   a :: name
  and   x :: name

  assumes PSimQ: ∀ y. P[x::=y] ∼ Q[x::=y]

  shows a<x>.P ∼ a<x>.Q
proof −
  let ?X = {(a<x>.P, a<x>.Q) | a x P Q. ∀ y. P[x::=y] ∼ Q[x::=y]}
  {
    fix axP axQ p
    assume (axP, axQ) ∈ ?X
     then obtain a x P Q where A: ∀ y. P[x::=y] ∼ Q[x::=y] and B: axP =
a<x>.P and C: axQ = a<x>.Q
      by auto
    have ⋀y. ((p::name prm) • P)[(p • x)::=y] ∼ (p • Q)[(p • x)::=y]
    proof −
      fix y
      from A have P[x::=(rev p • y)] ∼ Q[x::=(rev p • y)]
        by blast
      hence (p • (P[x::=(rev p • y)])) ∼ p • (Q[x::=(rev p • y)])
        by(rule bisimClosed)
      thus (p • P)[(p • x)::=y] ∼ (p • Q)[(p • x)::=y]
        by(simp add: eqvts pt-pi-rev[OF pt-name-inst, OF at-name-inst])
    qed
    hence ((p::name prm) • axP, p • axQ) ∈ ?X using B C
      by auto
  }
  hence eqvt ?X by(simp add: eqvt-def)

  from PSimQ have (a<x>.P, a<x>.Q) ∈ ?X by auto
  thus ?thesis
  proof(coinduct rule: bisimCoinduct)
    case(cSim P Q)
    thus ?case using ‹eqvt ?X›
      by(force intro: inputPres)
  next
    case(cSym P Q)
    thus ?case
      by(blast dest: symmetric)
  qed
qed
```

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

**assumes** $P \sim Q$

**shows** $a\{b\}.P \sim a\{b\}.Q$
**proof** −
**let** $?X = \{(a\{b\}.P,\ a\{b\}.Q),\ (a\{b\}.Q,\ a\{b\}.P)\}$
**have** $(a\{b\}.P,\ a\{b\}.Q) \in\ ?X$ **by** *auto*
**thus** *?thesis* **using** ‹$P \sim Q$›
**by**(*coinduct rule*: *bisimCoinduct*)
(*auto intro*: *Strong-Late-Sim-Pres.outputPres dest*: *symmetric*)
**qed**

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

**assumes** $P \sim Q$

**shows** $[a \frown b]P \sim [a \frown b]Q$
**proof** −
**let** $?X = \{([a \frown b]P,\ [a \frown b]Q),\ ([a \frown b]Q,\ [a \frown b]P)\}$
**have** $([a \frown b]P,\ [a \frown b]Q) \in\ ?X$ **by** *auto*
**thus** *?thesis* **using** ‹$P \sim Q$›
**by**(*coinduct rule*: *bisimCoinduct*)
(*auto intro*: *Strong-Late-Sim-Pres.matchPres dest*: *symmetric bisimE*)
**qed**

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

**assumes** $P \sim Q$

**shows** $[a \neq b]P \sim [a \neq b]Q$
**proof** −
**let** $?X = \{([a \neq b]P,\ [a \neq b]Q),\ ([a \neq b]Q,\ [a \neq b]P)\}$
**have** $([a \neq b]P,\ [a \neq b]Q) \in\ ?X$ **by** *auto*
**thus** *?thesis* **using** ‹$P \sim Q$›
**by**(*coinduct rule*: *bisimCoinduct*)
(*auto intro*: *Strong-Late-Sim-Pres.mismatchPres dest*: *symmetric bisimE*)

**qed**

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

  **assumes** $P \sim Q$

  **shows** $P \oplus R \sim Q \oplus R$
**proof** −
  **let** $?X = \{(P \oplus R, Q \oplus R), (Q \oplus R, P \oplus R)\}$
  **have** $(P \oplus R, Q \oplus R) \in ?X$ **by** *auto*
  **thus** *?thesis* **using** $\langle P \sim Q\rangle$
    **by**(*coinduct rule*: *bisimCoinduct*)
      (*auto intro*: *Strong-Late-Sim-Pres.sumPres reflexive dest*: *symmetric bisimE*)
**qed**

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

  **assumes** $P \sim Q$

  **shows** $<\nu x>P \sim <\nu x>Q$
**proof** −
  **let** $?X = \{x.\ \exists P\ Q.\ P \sim Q \wedge (\exists a.\ x = (<\nu a>P, <\nu a>Q))\}$
  **from** $\langle P \sim Q\rangle$ **have** $(<\nu x>P, <\nu x>Q) \in ?X$ **by** *blast*
  **thus** *?thesis*
  **proof**(*coinduct rule*: *bisimCoinduct*)
    **case**(*cSim xP xQ*)
    **{**
      **fix** $P\ Q\ a$
      **assume** *PSimQ*: $P \leadsto[bisim]\ Q$
      **moreover have** $\bigwedge P\ Q\ a.\ P \sim Q \Longrightarrow (<\nu a>P, <\nu a>Q) \in ?X \cup bisim$ **by** *blast*
      **moreover have** $bisim \subseteq ?X \cup bisim$ **by** *blast*
      **moreover have** *eqvt bisim* **by** *simp*
      **moreover have** *eqvt ?X*
        **by**(*auto simp add*: *eqvt-def*) (*blast intro*: *bisimClosed*)
      **hence** $eqvt\ (?X \cup bisim)$ **by** *auto*
      **ultimately have** $<\nu a>P \leadsto[(?X \cup bisim)]\ <\nu a>Q$
        **by**(*rule Strong-Late-Sim-Pres.resPres*)
    **}**
    **with** $\langle (xP, xQ) \in ?X\rangle$ **show** *?case*
      **by**(*auto dest*: *bisimE*)
  **next**
    **case**(*cSym xP xQ*)

**thus** *?case* **by**(*auto dest*: *symmetric*)
**qed**
**qed**

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

**assumes** *P* ∼ *Q*

**shows** *P* ∥ *R* ∼ *Q* ∥ *R*
**proof** −
**let** *?X* = {(*resChain lst* (*P* ∥ *R*), *resChain lst* (*Q* ∥ *R*)) | *lst P R Q*. *P* ∼ *Q*}
**have** *EmptyChain*: ⋀*P Q*. *P* ∥ *Q* = *resChain* [] (*P* ∥ *Q*) **by** *auto*
**with** ‹*P* ∼ *Q*› **have** (*P* ∥ *R*, *Q* ∥ *R*) ∈ *?X* **by** *blast*
**thus** *?thesis*
**proof**(*coinduct rule*: *bisimCoinduct*)
**case**(*cSim PR QR*)
**{**
**fix** *P Q R lst*

**assume** *P* ∼ *Q*

**hence** *P* ⇝[*bisim*] *Q* **by**(*rule bisimE*)
**moreover note** ‹*P* ∼ *Q*›
**moreover have** ⋀*P Q R*. *P* ∼ *Q* ⟹ (*P* ∥ *R*, *Q* ∥ *R*) ∈ *?X*
**by** *auto* (*blast intro*: *EmptyChain*)
**moreover**
**{**
**fix** *xP xQ x*
**assume** (*xP*, *xQ*) ∈ *?X*
**then obtain** *P Q R lst*
**where** *P* ∼ *Q* **and** *xP* = *resChain lst* (*P* ∥ *R*) **and** *xQeq*: *xQ* = *resChain lst* (*Q* ∥ *R*)
**by** *auto*
**moreover hence** (*resChain* (*x#lst*) (*P* ∥ *R*), *resChain* (*x#lst*) (*Q* ∥ *R*)) ∈ *?X*
**by** *blast*
**ultimately have** (<*νx*>*xP*, <*νx*>*xQ*) ∈ *?X* **by** *auto*
**}**
**note** *ResPres* = *this*
**moreover have** *eqvt bisim* **by** *simp*
**moreover have** *eqvt ?X*
**by**(*auto simp add*: *eqvt-def*) (*blast intro*: *bisimClosed*)
**ultimately have** *P* ∥ *R* ⇝[(*?X*)] *Q* ∥ *R* **by**(*rule parPres*)
**hence** *resChain lst* (*P* ∥ *R*) ⇝[*?X*] (*resChain lst* (*Q* ∥ *R*)) **using** ‹*eqvt ?X*› *ResPres*
**by**(*rule resChainI*)

```
      hence resChain lst (P ∥ R) ↝[(?X ∪ bisim)] (resChain lst (Q ∥ R))
        by(force intro: Strong-Late-Sim.monotonic)
    }
    with ‹(PR, QR) ∈ ?X› show ?case
      by auto
  next
    case(cSym PR QR)
    thus ?case by(blast dest: symmetric)
  qed
qed


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

  assumes PBiSimQ: P ∼ Q

  shows !P ∼ !Q
proof −
  let ?X = bangRel bisim
  from PBiSimQ have (!P, !Q) ∈ ?X by(rule Rel.BRBang)
  thus ?thesis
  proof(coinduct rule: bisimCoinduct)
    case(cSim bP bQ)
    {
      fix P Q
      assume (P, Q) ∈ ?X
      hence P ↝[?X] Q
      proof(induct)
        fix P Q
        assume P ∼ Q
        thus !P ↝[?X] !Q using bisimE bisimEqvt
          by(rule Strong-Late-Sim-Pres.bangPres)
      next
        fix P Q R T
        assume RBiSimT: R ∼ T
        assume PBangRelQ: (P, Q) ∈ ?X
        assume PSimQ: P ↝[?X] Q
        from RBiSimT have R ↝[bisim] T by(blast dest: bisimE)
        thus R ∥ P ↝[?X] T ∥ Q using PSimQ RBiSimT PBangRelQ Rel.BRPar
Rel.BRRes bisimEqvt eqvtBangRel
          by(blast intro: Strong-Late-Sim-Pres.parCompose)
      next
        fix P Q a
        assume P ↝[?X] Q
        moreover from eqvtBangRel bisimEqvt have eqvt ?X by blast
       ultimately show <νa>P ↝[?X] <νa>Q using Rel.BRRes by(blast intro:
Strong-Late-Sim-Pres.resPres)
```

```
      qed
     hence P ⇝[((bangRel bisim) ∪ bisim)] Q by(rule-tac Strong-Late-Sim.monotonic)
auto
    }
    with ‹(bP, bQ) ∈ ?X› show ?case by auto
  next
    case(cSym bP bQ)
    thus ?case by(metis bangRelSymetric symmetric)
  qed
qed

end

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

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

  assumes P ∼ˢ Q

  shows τ.(P) ∼ˢ τ.(Q)
using assms
by(force simp add: substClosed-def intro: Strong-Late-Bisim-Pres.tauPres)

lemma inputPres:
  fixes P :: pi
  and   Q :: pi
  and   a :: name
  and   x :: name

  assumes P ∼ˢ Q

  shows a<x>.P ∼ˢ a<x>.Q
proof(auto simp add: substClosed-def)
  fix σ :: (name × name) list
  {
    fix P Q a x σ
    assume P ∼ˢ Q
    then have P[<σ>] ∼ˢ Q[<σ>] by(rule partUnfold)
    then have ∀ y. (P[<σ>])[x::=y] ∼ (Q[<σ>])[x::=y]
      apply(auto simp add: substClosed-def)
      by(erule-tac x=[(x, y)] in allE) auto
    moreover assume x ♯ σ
    ultimately have (a<x>.P)[<σ>] ∼ (a<x>.Q)[<σ>] using bisimEqvt
      by(force intro: Strong-Late-Bisim-Pres.inputPres)
  }
```

**note** *Goal* = *this*

**obtain** $y$::*name* **where** $y \sharp P$ **and** $y \sharp Q$ **and** $y \sharp \sigma$
**by**(*generate-fresh name*) *auto*
**from** ‹$P \sim^s Q$› **have** $([(x, y)] \cdot P) \sim^s ([(x, y)] \cdot Q)$ **by**(*rule eqClosed*)
**hence** $(a<y>.([(x, y)] \cdot P))[<\sigma>] \sim (a<y>.([(x, y)] \cdot Q))[<\sigma>]$ **using** ‹$y \sharp \sigma$› **by**(*rule Goal*)
**moreover from** ‹$y \sharp P$› ‹$y \sharp Q$› **have** $a<x>.P = a<y>.([(x, y)] \cdot P)$ **and** $a<x>.Q = a<y>.([(x, y)] \cdot Q)$
**by**(*simp add*: *alphaInput*)+

**ultimately show** $(a<x>.P)[<\sigma>] \sim (a<x>.Q)[<\sigma>]$ **by** *simp*
**qed**

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

**assumes** $P \sim^s Q$

**shows** $a\{b\}.P \sim^s a\{b\}.Q$
**using** *assms*
**by**(*force simp add*: *substClosed-def intro*: *Strong-Late-Bisim-Pres.outputPres*)

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

**assumes** $P \sim^s Q$

**shows** $[a \frown b]P \sim^s [a \frown b]Q$
**using** *assms*
**by**(*force simp add*: *substClosed-def intro*: *Strong-Late-Bisim-Pres.matchPres*)

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

**assumes** $P \sim^s Q$

**shows** $[a \neq b]P \sim^s [a \neq b]Q$
**using** *assms*
**by**(*force simp add*: *substClosed-def intro*: *Strong-Late-Bisim-Pres.mismatchPres*)

**lemma** *sumPres*:

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

**assumes** $P \sim^s Q$

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

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

**assumes** $P \sim^s Q$

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

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

**assumes** *PeqQ*: $P \sim^s Q$

**shows** $<\nu x>P \sim^s <\nu x>Q$
**proof**(*auto simp add*: *substClosed-def*)
**fix** $s$::($name \times name$) *list*

**have** *Res*: $\bigwedge P\ Q\ x\ s.\ [\![P[<s>] \sim Q[<s>];\ x \sharp s]\!] \Longrightarrow (<\nu x>P)[<s>] \sim (<\nu x>Q)[<s>]$
**by**(*force intro*: *Strong-Late-Bisim-Pres.resPres*)

**have** $\exists c::name.\ c \sharp (P, Q, s)$ **by**(*blast intro*: *name-exists-fresh*)
**then obtain** $c::name$ **where** *cFreshP*: $c \sharp P$ **and** *cFreshQ*: $c \sharp Q$ **and** *cFreshs*: $c \sharp s$
**by**(*force simp add*: *fresh-prod*)

**from** *PeqQ* **have** $P[<([(x, c)] \cdot s)>] \sim Q[<([(x, c)] \cdot s)>]$ **by**(*simp add*: *substClosed-def*)
**hence** $([(x, c)] \cdot P[<([(x, c)] \cdot s)>]) \sim ([(x, c)] \cdot Q[<([(x, c)] \cdot s)>])$ **by**(*rule bisimClosed*)
**hence** $([(x, c)] \cdot P)[<s>] \sim ([(x, c)] \cdot Q)[<s>]$ **by** *simp*
**hence** $(<\nu c>([(x, c)] \cdot P))[<s>] \sim (<\nu c>([(x, c)] \cdot Q))[<s>]$ **using** *cFreshs* **by**(*rule Res*)

**moreover from** *cFreshP cFreshQ* **have** $<\nu x>P = <\nu c>([(x, c)] \cdot P)$ **and**

$<\nu x>Q = <\nu c>([(x, c)] \cdot Q)$
  **by**(*simp add*: *alphaRes*)+

**ultimately show** $(<\nu x>P)[<s>] \sim (<\nu x>Q)[<s>]$ **by** *simp*
**qed**

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

  **assumes** $P \sim^s Q$

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

**end**

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

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

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

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

  **shows** $P \Longrightarrow_\tau P'$
**using** *assms* **by**(*simp add*: *r-into-rtrancl*)

**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** $P \longmapsto \tau \prec P' \Longrightarrow P' \Longrightarrow_\tau P'' \Longrightarrow P \Longrightarrow_\tau P''$
**by**(*auto dest*: *singleTauChain*)

**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** $F\ P'$
**using** *assms*
**by**(*drule-tac rtrancl-induct*) *auto*

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

**assumes** $P \Longrightarrow_\tau P'$

**shows** $(perm \cdot P) \Longrightarrow_\tau (perm \cdot P')$
**using** *assms*
**proof**(*induct rule*: *tauChainInduct*)
**case** *id*
**thus** *?case* **by** *simp*
**next**
**case**(*ih* $P''\ P'''$)
**have** $P \Longrightarrow_\tau P''$ **and** $P'' \longmapsto \tau \prec P'''$ **by** *fact+*
**hence** $(perm \cdot P'') \longmapsto \tau \prec (perm \cdot P''')$ **by**(*force dest*: *transitions.eqvt*)
**moreover have** $(perm \cdot P) \Longrightarrow_\tau (perm \cdot P'')$ **by** *fact*
**ultimately show** *?case* **by** *auto*
**qed**

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

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

**shows** $P \Longrightarrow_\tau P'$
**proof** −
**have** *rev perm* $\cdot$ $(perm \cdot P) = P$ **by**(*simp add*: *pt-rev-pi*[*OF pt-name-inst*, *OF at-name-inst*])
**moreover have** *rev perm* $\cdot$ $(perm \cdot P') = P'$ **by**(*simp add*: *pt-rev-pi*[*OF pt-name-inst*, *OF at-name-inst*])
**ultimately show** *?thesis* **using** *assms*
**by**(*drule-tac perm=rev perm* **in** *eqvtChainI*, *simp*)
**qed**

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

**shows** $P \Longrightarrow_\tau P' = (perm \cdot P) \Longrightarrow_\tau (perm \cdot P')$
**by**(*blast intro*: *eqvtChainE eqvtChainI*)

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

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

**shows** $x \sharp P'$
**using** *assms*
**proof**(*induct rule*: *tauChainInduct*)
**case** *id*
**thus** *?case* **by** *simp*
**next**
**case**(*ih* $P'$ $P''$)
**have** $x \sharp P$ **and** $x \sharp P \Longrightarrow x \sharp P'$ **by** *fact+*
**hence** $x \sharp P'$ **by** *simp*
**moreover have** $P' \longmapsto \tau \prec P''$ **by** *fact*
**ultimately show** *?case* **by**(*force intro*: *freshFreeDerivative*)
**qed**

**lemma** *matchChain*:
**fixes** $b$ :: *name*
**and** $P$ :: *pi*
**and** $P'$ :: *pi*

**assumes** $P \Longrightarrow_\tau P'$
**and** $P \neq P'$

**shows** $[b\frown b]P \Longrightarrow_\tau P'$
**using** *assms*
**proof**(*induct rule*: *tauChainInduct*)
**case** *id*
**thus** *?case* **by** *simp*
**next**
**case**(*ih* $P''$ $P'''$)
**have** $P''TransP'''$: $P'' \longmapsto \tau \prec P'''$ **by** *fact*
**show** $[b\frown b]P \Longrightarrow_\tau P'''$
**proof**(*cases* $P = P''$)
**assume** $P=P''$
**moreover with** $P''TransP'''$ **have** $[b\frown b]P \longmapsto \tau \prec P'''$ **by**(*force intro*: *Match*)
**thus** $[b\frown b]P \Longrightarrow_\tau P'''$ **by**(*rule singleTauChain*)
**next**
**assume** $P \neq P''$
**moreover have** $P \neq P'' \Longrightarrow [b\frown b]P \Longrightarrow_\tau P''$ **by** *fact*
**ultimately show** $[b\frown b]P \Longrightarrow_\tau P'''$ **using** $P''TransP'''$ **by**(*blast*)

**qed**
**qed**

**lemma** *mismatchChain*:
**fixes** $a$ :: *name*
**and** $b$ :: *name*
**and** $P$ :: *pi*
**and** $P'$ :: *pi*

**assumes** *PChain*: $P \Longrightarrow_\tau P'$
**and** *aineqb*: $a \neq b$
**and** *PineqP'*: $P \neq P'$

**shows** $[a{\neq}b]P \Longrightarrow_\tau P'$
**using** *PChain PineqP'*
**proof**(*induct rule*: *tauChainInduct*)
**case** *id*
**thus** *?case* **by** *simp*
**next**
**case**(*ih* $P''$ $P'''$)
**have** *P''TransP'''*: $P'' \longmapsto\tau \prec P'''$ **by** *fact*
**show** $[a{\neq}b]P \Longrightarrow_\tau P'''$
**proof**(*cases* $P = P''$)
**assume** $P{=}P''$
**moreover with** *aineqb* *P''TransP'''* **have** $[a{\neq}b]P \longmapsto\tau \prec P'''$ **by**(*force intro*: *Mismatch*)
**thus** $[a{\neq}b]P \Longrightarrow_\tau P'''$ **by**(*rule singleTauChain*)
**next**
**assume** $P \neq P''$
**moreover have** $P \neq P'' \Longrightarrow [a{\neq}b]P \Longrightarrow_\tau P''$ **by** *fact+*
**ultimately show** $[a{\neq}b]P \Longrightarrow_\tau P'''$ **using** *P''TransP'''* **by**(*blast*)
**qed**
**qed**

**lemma** *sum1Chain*[*rule-format*]:
**fixes** $P$ :: *pi*
**and** $P'$ :: *pi*
**and** $Q$ :: *pi*

**assumes** $P \Longrightarrow_\tau P'$
**and** $P \neq P'$

**shows** $P \oplus Q \Longrightarrow_\tau P'$
**using** *assms*
**proof**(*induct rule*: *tauChainInduct*)
**case** *id*
**thus** *?case* **by** *simp*
**next**
**case**(*ih* $P''$ $P'''$)

**have** $P''TransP'''$: $P'' \longmapsto \tau \prec P'''$ **by** *fact*
**show** $P \oplus Q \Longrightarrow_\tau P'''$
**proof**(*cases* $P = P''$)
**assume** $P{=}P''$
**moreover with** $P''TransP'''$ **have** $P \oplus Q \longmapsto \tau \prec P'''$ **by**(*force intro*: *Sum1*)
**thus** $P \oplus Q \Longrightarrow_\tau P'''$ **by**(*force intro*: *singleTauChain*)
**next**
**assume** $P \neq P''$
**moreover have** $P \neq P'' \Longrightarrow P \oplus Q \Longrightarrow_\tau P''$ **by** *fact*
**ultimately show** $P \oplus Q \Longrightarrow_\tau P'''$ **using** $P''TransP'''$ **by**(*force*)
**qed**
**qed**

**lemma** *sum2Chain*[*rule-format*]:
**fixes** $P$ :: *pi*
**and** $Q$ :: *pi*
**and** $Q'$ :: *pi*

**assumes** $Q \Longrightarrow_\tau Q'$
**and** $Q \neq Q'$

**shows** $P \oplus Q \Longrightarrow_\tau Q'$
**using** *assms*
**proof**(*induct rule*: *tauChainInduct*)
**case** *id*
**thus** *?case* **by** *simp*
**next**
**case**(*ih* $Q''$ $Q'''$)
**have** $Q''TransQ'''$: $Q'' \longmapsto \tau \prec Q'''$ **by** *fact*
**show** $P \oplus Q \Longrightarrow_\tau Q'''$
**proof**(*cases* $Q = Q''$)
**assume** $Q{=}Q''$
**moreover with** $Q''TransQ'''$ **have** $P \oplus Q \longmapsto \tau \prec Q'''$ **by**(*force intro*: *Sum2*)
**thus** $P \oplus Q \Longrightarrow_\tau Q'''$ **by**(*force intro*: *singleTauChain*)
**next**
**assume** $Q \neq Q''$
**moreover have** $Q \neq Q'' \Longrightarrow P \oplus Q \Longrightarrow_\tau Q''$ **by** *fact*
**ultimately show** $P \oplus Q \Longrightarrow_\tau Q'''$ **using** $Q''TransQ'''$ **by** *blast*
**qed**
**qed**

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

**assumes** $P \Longrightarrow_\tau P'$

**shows** $P \parallel Q \Longrightarrow_\tau P' \parallel Q$
**using** *assms*
**proof**(*induct rule*: *tauChainInduct*)
**case** *id*
**thus** *?case* **by** *simp*
**next**
**case**(*ih* $P''$ $P'$)
**have** $P''$*TransP*$'$: $P'' \longmapsto \tau \prec P'$ **by** *fact*
**have** *IH*: $P \parallel Q \Longrightarrow_\tau P'' \parallel Q$ **by** *fact*

**have** $P'' \parallel Q \longmapsto \tau \prec P' \parallel Q$ **using** $P''$*TransP*$'$ **by**(*force intro*: *Par1F*)
**thus** $P \parallel Q \Longrightarrow_\tau P' \parallel Q$ **using** *IH* **by**(*force*)
**qed**

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

**assumes** $Q \Longrightarrow_\tau Q'$

**shows** $P \parallel Q \Longrightarrow_\tau P \parallel Q'$
**using** *assms*
**proof**(*induct rule*: *tauChainInduct*)
**case** *id*
**thus** *?case* **by** *simp*
**next**
**case**(*ih* $Q''$ $Q'$)
**have** $Q''$*TransQ*$'$: $Q'' \longmapsto \tau \prec Q'$ **by** *fact*
**have** *IH*: $P \parallel Q \Longrightarrow_\tau P \parallel Q''$ **by** *fact*

**have** $P \parallel Q'' \longmapsto \tau \prec P \parallel Q'$ **using** $Q''$*TransQ*$'$ **by**(*force intro*: *Par2F*)
**thus** $P \parallel Q \Longrightarrow_\tau P \parallel Q'$ **using** *IH* **by**(*force*)
**qed**

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

**assumes** $P \Longrightarrow_\tau P'$
**and** $Q \Longrightarrow_\tau Q'$

**shows** $P \parallel Q \Longrightarrow_\tau P' \parallel Q'$
**proof** −
**from** ‹$P \Longrightarrow_\tau P'$› **have** $P \parallel Q \Longrightarrow_\tau P' \parallel Q$ **by**(*rule Par1Chain*)
**moreover from** ‹$Q \Longrightarrow_\tau Q'$› **have** $P' \parallel Q \Longrightarrow_\tau P' \parallel Q'$ **by**(*rule Par2Chain*)
**ultimately show** *?thesis* **by** *auto*

**qed**

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

  **assumes** $P \Longrightarrow_\tau P'$

  **shows** $<\nu a>P \Longrightarrow_\tau <\nu a>P'$
**using** *assms*
**proof**(*induct rule*: *tauChainInduct*)
  **case** *id*
  **thus** *?case* **by** *simp*
**next**
  **case**(*ih* $P''$ $P'''$)
  **have** $P'' \longmapsto\tau \prec P'''$ **by** *fact*
  **hence** $<\nu a>P'' \longmapsto\tau \prec <\nu a>P'''$ **by**(*force intro*: *ResF*)
  **moreover have** $<\nu a>P \Longrightarrow_\tau <\nu a>P''$ **by** *fact*
  **ultimately show** *?case* **by** *force*
**qed**

**lemma** *substChain*:
  **fixes** $P$ :: *pi*
  **and** $x$ :: *name*
  **and** $b$ :: *name*
  **and** $P'$ :: *pi*

  **assumes** *PTrans*: $P[x::=b] \Longrightarrow_\tau P'$

  **shows** $P[x::=b] \Longrightarrow_\tau P'[x::=b]$
**proof**(*cases* $x=b$)
  **assume** $x = b$
  **with** *PTrans* **show** *?thesis* **by** *simp*
**next**
  **assume** $x \neq b$
  **hence** $x \sharp P[x::=b]$ **by**(*simp add*: *fresh-fact2*)
  **with** *PTrans* **have** $x \sharp P'$ **by**(*force intro*: *freshChain*)
  **hence** $P' = P'[x::=b]$ **by**(*simp add*: *forget*)
  **with** *PTrans* **show** *?thesis* **by** *simp*
**qed**

**lemma** *bangChain*:
  **fixes** $P$ :: *pi*
  **and** $P'$ :: *pi*

  **assumes** *PTrans*: $P \parallel !P \Longrightarrow_\tau P'$
  **and** *P'ineq*: $P' \neq P \parallel !P$

**shows** $!P \Longrightarrow_\tau P'$
**using** *assms*
**proof**(*induct rule*: *tauChainInduct*)
**case** *id*
**thus** *?case* **by** *simp*
**next**
**case**(*ih* $P'$ $P''$)
**show** *?case*
**proof**(*cases* $P' = P \parallel !P$)
**case** *True*
**from** ‹$P' \longmapsto\tau \prec P''$› ‹$P' = P \parallel !P$› **have** $!P \longmapsto\tau \prec P''$ **by**(*blast intro*: *Bang*)
**thus** *?thesis* **by** *auto*
**next**
**case** *False*
**from** ‹$P' \neq P \parallel !P$› **have** $!P \Longrightarrow_\tau P'$ **by**(*rule ih*)
**with** ‹$P' \longmapsto\tau \prec P''$› **show** *?thesis* **by** *auto*
**qed**
**qed**

**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*

**using** *assms*
**by**(*auto simp add*: *transition-def residual.inject*)

**lemma** *transitionI*:
**fixes** $P$ :: *pi*
**and** $P'''$ :: *pi*
**and** $\alpha$ :: *freeRes*
**and** $P''$ :: *pi*
**and** $P'$ :: *pi*
**and** $a$ :: *name*
**and** $x$ :: *name*
**and** $u$ :: *name*

**shows** $[\![P \Longrightarrow_\tau P'''; P''' \longmapsto \alpha \prec P''; P'' \Longrightarrow_\tau P']\!] \Longrightarrow P \Longrightarrow_l \alpha \prec P'$
**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** $[\![P \Longrightarrow_\tau P'''; P''' \longmapsto a{<}x{>} \prec P''; P''[x{::=}u] \Longrightarrow_\tau P']\!] \Longrightarrow P \Longrightarrow_l u\ in\ P'' \rightarrow a{<}x{>} \prec P'$
**proof** −
**assume** $P \Longrightarrow_\tau P'''$ **and** $P''' \longmapsto \alpha \prec P''$ **and** $P'' \Longrightarrow_\tau P'$
**thus** $P \Longrightarrow_l \alpha \prec P'$
**by**(*force simp add*: *transition-def*)
**next**
**assume** $P \Longrightarrow_\tau P'''$ **and** $P''' \longmapsto a{<}\nu x{>} \prec P''$ **and** $P'' \Longrightarrow_\tau P'$
**thus** $P \Longrightarrow_l a{<}\nu x{>} \prec P'$
**by**(*force simp add*: *transition-def*)
**next**
**assume** $P \Longrightarrow_\tau P'''$ **and** $P''' \longmapsto a{<}x{>} \prec P''$ **and** $P''[x{::=}u] \Longrightarrow_\tau P'$
**thus** $P \Longrightarrow_l u\ in\ P'' \rightarrow a{<}x{>} \prec P'$
**by**(*force simp add*: *inputTransition-def*)
**qed**

**lemma** *transitionE*:
**fixes** $P$ :: *pi*
**and** $\alpha$ :: *freeRes*
**and** $P'$ :: *pi*
**and** $P''$ :: *pi*
**and** $a$ :: *name*
**and** $u$ :: *name*
**and** $x$ :: *name*

**shows** $P \Longrightarrow_l \alpha \prec P' \Longrightarrow \exists P''\ P'''.\ P \Longrightarrow_\tau P'' \wedge P'' \longmapsto \alpha \prec P''' \wedge P''' \Longrightarrow_\tau P'$ (**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** $[\![P \Longrightarrow_l u\ in\ P'' \rightarrow a{<}x{>} \prec P']\!] \Longrightarrow \exists P'''.\ P \Longrightarrow_\tau P''' \wedge P''' \longmapsto a{<}x{>} \prec P'' \wedge P''[x{::=}u] \Longrightarrow_\tau P'$
**proof** −
**assume** $P \Longrightarrow_l \alpha \prec P'$

**thus** *?thesis1* **by**(*auto simp add*: *transition-def residual.inject*)
**next**
**assume** $P \Longrightarrow_l a{<}\nu x{>} \prec P'$ **and** $x \sharp P$
**thus** $\exists P'' P'''. P \Longrightarrow_\tau P''' \wedge P''' \longmapsto a{<}\nu x{>} \prec P'' \wedge P'' \Longrightarrow_\tau P'$
**using** [[*hypsubst-thin* = *true*]]
**apply**(*auto simp add*: *transition-def residualInject name-abs-eq*)
**apply**(*rule-tac* $x=[(x, y)] \cdot P''$ **in** *exI*)
**apply**(*rule-tac* $x=P'$ **in** *exI*)
**apply**(*clarsimp*)
**apply**(*auto*)
**apply**(*subgoal-tac* $x \sharp P''$)
**apply**(*simp add*: *alphaBoundResidual name-swap*)
**using** *freshChain*
**apply**(*force dest*: *freshBoundDerivative*)
**using** *eqvtChainI*
**by** *simp*
**next**
**assume** *PTrans*: $P \Longrightarrow_l u$ *in* $P''{\rightarrow}a{<}x{>} \prec P'$
**thus** $\exists P'''. P \Longrightarrow_\tau P''' \wedge P''' \longmapsto a{<}x{>} \prec P'' \wedge P''[x{::=}u] \Longrightarrow_\tau P'$
**by**(*auto simp add*: *inputTransition-def*)
**qed**

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

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

**shows** $P \Longrightarrow_l u$ *in* $([(x, y)] \cdot P''){\rightarrow}a{<}y{>} \prec P'$
**proof**(*cases* $x=y$)
**assume** $x=y$
**with** *PTrans* **show** *?thesis* **by** *simp*
**next**
**assume** *xineqy*: $x \neq y$
**from** *PTrans* **obtain** $P'''$ **where** *PChain*: $P \Longrightarrow_\tau P'''$
**and** $P'''$*Trans*: $P''' \longmapsto a{<}x{>} \prec P''$
**and** $P''$*Chain*: $P''[x{::=}u] \Longrightarrow_\tau P'$
**by**(*blast dest*: *transitionE*)

**from** *PChain yFreshP* **have** $y \sharp P'''$ **by**(*rule freshChain*)
**with** $P'''$*Trans xineqy* **have** *yFreshP''*: $y \sharp P''$ **by**(*blast dest*: *freshBoundDerivative*)

**with** $P'''$*Trans* **have** $P''' \longmapsto a{<}y{>} \prec [(x,\ y)] \cdot P''$ **by**(*simp add*: *alphaBound-Residual*)
**moreover from** $P''$*Chain yFresh*$P''$ **have** $([(x,\ y)] \cdot P'')[y{::=}u] \Longrightarrow_\tau P'$
**by**(*simp add*: *renaming name-swap*)
**ultimately show** *?thesis* **using** *PChain* **by**(*blast intro*: *transitionI*)
**qed**

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

**shows** $P \Longrightarrow_l \tau \prec P' \Longrightarrow P \Longrightarrow_\tau P'$
**and** $P \neq P' \Longrightarrow P \Longrightarrow_\tau P' \Longrightarrow P \Longrightarrow_l \tau \prec P'$
**proof** −
**assume** $P \Longrightarrow_l \tau \prec P'$
**then obtain** $P''$ $P'''$ **where** $P \Longrightarrow_\tau P''$
**and** $P'' \longmapsto \tau \prec P'''$
**and** $P''' \Longrightarrow_\tau P'$
**by**(*blast dest*: *transitionE*)
**thus** $P \Longrightarrow_\tau P'$ **by** *auto*
**next**
**assume** $P \Longrightarrow_\tau P'$ **and** $P \neq P'$
**thus** $P \Longrightarrow_l \tau \prec P'$
**proof**(*induct rule*: *tauChainInduct*)
**case** *id*
**thus** *?case* **by** *simp*
**next**
**case**(*ih* $P''$ $P'''$)
**have** $P \Longrightarrow_\tau P''$ **and** $P'' \longmapsto \tau \prec P'''$ **by** *fact+*
**moreover have** $P''' \Longrightarrow_\tau P'''$ **by** *simp*
**ultimately show** *?case* **by**(*rule transitionI*)
**qed**
**qed**

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

**shows** $P \longmapsto a{<}\nu x{>} \prec P' \Longrightarrow P \Longrightarrow_l a{<}\nu x{>} \prec P'$
**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** −
**assume** $P \longmapsto a{<}\nu x{>} \prec P'$
**moreover have** $P \Longrightarrow_\tau P$ **by** *simp*
**moreover have** $P' \Longrightarrow_\tau P'$ **by** *simp*
**ultimately show** $P \Longrightarrow_l a{<}\nu x{>} \prec P'$ **by**(*blast intro*: *transitionI*)

**next**
  **assume** $P \longmapsto a{<}x{>} \prec P'$
  **moreover have** $P \Longrightarrow_\tau P$ **by** *simp*
  **moreover have** $P'[x{::=}u] \Longrightarrow_\tau P'[x{::=}u]$ **by** *simp*
  **ultimately show** $P \Longrightarrow_l u\ in\ P' \rightarrow a{<}x{>} \prec P'[x{::=}u]$ **by**(*blast intro*: *transitionI*)
**next**
  **assume** $P \longmapsto \alpha \prec P'$
  **moreover have** $P \Longrightarrow_\tau P$ **by** *simp*
  **moreover have** $P' \Longrightarrow_\tau P'$ **by** *simp*
  **ultimately show** $P \Longrightarrow_l \alpha \prec P'$ **by**(*blast intro*: *transitionI*)
**qed**

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

  **shows** $\tau.(P) \Longrightarrow_l \tau \prec P$
**proof** −
  **have** $\tau.(P) \Longrightarrow_\tau \tau.(P)$ **by** *simp*
  **moreover have** $\tau.(P) \longmapsto \tau \prec P$ **by**(*rule transitions.Tau*)
  **moreover have** $P \Longrightarrow_\tau P$ **by** *simp*
  **ultimately show** *?thesis* **by**(*rule transitionI*)
**qed**

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

  **shows** $a{<}x{>}.P \Longrightarrow_l u\ in\ P \rightarrow a{<}x{>} \prec P[x{::=}u]$
**proof** −
  **have** $a{<}x{>}.P \Longrightarrow_\tau a{<}x{>}.P$ **by** *simp*
  **moreover have** $a{<}x{>}.P \longmapsto a{<}x{>} \prec P$ **by**(*rule Input*)
  **moreover have** $P[x{::=}u] \Longrightarrow_\tau P[x{::=}u]$ **by** *simp*
  **ultimately show** *?thesis* **by**(*rule transitionI*)
**qed**

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

  **shows** $a\{b\}.P \Longrightarrow_l a[b] \prec P$
**proof** −
  **have** $a\{b\}.P \Longrightarrow_\tau a\{b\}.P$ **by** *simp*
  **moreover have** $a\{b\}.P \longmapsto a[b] \prec P$ **by**(*rule transitions.Output*)
  **moreover have** $P \Longrightarrow_\tau P$ **by** *simp*
  **ultimately show** *?thesis* **by**(*rule transitionI*)
**qed**

**lemma** *Match*:
**fixes** $P$ :: *pi*
**and** $Rs$ :: *residual*
**and** $a$ :: *name*
**and** $u$ :: *name*
**and** $b$ :: *name*
**and** $x$ :: *name*
**and** $P'$ :: *pi*

**shows** $P \Longrightarrow_l Rs \Longrightarrow [a\frown a]P \Longrightarrow_l Rs$
**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** −
**assume** *PTrans*: $P \Longrightarrow_l Rs$
**thus** $[a\frown a]P \Longrightarrow_l Rs$
**proof**(*nominal-induct avoiding*: $P$ *rule*: *residual.strong-inducts*)
**case**(*BoundR* $b$ $x$ $P'$)
**have** *PTrans*: $P \Longrightarrow_l b«x» \prec P'$ **and** *xFreshP*: $x \sharp P$ **by** *fact+*
**from** *PTrans* **obtain** $b'$ **where** *beq*: $b = BoundOutputS\ b'$ **by**(*cases* $b$) *auto*
**with** *PTrans xFreshP* **obtain** $P''$ $P'''$ **where** *PTrans*: $P \Longrightarrow_\tau P''$
**and** $P''$*Trans*: $P'' \longmapsto b'{<}\nu x{>} \prec P'''$
**and** $P'''$*Trans*: $P''' \Longrightarrow_\tau P'$
**by**(*blast dest*: *transitionE*)
**show** *?case*
**proof**(*cases* $P = P''$)
**assume** $P = P''$
**moreover have** $[a\frown a]P \Longrightarrow_\tau [a\frown a]P$ **by** *simp*
**moreover from** $P''$*Trans* **have** $[a\frown a]P'' \longmapsto b'{<}\nu x{>} \prec P'''$ **by**(*rule Match*)
**ultimately show** *?thesis* **using** *beq* $P'''$*Trans* **by**(*blast intro*: *transitionI*)
**next**
**assume** $P \neq P''$
**with** *PTrans* **have** $[a\frown a]P \Longrightarrow_\tau P''$ **by**(*rule matchChain*)
**thus** *?thesis* **using** *beq* $P''$*Trans* $P'''$*Trans* **by**(*blast intro*: *transitionI*)
**qed**
**next**
**case**(*FreeR* $\alpha$ $P'$)
**have** $P \Longrightarrow_l \alpha \prec P'$ **by** *fact*

**then obtain** $P''$ $P'''$ **where** *PTrans*: $P \Longrightarrow_\tau P''$
**and** $P''$*Trans*: $P'' \longmapsto \alpha \prec P'''$
**and** $P'''$*Trans*: $P''' \Longrightarrow_\tau P'$
**by**(*blast dest*: *transitionE*)
**show** *?case*
**proof**(*cases* $P = P''$)
**assume** $P = P''$
**moreover have** $[a\frown a]P \Longrightarrow_\tau [a\frown a]P$ **by** *simp*
**moreover from** $P''$*Trans* **have** $[a\frown a]P'' \longmapsto \alpha \prec P'''$ **by**(*rule transitions.Match*)
**ultimately show** *?thesis* **using** $P'''$*Trans* **by**(*blast intro*: *transitionI*)

```
    next
      assume P ≠ P''
      with PTrans have [a⌢a]P ⟹τ P'' by(rule matchChain)
      thus ?thesis using P''Trans P'''Trans by(rule transitionI)
    qed
  qed
next
  assume P ⟹lu in P''→b<x> ≺ P'
  then obtain P''' where PChain: P ⟹τ P'''
                   and P'''Trans: P''' ⟼b<x> ≺ P''
                   and P''Chain: P''[x::=u] ⟹τ P'
    by(blast dest: transitionE)
  show [a⌢a]P ⟹lu in P''→b<x> ≺ P'
  proof(cases P=P''')
    assume P=P'''
    moreover have [a⌢a]P ⟹τ [a⌢a]P by simp
   moreover from P'''Trans have [a⌢a]P''' ⟼b<x> ≺ P'' by(rule Late-Semantics.Match)
    ultimately show ?thesis using P''Chain by(blast intro: transitionI)
  next
    assume P ≠ P'''
    with PChain have [a⌢a]P ⟹τ P''' by(rule matchChain)
    thus ?thesis using P'''Trans P''Chain by(rule transitionI)
  qed
qed

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

  shows ⟦P ⟹l Rs; a ≠ c⟧ ⟹ [a≠c]P ⟹l Rs
  and   ⟦P ⟹lu in P''→b<x> ≺ P'; a ≠ c⟧ ⟹ [a≠c]P ⟹lu in P''→b<x> ≺
P'
proof -
  assume PTrans: P ⟹l Rs
     and aineqc: a ≠ c
  thus [a≠c]P ⟹l Rs
  proof(nominal-induct avoiding: P rule: residual.strong-inducts)
    case(BoundR b x P')
    have PTrans: P ⟹l b«x» ≺ P' and xFreshP: x ♯ P by fact+
    from PTrans obtain b' where beq: b = BoundOutputS b' by(cases b, auto)
    with PTrans xFreshP obtain P'' P''' where PTrans: P ⟹τ P''
                                        and P''Trans: P'' ⟼b'<νx> ≺ P'''
                                        and P'''Trans: P''' ⟹τ P'
```

**by**(*blast dest*: *transitionE*)
**show** *?case*
**proof**(*cases* $P = P''$)
**assume** $P = P''$
**moreover have** $[a{\neq}c]P \Longrightarrow_\tau [a{\neq}c]P$ **by** *simp*
**moreover from** $P''$*Trans aineqc* **have** $[a{\neq}c]P'' \longmapsto b'{<}\nu x{>} \prec P'''$ **by**(*rule transitions.Mismatch*)
**ultimately show** *?thesis* **using** *beq* $P'''$*Trans* **by**(*blast intro*: *transitionI*)
**next**
**assume** $P \neq P''$
**with** *PTrans aineqc* **have** $[a{\neq}c]P \Longrightarrow_\tau P''$ **by**(*rule mismatchChain*)
**thus** *?thesis* **using** *beq* $P''$*Trans* $P'''$*Trans* **by**(*blast intro*: *transitionI*)
**qed**
**next**
**case**(*FreeR* $\alpha$ $P'$)
**have** $P \Longrightarrow_l \alpha \prec P'$ **by** *fact*

**then obtain** $P''$ $P'''$ **where** *PTrans*: $P \Longrightarrow_\tau P''$
**and** $P''$*Trans*: $P'' \longmapsto \alpha \prec P'''$
**and** $P'''$*Trans*: $P''' \Longrightarrow_\tau P'$
**by**(*blast dest*: *transitionE*)
**show** *?case*
**proof**(*cases* $P = P''$)
**assume** $P = P''$
**moreover have** $[a{\neq}c]P \Longrightarrow_\tau [a{\neq}c]P$ **by** *simp*
**moreover from** $P''$*Trans* ‹$a \neq c$› **have** $[a{\neq}c]P'' \longmapsto \alpha \prec P'''$ **by**(*rule transitions.Mismatch*)
**ultimately show** *?thesis* **using** $P'''$*Trans* **by**(*blast intro*: *transitionI*)
**next**
**assume** $P \neq P''$
**with** *PTrans aineqc* **have** $[a{\neq}c]P \Longrightarrow_\tau P''$ **by**(*rule mismatchChain*)
**thus** *?thesis* **using** $P''$*Trans* $P'''$*Trans* **by**(*rule transitionI*)
**qed**
**qed**
**next**
**assume** *aineqc*: $a \neq c$
**assume** $P \Longrightarrow_l u$ *in* $P'' \rightarrow b{<}x{>} \prec P'$
**then obtain** $P'''$ **where** *PChain*: $P \Longrightarrow_\tau P'''$
**and** $P'''$*Trans*: $P''' \longmapsto b{<}x{>} \prec P''$
**and** $P''$*Chain*: $P''[x{::=}u] \Longrightarrow_\tau P'$
**by**(*blast dest*: *transitionE*)
**show** $[a{\neq}c]P \Longrightarrow_l u$ *in* $P'' \rightarrow b{<}x{>} \prec P'$
**proof**(*cases* $P{=}P'''$)
**assume** $P{=}P'''$
**moreover have** $[a{\neq}c]P \Longrightarrow_\tau [a{\neq}c]P$ **by** *simp*
**moreover from** $P'''$*Trans aineqc* **have** $[a{\neq}c]P''' \longmapsto b{<}x{>} \prec P''$ **by**(*rule Late-Semantics.Mismatch*)
**ultimately show** *?thesis* **using** $P''$*Chain* **by**(*blast intro*: *transitionI*)
**next**

**assume** $P \neq P'''$
**with** *PChain aineqc* **have** $[a{\neq}c]P \Longrightarrow_\tau P'''$ **by**(*rule mismatchChain*)
**thus** *?thesis* **using** *P'''Trans P''Chain* **by**(*rule transitionI*)
**qed**
**qed**

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

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

**shows** $<\nu b>P \Longrightarrow_l a<\nu b> \prec P'$
**proof** −
**from** *Trans* **obtain** $P''$ $P'''$ **where** $A$: $P \Longrightarrow_\tau P''$
**and** $B$: $P'' \longmapsto a[b] \prec P'''$
**and** $C$: $P''' \Longrightarrow_\tau P'$
**by**(*force dest*: *transitionE*)
**from** $A$ **have** $<\nu b>P \Longrightarrow_\tau <\nu b>P''$ **by**(*rule ResChain*)
**moreover from** *B aInEqb* **have** $<\nu b>P'' \longmapsto a<\nu b> \prec P'''$ **by**(*rule Open*)
**ultimately show** *?thesis* **using** $C$ **by**(*force simp add*: *transition-def*)
**qed**

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

**shows** $P \Longrightarrow_l Rs \Longrightarrow P \oplus Q \Longrightarrow_l Rs$
**and** $P \Longrightarrow_l u\ in\ P'' \rightarrow a<x> \prec P' \Longrightarrow P \oplus Q \Longrightarrow_l u\ in\ P'' \rightarrow a<x> \prec P'$
**proof** −
**assume** $P \Longrightarrow_l Rs$
**thus** $P \oplus Q \Longrightarrow_l Rs$
**proof**(*nominal-induct avoiding*: $P$ *rule*: *residual.strong-inducts*)
**case**(*BoundR a x P' P*)
**have** *PTrans*: $P \Longrightarrow_l a«x» \prec P'$
**and** *xFreshP*: $x \sharp P$ **by** *fact+*
**from** *PTrans* **obtain** $a'$ **where** *aeq*: $a = BoundOutputS\ a'$ **by**(*cases a*, *auto*)
**with** *PTrans xFreshP* **obtain** $P''$ $P'''$ **where** *PTrans*: $P \Longrightarrow_\tau P''$
**and** *P''Trans*: $P'' \longmapsto a'<\nu x> \prec P'''$
**and** *P'''Trans*: $P''' \Longrightarrow_\tau P'$

**by**(*blast dest*: *transitionE*)
**show** *?case*
**proof**(*cases* $P = P''$)
**assume** $P = P''$
**moreover have** $P \oplus Q \Longrightarrow_\tau P \oplus Q$ **by** *simp*
**moreover from** *P''Trans* **have** $P'' \oplus Q \longmapsto a'<\nu x> \prec P'''$ **by**(*rule transitions.Sum1*)
**ultimately show** *?thesis* **using** *P'''Trans aeq* **by**(*blast intro*: *transitionI*)
**next**
**assume** $P \neq P''$
**with** *PTrans* **have** $P \oplus Q \Longrightarrow_\tau P''$ **by**(*rule sum1Chain*)
**thus** *?thesis* **using** *P''Trans P'''Trans aeq* **by**(*blast intro*: *transitionI*)
**qed**
**next**
**case**(*FreeR* $\alpha$ $P'$)
**have** $P \Longrightarrow_l \alpha \prec P'$ **by** *fact*

**then obtain** $P''$ $P'''$ **where** *PTrans*: $P \Longrightarrow_\tau P''$
**and** *P''Trans*: $P'' \longmapsto \alpha \prec P'''$
**and** *P'''Trans*: $P''' \Longrightarrow_\tau P'$
**by**(*blast dest*: *transitionE*)
**show** *?case*
**proof**(*cases* $P = P''$)
**assume** $P = P''$
**moreover have** $P \oplus Q \Longrightarrow_\tau P \oplus Q$ **by** *simp*
**moreover from** *P''Trans* **have** $P'' \oplus Q \longmapsto \alpha \prec P'''$ **by**(*rule transitions.Sum1*)
**ultimately show** *?thesis* **using** *P'''Trans* **by**(*blast intro*: *transitionI*)
**next**
**assume** $P \neq P''$
**with** *PTrans* **have** $P \oplus Q \Longrightarrow_\tau P''$ **by**(*rule sum1Chain*)
**thus** *?thesis* **using** *P''Trans P'''Trans* **by**(*rule transitionI*)
**qed**
**qed**
**next**
**assume** $P \Longrightarrow_l u \textit{ in } P'' \rightarrow a<x> \prec P'$
**then obtain** $P'''$ **where** *PChain*: $P \Longrightarrow_\tau P'''$
**and** *P'''Trans*: $P''' \longmapsto a<x> \prec P''$
**and** *P''Chain*: $P''[x::=u] \Longrightarrow_\tau P'$
**by**(*blast dest*: *transitionE*)
**show** $P \oplus Q \Longrightarrow_l u \textit{ in } P'' \rightarrow a<x> \prec P'$
**proof**(*cases* $P = P'''$)
**assume** $P = P'''$
**moreover have** $P \oplus Q \Longrightarrow_\tau P \oplus Q$ **by** *simp*
**moreover from** *P'''Trans* **have** $P''' \oplus Q \longmapsto a<x> \prec P''$ **by**(*rule transitions.Sum1*)
**ultimately show** *?thesis* **using** *P''Chain* **by**(*blast intro*: *transitionI*)
**next**
**assume** $P \neq P'''$

**with** *PChain* **have** $P \oplus Q \Longrightarrow_\tau P'''$ **by**(*rule sum1Chain*)
**thus** *?thesis* **using** *P'''Trans P''Chain* **by**(*blast intro*: *transitionI*)
**qed**
**qed**

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

**shows** $Q \Longrightarrow_l Rs \Longrightarrow P \oplus Q \Longrightarrow_l Rs$
**and** $Q \Longrightarrow_l u\ in\ Q'' \rightarrow a<x> \prec Q' \Longrightarrow P \oplus Q \Longrightarrow_l u\ in\ Q'' \rightarrow a<x> \prec Q'$
**proof** −
**assume** $Q \Longrightarrow_l Rs$
**thus** $P \oplus Q \Longrightarrow_l Rs$
**proof**(*nominal-induct avoiding*: *Q rule*: *residual.strong-inducts*)
**case**(*BoundR a x Q' Q*)
**have** *QTrans*: $Q \Longrightarrow_l a«x» \prec Q'$
**and** *xFreshQ*: $x \sharp Q$ **by** *fact+*
**from** *QTrans* **obtain** $a'$ **where** *aeq*: $a = BoundOutputS\ a'$ **by**(*cases a*, *auto*)
**with** *QTrans xFreshQ* **obtain** $Q''\ Q'''$ **where** *QTrans*: $Q \Longrightarrow_\tau Q''$
**and** *Q''Trans*: $Q'' \longmapsto a'<\nu x> \prec Q'''$
**and** *Q'''Trans*: $Q''' \Longrightarrow_\tau Q'$
**by**(*blast dest*: *transitionE*)
**show** *?case*
**proof**(*cases* $Q = Q''$)
**assume** $Q = Q''$
**moreover have** $P \oplus Q \Longrightarrow_\tau P \oplus Q$ **by** *simp*
**moreover from** *Q''Trans* **have** $P \oplus Q'' \longmapsto a'<\nu x> \prec Q'''$ **by**(*rule transitions.Sum2*)
**ultimately show** *?thesis* **using** *Q'''Trans aeq* **by**(*blast intro*: *transitionI*)
**next**
**assume** $Q \neq Q''$
**with** *QTrans* **have** $P \oplus Q \Longrightarrow_\tau Q''$ **by**(*rule sum2Chain*)
**thus** *?thesis* **using** *Q''Trans Q'''Trans aeq* **by**(*blast intro*: *transitionI*)
**qed**
**next**
**case**(*FreeR* $\alpha$ $Q'$)
**have** $Q \Longrightarrow_l \alpha \prec Q'$ **by** *fact*

**then obtain** $Q''\ Q'''$ **where** *QTrans*: $Q \Longrightarrow_\tau Q''$
**and** *Q''Trans*: $Q'' \longmapsto \alpha \prec Q'''$
**and** *Q'''Trans*: $Q''' \Longrightarrow_\tau Q'$
**by**(*blast dest*: *transitionE*)

**show** *?case*
**proof**(*cases* $Q = Q''$)
**assume** $Q = Q''$
**moreover have** $P \oplus Q \Longrightarrow_\tau P \oplus Q$ **by** *simp*
**moreover from** $Q''Trans$ **have** $P \oplus Q'' \longmapsto \alpha \prec Q'''$ **by**(*rule transitions.Sum2*)
**ultimately show** *?thesis* **using** $Q'''Trans$ **by**(*blast intro*: *transitionI*)
**next**
**assume** $Q \neq Q''$
**with** *QTrans* **have** $P \oplus Q \Longrightarrow_\tau Q''$ **by**(*rule sum2Chain*)
**thus** *?thesis* **using** $Q''Trans$ $Q'''Trans$ **by**(*rule transitionI*)
**qed**
**qed**
**next**
**assume** $Q \Longrightarrow_l u\ in\ Q'' \rightarrow a{<}x{>} \prec Q'$
**then obtain** $Q'''$ **where** *QChain*: $Q \Longrightarrow_\tau Q'''$
**and** $Q'''Trans$: $Q''' \longmapsto a{<}x{>} \prec Q''$
**and** $Q''Chain$: $Q''[x{::=}u] \Longrightarrow_\tau Q'$
**by**(*blast dest*: *transitionE*)
**show** $P \oplus Q \Longrightarrow_l u\ in\ Q'' \rightarrow a{<}x{>} \prec Q'$
**proof**(*cases* $Q = Q'''$)
**assume** $Q = Q'''$
**moreover have** $P \oplus Q \Longrightarrow_\tau P \oplus Q$ **by** *simp*
**moreover from** $Q'''Trans$ **have** $P \oplus Q''' \longmapsto a{<}x{>} \prec Q''$ **by**(*rule transitions.Sum2*)
**ultimately show** *?thesis* **using** $Q''Chain$ **by**(*blast intro*: *transitionI*)
**next**
**assume** $Q \neq Q'''$
**with** *QChain* **have** $P \oplus Q \Longrightarrow_\tau Q'''$ **by**(*rule sum2Chain*)
**thus** *?thesis* **using** $Q'''Trans$ $Q''Chain$ **by**(*blast intro*: *transitionI*)
**qed**
**qed**

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

**shows** $[\![P \Longrightarrow_l a{<}\nu x{>} \prec P';\ x \sharp Q]\!] \Longrightarrow P \parallel Q \Longrightarrow_l a{<}\nu x{>} \prec (P' \parallel Q)$
**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** −
**assume** *PTrans*: $P \Longrightarrow_l a{<}\nu x{>} \prec P'$
**assume** *xFreshQ*: $x \sharp Q$

**have** *Goal*: $\bigwedge P\ a\ x\ P'\ Q.\ [\![P \Longrightarrow_l a{<}\nu x{>} \prec P';\ x \sharp P;\ x \sharp Q]\!] \Longrightarrow P \parallel Q$

$\Longrightarrow_l a{<}\nu x{>} \prec (P' \parallel Q)$
  **proof** −
    **fix** *P a x P' Q*
    **assume** *PTrans*: $P \Longrightarrow_l a{<}\nu x{>} \prec P'$
    **assume** *xFreshP*: $x \sharp P$
    **assume** *xFreshQ*: $x \sharp (Q{::}pi)$

    **from** *PTrans xFreshP* **obtain** $P''$ $P'''$ **where** *PTrans*: $P \Longrightarrow_\tau P''$
                                              **and** *P''Trans*: $P'' \longmapsto a{<}\nu x{>} \prec P'''$
                                              **and** *P'''Trans*: $P''' \Longrightarrow_\tau P'$
      **by**(*blast dest*: *transitionE*)
    **from** *PTrans* **have** $P \parallel Q \Longrightarrow_\tau P'' \parallel Q$ **by**(*rule Par1Chain*)
     **moreover from** *P''Trans xFreshQ* **have** $P'' \parallel Q \longmapsto a{<}\nu x{>} \prec (P''' \parallel Q)$ **by**(*rule Par1B*)
    **moreover from** *P'''Trans* **have** $P''' \parallel Q \Longrightarrow_\tau P' \parallel Q$ **by**(*rule Par1Chain*)
    **ultimately show** $P \parallel Q \Longrightarrow_l a{<}\nu x{>} \prec (P' \parallel Q)$ **by**(*rule transitionI*)
  **qed**

  **have** $\exists$ *c::name*. $c \sharp (P, P', Q)$ **by**(*blast intro*: *name-exists-fresh*)
  **then obtain** *c::name* **where** *cFreshP*: $c \sharp P$ **and** *cFreshP'*: $c \sharp P'$ **and** *cFreshQ*: $c \sharp Q$
    **by**(*force simp add*: *fresh-prod*)

 **from** *cFreshP'* **have** $a{<}\nu x{>} \prec P' = a{<}\nu c{>} \prec ([(x, c)] \cdot P')$ **by**(*rule alphaBoundResidual*)
  **moreover have** $a{<}\nu x{>} \prec (P' \parallel Q) = a{<}\nu c{>} \prec (([(x, c)] \cdot P') \parallel Q)$
  **proof** −
    **from** *cFreshP' cFreshQ* **have** $c \sharp P' \parallel Q$ **by** *simp*
   **hence** $a{<}\nu x{>} \prec (P' \parallel Q) = a{<}\nu c{>} \prec ([(x, c)] \cdot (P' \parallel Q))$ **by**(*rule alphaBoundResidual*)
    **with** *cFreshQ xFreshQ* **show** *?thesis* **by**(*simp add*: *name-fresh-fresh*)
  **qed**
  **ultimately show** $P \parallel Q \Longrightarrow_l a{<}\nu x{>} \prec P' \parallel Q$ **using** *PTrans cFreshP cFreshQ* **by**(*force intro*: *Goal*)
**next**
  **assume** *PTrans*: $P \Longrightarrow_l u$ *in* $P'' \rightarrow a{<}x{>} \prec P'$
    **and** *xFreshQ*: $x \sharp Q$
  **from** *PTrans* **obtain** $P'''$ **where** *PChain*: $P \Longrightarrow_\tau P'''$
                          **and** *P'''Trans*: $P''' \longmapsto a{<}x{>} \prec P''$
                          **and** *P''Chain*: $P''[x{::=}u] \Longrightarrow_\tau P'$
    **by**(*blast dest*: *transitionE*)
  **from** *PChain* **have** $P \parallel Q \Longrightarrow_\tau P''' \parallel Q$ **by**(*rule Par1Chain*)
 **moreover from** *P'''Trans xFreshQ* **have** $P''' \parallel Q \longmapsto a{<}x{>} \prec (P'' \parallel Q)$ **by**(*rule Par1B*)
  **moreover have** $(P'' \parallel Q)[x{::=}u] \Longrightarrow_\tau P' \parallel Q$
  **proof** −
    **from** *P''Chain* **have** $P''[x{::=}u] \parallel Q \Longrightarrow_\tau P' \parallel Q$ **by**(*rule Par1Chain*)
    **with** *xFreshQ* **show** *?thesis* **by**(*simp add*: *forget*)
  **qed**

**ultimately show** $P \parallel Q \Longrightarrow_l u$ *in* $(P'' \parallel Q) \rightarrow a{<}x{>} \prec (P' \parallel Q)$ **by**(*rule transitionI*)
**qed**

**lemma** *Par1F*:
**fixes** $P$ :: *pi*
**and** $\alpha$ :: *freeRes*
**and** $P'$ :: *pi*

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

**shows** $P \parallel Q \Longrightarrow_l \alpha \prec (P' \parallel Q)$
**proof** −
**from** *PTrans* **obtain** $P''$ $P'''$ **where** *PTrans*: $P \Longrightarrow_\tau P''$
**and** $P''$*Trans*: $P'' \longmapsto \alpha \prec P'''$
**and** $P'''$*Trans*: $P''' \Longrightarrow_\tau P'$
**by**(*blast dest*: *transitionE*)
**from** *PTrans* **have** $P \parallel Q \Longrightarrow_\tau P'' \parallel Q$ **by**(*rule Par1Chain*)
**moreover from** $P''$*Trans* **have** $P'' \parallel Q \longmapsto \alpha \prec (P''' \parallel Q)$ **by**(*rule transitions.Par1F*)
**moreover from** $P'''$*Trans* **have** $P''' \parallel Q \Longrightarrow_\tau P' \parallel Q$ **by**(*rule Par1Chain*)
**ultimately show** *?thesis* **by**(*rule transitionI*)
**qed**

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

**shows** $Q \Longrightarrow_l a{<}\nu x{>} \prec Q' \Longrightarrow x \sharp P \Longrightarrow P \parallel Q \Longrightarrow_l a{<}\nu x{>} \prec (P \parallel Q')$
**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** −
**assume** *QTrans*: $Q \Longrightarrow_l a{<}\nu x{>} \prec Q'$
**assume** *xFreshP*: $x \sharp P$

**have** *Goal*: $\bigwedge Q\ a\ x\ Q'\ P.\ [\![Q \Longrightarrow_l a{<}\nu x{>} \prec Q';\ x \sharp Q;\ x \sharp P]\!] \Longrightarrow P \parallel Q \Longrightarrow_l a{<}\nu x{>} \prec (P \parallel Q')$
**proof** −
**fix** $Q\ a\ x\ Q'\ P$
**assume** *QTrans*: $Q \Longrightarrow_l a{<}\nu x{>} \prec Q'$
**assume** *xFreshQ*: $x \sharp Q$
**assume** *xFreshP*: $x \sharp (P{::}pi)$

**from** *QTrans xFreshQ* **obtain** $Q''$ $Q'''$ **where** *QTrans*: $Q \Longrightarrow_\tau Q''$

**and** $Q''$*Trans*: $Q'' \longmapsto a{<}\nu x{>} \prec Q'''$
**and** $Q'''$*Trans*: $Q''' \Longrightarrow_\tau Q'$
**by**(*blast dest*: *transitionE*)
**from** *QTrans* **have** $P \parallel Q \Longrightarrow_\tau P \parallel Q''$ **by**(*rule Par2Chain*)
**moreover from** $Q''$*Trans xFreshP* **have** $P \parallel Q'' \longmapsto a{<}\nu x{>} \prec (P \parallel Q''')$ **by**(*rule Par2B*)
**moreover from** $Q'''$*Trans* **have** $P \parallel Q''' \Longrightarrow_\tau P \parallel Q'$ **by**(*rule Par2Chain*)
**ultimately show** $P \parallel Q \Longrightarrow_l a{<}\nu x{>} \prec (P \parallel Q')$ **by**(*rule transitionI*)
**qed**

**have** $\exists$ *c*::*name*. $c \sharp (Q, Q', P)$ **by**(*blast intro*: *name-exists-fresh*)
**then obtain** *c*::*name* **where** *cFreshQ*: $c \sharp Q$ **and** *cFreshQ'*: $c \sharp Q'$ **and** *cFreshP*: $c \sharp P$
**by**(*force simp add*: *fresh-prod*)

**from** *cFreshQ'* **have** $a{<}\nu x{>} \prec Q' = a{<}\nu c{>} \prec ([(x, c)] \cdot Q')$ **by**(*rule alphaBoundResidual*)
**moreover have** $a{<}\nu x{>} \prec (P \parallel Q') = a{<}\nu c{>} \prec (P \parallel ([(x, c)] \cdot Q'))$
**proof** −
**from** *cFreshQ' cFreshP* **have** $c \sharp P \parallel Q'$ **by** *simp*
**hence** $a{<}\nu x{>} \prec (P \parallel Q') = a{<}\nu c{>} \prec ([(x, c)] \cdot (P \parallel Q'))$ **by**(*rule alphaBoundResidual*)
**with** *cFreshP xFreshP* **show** *?thesis* **by**(*simp add*: *name-fresh-fresh*)
**qed**
**ultimately show** $P \parallel Q \Longrightarrow_l a{<}\nu x{>} \prec P \parallel Q'$ **using** *QTrans cFreshQ cFreshP* **by**(*force intro*: *Goal*)
**next**
**assume** *QTrans*: $Q \Longrightarrow_l u$ *in* $Q'' \rightarrow a{<}x{>} \prec Q'$
**and** *xFreshP*: $x \sharp P$
**from** *QTrans* **obtain** $Q'''$ **where** *QChain*: $Q \Longrightarrow_\tau Q'''$
**and** $Q'''$*Trans*: $Q''' \longmapsto a{<}x{>} \prec Q''$
**and** $Q''$*Chain*: $Q''[x{::=}u] \Longrightarrow_\tau Q'$
**by**(*blast dest*: *transitionE*)
**from** *QChain* **have** $P \parallel Q \Longrightarrow_\tau P \parallel Q'''$ **by**(*rule Par2Chain*)
**moreover from** $Q'''$*Trans xFreshP* **have** $P \parallel Q''' \longmapsto a{<}x{>} \prec (P \parallel Q'')$ **by**(*rule Par2B*)
**moreover have** $(P \parallel Q'')[x{::=}u] \Longrightarrow_\tau P \parallel Q'$
**proof** −
**from** $Q''$*Chain* **have** $P \parallel (Q''[x{::=}u]) \Longrightarrow_\tau P \parallel Q'$ **by**(*rule Par2Chain*)
**with** *xFreshP* **show** *?thesis* **by**(*simp add*: *forget*)
**qed**
**ultimately show** $P \parallel Q \Longrightarrow_l u$ *in* $(P \parallel Q'') \rightarrow a{<}x{>} \prec (P \parallel Q')$ **by**(*rule transitionI*)
**qed**

**lemma** *Par2F*:
**fixes** $Q$ :: *pi*
**and** $\alpha$ :: *freeRes*
**and** $Q'$ :: *pi*

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

**shows** $P \parallel Q \Longrightarrow_l \alpha \prec (P \parallel Q')$
**proof** −
**from** *QTrans* **obtain** $Q''$ $Q'''$ **where** *QTrans*: $Q \Longrightarrow_\tau Q''$
**and** $Q''$*Trans*: $Q'' \longmapsto \alpha \prec Q'''$
**and** $Q'''$*Trans*: $Q''' \Longrightarrow_\tau Q'$
**by**(*blast dest*: *transitionE*)
**from** *QTrans* **have** $P \parallel Q \Longrightarrow_\tau P \parallel Q''$ **by**(*rule Par2Chain*)
**moreover from** $Q''$*Trans* **have** $P \parallel Q'' \longmapsto \alpha \prec (P \parallel Q''')$ **by**(*rule transitions.Par2F*)
**moreover from** $Q'''$*Trans* **have** $P \parallel Q''' \Longrightarrow_\tau P \parallel Q'$ **by**(*rule Par2Chain*)
**ultimately show** *?thesis* **by**(*rule transitionI*)
**qed**

**lemma** *Comm1*:
**fixes** $P$ :: *pi*
**and** $b$ :: *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** −
**from** *PTrans* **obtain** $P'''$ **where** *PChain*: $P \Longrightarrow_\tau P'''$
**and** $P'''$*Trans*: $P''' \longmapsto a{<}x{>} \prec P''$
**and** $P''$*Chain*: $P''[x::=b] \Longrightarrow_\tau P'$
**by**(*blast dest*: *transitionE*)
**from** *QTrans* **obtain** $Q''$ $Q'''$ **where** *QChain*: $Q \Longrightarrow_\tau Q'''$
**and** $Q'''$*Trans*: $Q''' \longmapsto a[b] \prec Q''$
**and** $Q''$*Chain*: $Q'' \Longrightarrow_\tau Q'$
**by**(*blast dest*: *transitionE*)

**from** *PChain QChain* **have** $P \parallel Q \Longrightarrow_\tau P''' \parallel Q'''$ **by**(*rule chainPar*)
**moreover from** $P'''$*Trans* $Q'''$*Trans* **have** $P''' \parallel Q''' \longmapsto \tau \prec P''[x::=b] \parallel Q''$
**by**(*rule Comm1*)
**moreover from** $P''$*Chain* $Q''$*Chain* **have** $P''[x::=b] \parallel Q'' \Longrightarrow_\tau P' \parallel Q'$ **by**(*rule chainPar*)
**ultimately show** *?thesis* **by**(*rule transitionI*)
**qed**

**lemma** *Comm2*:

**fixes** $P$ :: *pi*
**and** $a$ :: *name*
**and** $b$ :: *name*
**and** $P'$ :: *pi*
**and** $Q$ :: *pi*
**and** $x$ :: *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** −
**from** *PTrans* **obtain** $P''$ $P'''$ **where** *PChain*: $P \Longrightarrow_\tau P'''$
**and** $P'''$*Trans*: $P''' \longmapsto a[b] \prec P''$
**and** $P''$*Chain*: $P'' \Longrightarrow_\tau P'$
**by**(*blast dest*: *transitionE*)
**from** *QTrans* **obtain** $Q'''$ **where** *QChain*: $Q \Longrightarrow_\tau Q'''$
**and** $Q'''$*Trans*: $Q''' \longmapsto a{<}x{>} \prec Q''$
**and** $Q''$*Chain*: $Q''[x{::=}b] \Longrightarrow_\tau Q'$
**by**(*blast dest*: *transitionE*)

**from** *PChain QChain* **have** $P \parallel Q \Longrightarrow_\tau P''' \parallel Q'''$ **by**(*rule chainPar*)
**moreover from** $P'''$*Trans* $Q'''$*Trans* **have** $P''' \parallel Q''' \longmapsto \tau \prec P'' \parallel (Q''[x{::=}b])$
**by**(*rule Comm2*)
**moreover from** $P''$*Chain* $Q''$*Chain* **have** $P'' \parallel (Q''[x{::=}b]) \Longrightarrow_\tau P' \parallel Q'$ **by**(*rule chainPar*)
**ultimately show** *?thesis* **by**(*rule transitionI*)
**qed**

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

**assumes** *PTrans*: $P \Longrightarrow_l y \ in \ P'' \rightarrow a{<}x{>} \prec P'$
**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** −
**from** *PTrans* **obtain** $P'''$ **where** *PChain*: $P \Longrightarrow_\tau P'''$

**and** $P'''Trans$: $P''' \longmapsto a{<}x{>} \prec P''$
**and** $P''Chain$: $P''[x{::=}y] \Longrightarrow_\tau P'$
**by**(*blast dest*: *transitionE*)
**from** *QTrans yFreshQ* **obtain** $Q''$ $Q'''$ **where** $QChain$: $Q \Longrightarrow_\tau Q'''$
**and** $Q'''Trans$: $Q''' \longmapsto a{<}\nu y{>} \prec Q''$
**and** $Q''Chain$: $Q'' \Longrightarrow_\tau Q'$
**by**(*blast dest*: *transitionE*)

**from** *PChain yFreshP* **have** $yFreshP'''$: $y \sharp P'''$ **by**(*rule freshChain*)

**from** *PChain QChain* **have** $P \parallel Q \Longrightarrow_\tau P''' \parallel Q'''$ **by**(*rule chainPar*)
**moreover from** $P'''Trans$ $Q'''Trans$ $yFreshP'''$ **have** $P''' \parallel Q''' \longmapsto \tau \prec {<}\nu y{>}(P''[x{::=}y]$
$\parallel Q'')$
**by**(*rule Close1*)
**moreover have** ${<}\nu y{>}(P''[x{::=}y] \parallel Q'') \Longrightarrow_\tau {<}\nu y{>}(P' \parallel Q')$
**proof** −
**from** $P''Chain$ $Q''Chain$ **have** $P''[x{::=}y] \parallel Q'' \Longrightarrow_\tau P' \parallel Q'$ **by**(*rule chainPar*)
**thus** *?thesis* **by**(*rule ResChain*)
**qed**
**ultimately show** $P \parallel Q \Longrightarrow_l \tau \prec {<}\nu y{>}(P' \parallel Q')$ **by**(*rule transitionI*)
**qed**

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

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

**shows** $P \parallel Q \Longrightarrow_l \tau \prec {<}\nu y{>}(P' \parallel Q')$
**proof** −
**from** *PTrans yFreshP* **obtain** $P''$ $P'''$ **where** *PChain*: $P \Longrightarrow_\tau P'''$
**and** $P'''Trans$: $P''' \longmapsto a{<}\nu y{>} \prec P''$
**and** $P''Chain$: $P'' \Longrightarrow_\tau P'$
**by**(*blast dest*: *transitionE*)

**from** *QTrans* **obtain** $Q'''$ **where** *QChain*: $Q \Longrightarrow_\tau Q'''$
**and** $Q'''Trans$: $Q''' \longmapsto a{<}x{>} \prec Q''$
**and** $Q''Chain$: $Q''[x{::=}y] \Longrightarrow_\tau Q'$
**by**(*blast dest*: *transitionE*)

**from** *QChain yFreshQ* **have** *yFreshQ'''*: $y \sharp Q'''$ **by**(*rule freshChain*)

**from** *PChain QChain* **have** $P \parallel Q \Longrightarrow_\tau P''' \parallel Q'''$ **by**(*rule chainPar*)
**moreover from** *P'''Trans Q'''Trans yFreshQ'''* **have** $P''' \parallel Q''' \longmapsto \tau \prec <\nu y>(P'' \parallel (Q''[x::=y]))$
**by**(*rule Close2*)
**moreover have** $<\nu y>(P'' \parallel (Q''[x::=y])) \Longrightarrow_\tau <\nu y>(P' \parallel Q')$
**proof** −
**from** *P''Chain Q''Chain* **have** $P'' \parallel (Q''[x::=y]) \Longrightarrow_\tau P' \parallel Q'$ **by**(*rule chainPar*)
**thus** *?thesis* **by**(*rule ResChain*)
**qed**
**ultimately show** $P \parallel Q \Longrightarrow_l \tau \prec <\nu y>(P' \parallel Q')$ **by**(*rule transitionI*)
**qed**

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

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

**shows** $<\nu x>P \Longrightarrow_l \alpha \prec <\nu x>P'$
**proof** −
**from** *PTrans* **obtain** $P''$ $P'''$ **where** *PChain*: $P \Longrightarrow_\tau P''$
**and** *P''Trans*: $P'' \longmapsto \alpha \prec P'''$
**and** *P'''Chain*: $P''' \Longrightarrow_\tau P'$
**by**(*blast dest*: *transitionE*)

**from** *PChain* **have** $<\nu x>P \Longrightarrow_\tau <\nu x>P''$ **by**(*rule ResChain*)
**moreover from** *P''Trans xFreshAlpha* **have** $<\nu x>P'' \longmapsto \alpha \prec <\nu x>P'''$
**by**(*rule transitions.ResF*)
**moreover from** *P'''Chain* **have** $<\nu x>P''' \Longrightarrow_\tau <\nu x>P'$ **by**(*rule ResChain*)
**ultimately show** *?thesis* **by**(*rule transitionI*)
**qed**

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

**shows** $[\![P \Longrightarrow_l a<\nu x> \prec P'; y \neq a; y \neq x; x \sharp P]\!] \Longrightarrow <\nu y>P \Longrightarrow_l a<\nu x> \prec (<\nu y>P')$

**and** $[\![P \Longrightarrow_l u\ in\ P''{\rightarrow}a{<}x{>} \prec P';\ y \neq a;\ y \neq x;\ y \neq u]\!] \Longrightarrow {<}\nu y{>}P \Longrightarrow_l u\ in$ $({<}\nu y{>}P''){\rightarrow}a{<}x{>} \prec ({<}\nu y{>}P')$
**proof** −
  **assume** *PTrans*: $P \Longrightarrow_l a{<}\nu x{>} \prec P'$
    **and** *yineqa*: $y \neq a$
    **and** *yineqx*: $y \neq x$
    **and** *xFreshP*: $x \sharp P$

  **from** *PTrans xFreshP* **obtain** $P''$ $P'''$ **where** *PChain*: $P \Longrightarrow_\tau P''$
    **and** $P''$*Trans*: $P'' \longmapsto a{<}\nu x{>} \prec P'''$
    **and** $P'''$*Chain*: $P''' \Longrightarrow_\tau P'$
   **by**(*blast dest*: *transitionE*)

  **from** *PChain* **have** ${<}\nu y{>}P \Longrightarrow_\tau {<}\nu y{>}P''$ **by**(*rule ResChain*)
  **moreover from** $P''$*Trans yineqa yineqx* **have** ${<}\nu y{>}P'' \longmapsto a{<}\nu x{>} \prec ({<}\nu y{>}P''')$
   **by**(*force intro*: *ResB*)
  **moreover from** $P'''$*Chain* **have** ${<}\nu y{>}P''' \Longrightarrow_\tau {<}\nu y{>}P'$ **by**(*rule ResChain*)
  **ultimately show** ${<}\nu y{>}P \Longrightarrow_l a{<}\nu x{>} \prec {<}\nu y{>}P'$ **by**(*rule transitionI*)
**next**
  **assume** *PTrans*: $P \Longrightarrow_l u\ in\ P''{\rightarrow}a{<}x{>} \prec P'$
    **and** *yineqa*: $y \neq a$
    **and** *yineqx*: $y \neq x$
    **and** *yinequ*: $y \neq u$

  **from** *PTrans* **obtain** $P'''$ **where** *PChain*: $P \Longrightarrow_\tau P'''$
    **and** $P'''$*Trans*: $P''' \longmapsto a{<}x{>} \prec P''$
    **and** $P''$*Chain*: $P''[x{::=}u] \Longrightarrow_\tau P'$
   **by**(*blast dest*: *transitionE*)

  **from** *PChain* **have** ${<}\nu y{>}P \Longrightarrow_\tau {<}\nu y{>}P'''$ **by**(*rule ResChain*)
  **moreover from** $P'''$*Trans yineqa yineqx* **have** ${<}\nu y{>}P''' \longmapsto a{<}x{>} \prec ({<}\nu y{>}P'')$
   **by**(*force intro*: *ResB*)
  **moreover have** $({<}\nu y{>}P'')[x{::=}u] \Longrightarrow_\tau {<}\nu y{>}P'$
  **proof** −
   **from** $P''$*Chain* **have** ${<}\nu y{>}(P''[x{::=}u]) \Longrightarrow_\tau {<}\nu y{>}P'$ **by**(*rule ResChain*)
   **with** *yineqx yinequ* **show** *?thesis* **by**(*simp add*: *eqvt-subs*[*THEN sym*])
  **qed**
  **ultimately show** ${<}\nu y{>}P \Longrightarrow_l u\ in\ ({<}\nu y{>}P''){\rightarrow}a{<}x{>} \prec {<}\nu y{>}P'$ **by**(*rule transitionI*)
**qed**

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

**shows** $P \parallel !P \Longrightarrow_l Rs \Longrightarrow !P \Longrightarrow_l Rs$
**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** −
**assume** $P \parallel !P \Longrightarrow_l Rs$
**thus** $!P \Longrightarrow_l Rs$
**proof**(*nominal-induct avoiding*: $P$ *rule*: *residual.strong-inducts*)
**case**(*BoundR a x P′ P*)
**assume** *xFreshP*: $x \sharp P$
**assume** *PTrans*: $P \parallel !P \Longrightarrow_l a«x» \prec P'$
**from** *PTrans* **obtain** $a'$ **where** *aeq*: $a = BoundOutputS\ a'$ **by**(*cases a*, *auto*)

**with** *PTrans xFreshP* **obtain** $P''$ $P'''$ **where** *PChain*: $P \parallel !P \Longrightarrow_\tau P''$
**and** *P″Trans*: $P'' \longmapsto a'{<}\nu x{>} \prec P'''$
**and** *P‴Chain*: $P''' \Longrightarrow_\tau P'$
**by**(*force dest*: *transitionE*)

**show** $!P \Longrightarrow_l a«x» \prec P'$
**proof**(*cases* $P'' = P \parallel !P$)
**assume** $P'' = P \parallel !P$
**moreover with** *P″Trans* **have** $!P \longmapsto a'{<}\nu x{>} \prec P'''$ **by**(*blast intro*: *transitions.Bang*)
**ultimately show** *?thesis* **using** *PChain P‴Chain aeq* **by**(*simp*, *rule-tac transitionI*, *auto*)
**next**
**assume** $P'' \neq P \parallel !P$
**with** *PChain* **have** $!P \Longrightarrow_\tau P''$ **by**(*rule bangChain*)
**with** *P″Trans P‴Chain aeq* **show** *?thesis* **by**(*blast intro*: *transitionI*)
**qed**
**next**
**fix** $\alpha$ $P'$ $P$
**assume** $P \parallel !P \Longrightarrow_l \alpha \prec P'$

**then obtain** $P''$ $P'''$ **where** *PChain*: $P \parallel !P \Longrightarrow_\tau P''$
**and** *P″Trans*: $P'' \longmapsto \alpha \prec P'''$
**and** *P‴Chain*: $P''' \Longrightarrow_\tau P'$
**by**(*force dest*: *transitionE*)

**show** $!P \Longrightarrow_l \alpha \prec P'$
**proof**(*cases* $P'' = P \parallel !P$)
**assume** $P'' = P \parallel !P$
**moreover with** *P″Trans* **have** $!P \longmapsto \alpha \prec P'''$ **by**(*blast intro*: *transitions.Bang*)
**ultimately show** *?thesis* **using** *PChain P‴Chain* **by**(*rule-tac transitionI*, *auto*)
**next**
**assume** $P'' \neq P \parallel !P$
**with** *PChain* **have** $!P \Longrightarrow_\tau P''$ **by**(*rule bangChain*)

**with** *P''Trans P'''Chain* **show** *?thesis* **by**(*blast intro*: *transitionI*)
**qed**
**qed**
**next**
**assume** $P \parallel !P \Longrightarrow_l u\ in\ P''{\rightarrow}a{<}x{>} \prec P'$

**then obtain** $P'''$ **where** *PChain*: $P \parallel !P \Longrightarrow_\tau P'''$
**and** *P'''Trans*: $P''' \longmapsto a{<}x{>} \prec P''$
**and** *P''Chain*: $P''[x{::=}u] \Longrightarrow_\tau P'$
**by**(*force dest*: *transitionE*)

**show** $!P \Longrightarrow_l u\ in\ P''{\rightarrow}a{<}x{>} \prec P'$
**proof**(*cases* $P''' = P \parallel !P$)
**assume** $P''' = P \parallel !P$
**moreover with** *P'''Trans* **have** $!P \longmapsto a{<}x{>} \prec P''$ **by**(*blast intro*: *transitions.Bang*)
**ultimately show** *?thesis* **using** *PChain P''Chain* **by**(*rule-tac transitionI*, *auto*)
**next**
**assume** $P''' \neq P \parallel !P$
**with** *PChain* **have** $!P \Longrightarrow_\tau P'''$ **by**(*rule bangChain*)
**with** *P'''Trans P''Chain* **show** *?thesis* **by**(*blast intro*: *transitionI*)
**qed**
**qed**

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

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

**shows** $P \Longrightarrow_\tau P'$
**using** *assms*
**by**(*auto simp add*: *transition-def residualInject*)

**lemma** *chainTransitionAppend*:
**fixes** $P$ :: *pi*
**and** $P'$ :: *pi*
**and** $Rs$ :: *residual*
**and** $a$ :: *name*
**and** $x$ :: *name*
**and** $P''$ :: *pi*
**and** $u$ :: *name*
**and** $P'''$ :: *pi*
**and** $\alpha$ :: *freeRes*

**shows** $P \Longrightarrow_\tau P' \Longrightarrow P' \Longrightarrow_l Rs \Longrightarrow P \Longrightarrow_l Rs$
**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** −
**assume** $P \Longrightarrow_\tau P'$ **and** $P' \Longrightarrow_l Rs$
**thus** $P \Longrightarrow_l Rs$
**by**(*auto simp add*: *transition-def residualInject*) (*blast dest*: *rtrancl-trans*)+
**next**
**assume** $P \Longrightarrow_\tau P''$ **and** $P'' \Longrightarrow_l u\ in\ P'''{\rightarrow}a{<}x{>} \prec P'$
**thus** $P \Longrightarrow_l u\ in\ P'''{\rightarrow}a{<}x{>} \prec P'$
**apply**(*auto simp add*: *inputTransition-def residualInject*)
**by**(*blast dest*: *rtrancl-trans*)+
**next**
**assume** *PTrans*: $P \Longrightarrow_l a{<}\nu x{>} \prec P''$
**assume** $P''$*Chain*: $P'' \Longrightarrow_\tau P'$
**assume** *xFreshP*: $x \sharp P$

**from** *PTrans xFreshP* **obtain** $P'''\ P''''$ **where** *PChain*: $P \Longrightarrow_\tau P'''$
**and** $P'''$*Trans*: $P''' \longmapsto a{<}\nu x{>} \prec P''''$
**and** $P''''$*Chain*: $P'''' \Longrightarrow_\tau P''$
**by**(*blast dest*: *transitionE*)

**from** $P''''$*Chain* $P''$*Chain* **have** $P'''' \Longrightarrow_\tau P'$ **by** *auto*
**with** *PChain* $P'''$*Trans* **show** $P \Longrightarrow_l a{<}\nu x{>} \prec P'$ **by**(*rule transitionI*)
**next**
**assume** *PTrans*: $P \Longrightarrow_l u\ in\ P'''{\rightarrow}a{<}x{>} \prec P''$
**assume** $P''$*Chain*: $P'' \Longrightarrow_\tau P'$

**from** *PTrans* **obtain** $P''''$ **where** *PChain*: $P \Longrightarrow_\tau P''''$
**and** $P''''$*Trans*: $P'''' \longmapsto a{<}x{>} \prec P'''$
**and** $P'''$*Chain*: $P'''[x::=u] \Longrightarrow_\tau P''$
**by**(*blast dest*: *transitionE*)

**from** $P'''$*Chain* $P''$*Chain* **have** $P'''[x::=u] \Longrightarrow_\tau P'$ **by** *auto*
**with** *PChain* $P''''$*Trans* **show** $P \Longrightarrow_l u\ in\ P'''{\rightarrow}a{<}x{>} \prec P'$ **by**(*blast intro*: *transitionI*)
**next**
**assume** *PTrans*: $P \Longrightarrow_l \alpha \prec P''$
**assume** $P''$*Chain*: $P'' \Longrightarrow_\tau P'$

**from** *PTrans* **obtain** $P'''\ P''''$ **where** *PChain*: $P \Longrightarrow_\tau P'''$
**and** $P'''$*Trans*: $P''' \longmapsto \alpha \prec P''''$
**and** $P''''$*Chain*: $P'''' \Longrightarrow_\tau P''$
**by**(*blast dest*: *transitionE*)

**from** $P''''$*Chain* $P''$*Chain* **have** $P'''' \Longrightarrow_\tau P'$ **by** *auto*
**with** *PChain* $P'''$*Trans* **show** $P \Longrightarrow_l \alpha \prec P'$ **by**(*rule transitionI*)
**qed**

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

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

  **shows** $c \sharp P'$
**proof** −

  **from** *PTrans* **obtain** $P'''$ **where** *PChain*: $P \Longrightarrow_{\tau} P'''$
    **and** $P'''$*Trans*: $P''' \longmapsto a<x> \prec P''$
    **and** $P''$*Chain*: $P''[x::=u] \Longrightarrow_{\tau} P'$
    **by**(*blast dest*: *transitionE*)

  **from** *PChain cFreshP* **have** *cFreshP*$'''$: $c \sharp P'''$ **by**(*rule freshChain*)
  **show** $c \sharp P'$
  **proof**(*cases* $x=c$)
    **assume** *xeqc*: $x=c$
    **from** *cinequ* **have** $c \sharp P''[c::=u]$ **apply** − **by**(*rule fresh-fact2*)
    **with** $P''$*Chain xeqc* **show** *?thesis* **by**(*force intro*: *freshChain*)
  **next**
    **assume** *xineqc*: $x \neq c$
    **with** $P'''$*Trans cFreshP*$'''$ **have** $c \sharp P''$ **by**(*blast dest*: *freshBoundDerivative*)
    **with** *cinequ* **have** $c \sharp P''[x::=u]$
      **apply** −
      **apply**(*rule fresh-fact1*)
      **by** *simp*
    **with** $P''$*Chain* **show** *?thesis* **by**(*rule freshChain*)
  **qed**
**qed**

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

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

**shows** $c \sharp P'$
**proof** −
**have** *Goal*: $\bigwedge P\ a\ x\ P'\ c.\ [\![P \Longrightarrow_l a{<}\nu x{>} \prec P';\ x \sharp P;\ c \sharp P;\ c \neq x]\!] \Longrightarrow c \sharp P'$
**proof** −
**fix** $P\ a\ x\ P'\ c$
**assume** *PTrans*: $P \Longrightarrow_l a{<}\nu x{>} \prec P'$
**assume** *xFreshP*: $x \sharp P$
**assume** *cFreshP*: $(c::name) \sharp P$
**assume** *cineqx*: $c \neq x$

**from** *PTrans xFreshP* **obtain** $P''\ P'''$ **where** *PTrans*: $P \Longrightarrow_\tau P''$
**and** $P''$*Trans*: $P'' \longmapsto a{<}\nu x{>} \prec P'''$
**and** $P'''$*Trans*: $P''' \Longrightarrow_\tau P'$
**by**(*blast dest*: *transitionE*)

**from** *PTrans cFreshP* **have** $c \sharp P''$ **by**(*rule freshChain*)
**with** $P''$*Trans cineqx* **have** $c \sharp P'''$ **by**(*blast dest*: *Late-Semantics.freshBoundDerivative*)
**with** $P'''$*Trans* **show** $c \sharp P'$ **by**(*rule freshChain*)
**qed**

**have** $\exists\, d::name.\ d \sharp (P,\ P',\ c)$ **by**(*blast intro*: *name-exists-fresh*)
**then obtain** $d::name$ **where** *dFreshP*: $d \sharp P$ **and** *dFreshP'*: $d \sharp P'$ **and** *cineqd*: $c \neq d$
**by**(*force simp add*: *fresh-prod*)

**from** *PTrans dFreshP'* **have** $P \Longrightarrow_l a{<}\nu d{>} \prec ([(x,\ d)] \cdot P')$ **by**(*simp add*: *alphaBoundResidual*)
**hence** $c \sharp [(x,\ d)] \cdot P'$ **using** *dFreshP cFreshP cineqd* **by**(*rule Goal*)
**with** *cineqd cineqx* **show** *?thesis* **by**(*simp add*: *name-fresh-left name-calc*)
**qed**

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

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

**shows** $c \sharp P'$
**proof** −
**from** *PTrans* **have** $P \Longrightarrow_\tau P'$ **by**(*rule tauTransitionChain*)
**thus** *?thesis* **using** *cFreshP* **by**(*rule freshChain*)
**qed**

**lemma** *freshOutputTransition*:
**fixes** $P :: pi$
**and** $a :: name$
**and** $b :: name$
**and** $P' :: pi$

**and** $c$ :: *name*

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

**shows** $c \sharp P'$
**proof** −
**from** *PTrans* **obtain** $P''$ $P'''$ **where** *PTrans*: $P \Longrightarrow_\tau P''$
**and** $P''$*Trans*: $P'' \longmapsto a[b] \prec P'''$
**and** $P'''$*Trans*: $P''' \Longrightarrow_\tau P'$
**by**(*blast dest*: *transitionE*)

**from** *PTrans cFreshP* **have** $c \sharp P''$ **by**(*rule freshChain*)
**with** $P''$*Trans* **have** $c \sharp P'''$ **by**(*blast dest*: *Late-Semantics.freshFreeDerivative*)
**with** $P'''$*Trans* **show** *?thesis* **by**(*rule freshChain*)
**qed**

**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** $P \Longrightarrow_l u \text{ in } P'' \rightarrow a{<}x{>} \prec P' \Longrightarrow (perm \cdot P) \Longrightarrow_l (perm \cdot u) \text{ in } (perm \cdot P'') \rightarrow (perm \cdot a){<}(perm \cdot x){>} \prec (perm \cdot P')$
**proof** −
**assume** $P \Longrightarrow_l Rs$
**thus** $(perm \cdot P) \Longrightarrow_l (perm \cdot Rs)$
**proof**(*nominal-induct Rs avoiding*: *P rule*: *residual.strong-inducts*)
**case**(*BoundR* $a$ $x$ $P'$ $P$)
**have** *PTrans*: $P \Longrightarrow_l a«x» \prec P'$ **by** *fact*
**moreover then obtain** $b$ **where** *aeqb*: $a = BoundOutputS\ b$ **by**(*cases a*, *auto*)
**moreover have** $x \sharp P$ **by** *fact*
**ultimately obtain** $P''$ $P'''$ **where** *PTrans*: $P \Longrightarrow_\tau P''$
**and** $P''$*Trans*: $P'' \longmapsto b{<}\nu x{>} \prec P'''$
**and** $P'''$*Trans*: $P''' \Longrightarrow_\tau P'$
**by**(*blast dest*: *transitionE*)

**from** *PTrans* **have** $(perm \cdot P) \Longrightarrow_\tau (perm \cdot P'')$ **by**(*rule eqvtChainI*)
**moreover from** $P''$*Trans* **have** $(perm \cdot P'') \longmapsto (perm \cdot (b{<}\nu x{>} \prec P'''))$
**by**(*rule eqvts*)
**moreover from** $P'''$*Trans* **have** $(perm \cdot P''') \Longrightarrow_\tau (perm \cdot P')$ **by**(*rule eqvtChainI*)
**ultimately show** *?case* **using** *aeqb* **by**(*force intro*: *transitionI*)

**next**
  **case**(*FreeR* $\alpha$ $P'$ $P$)
  **have** $P \Longrightarrow_l \alpha \prec P'$ **by** *fact*
  **then obtain** $P''$ $P'''$ **where** *PTrans*: $P \Longrightarrow_\tau P''$
    **and** $P''$*Trans*: $P'' \longmapsto \alpha \prec P'''$
    **and** $P'''$*Trans*: $P''' \Longrightarrow_\tau P'$
   **by**(*blast dest*: *transitionE*)

  **from** *PTrans* **have** $(perm \cdot P) \Longrightarrow_\tau (perm \cdot P'')$ **by**(*rule eqvtChainI*)
  **moreover from** $P''$*Trans* **have** $(perm \cdot P'') \longmapsto (perm \cdot (\alpha \prec P'''))$
   **by**(*rule eqvts*)
   **moreover from** $P'''$*Trans* **have** $(perm \cdot P''') \Longrightarrow_\tau (perm \cdot P')$ **by**(*rule eqvtChainI*)
  **ultimately show** *?case* **by**(*force intro*: *transitionI*)
 **qed**
**next**
 **assume** $P \Longrightarrow_l u$ *in* $P'' \rightarrow a{<}x{>} \prec P'$

 **then obtain** $P'''$ **where** *PChain*: $P \Longrightarrow_\tau P'''$
   **and** $P'''$*Trans*: $P''' \longmapsto a{<}x{>} \prec P''$
   **and** $P''$*Chain*: $P''[x{::=}u] \Longrightarrow_\tau P'$
   **by**(*blast dest*: *transitionE*)

  **from** *PChain* **have** $(perm \cdot P) \Longrightarrow_\tau (perm \cdot P''')$ **by**(*rule eqvtChainI*)
  **moreover from** $P'''$*Trans* **have** $(perm \cdot P''') \longmapsto (perm \cdot (a{<}x{>} \prec P''))$
   **by**(*rule eqvts*)
  **moreover from** $P''$*Chain* **have** $(perm \cdot P''[x{::=}u]) \Longrightarrow_\tau (perm \cdot P')$ **by**(*rule eqvtChainI*)
  **ultimately show** $(perm \cdot P) \Longrightarrow_l (perm \cdot u)$ *in* $(perm \cdot P'') \rightarrow (perm \cdot a){<}(perm \cdot x){>} \prec (perm \cdot P')$
   **by**(*force intro*: *transitionI simp add*: *eqvt-subs*[*THEN sym*] *perm-bij*)
**qed**

**lemmas** *freshTransition* = *freshBoundOutputTransition freshOutputTransition*
    *freshInputTransition freshTauTransition*

**end**

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

**definition** *weakTransition* :: (*pi* $\times$ *residual*) *set*
 **where** *weakTransition* $\equiv$ *Weak-Late-Step-Semantics.transition* $\cup$ $\{x.\ \exists P.\ x = (P, \tau \prec P)\}$

**abbreviation** *weakLateTransition-judge* :: *pi* $\Rightarrow$ *residual* $\Rightarrow$ *bool* (‹- $\Longrightarrow_l\hat{}$ -› [*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** −
**assume** $P \Longrightarrow_l Rs$
**thus** $P \Longrightarrow_l\hat{}\ Rs$ **by**(*simp add*: *weakTransition-def*)
**next**
**show** $P \Longrightarrow_l\hat{}\ \tau \prec P$ **by**(*simp add*: *weakTransition-def*)
**qed**

**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** $F\ Rs$
**using** *assms*
**by**(*auto simp add*: *weakTransition-def*)

**lemma** *singleActionChain*:
**fixes** $P$ :: *pi*
**and** $\alpha$ :: *freeRes*
**and** $P'$ :: *pi*

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

**shows** $P \Longrightarrow_l\hat{}\ (\alpha \prec P')$
**using** *assms*
**by**(*auto intro*: *Weak-Late-Step-Semantics.singleActionChain*
*simp add*: *weakTransition-def*)

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

**shows** $\tau.(P) \Longrightarrow_l\hat{}\ \tau \prec P$
**by**(*auto intro*: *Weak-Late-Step-Semantics.Tau*
*simp add*: *weakTransition-def*)

**lemma** *Output*:

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

**shows** $a\{b\}.P \Longrightarrow_l\hat{}\ a[b] \prec P$
**by**(*auto intro*: *Weak-Late-Step-Semantics.Output*
*simp add*: *weakTransition-def*)

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

**shows** $P \Longrightarrow_l\hat{}\ b{<}\nu x{>} \prec P' \Longrightarrow [a{\frown}a]P \Longrightarrow_l\hat{}\ b{<}\nu x{>} \prec P'$
**and** $P \Longrightarrow_l\hat{}\ \alpha \prec P' \Longrightarrow P \neq P' \Longrightarrow [a{\frown}a]P \Longrightarrow_l\hat{}\ \alpha \prec P'$
**by**(*auto simp add*: *residual.inject weakTransition-def intro*: *Weak-Late-Step-Semantics.Match*)

**lemma** *Mismatch*:
**fixes** $a :: name$
**and** $c :: name$
**and** $P :: pi$
**and** $b :: name$
**and** $x :: name$
**and** $P' :: pi$
**and** $\alpha :: freeRes$

**shows** $[\![P \Longrightarrow_l\hat{}\ b{<}\nu x{>} \prec P';\ a \neq c]\!] \Longrightarrow [a{\neq}c]P \Longrightarrow_l\hat{}\ b{<}\nu x{>} \prec P'$
**and** $P \Longrightarrow_l\hat{}\ \alpha \prec P' \Longrightarrow P \neq P' \Longrightarrow a \neq c \Longrightarrow [a{\neq}c]P \Longrightarrow_l\hat{}\ \alpha \prec P'$
**by**(*auto simp add*: *residual.inject weakTransition-def intro*: *Weak-Late-Step-Semantics.Mismatch*)

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

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

**shows** ${<}\nu b{>}P \Longrightarrow_l\hat{}\ a{<}\nu b{>} \prec P'$
**using** *assms*
**by**(*auto intro*: *Weak-Late-Step-Semantics.Open*
*simp add*: *weakTransition-def residual.inject*)

**lemma** *Par1B*:
**fixes** $P :: pi$

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

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

**shows** $P \parallel Q \Longrightarrow_l\hat{} a{<}\nu x{>} \prec (P' \parallel Q)$
**using** *assms*
**by**(*auto intro*: *Weak-Late-Step-Semantics.Par1B*
*simp add*: *weakTransition-def residual.inject*)

**lemma** *Par1F*:
**fixes** $P$ :: *pi*
**and** $\alpha$ :: *freeRes*
**and** $P'$ :: *pi*

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

**shows** $P \parallel Q \Longrightarrow_l\hat{} \alpha \prec (P' \parallel Q)$
**using** *assms*
**by**(*auto intro*: *Weak-Late-Step-Semantics.Par1F*
*simp add*: *weakTransition-def residual.inject*)

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

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

**shows** $P \parallel Q \Longrightarrow_l\hat{} a{<}\nu x{>} \prec (P \parallel Q')$
**using** *assms*
**by**(*auto intro*: *Weak-Late-Step-Semantics.Par2B*
*simp add*: *weakTransition-def residual.inject*)

**lemma** *Par2F*:
**fixes** $Q$ :: *pi*
**and** $\alpha$ :: *freeRes*
**and** $Q'$ :: *pi*

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

**shows** $P \parallel Q \Longrightarrow_l\hat{} \alpha \prec (P \parallel Q')$
**using** *assms*
**by**(*auto intro*: *Weak-Late-Step-Semantics.Par2F*
*simp add*: *weakTransition-def residual.inject*)

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

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

**shows** $P \parallel Q \Longrightarrow_l\hat{}\ \tau \prec P' \parallel Q'$
**using** *assms*
**by**(*auto intro*: *Weak-Late-Step-Semantics.Comm1*
*simp add*: *weakTransition-def residual.inject*)

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

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

**shows** $P \parallel Q \Longrightarrow_l\hat{}\ \tau \prec P' \parallel Q'$
**using** *assms*
**by**(*auto intro*: *Weak-Late-Step-Semantics.Comm2*
*simp add*: *weakTransition-def residual.inject*)

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

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

**shows** $P \parallel Q \Longrightarrow_l\hat{}\ \tau \prec <\nu y>(P' \parallel Q')$
**using** *assms*
**by**(*auto intro*: *Weak-Late-Step-Semantics.Close1*
*simp add*: *weakTransition-def residual.inject*)

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

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

**shows** $P \parallel Q \Longrightarrow_l\hat{}\ \tau \prec <\nu y>(P' \parallel Q')$
**using** *assms*
**by**(*auto intro*: *Weak-Late-Step-Semantics.Close2*
*simp add*: *weakTransition-def residual.inject*)

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

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

**shows** $<\nu x>P \Longrightarrow_l\hat{}\ \alpha \prec <\nu x>P'$
**using** *assms*
**by**(*auto intro*: *Weak-Late-Step-Semantics.ResF*
*simp add*: *weakTransition-def residual.inject*)

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

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

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

**shows** $<\nu y>P \Longrightarrow_l\hat{}\ a<\nu x> \prec (<\nu y>P')$
**using** *assms*
**by**(*auto intro*: *Weak-Late-Step-Semantics.ResB*
*simp add*: *weakTransition-def residual.inject*)

**lemma** *Bang*:
**fixes** $P$ :: *pi*
**and** $Rs$ :: *residual*

**assumes** $P \parallel !P \Longrightarrow_l\hat{}\ Rs$
**and** $Rs \neq \tau \prec P \parallel !P$

**shows** $!P \Longrightarrow_l\hat{}\ Rs$
**using** *assms*
**by**(*auto intro*: *Weak-Late-Step-Semantics.Bang*
*simp add*: *weakTransition-def residual.inject*)

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

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

**shows** $P \Longrightarrow_\tau P'$
**using** *assms*
**by**(*auto intro*: *Weak-Late-Step-Semantics.tauTransitionChain*
*simp add*: *weakTransition-def residual.inject transition-def*)

**lemma** *chainTransitionAppend*:
**fixes** $P$ :: *pi*
**and** $P'$ :: *pi*
**and** $Rs$ :: *residual*
**and** $a$ :: *name*
**and** $x$ :: *name*
**and** $P''$ :: *pi*
**and** $\alpha$ :: *freeRes*

**shows** $P \Longrightarrow_\tau P' \Longrightarrow P' \Longrightarrow_l\hat{}\ Rs \Longrightarrow P \Longrightarrow_l\hat{}\ Rs$
**and** $P \Longrightarrow_l\hat{}\ a<\nu x> \prec P'' \Longrightarrow P'' \Longrightarrow_\tau P' \Longrightarrow x \sharp P \Longrightarrow P \Longrightarrow_l\hat{}\ a<\nu x> \prec P'$
**and** $P \Longrightarrow_l\hat{}\ \alpha \prec P'' \Longrightarrow P'' \Longrightarrow_\tau P' \Longrightarrow P \Longrightarrow_l\hat{}\ \alpha \prec P'$
**proof** –
**assume** $P \Longrightarrow_\tau P'$ **and** $P' \Longrightarrow_l\hat{}\ Rs$
**thus** $P \Longrightarrow_l\hat{}\ Rs$
**by**(*auto intro*: *Weak-Late-Step-Semantics.chainTransitionAppend*
*Weak-Late-Step-Semantics.tauActionChain*

*simp add*: *weakTransition-def residual.inject*)
**next**
**assume** $P \Longrightarrow_l \hat{} a{<}\nu x{>} \prec P''$ **and** $P'' \Longrightarrow_\tau P'$ **and** $x \sharp P$
**thus** $P \Longrightarrow_l \hat{} a{<}\nu x{>} \prec P'$
**by**(*auto intro*: *Weak-Late-Step-Semantics.chainTransitionAppend*
*simp add*: *weakTransition-def residual.inject*)
**next**
**assume** $P \Longrightarrow_l \hat{} \alpha \prec P''$ **and** $P'' \Longrightarrow_\tau P'$
**thus** $P \Longrightarrow_l \hat{} \alpha \prec P'$
**apply**(*case-tac* $P''=P'$)
**by**(*auto dest*: *Weak-Late-Step-Semantics.chainTransitionAppend*
*Weak-Late-Step-Semantics.tauActionChain*
*simp add*: *weakTransition-def residual.inject*)
**qed**

**lemma** *weakEqWeakTransitionAppend*:
**fixes** $P$ :: *pi*
**and** $P'$ :: *pi*
**and** $\alpha$ :: *freeRes*
**and** $P''$ :: *pi*

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

**shows** $P \Longrightarrow_l \alpha \prec P''$
**proof**(*cases* $\alpha=\tau$)
**assume** *alphaEqTau*: $\alpha = \tau$
**with** *P'Trans* **have** $P' \Longrightarrow_\tau P''$ **by**(*blast intro*: *tauTransitionChain*)
**with** *PTrans alphaEqTau* **show** *?thesis*
**by**(*blast intro*: *Weak-Late-Step-Semantics.chainTransitionAppend*)
**next**
**assume** *alphaIneqTau*: $\alpha \neq \tau$
**from** *PTrans* **have** $P \Longrightarrow_\tau P'$ **by**(*rule Weak-Late-Step-Semantics.tauTransitionChain*)
**moreover from** *P'Trans alphaIneqTau* **have** $P' \Longrightarrow_l \alpha \prec P''$
**by**(*auto simp add*: *weakTransition-def residual.inject*)
**ultimately show** *?thesis*
**by**(*rule Weak-Late-Step-Semantics.chainTransitionAppend*)
**qed**

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

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

**shows** $c \sharp P'$
**using** *assms*
**by**(*auto intro*: *Weak-Late-Step-Semantics.freshBoundOutputTransition*
*simp add*: *weakTransition-def residual.inject*)

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

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

**shows** $c \sharp P'$
**using** *assms*
**by**(*auto intro*: *Weak-Late-Step-Semantics.freshTauTransition*
*simp add*: *weakTransition-def residual.inject*)

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

**assumes** *PTrans*: $P \Longrightarrow_l\hat{}a[b] \prec P'$
**and** *cFreshP*: $c \sharp P$

**shows** $c \sharp P'$
**using** *assms*
**by**(*auto intro*: *Weak-Late-Step-Semantics.freshOutputTransition*
*simp add*: *weakTransition-def residual.inject*)

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

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

**shows** $(perm \cdot P) \Longrightarrow_l\hat{}\ (perm \cdot Rs)$
**using** *assms*
**by**(*auto intro*: *Weak-Late-Step-Semantics.eqvtI*
*simp add*: *weakTransition-def residual.inject*)

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

**and** $P'$ :: *pi*
**and** $c$ :: *name*

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

**shows** $c \sharp P'$
**using** *assms*
**by**(*auto intro*: *Weak-Late-Step-Semantics.freshInputTransition*
*simp add*: *weakTransition-def residual.inject*)

**lemmas** *freshTransition* = *freshBoundOutputTransition freshOutputTransition freshInputTransition freshTauTransition*

**end**

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

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

**definition** *weakSimAux* :: $pi \Rightarrow (pi \times pi)\ set \Rightarrow pi \Rightarrow bool$ **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* :: $pi \Rightarrow (pi \times pi)\ set \Rightarrow pi \Rightarrow bool$ (‹- $\leadsto\hat{}$<->  -› [*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*
**by**(*auto simp add*: *weakSimAux-def simDef*)

**lemma** *monotonic*:
**fixes** $A$ :: $(pi \times pi)\ set$

**and** $B$ :: $(pi \times pi)$ *set*
**and** $P$ :: *pi*
**and** $P'$ :: *pi*

**assumes** $P \leadsto\hat{}<A> P'$
**and** $A \subseteq B$

**shows** $P \leadsto\hat{}<B> P'$
**using** *assms*
**apply**(*auto simp add*: *simDef*)
**apply** *blast*
**apply**(*erule-tac* $x{=}a{<}x{>} \prec Q'$ **in** *allE*)
**apply**(*clarsimp*)
**apply**(*rotate-tac 4*)
**apply**(*erule-tac* $x{=}Q'$ **in** *allE*)
**apply**(*erule-tac* $x{=}a$ **in** *allE*)
**apply**(*erule-tac* $x{=}x$ **in** *allE*)
**by** *blast+*

**lemma** *simCasesCont*[*consumes 1*, *case-names Bound Input Free*]:
**fixes** $P$ :: *pi*
**and** $Q$ :: *pi*
**and** *Rel* :: $(pi \times pi)$ *set*
**and** $C$ :: *'a::fs-name*

**assumes** *Eqvt*: *eqvt Rel*
**and** *Bound*: $\bigwedge Q'\ a\ x.\ [\![x \sharp C;\ Q \longmapsto a{<}\nu x{>} \prec Q']\!] \Longrightarrow \exists P'.\ P \Longrightarrow_l\hat{}\ a{<}\nu x{>} \prec P' \wedge (P',\ Q') \in Rel$
**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** $P \leadsto\hat{}<Rel> Q$
**using** *Free*
**proof**(*auto simp add*: *simDef*)
**fix** $Q'\ a\ x$
**assume** *xFreshP*: $(x{::}name) \sharp P$
**assume** *Trans*: $Q \longmapsto a{<}\nu x{>} \prec Q'$
**have** $\exists c{::}name.\ c \sharp (P,\ Q',\ x,\ C)$ **by**(*blast intro*: *name-exists-fresh*)
**then obtain** *c::name* **where** *cFreshP*: $c \sharp P$ **and** *cFreshQ'*: $c \sharp Q'$ **and** *cFreshC*: $c \sharp C$
**and** *cineqx*: $c \neq x$
**by**(*force simp add*: *fresh-prod*)

**from** *Trans cFreshQ'* **have** $Q \longmapsto a{<}\nu c{>} \prec ([(x,\ c)] \cdot Q')$ **by**(*simp add*: *alphaBoundResidual*)
**with** *cFreshC* **have** $\exists P'.\ P \Longrightarrow_l\hat{}\ a{<}\nu c{>} \prec P' \wedge (P',\ [(x,\ c)] \cdot Q') \in Rel$
**by**(*rule Bound*)

**then obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l\hat{}\ a{<}\nu c{>} \prec P'$ **and** *P'RelQ'*: $(P', [(x, c)] \cdot Q') \in Rel$
  **by** *blast*

  **from** *PTrans xFreshP cineqx* **have** *xFreshP'*: $x \sharp P'$ **by**(*force dest*: *freshTransition*)
  **with** *PTrans* **have** $P \Longrightarrow_l\hat{}\ a{<}\nu x{>} \prec ([(x, c)] \cdot P')$ **by**(*simp add*: *alphaBoundResidual name-swap*)
  **moreover have** $([(x, c)] \cdot P', Q') \in Rel$ (**is** *?goal*)
  **proof** −
    **from** *Eqvt P'RelQ'* **have** $([(x, c)] \cdot P', [(x, c)] \cdot [(x, c)] \cdot Q') \in Rel$
      **by**(*rule eqvtRelI*)
    **with** *cineqx* **show** *?goal* **by**(*simp add*: *name-calc*)
  **qed**
  **ultimately show** $\exists P'.\ P \Longrightarrow_l\hat{}\ a{<}\nu x{>} \prec P' \wedge (P', Q') \in Rel$ **by** *blast*
**next**
  **fix** $Q'$ $a$ $x$ $u$
  **assume** *QTrans*: $Q \longmapsto a{<}x{>} \prec (Q'{::}pi)$**and** *xFreshP*: $x \sharp P$

  **have** $\exists c{::}name.\ c \sharp (P, Q', C, x)$ **by**(*blast intro*: *name-exists-fresh*)
  **then obtain** $c{::}name$ **where** *cFreshP*: $c \sharp P$ **and** *cFreshQ'*: $c \sharp Q'$ **and** *cFreshC*: $c \sharp C$
      **and** *cineqx*: $c \neq x$
    **by**(*force simp add*: *fresh-prod*)

  **from** *QTrans cFreshQ'* **have** $Q \longmapsto a{<}c{>} \prec ([(x, c)] \cdot Q')$ **by**(*simp add*: *alphaBoundResidual*)
  **with** *cFreshC* **have** $\exists P''.\ \forall u.\ \exists P'.\ P \Longrightarrow_l u\ in\ P'' {\rightarrow} a{<}c{>} \prec P' \wedge (P', ([(x, c)] \cdot Q')[c{::=}u]) \in Rel$
    **by**(*rule Input*)

  **then obtain** $P''$ **where** *L1*: $\forall u.\ \exists P'.\ P \Longrightarrow_l u\ in\ P'' {\rightarrow} a{<}c{>} \prec P' \wedge (P', ([(x, c)] \cdot Q')[c{::=}u]) \in Rel$ **by** *blast*

  **have** $\forall u.\ \exists P'.\ P \Longrightarrow_l u\ in\ ([(c, x)] \cdot P''){\rightarrow} a{<}x{>} \prec P' \wedge (P', Q'[x{::=}u]) \in Rel$
  **proof**(*auto*)
    **fix** $u$
    **from** *L1* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l u\ in\ P'' {\rightarrow} a{<}c{>} \prec P'$ **and** *P'RelQ'*: $(P', ([(x, c)] \cdot Q')[c{::=}u]) \in Rel$
      **by** *blast*

    **from** *PTrans xFreshP* **have** $P \Longrightarrow_l u\ in\ ([(c, x)] \cdot P''){\rightarrow} a{<}x{>} \prec P'$ **by**(*rule alphaInput*)
    **moreover from** *P'RelQ' cFreshQ'* **have** $(P', Q'[x{::=}u]) \in Rel$ **by**(*simp add*: *renaming*[*THEN sym*] *name-swap*)

    **ultimately show** $\exists P'.\ P \Longrightarrow_l u\ in\ ([(c, x)] \cdot P''){\rightarrow} a{<}x{>} \prec P' \wedge (P', Q'[x{::=}u]) \in Rel$ **by** *blast*
  **qed**

**thus** $\exists P''. \forall u. \exists P'. P \Longrightarrow_l u$ *in* $P'' \rightarrow a{<}x{>} \prec P' \wedge (P', Q'[x{::=}u]) \in Rel$ **by** *blast*
**qed**

**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** $P \leadsto\hat{}{<}Rel{>}\ Q$
**using** *assms*
**by**(*auto simp add*: *simDef*)

**lemma** *simActBoundCases*[*consumes 1*, *case-names Input BoundOutput*]:
**fixes** $P$ :: *pi*
**and** $a$ :: *subject*
**and** $x$ :: *name*
**and** $Q'$ :: *pi*
**and** $C$ :: *'a::fs-name*
**and** $Rel$ :: $(pi \times pi)$ *set*

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

**shows** *weakSimAct* $P$ $(a«x» \prec Q')$ $P$ *Rel*
**proof**(*simp add*: *weakSimAct-def fresh-prod*, *auto*)
**fix** $Q''\ b\ y$
**assume** *Eq*: $a«x» \prec Q' = b{<}\nu y{>} \prec Q''$
**assume** *yFreshP*: $y \sharp P$

**from** *Eq* **have** $a = BoundOutputS\ b$ **by**(*simp add*: *residual.inject*)

**from** *yFreshP DerBoundOutput*[*OF this*] *Eq* **show** $\exists P'. P \Longrightarrow_l\hat{}\ b{<}\nu y{>} \prec P' \wedge (P', Q'') \in Rel$
**proof**(*cases x=y*, *auto simp add*: *residual.inject name-abs-eq*)
**fix** $P'$
**assume** *PTrans*: $P \Longrightarrow_l\hat{}\ b{<}\nu x{>} \prec P'$

**assume** *P′RelQ′*: $(P', ([(x, y)] \cdot Q'')) \in Rel$
**assume** *xineqy*: $x \neq y$

**with** *PTrans yFreshP* **have** *yFreshP′*: $y \sharp P'$
**by**(*force intro*: *freshTransition*)

**hence** $b<\nu x> \prec P' = b<\nu y> \prec [(x, y)] \cdot P'$ **by**(*rule alphaBoundResidual*)
**moreover have** $([(x, y)] \cdot P', Q'') \in Rel$
**proof** −
**from** *EqvtRel P′RelQ′* **have** $([(x, y)] \cdot P', [(x, y)] \cdot ([(x, y)] \cdot Q''))\in Rel$
**by**(*rule eqvtRelI*)
**thus** *?thesis* **by**(*simp add*: *name-calc*)
**qed**

**ultimately show** $\exists P'.\ P \Longrightarrow_l\hat{}\ b<\nu y> \prec P' \wedge (P', Q'') \in Rel$ **using** *PTrans* **by** *auto*
**qed**
**next**
**fix** $Q''\ b\ y\ u$
**assume** *Eq*: $a«x» \prec Q' = b<y> \prec Q''$
**assume** *yFreshP*: $y \sharp P$

**from** *Eq* **have** $a = InputS\ b$ **by**(*simp add*: *residual.inject*)
**from** *DerInput*[*OF this*] **obtain** $P''$ **where** *L1*: $\forall u.\ \exists P'.\ P \Longrightarrow_l u\ in\ P''\rightarrow b<x> \prec P' \wedge$
$(P', Q'[x::=u]) \in Rel$
**by** *blast*
**have** $\forall u.\ \exists P'.\ P \Longrightarrow_l u\ in\ ([(x, y)] \cdot P'')\rightarrow b<y> \prec P' \wedge (P', Q''[y::=u]) \in Rel$
**proof**(*rule allI*)
**fix** $u$
**from** *L1 Eq* **show** $\exists P'.\ P \Longrightarrow_l u\ in\ ([(x, y)] \cdot P'')\rightarrow b<y> \prec P' \wedge (P', Q''[y::=u]) \in Rel$
**proof**(*cases x=y*, *auto simp add*: *residual.inject name-abs-eq*)
**assume** *Der*: $\forall u.\ \exists P'.\ P \Longrightarrow_l u\ in\ P''\rightarrow b<x> \prec P' \wedge (P', ([(x, y)] \cdot Q'')[x::=u]) \in Rel$
**assume** *xFreshQ′′*: $x \sharp Q''$

**from** *Der* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l u\ in\ P''\rightarrow b<x> \prec P'$
**and** *P′RelQ′*: $(P', ([(x, y)] \cdot Q'')[x::=u]) \in Rel$
**by** *force*

**from** *PTrans yFreshP* **have** $P \Longrightarrow_l u\ in\ ([(x, y)] \cdot P'')\rightarrow b<y> \prec P'$ **by**(*rule alphaInput*)
**moreover from** *xFreshQ′′ P′RelQ′* **have** $(P', Q''[y::=u]) \in Rel$
**by**(*simp add*: *renaming*)
**ultimately show** *?thesis* **by** *force*
**qed**
**qed**
**thus** $\exists P''.\ \forall u.\ \exists P'.\ P \Longrightarrow_l u\ in\ P''\rightarrow b<y> \prec P' \wedge (P', Q''[y::=u]) \in Rel$

**by** *blast*
**qed**

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

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

**shows** *weakSimAct* $P$ $(\alpha \prec Q')$ $P$ *Rel*
**using** *assms*
**by**(*simp add*: *residual.inject weakSimAct-def fresh-prod*)

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

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

**shows** $Q \longmapsto a<\nu x> \prec Q' \Longrightarrow x \sharp P \Longrightarrow \exists P'.\ P \Longrightarrow_l\hat{}\ a<\nu x> \prec P' \wedge (P', Q')$
$\in Rel$
**and** $Q \longmapsto a<x> \prec Q' \Longrightarrow x \sharp P \Longrightarrow \exists P''.\ \forall u.\ \exists P'.\ P \Longrightarrow_l u\ in\ P'' \rightarrow a<x>$
$\prec P' \wedge (P', Q'[x::=u]) \in Rel$
**and** $Q \longmapsto \alpha \prec Q' \Longrightarrow (\exists P'.\ P \Longrightarrow_l\hat{}\ \alpha \prec P' \wedge (P', Q') \in Rel)$
**using** *assms* **by**(*simp add*: *simDef*)+

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

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

**shows** $\exists P'.\ P \Longrightarrow_\tau P' \wedge (P', Q') \in Rel$
**proof** −
**from** *QChain* **show** *?thesis*
**proof**(*induct rule*: *tauChainInduct*)
**case** *id*
**have** $P \Longrightarrow_\tau P$ **by** *simp*

**with** *PRelQ* **show** *?case* **by** *blast*
**next**
**case**(*ih* $Q'$ $Q''$)
**have** *IH*: $\exists P'.\ P \Longrightarrow_\tau P' \wedge (P', Q') \in Rel$ **by** *fact*
**then obtain** $P'$ **where** *PChain*: $P \Longrightarrow_\tau P'$ **and** $P'RelQ'$: $(P', Q') \in Rel$ **by** *blast*
**from** $P'RelQ'$ **have** $P \rightsquigarrow\hat{}<Rel>\ Q'$ **by**(*rule Sim*)
**moreover have** $Q'Trans$: $Q' \longmapsto\tau \prec Q''$ **by** *fact*
**ultimately have** $\exists P''.\ P' \Longrightarrow_l\hat{}\ \tau \prec P'' \wedge (P'', Q'') \in Rel$ **by**(*rule simE*)
**then obtain** $P''$ **where** $P'Trans$: $P' \Longrightarrow_l\hat{}\ \tau \prec P''$ **and** $P''RelQ''$: $(P'', Q'') \in Rel$ **by** *blast*
**from** $P'Trans$ **have** $P' \Longrightarrow_\tau P''$ **by**(*rule tauTransitionChain*)
**with** *PChain* **have** $P \Longrightarrow_\tau P''$ **by** *auto*
**with** $P''RelQ''$ **show** *?case* **by** *blast*
**qed**
**qed**

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

**assumes** *PSimQ*: $P \rightsquigarrow\hat{}<Rel>\ Q$
**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** $Q \Longrightarrow_l\hat{}\ a<\nu x> \prec Q' \Longrightarrow x \sharp P \Longrightarrow \exists P'.\ P \Longrightarrow_l\hat{}\ a<\nu x> \prec P' \wedge (P', Q') \in Rel$
**and** $Q \Longrightarrow_l\hat{}\ \alpha \prec Q' \Longrightarrow \exists P'.\ P \Longrightarrow_l\hat{}\ \alpha \prec P' \wedge (P', Q') \in Rel$
**proof** −
**assume** *QTrans*: $Q \Longrightarrow_l\hat{}\ a<\nu x> \prec Q'$
**assume** *xFreshP*: $x \sharp P$
**have** *Goal*: $\bigwedge P\ Q\ a\ x\ Q'.\ [\![P \rightsquigarrow\hat{}<Rel>\ Q;\ Q \Longrightarrow_l\hat{}\ a<\nu x> \prec Q';\ x \sharp P;\ x \sharp Q;\ (P, Q) \in Rel]\!] \Longrightarrow$
$\exists P'.\ P \Longrightarrow_l\hat{}\ a<\nu x> \prec P' \wedge (P', Q') \in Rel$
**proof** −
**fix** $P\ Q\ a\ x\ Q'$
**assume** *PSimQ*: $P \rightsquigarrow\hat{}<Rel>\ Q$
**assume** *QTrans*: $Q \Longrightarrow_l\hat{}\ a<\nu x> \prec Q'$
**assume** *xFreshP*: $x \sharp P$
**assume** *xFreshQ*: $x \sharp Q$
**assume** *PRelQ*: $(P, Q) \in Rel$

**from** *QTrans xFreshQ* **obtain** $Q''$ $Q'''$ **where** *QChain*: $Q \Longrightarrow_\tau Q''$
**and** $Q''Trans$: $Q'' \longmapsto a<\nu x> \prec Q'''$

**and** $Q'''Chain$: $Q''' \Longrightarrow_\tau Q'$
**by**(*force dest*: *Weak-Late-Step-Semantics.transitionE simp add*: *weakTransition-def*)

**from** *QChain PRelQ Sim* **have** $\exists P''. P \Longrightarrow_\tau P'' \wedge (P'', Q'') \in Rel$
**by**(*rule weakSimTauChain*)
**then obtain** $P''$ **where** *PChain*: $P \Longrightarrow_\tau P''$ **and** $P''RelQ''$: $(P'', Q'') \in Rel$ **by** *blast*
**from** *PChain xFreshP* **have** $xFreshP''$: $x \sharp P''$ **by**(*rule freshChain*)

**from** $P''RelQ''$ **have** $P'' \leadsto\hat{}<Rel> Q''$ **by**(*rule Sim*)
**hence** $\exists P'''. P'' \Longrightarrow_l\hat{} a<\nu x> \prec P''' \wedge (P''', Q''') \in Rel$ **using** $Q''Trans$ $xFreshP''$
**by**(*rule simE*)
**then obtain** $P'''$ **where** $P''Trans$: $P'' \Longrightarrow_l\hat{} a<\nu x> \prec P'''$ **and** $P'''RelQ'''$: $(P''', Q''') \in Rel$
**by** *blast*

**from** $P'''RelQ'''$ **have** $P''' \leadsto\hat{}<Rel> Q'''$ **by**(*rule Sim*)
**have** $\exists P'. P''' \Longrightarrow_\tau P' \wedge (P', Q') \in Rel$ **using** $Q'''Chain$ $P'''RelQ'''$ *Sim*
**by**(*rule weakSimTauChain*)
**then obtain** $P'$ **where** $P'''Chain$: $P''' \Longrightarrow_\tau P'$ **and** $P'RelQ'$: $(P', Q') \in Rel$ **by** *blast*

**from** *PChain* $P''Trans$ $P'''Chain$ $xFreshP''$ **have** $P \Longrightarrow_l\hat{} a<\nu x> \prec P'$
**by**(*blast dest*: *chainTransitionAppend*)
**with** $P'RelQ'$ **show** $\exists P'. P \Longrightarrow_l\hat{} a<\nu x> \prec P' \wedge (P', Q') \in Rel$ **by** *blast*
**qed**

**have** $\exists c{::}name.\ c \sharp (Q, Q', P, x)$ **by**(*blast intro*: *name-exists-fresh*)
**then obtain** $c{::}name$ **where** *cFreshQ*: $c \sharp Q$ **and** $cFreshQ'$: $c \sharp Q'$ **and** *cFreshP*: $c \sharp P$
**and** *xineqc*: $x \neq c$
**by**(*force simp add*: *fresh-prod*)

**from** *QTrans* $cFreshQ'$ **have** $Q \Longrightarrow_l\hat{} a<\nu c> \prec ([(x, c)] \cdot Q')$ **by**(*simp add*: *alphaBoundResidual*)
**with** *PSimQ* **have** $\exists P'. P \Longrightarrow_l\hat{} a<\nu c> \prec P' \wedge (P', [(x, c)] \cdot Q') \in Rel$ **using** *cFreshP cFreshQ* ‹$(P, Q) \in Rel$›
**by**(*rule Goal*)
**then obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l\hat{} a<\nu c> \prec P'$ **and** $P'RelQ'$: $(P', [(x, c)] \cdot Q') \in Rel$
**by** *force*
**have** $P \Longrightarrow_l\hat{} a<\nu x> \prec ([(x, c)] \cdot P')$
**proof** −
**from** *PTrans xFreshP xineqc* **have** $x \sharp P'$ **by**(*rule freshTransition*)
**with** *PTrans* **show** *?thesis* **by**(*simp add*: *alphaBoundResidual name-swap*)
**qed**
**moreover have** $([(x, c)] \cdot P', Q') \in Rel$

**proof** −
**from** *Eqvt P′RelQ′* **have** $([(x, c)] \cdot P', [(x, c)] \cdot [(x, c)] \cdot Q') \in Rel$
**by**(*rule eqvtRelI*)
**thus** *?thesis* **by** *simp*
**qed**

**ultimately show** $\exists P'. \ P \Longrightarrow_l\hat{} \ a<\nu x> \prec P' \wedge (P', Q') \in Rel$ **by** *blast*
**next**
**assume** *QTrans*: $Q \Longrightarrow_l\hat{}\alpha \prec Q'$
**thus** $\exists P'. \ P \Longrightarrow_l\hat{}\alpha \prec P' \wedge (P', Q') \in Rel$
**proof**(*induct rule*: *transitionCases*)
**case** *Step*
**have** $Q \Longrightarrow_l\alpha \prec Q'$ **by** *fact*
**then obtain** $Q''$ $Q'''$ **where** *QChain*: $Q \Longrightarrow_\tau Q''$
**and** *Q′′Trans*: $Q'' \longmapsto\alpha \prec Q'''$
**and** *Q′′′Chain*: $Q''' \Longrightarrow_\tau Q'$
**by**(*blast dest*: *Weak-Late-Step-Semantics.transitionE*)

**from** *QChain PRelQ Sim* **have** $\exists P''. \ P \Longrightarrow_\tau P'' \wedge (P'', Q'') \in Rel$
**by**(*rule weakSimTauChain*)
**then obtain** $P''$ **where** *PChain*: $P \Longrightarrow_\tau P''$ **and** *P′′RelQ′′*: $(P'', Q'') \in Rel$
**by** *blast*
**from** *P′′RelQ′′* **have** $P'' \leadsto\hat{}<Rel> Q''$ **by**(*rule Sim*)
**hence** $\exists P'''. \ P'' \Longrightarrow_l\hat{}\alpha \prec P''' \wedge (P''', Q''') \in Rel$ **using** *Q′′Trans*
**by**(*rule simE*)
**then obtain** $P'''$ **where** *P′′Trans*: $P'' \Longrightarrow_l\hat{}\alpha \prec P'''$ **and** *P′′′RelQ′′′*: $(P''', Q''') \in Rel$
**by** *blast*

**from** *P′′′RelQ′′′* **have** $P''' \leadsto\hat{}<Rel> Q'''$ **by**(*rule Sim*)
**have** $\exists P'. \ P''' \Longrightarrow_\tau P' \wedge (P', Q') \in Rel$ **using** *Q′′′Chain P′′′RelQ′′′ Sim*
**by**(*rule weakSimTauChain*)
**then obtain** $P'$ **where** *P′′′Chain*: $P''' \Longrightarrow_\tau P'$ **and** *P′RelQ′*: $(P', Q') \in Rel$
**by** *blast*

**from** *PChain P′′Trans P′′′Chain* **have** $P \Longrightarrow_l\hat{}\alpha \prec P'$
**by**(*blast dest*: *chainTransitionAppend*)
**with** *P′RelQ′* **show** *?case* **by** *blast*
**next**
**case** *Stay*
**have** $\alpha \prec Q' = \tau \prec Q$ **by** *fact*
**hence** $Q = Q'$ **and** $\alpha = \tau$ **by**(*simp add*: *residual.inject*)+
**moreover have** $P \Longrightarrow_l\hat{}\tau \prec P$ **by**(*simp add*: *weakTransition-def*)
**ultimately show** *?case* **using** *PRelQ* **by** *blast*
**qed**
**qed**

**lemma** *eqvtI*:
**fixes** $P$ :: *pi*

**and** $Q$ :: *pi*
**and** $Rel$ :: $(pi \times pi)$ *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** −
**from** *EqvtRel′* **show** *?thesis*
**proof**(*induct rule*: *simCasesCont*[*of* - $(perm \cdot P)$])
**case**(*Bound* $Q'$ $a$ $x$)
**have** *Trans*: $(perm \cdot Q) \longmapsto a{<}\nu x{>} \prec Q'$ **and** *xFreshP*: $x \sharp\ perm \cdot P$ **by** *fact+*

**from** *Trans* **have** $(rev\ perm \cdot (perm \cdot Q)) \longmapsto rev\ perm \cdot (a{<}\nu x{>} \prec Q')$
**by**(*rule eqvts*)
**hence** $Q \longmapsto (rev\ perm \cdot a){<}\nu(rev\ perm \cdot x){>} \prec (rev\ perm \cdot Q')$
**by**(*simp add*: *name-rev-per*)
**moreover from** *xFreshP* **have** $(rev\ perm \cdot x) \sharp\ P$ **by**(*simp add*: *name-fresh-left*)
**ultimately have** $\exists P'.\ P \Longrightarrow_l\hat{}\ (rev\ perm \cdot a){<}\nu(rev\ perm \cdot x){>} \prec P' \wedge (P', rev\ perm \cdot Q') \in Rel$ **using** *Sim*
**by**(*force intro*: *simE*)
**then obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l\hat{}\ (rev\ perm \cdot a){<}\nu(rev\ perm \cdot x){>} \prec P'$ **and** *P′RelQ′*: $(P', rev\ perm \cdot Q') \in Rel$ **by** *blast*

**from** *PTrans* **have** $(perm \cdot P) \Longrightarrow_l\hat{}\ perm \cdot ((rev\ perm \cdot a){<}\nu(rev\ perm \cdot x){>} \prec P')$
**by**(*rule Weak-Late-Semantics.eqvtI*)
**hence** *L1*: $(perm \cdot P) \Longrightarrow_l\hat{}\ a{<}\nu x{>} \prec (perm \cdot P')$ **by**(*simp add*: *name-per-rev*)
**from** *P′RelQ′ RelRel′* **have** $(P', rev\ perm \cdot Q') \in Rel'$ **by** *blast*
**with** *EqvtRel′* **have** $(perm \cdot P', perm \cdot (rev\ perm \cdot Q')) \in Rel'$
**by**(*rule eqvtRelI*)
**hence** $(perm \cdot P', Q') \in Rel'$ **by**(*simp add*: *name-per-rev*)
**with** *L1* **show** *?case* **by** *blast*
**next**
**case**(*Input* $Q'$ $a$ $x$)
**have** *Trans*: $(perm \cdot Q) \longmapsto a{<}x{>} \prec Q'$ **and** *xFreshP*: $x \sharp\ perm \cdot P$ **by** *fact+*

**from** *Trans* **have** $(rev\ perm \cdot (perm \cdot Q)) \longmapsto rev\ perm \cdot (a{<}x{>} \prec Q')$
**by**(*rule eqvts*)
**hence** $Q \longmapsto (rev\ perm \cdot a){<}(rev\ perm \cdot x){>} \prec (rev\ perm \cdot Q')$
**by**(*simp add*: *name-rev-per*)
**moreover from** *xFreshP* **have** *xFreshP*: $(rev\ perm \cdot x) \sharp\ P$ **by**(*simp add*: *name-fresh-left*)
**ultimately have** $\exists P''.\ \forall u.\ \exists P'.\ P \Longrightarrow_l u\ in\ P''{\rightarrow}(rev\ perm \cdot a){<}(rev\ perm \cdot x){>} \prec P' \wedge (P', (rev\ perm \cdot Q')[(rev\ perm \cdot x){::=}u]) \in Rel$ **using** *Sim*
**by**(*force intro*: *simE*)
**then obtain** $P''$ **where** *L1*: $\forall u.\ \exists P'.\ P \Longrightarrow_l u\ in\ P''{\rightarrow}(rev\ perm \cdot a){<}(rev$

$perm \cdot x)\!> \prec P' \wedge (P', (rev\ perm \cdot Q')[(rev\ perm \cdot x)::=u]) \in Rel$
**by** *blast*
**have** $\forall u.\ \exists P'.\ (perm \cdot P) \Longrightarrow_l u\ in\ (perm \cdot P'')\rightarrow a\!<\!x\!> \prec P' \wedge (P', Q'[x::=u]) \in Rel'$
**proof**(*rule allI*)
**fix** $u$
**from** *L1* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l (rev\ perm \cdot u)\ in\ P''\rightarrow(rev\ perm \cdot a)\!<\!(rev\ perm \cdot x)\!> \prec P'$
**and** *P'RelQ'*: $(P', (rev\ perm \cdot Q')[(rev\ perm \cdot x)::=(rev\ perm \cdot u)]) \in Rel$ **by** *blast*
**from** *PTrans* **have** $(perm \cdot P) \Longrightarrow_l (perm \cdot (rev\ perm \cdot u))\ in\ (perm \cdot P'')\rightarrow(perm \cdot rev\ perm \cdot a)\!<\!(perm \cdot rev\ perm \cdot x)\!> \prec (perm \cdot P')$
**by**(*rule-tac Weak-Late-Step-Semantics.eqvtI*, *auto*)
**hence** *L2*: $(perm \cdot P) \Longrightarrow_l u\ in\ (perm \cdot P'')\rightarrow a\!<\!x\!> \prec (perm \cdot P')$ **by**(*simp add*: *name-per-rev*)
**from** *P'RelQ' RelRel'* **have** $(P', (rev\ perm \cdot Q')[(rev\ perm \cdot x)::=(rev\ perm \cdot u)]) \in Rel'$ **by** *blast*
**with** *EqvtRel'* **have** $(perm \cdot P', perm \cdot ((rev\ perm \cdot Q')[(rev\ perm \cdot x)::=(rev\ perm \cdot u)])) \in Rel'$
**by**(*rule eqvtRelI*)
**hence** $(perm \cdot P', Q'[x::=u]) \in Rel'$ **by**(*simp add*: *name-per-rev eqvt-subs*[*THEN sym*] *name-calc*)
**with** *L2* **show** $\exists P'.\ (perm \cdot P) \Longrightarrow_l u\ in\ (perm \cdot P'')\rightarrow a\!<\!x\!> \prec P' \wedge (P', Q'[x::=u]) \in Rel'$ **by** *blast*
**qed**

**thus** *?case* **by** *blast*
**next**
**case**(*Free* $Q'$ $\alpha$)
**have** *Trans*: $(perm \cdot Q) \longmapsto \alpha \prec Q'$ **by** *fact*

**from** *Trans* **have** $(rev\ perm \cdot (perm \cdot Q)) \longmapsto rev\ perm \cdot (\alpha \prec Q')$
**by**(*rule eqvts*)
**hence** $Q \longmapsto (rev\ perm \cdot \alpha) \prec (rev\ perm \cdot Q')$
**by**(*simp add*: *name-rev-per*)
**with** *Sim* **have** $(\exists P'.\ P \Longrightarrow_l\hat{}\ (rev\ perm \cdot \alpha) \prec P' \wedge (P', (rev\ perm \cdot Q')) \in Rel)$
**by**(*rule simE*)
**then obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l\hat{}\ (rev\ perm \cdot \alpha) \prec P'$ **and** *PRel*: $(P', (rev\ perm \cdot Q')) \in Rel$ **by** *blast*
**from** *PTrans* **have** $(perm \cdot P) \Longrightarrow_l\hat{}\ perm \cdot ((rev\ perm \cdot \alpha)\prec P')$
**by**(*rule Weak-Late-Semantics.eqvtI*)
**hence** *L1*: $(perm \cdot P) \Longrightarrow_l\hat{}\ \alpha \prec (perm \cdot P')$ **by**(*simp add*: *name-per-rev*)
**from** *PRel EqvtRel' RelRel'* **have** $((perm \cdot P'), (perm \cdot (rev\ perm \cdot Q'))) \in Rel'$
**by**(*force intro*: *eqvtRelI*)
**hence** $((perm \cdot P'), Q') \in Rel'$ **by**(*simp add*: *name-per-rev*)
**with** *L1* **show** *?case* **by** *blast*
**qed**

**qed**

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

**assumes** *Id* ⊆ *Rel*

**shows** $P \leadsto\hat{}$ <*Rel*> $P$
**using** *assms*
**by**(*auto intro*: *Weak-Late-Step-Semantics.singleActionChain*
*simp add*: *simDef weakTransition-def*)

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

**assumes** *QSimR*: $Q \leadsto\hat{}$ <*Rel′*> $R$
**and** *Eqvt*: *eqvt Rel*
**and** *Eqvt′*: *eqvt Rel″*
**and** *Trans*: *Rel O Rel′* ⊆ *Rel″*
**and** *Sim*: $\bigwedge P\ Q.\ (P, Q) \in Rel \Longrightarrow P \leadsto\hat{}$ <*Rel*> $Q$
**and** *PRelQ*: $(P, Q) \in Rel$

**shows** $P \leadsto\hat{}$ <*Rel″*> $R$
**proof** −
**from** *PRelQ* **have** *PSimQ*: $P \leadsto\hat{}$ <*Rel*> $Q$ **by**(*rule Sim*)
**from** *Eqvt′* **show** *?thesis*
**proof**(*induct rule*: *simCasesCont*[*of* - (*P*, *Q*)])
**case**(*Bound R′ a x*)
**have** *RTrans*: $R \longmapsto a{<}\nu x{>} \prec R'$ **by** *fact*
**have** $x \sharp (P, Q)$ **by** *fact*
**hence** *xFreshP*: $x \sharp P$ **and** *xFreshQ*: $x \sharp Q$ **by**(*simp add*: *fresh-prod*)+

**from** *QSimR RTrans xFreshQ* **have** $\exists Q'.\ Q \Longrightarrow_l\hat{}\ a{<}\nu x{>} \prec Q' \wedge (Q', R') \in$ *Rel′*
**by**(*rule simE*)
**then obtain** $Q'$ **where** *QTrans*: $Q \Longrightarrow_l\hat{}\ a{<}\nu x{>} \prec Q'$ **and** *Q′RelR′*: $(Q', R')$ ∈ *Rel′* **by** *blast*
**from** *PSimQ Sim Eqvt PRelQ QTrans xFreshP* **have** $\exists P'.\ P \Longrightarrow_l\hat{}\ a{<}\nu x{>} \prec P' \wedge (P', Q') \in Rel$
**by**(*rule simE2*)
**then obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l\hat{}\ a{<}\nu x{>} \prec P'$ **and** *P′RelQ′*: $(P', Q')$

$\in$ *Rel* **by** *blast*
**moreover from** $P'RelQ'$ $Q'RelR'$ *Trans* **have** $(P', R') \in Rel''$ **by** *blast*
**ultimately show** *?case* **by** *blast*
**next**
**case**(*Input* $R'$ *a x*)
**have** *RTrans*: $R \longmapsto a{<}x{>} \prec R'$ **by** *fact*
**have** $x \sharp (P, Q)$ **by** *fact*
**hence** *xFreshP*: $x \sharp P$ **and** *xFreshQ*: $x \sharp Q$ **by**(*simp add*: *fresh-prod*)+

**from** *QSimR RTrans xFreshQ* **obtain** $Q''$ **where** $\forall u. \exists Q'. Q \Longrightarrow_l u$ *in* $Q''{\rightarrow}a{<}x{>} \prec Q' \wedge (Q', R'[x::=u]) \in Rel'$
**by**(*blast dest*: *simE*)
**hence** $\exists Q'''. Q \Longrightarrow_\tau Q''' \wedge Q'''{\longmapsto}a{<}x{>} \prec Q'' \wedge (\forall u. \exists Q'. Q''[x::=u]{\Longrightarrow_\tau} Q' \wedge (Q', R'[x::=u]) \in Rel')$
**by**(*simp add*: *inputTransition-def*, *blast*)
**then obtain** $Q'''$ **where** *QChain*: $Q \Longrightarrow_\tau Q'''$
**and** $Q'''$*Trans*: $Q''' \longmapsto a{<}x{>} \prec Q''$
**and** *L1*: $\forall u. \exists Q'. Q''[x::=u]{\Longrightarrow_\tau} Q' \wedge (Q', R'[x::=u]) \in Rel'$
**by** *blast*
**from** *QChain PRelQ Sim* **have** $\exists P'''. P \Longrightarrow_\tau P''' \wedge (P''', Q''') \in Rel$
**by**(*rule weakSimTauChain*)
**then obtain** $P'''$ **where** *PChain*: $P \Longrightarrow_\tau P'''$ **and** $P'''RelQ'''$: $(P''', Q''') \in Rel$ **by** *blast*
**from** *PChain xFreshP* **have** *xFreshP*$'''$: $x \sharp P'''$ **by**(*rule freshChain*)
**from** $P'''RelQ'''$ **have** $P''' \leadsto\hat{} {<}Rel{>} Q'''$ **by**(*rule Sim*)
**hence** $\exists P''''. \forall u. \exists P''. P''' \Longrightarrow_l u$ *in* $P''''{\rightarrow}a{<}x{>} \prec P'' \wedge (P'', Q''[x::=u]) \in Rel$ **using** $Q'''$*Trans xFreshP*$'''$
**by**(*rule simE*)
**then obtain** $P''''$ **where** *L2*: $\forall u. \exists P''. P''' \Longrightarrow_l u$ *in* $P''''{\rightarrow}a{<}x{>} \prec P'' \wedge (P'', Q''[x::=u]) \in Rel$
**by** *blast*
**have** $\forall u. \exists P'\ Q'. P \Longrightarrow_l u$ *in* $P''''{\rightarrow}a{<}x{>} \prec P' \wedge (P', R'[x::=u]) \in Rel''$
**proof**(*rule allI*)
**fix** *u*
**from** *L1* **obtain** $Q'$ **where** $Q''$*Chain*: $Q''[x::=u] \Longrightarrow_\tau Q'$ **and** $Q'RelR'$: $(Q', R'[x::=u]) \in Rel'$
**by** *blast*
**from** *L2* **obtain** $P''$ **where** $P'''$*Trans*: $P''' \Longrightarrow_l u$ *in* $P''''{\rightarrow}a{<}x{>} \prec P''$
**and** $P''RelQ''$: $(P'', Q''[x::=u]) \in Rel$
**by** *blast*
**from** $P''RelQ''$ **have** $P'' \leadsto\hat{} {<}Rel{>} Q''[x::=u]$ **by**(*rule Sim*)
**have** $\exists P'. P'' \Longrightarrow_\tau P' \wedge (P', Q') \in Rel$ **using** $Q''$*Chain* $P''RelQ''$ *Sim*
**by**(*rule weakSimTauChain*)
**then obtain** $P'$ **where** $P''$*Chain*: $P'' \Longrightarrow_\tau P'$ **and** $P'RelQ'$: $(P', Q') \in Rel$ **by** *blast*
**from** *PChain* $P'''$*Trans* $P''$*Chain* **have** $P \Longrightarrow_l u$ *in* $P''''{\rightarrow}a{<}x{>} \prec P'$
**by**(*blast dest*: *Weak-Late-Step-Semantics.chainTransitionAppend*)
**moreover from** $P'RelQ'$ $Q'RelR'$ **have** $(P', R'[x::=u]) \in Rel''$ **by**(*insert Trans*, *auto*)

**ultimately show** $\exists P'\ Q'.\ P \Longrightarrow_l u\ in\ P''''\rightarrow a{<}x{>} \prec P' \wedge (P', R'[x{::=}u]) \in Rel''$ **by** *blast*
**qed**
**thus** *?case* **by** *force*
**next**
**case**(*Free* $R'$ $\alpha$)
**have** *RTrans*: $R \longmapsto \alpha \prec R'$ **by** *fact*
**with** *QSimR* **have** $\exists Q'.\ Q \Longrightarrow_l\hat{}\ \alpha \prec Q' \wedge (Q', R') \in Rel'$ **by**(*rule simE*)
**then obtain** $Q'$ **where** *QTrans*: $Q \Longrightarrow_l\hat{}\ \alpha \prec Q'$ **and** *Q'RelR'*: $(Q', R') \in Rel'$ **by** *blast*
**from** *PSimQ Sim Eqvt PRelQ QTrans* **have** $\exists P'.\ P \Longrightarrow_l\hat{}\ \alpha \prec P' \wedge (P', Q') \in Rel$ **by**(*rule simE2*)
**then obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l\hat{}\ \alpha \prec P'$ **and** *P'RelQ'*: $(P', Q') \in Rel$ **by** *blast*
**from** *P'RelQ' Q'RelR' Trans* **have** $(P', R') \in Rel''$ **by** *blast*
**with** *PTrans* **show** $\exists P'.\ P \Longrightarrow_l\hat{}\ \alpha \prec P' \wedge (P', R') \in Rel''$ **by** *blast*
**qed**
**qed**

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

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

**shows** $P \leadsto\hat{}{<}Rel{>}\ Q$
**proof**(*induct rule*: *simCases*)
**case**(*Bound* $Q'$ *a x*)
**have** $Q \longmapsto a{<}\nu x{>} \prec Q'$ **and** $x \sharp P$ **by** *fact+*
**with** *PSimQ* **obtain** $P'$ **where** *PTrans*: $P \longmapsto a{<}\nu x{>} \prec P'$ **and** *P'RelQ'*: $(P', Q') \in Rel$
**by**(*force dest*: *Strong-Late-Sim.simE simp add*: *derivative-def*)

**from** *PTrans* **have** $P \Longrightarrow_l\hat{}\ a{<}\nu x{>} \prec P'$
**by**(*force intro*: *Weak-Late-Step-Semantics.singleActionChain simp add*: *weak-Transition-def*)
**with** *P'RelQ'* **show** *?case* **by** *blast*
**next**
**case**(*Input* $Q'$ *a x*)
**assume** $Q \longmapsto a{<}x{>} \prec Q'$ **and** $x \sharp P$
**with** *PSimQ* **obtain** $P'$ **where** *PTrans*: $P \longmapsto a{<}x{>} \prec P'$ **and** *PDer*: *derivative* $P'$ $Q'$ (*InputS a*) *x Rel*
**by**(*blast dest*: *Strong-Late-Sim.simE*)

**have** $\forall u.\ \exists P''.\ P \Longrightarrow_l u\ in\ P'\rightarrow a{<}x{>} \prec P'' \wedge (P'', Q'[x{::=}u]) \in Rel$
**proof**(*rule allI*)
**fix** *u*
**from** *PTrans* **have** $P \Longrightarrow_l u\ in\ P'\rightarrow a{<}x{>} \prec P'[x{::=}u]$ **by**(*blast intro*: *Weak-Late-Step-Semantics.singleActio*

**moreover from** *PDer* **have** $(P'[x::=u],\ Q'[x::=u]) \in Rel$ **by**(*force simp add*: *derivative-def*)
**ultimately show** $\exists P''.\ P \Longrightarrow_l u\ in\ P' \rightarrow a{<}x{>} \prec P'' \wedge (P'',\ Q'[x::=u]) \in Rel$ **by** *auto*
**qed**
**thus** *?case* **by** *blast*
**next**
**case**(*Free* $Q'$ $\alpha$)
**have** $Q \longmapsto \alpha \prec Q'$ **by** *fact*
**with** *PSimQ* **obtain** $P'$ **where** *PTrans*: $P \longmapsto \alpha \prec P'$ **and** $P'RelQ'$: $(P',\ Q') \in Rel$
**by**(*blast dest*: *Strong-Late-Sim.simE*)
**from** *PTrans* **have** $P \Longrightarrow_l\hat{}\ \alpha \prec P'$ **by**(*rule Weak-Late-Semantics.singleActionChain*)
**with** $P'RelQ'$ **show** *?case* **by** *blast*
**qed**

**lemma** *strongAppend*:
**fixes** $P$ :: *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*: $P \leadsto\hat{}{<}Rel{>}\ Q$
**and** *QSimR*: $Q \leadsto[Rel']\ R$
**and** $Eqvt''$: *eqvt* $Rel''$
**and** *Trans*: $Rel\ O\ Rel' \subseteq Rel''$

**shows** $P \leadsto\hat{}{<}Rel''{>}\ R$
**proof** $-$
**from** $Eqvt''$ **show** *?thesis*
**proof**(*induct rule*: *simCasesCont*[*of* - $(P,\ Q)$])
**case**(*Bound* $R'$ $a$ $x$)
**have** $x \sharp (P,\ Q)$ **by** *fact*
**hence** *xFreshP*: $x \sharp P$ **and** *xFreshQ*: $x \sharp Q$ **by**(*simp add*: *fresh-prod*)+
**have** *RTrans*: $R \longmapsto a{<}\nu x{>} \prec R'$ **by** *fact*
**from** *xFreshQ QSimR RTrans* **obtain** $Q'$ **where** *QTrans*: $Q \longmapsto a{<}\nu x{>} \prec Q'$
**and** $Q'Rel'R'$: $(Q',\ R') \in Rel'$
**by**(*force dest*: *Strong-Late-Sim.simE simp add*: *derivative-def*)

**with** *PSimQ QTrans xFreshP* **have** $\exists P'.\ P \Longrightarrow_l\hat{}\ a{<}\nu x{>} \prec P' \wedge (P',\ Q') \in Rel$
**by**(*blast intro*: *simE*)
**then obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l\hat{}\ a{<}\nu x{>} \prec P'$ **and** $P'RelQ'$: $(P',\ Q') \in Rel$ **by** *blast*
**moreover from** $P'RelQ'$ $Q'Rel'R'$ *Trans* **have** $(P',\ R') \in Rel''$ **by** *blast*
**ultimately show** *?case* **by** *blast*
**next**

**case**(*Input R′ a x*)
**have** *RTrans*: $R \longmapsto a{<}x{>} \prec R'$ **by** *fact*
**have** $x \sharp (P, Q)$ **by** *fact*
**hence** *xFreshP*: $x \sharp P$ **and** *xFreshQ*: $x \sharp Q$ **by**(*simp add*: *fresh-prod*)+

**from** *QSimR RTrans xFreshQ* **obtain** $Q'$ **where** *QTrans*: $Q \longmapsto a{<}x{>} \prec Q'$ **and** *Q′Der*: *derivative* $Q'$ $R'$ (*InputS a*) *x Rel′*
**by**(*blast dest*: *Strong-Late-Sim.simE*)
**from** *QTrans PSimQ xFreshP* **obtain** $P''$ **where** *L2*: $\forall u. \exists P'. P \Longrightarrow_l u$ *in* $P'' \rightarrow a{<}x{>} \prec P' \wedge (P', Q'[x::=u]) \in Rel$
**by**(*blast dest*: *simE*)
**have** $\forall u. \exists P'. P \Longrightarrow_l u$ *in* $P'' \rightarrow a{<}x{>} \prec P' \wedge (P', R'[x::=u]) \in Rel''$
**proof**(*rule allI*)
**fix** *u*
**from** *L2* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l u$ *in* $P'' \rightarrow a{<}x{>} \prec P'$
**and** *P′RelQ′*: $(P', Q'[x::=u]) \in Rel$
**by** *blast*
**moreover from** *Q′Der* **have** $(Q'[x::=u], R'[x::=u]) \in Rel'$ **by**(*simp add*: *derivative-def*)
**ultimately show** $\exists P'. P \Longrightarrow_l u$ *in* $P'' \rightarrow a{<}x{>} \prec P' \wedge (P', R'[x::=u]) \in Rel''$ **using** *Trans* **by** *blast*
**qed**
**thus** *?case* **by** *force*
**next**
**case**(*Free R′ α*)
**have** *RTrans*: $R \longmapsto \alpha \prec R'$ **by** *fact*
**with** *QSimR* **obtain** $Q'$ **where** *QTrans*: $Q \longmapsto \alpha \prec Q'$ **and** *Q′RelR′*: $(Q', R') \in Rel'$
**by**(*blast dest*: *Strong-Late-Sim.simE*)
**from** *PSimQ QTrans* **have** $\exists P'. P \Longrightarrow_l\hat{}\ \alpha \prec P' \wedge (P', Q') \in Rel$
**by**(*blast intro*: *simE*)
**then obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l\hat{}\ \alpha \prec P'$ **and** *P′RelQ′*: $(P', Q') \in Rel$ **by** *blast*
**from** *P′RelQ′ Q′RelR′ Trans* **have** $(P', R') \in Rel''$ **by** *blast*
**with** *PTrans* **show** *?case* **by** *blast*
**qed**
**qed**

**end**

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

**lemma** *monoAux*: $A \subseteq B \Longrightarrow P \leadsto\hat{} {<}A{>}\ Q \longrightarrow P \leadsto\hat{} {<}B{>}\ Q$
**by**(*auto intro*: *Weak-Late-Sim.monotonic*)

**coinductive-set** *weakBisim* :: $(pi \times pi)$ *set*
**where**

*step*: $[\![P \leadsto\hat{}<weakBisim> Q; (Q, P) \in weakBisim]\!] \Longrightarrow (P, Q) \in weakBisim$
**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*: $\bigwedge P\ Q.\ (P, Q) \in X \Longrightarrow P \leadsto\hat{}<(X \cup weakBisim)> Q \wedge ((Q, P) \in X \vee Q \approx P)$

**shows** $P \approx Q$
**proof** −
**have** *aux*: $X \cup weakBisim = \{(P, Q).\ (P, Q) \in X \vee P \approx Q\}$ **by** *blast*

**from** *p* **show** *?thesis*
**by**(*coinduct*, *force dest*: *step simp add*: *aux*)
**qed**

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

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

**shows** $P \approx Q$
**using** *assms*
**by**(*coinduct rule*: *weakBisimCoinductAux*) *auto*

**lemma** *weak-coinduct*[*case-names weakBisim*, *case-conclusion weakBisim step*, *consumes 1*]:
**assumes** *p*: $(P, Q) \in X$
**and** *step*: $\bigwedge P\ Q.\ (P, Q) \in X \Longrightarrow P \leadsto\hat{}<X> Q \wedge (Q, P) \in X$

**shows** $P \approx Q$
**using** *p*
**proof**(*coinduct rule*: *weakBisimCoinductAux*)
**case** (*weakBisim P Q*)
**from** *step*[*OF this*] **show** *?case* **using** *Weak-Late-Sim.monotonic* **by** *blast*
**qed**

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

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

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

**shows** $P \approx Q$
**using** *assms*
**by**(*coinduct rule*: *weak-coinduct*) *auto*
**lemma** *monotonic*: $mono(\lambda p\ x1\ x2.\ \exists P\ Q.\ x1 = P \wedge x2 = Q \wedge P \leadsto\hat{}{<}\{(xa,\ x).\ p\ xa\ x\}{>}\ Q \wedge Q \leadsto\hat{}{<}\{(xa,\ x).\ p\ xa\ x\}{>}\ P)$
**by**(*auto intro*: *monoI Weak-Late-Sim.monotonic*)

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

**assumes** $P \approx Q$

**shows** $P \leadsto\hat{}{<}weakBisim{>}\ Q$
**and** $Q \approx P$
**using** *assms*
**by**(*auto intro*: *weakBisim.cases*)

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

**assumes** $P \leadsto\hat{}{<}weakBisim{>}\ Q$
**and** $Q \approx P$

**shows** $P \approx Q$
**using** *assms*
**by**(*auto intro*: *weakBisim.cases*)

**lemma** *eqvt*:
**shows** *eqvt weakBisim*
**proof**(*auto simp add*: *eqvt-def*)
**let** $?X = \{x.\ \exists P\ Q\ (perm{::}name\ prm).\ P \approx Q \wedge x = (perm \cdot P,\ perm \cdot Q)\}$
**fix** $P\ Q$
**fix** *perm*::*name prm*
**assume** *PBiSimQ*: $P \approx Q$

**hence** $(perm \cdot P,\ perm \cdot Q) \in ?X$ **by** *blast*
**moreover have** $\bigwedge P\ Q\ perm{::}name\ prm.\ [\![P \leadsto\hat{}{<}weakBisim{>}\ Q]\!] \Longrightarrow (perm \cdot P) \leadsto\hat{}{<}?X{>}\ (perm \cdot Q)$
**proof** −
**fix** $P\ Q$
**fix** *perm*::*name prm*
**assume** $P \leadsto\hat{}{<}weakBisim{>}\ Q$

**moreover have** $weakBisim \subseteq ?X$

**proof**(*auto*)
**fix** $P$ $Q$
**assume** $P \approx Q$
**moreover have** $P = ([]{::}name\ prm) \cdot P$ **and** $Q = ([]{::}name\ prm) \cdot Q$ **by** *auto*
**ultimately show** $\exists P'\ Q'.\ P' \approx Q' \wedge (\exists (perm{::}name\ prm).\ P = perm \cdot P' \wedge Q = perm \cdot Q')$
**by** *blast*
**qed**

**moreover have** *eqvt ?X*
**proof**(*auto simp add*: *eqvt-def*)
**fix** $P$ $Q$
**fix** *perm1*::*name prm*
**fix** *perm2*::*name prm*

**assume** $P \approx Q$
**moreover have** $perm1 \cdot perm2 \cdot P = (perm1 \ @\ perm2) \cdot P$ **by**(*simp add*: *pt2*[*OF pt-name-inst*])
**moreover have** $perm1 \cdot perm2 \cdot Q = (perm1 \ @\ perm2) \cdot Q$ **by**(*simp add*: *pt2*[*OF pt-name-inst*])

**ultimately show** $\exists P'\ Q'.\ P' \approx Q' \wedge (\exists (perm{::}name\ prm).\ perm1 \cdot perm2 \cdot P = perm \cdot P' \wedge$
$perm1 \cdot perm2 \cdot Q = perm \cdot Q')$
**by** *blast*
**qed**

**ultimately show** $(perm \cdot P) \rightsquigarrow\hat{}$<*?X*> $(perm \cdot Q)$
**by**(*rule Weak-Late-Sim.eqvtI*)
**qed**

**ultimately show** $(perm \cdot P) \approx (perm \cdot Q)$ **by**(*coinduct rule*: *weak-coinduct*, *blast dest*: *unfoldE*)
**qed**

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

**assumes** $P \approx Q$

**shows** $(perm \cdot P) \approx (perm \cdot Q)$
**using** *assms*
**by**(*rule eqvtRelI*[*OF eqvt*])

**lemma** *weakBisimEqvt*[*simp*]:
**shows** *eqvt weakBisim*

**by**(*auto simp add*: *eqvt-def eqvtI*)

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

**assumes** *PSimQ*: $P \sim Q$

**shows** $P \approx Q$
**proof** −
**have** $\bigwedge P\ Q.\ P \leadsto[bisim]\ Q \Longrightarrow P \leadsto\hat{}<(bisim \cup weakBisim)>\ Q$
**proof** −
**fix** $P\ Q$
**assume** $P \leadsto[bisim]\ Q$
**hence** $P \leadsto\hat{}<bisim>\ Q$ **by**(*rule strongSimWeakSim*)
**thus** $P \leadsto\hat{}<(bisim \cup weakBisim)>\ Q$
**by**(*blast intro*: *Weak-Late-Sim.monotonic*)
**qed**

**with** *PSimQ* **show** *?thesis*
**by**(*coinduct rule*: *weakBisimCoinductAux*, *force dest*: *Strong-Late-Bisim.bisimE symmetric*)
**qed**

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

**shows** $P \approx P$
**proof** −
**have** $(P, P) \in Id$ **by** *simp*
**then show** *?thesis*

**proof** (*coinduct rule*: *weak-coinduct*)
**case** (*weakBisim P Q*)
**have** $(P, Q) \in Id$ **by** *fact*
**thus** *?case* **by**(*auto intro*: *Weak-Late-Sim.reflexive*)
**qed**
**qed**

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

**assumes** $P \approx Q$

**shows** $Q \approx P$
**using** *assms*
**by**(*auto dest*: *unfoldE intro*: *unfoldI*)

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

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

**shows** $P \approx R$
**proof** −
**let** *?X* = *weakBisim O weakBisim*
**from** *assms* **have** $(P, R) \in$ *?X* **by** *blast*
**moreover have** $\bigwedge P\ Q\ R.\ [\![Q \rightsquigarrow\hat{}$ <*weakBisim*> $R;\ P \approx Q]\!] \Longrightarrow$
$P \rightsquigarrow\hat{}$ <(*?X* $\cup$ *weakBisim*)> $R$
**proof** −
**fix** $P\ Q\ R$
**assume** *PBiSimQ*: $P \approx Q$
**assume** $Q \rightsquigarrow\hat{}$ <*weakBisim*> $R$
**moreover have** *eqvt weakBisim* **by**(*rule eqvt*)
**moreover from** *eqvt* **have** *eqvt* (*?X* $\cup$ *weakBisim*) **by**(*auto simp add*: *eqvt-Trans*)
**moreover have** *weakBisim O weakBisim* $\subseteq$ *?X* $\cup$ *weakBisim* **by** *auto*
**moreover have** $\bigwedge P\ Q.\ P \approx Q \Longrightarrow P \rightsquigarrow\hat{}$ <*weakBisim*> $Q$ **by**(*rule unfoldE*)

**ultimately show** $P \rightsquigarrow\hat{}$ <(*?X* $\cup$ *weakBisim*)> $R$ **using** *PBiSimQ*
**by**(*rule Weak-Late-Sim.transitive*)
**qed**

**ultimately show** *?thesis*
**apply**(*coinduct rule*: *weakBisimCoinduct*, *auto*)
**by**(*blast dest*: *unfoldE symmetric*)+
**qed**

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

**shows** $P \approx Q$
**proof** −
**let** *?X* = *bisim O X O bisim*

**have** *Sim*: $\bigwedge P\ P'\ Q'\ Q.\ [\![P \sim P';\ P' \rightsquigarrow\hat{}$ <*?X*> $Q';\ Q' \rightsquigarrow[bisim]\ Q]\!] \Longrightarrow$
$P \rightsquigarrow\hat{}$ <*?X*> $Q$
**proof** −
**fix** $P\ P'\ Q'\ Q$

**assume** *PBisimP′*: $P \sim P'$
**assume** *P′SimQ′*: $P' \rightsquigarrow\hat{}$ *<?X>* $Q'$
**assume** *Q′SimQ*: $Q' \rightsquigarrow$[*bisim*] $Q$

**show** $P \rightsquigarrow\hat{}$ *<?X>* $Q$
**proof** −
**have** $P' \rightsquigarrow\hat{}$ *<?X>* $Q$
**proof** −
**have** *?X O bisim* ⊆ *?X* **by**(*blast intro*: *Strong-Late-Bisim.transitive*)
**moreover from** *Strong-Late-Bisim.bisimEqvt Eqvt* **have** *eqvt ?X* **by** *blast*
**ultimately show** *?thesis* **using** *P′SimQ′ Q′SimQ* **by**(*blast intro*: *strongAppend*)
**qed**
**moreover have** *eqvt bisim* **by**(*rule Strong-Late-Bisim.bisimEqvt*)
**moreover from** *Strong-Late-Bisim.bisimEqvt Eqvt* **have** *eqvt ?X* **by** *blast*
**moreover have** *bisim O ?X* ⊆ *?X* **by**(*blast intro*: *Strong-Late-Bisim.transitive*)
**moreover have** $\bigwedge P\ Q.\ P \sim Q \Longrightarrow P \rightsquigarrow\hat{}$ *<bisim>* $Q$ **by**(*blast dest*: *Strong-Late-Bisim.bisimE strongSimWeakSim*)
**ultimately show** *?thesis* **using** *PBisimP′* **by**(*rule Weak-Late-Sim.transitive*)
**qed**
**qed**

**from** *p* **have** $(P,\ Q) \in$ *?X* **by**(*blast intro*: *Strong-Late-Bisim.reflexive*)
**moreover from** *step Sim* **have** $\bigwedge P\ Q.\ (P,\ Q) \in$ *?X* $\Longrightarrow P \rightsquigarrow\hat{}$ *<?X>* $Q \wedge (Q,\ P) \in$ *?X*
**by**(*blast dest*: *Strong-Late-Bisim.bisimE Strong-Late-Bisim.symmetric*)

**ultimately show** *?thesis* **by**(*rule weak-coinduct*)
**qed**

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

**shows** $P \approx Q$
**using** *assms*
**by**(*coinduct rule*: *transitive-coinduct-weak*) *auto*

**end**

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

**definition** *weakStepSimAct* :: *pi* ⇒ *residual* ⇒ (*′a::fs-name*) ⇒ (*pi* × *pi*) *set* ⇒ *bool* **where**
*weakStepSimAct P Rs C Rel* ≡ (∀ $Q'$ *a x*. *Rs* = *a<νx>* ≺ $Q'$ ⟶ *x* ♯ *C* ⟶

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

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

**lemmas** *weakStepSimDef* = *weakStepSimAct-def weakStepSim-def*
**lemma** *weakStepSimAux P Rel Q* = *weakStepSim P Rel Q*
**by**(*auto simp add*: *weakStepSimDef weakStepSimAux-def*)

**lemma** *monotonic*:
  **fixes** $A$ :: $(pi \times pi)\ set$
  **and** $B$ :: $(pi \times pi)\ set$
  **and** $P$ :: $pi$
  **and** $P'$ :: $pi$

  **assumes** $P \leadsto{<}A{>}\ P'$
  **and** $A \subseteq B$

  **shows** $P \leadsto{<}B{>}\ P'$
**using** *assms*
**apply**(*auto simp add*: *weakStepSimDef*)
**apply** *blast*
**apply**(*erule-tac* $x{=}a{<}x{>} \prec Q'$ **in** *allE*)
**apply**(*clarsimp*)
**apply**(*rotate-tac 4*)
**apply**(*erule-tac* $x{=}Q'$ **in** *allE*)
**apply**(*erule-tac* $x{=}a$ **in** *allE*)
**apply**(*erule-tac* $x{=}x$ **in** *allE*)
**by** *blast+*

**lemma** *simCasesCont*[*consumes 1*, *case-names Bound Input Free*]:
  **fixes** $P$ :: $pi$
  **and** $Q$ :: $pi$
  **and** $Rel$ :: $(pi \times pi)\ set$
  **and** $C$ :: $'a$::*fs-name*

**assumes** *Eqvt*: *eqvt Rel*
**and** *Bound*: $\bigwedge Q'$ $a$ $x$. $[\![x \sharp C; Q \longmapsto a{<}\nu x{>} \prec Q']\!] \Longrightarrow \exists P'. P \Longrightarrow_l a{<}\nu x{>} \prec P' \wedge (P', Q') \in Rel$
**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** $P \leadsto{<}Rel{>}$ $Q$
**using** *Free*
**proof**(*auto simp add*: *weakStepSimDef*)
**fix** $Q'$ $a$ $x$
**assume** *xFreshP*: $(x::name) \sharp P$
**assume** *Trans*: $Q \longmapsto a{<}\nu x{>} \prec Q'$
**have** $\exists c::name.\ c \sharp (P, Q', x, C)$ **by**(*blast intro*: *name-exists-fresh*)
**then obtain** $c::name$ **where** *cFreshP*: $c \sharp P$ **and** *cFreshQ'*: $c \sharp Q'$ **and** *cFreshC*: $c \sharp C$
**and** *cineqx*: $c \neq x$
**by**(*force simp add*: *fresh-prod*)

**from** *Trans cFreshQ'* **have** $Q \longmapsto a{<}\nu c{>} \prec ([(x, c)] \cdot Q')$ **by**(*simp add*: *alphaBoundResidual*)
**with** *cFreshC* **have** $\exists P'. P \Longrightarrow_l a{<}\nu c{>} \prec P' \wedge (P', [(x, c)] \cdot Q') \in Rel$
**by**(*rule Bound*)
**then obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l a{<}\nu c{>} \prec P'$ **and** *P'RelQ'*: $(P', [(x, c)] \cdot Q') \in Rel$
**by** *blast*

**from** *PTrans xFreshP cineqx* **have** *xFreshP'*: $x \sharp P'$ **by**(*force dest*: *Weak-Late-Step-Semantics.freshTransition*)
**with** *PTrans* **have** $P \Longrightarrow_l a{<}\nu x{>} \prec ([(x, c)] \cdot P')$ **by**(*simp add*: *alphaBoundResidual name-swap*)
**moreover have** $([(x, c)] \cdot P', Q') \in Rel$ (**is** *?goal*)
**proof** −
**from** *Eqvt P'RelQ'* **have** $([(x, c)] \cdot P', [(x, c)] \cdot [(x, c)] \cdot Q') \in Rel$
**by**(*rule eqvtRelI*)
**with** *cineqx* **show** *?goal* **by**(*simp add*: *name-calc*)
**qed**
**ultimately show** $\exists P'. P \Longrightarrow_l a{<}\nu x{>} \prec P' \wedge (P', Q') \in Rel$ **by** *blast*
**next**
**fix** $Q'$ $a$ $x$ $u$
**assume** *QTrans*: $Q \longmapsto a{<}x{>} \prec (Q'::pi)$
**and** *xFreshP*: $x \sharp P$

**have** $\exists c::name.\ c \sharp (P, Q', C, x)$ **by**(*blast intro*: *name-exists-fresh*)
**then obtain** $c::name$ **where** *cFreshP*: $c \sharp P$ **and** *cFreshQ'*: $c \sharp Q'$ **and** *cFreshC*: $c \sharp C$
**and** *cineqx*: $c \neq x$
**by**(*force simp add*: *fresh-prod*)

**from** *QTrans cFreshQ'* **have** $Q \longmapsto a{<}c{>} \prec ([(x, c)] \cdot Q')$ **by**(*simp add*: *al-*

*phaBoundResidual*)
**with** *cFreshC* **have** $\exists P''. \forall u. \exists P'. P \Longrightarrow_l u$ *in* $P''\rightarrow a{<}c{>} \prec P' \wedge (P', ([(x, c)] \cdot Q')[c{::=}u]) \in Rel$
**by**(*rule Input*)

**then obtain** $P''$ **where** *L1*: $\forall u. \exists P'. P \Longrightarrow_l u$ *in* $P''\rightarrow a{<}c{>} \prec P' \wedge (P', ([(x, c)] \cdot Q')[c{::=}u]) \in Rel$ **by** *blast*

**have** $\forall u. \exists P'. P \Longrightarrow_l u$ *in* $([(c, x)] \cdot P'')\rightarrow a{<}x{>} \prec P' \wedge (P', Q'[x{::=}u]) \in Rel$
**proof**(*auto*)
**fix** $u$
**from** *L1* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l u$ *in* $P''\rightarrow a{<}c{>} \prec P'$ **and** *P'RelQ'*: $(P', ([(x, c)] \cdot Q')[c{::=}u]) \in Rel$
**by** *blast*

**from** *PTrans xFreshP* **have** $P \Longrightarrow_l u$ *in* $([(c, x)] \cdot P'')\rightarrow a{<}x{>} \prec P'$ **by**(*rule alphaInput*)
**moreover from** *P'RelQ' cFreshQ'* **have** $(P', Q'[x{::=}u]) \in Rel$ **by**(*simp add*: *renaming*[*THEN sym*] *name-swap*)

**ultimately show** $\exists P'. P \Longrightarrow_l u$ *in* $([(c, x)] \cdot P'')\rightarrow a{<}x{>} \prec P' \wedge (P', Q'[x{::=}u]) \in Rel$ **by** *blast*
**qed**

**thus** $\exists P''. \forall u. \exists P'. P \Longrightarrow_l u$ *in* $P''\rightarrow a{<}x{>} \prec P' \wedge (P', Q'[x{::=}u]) \in Rel$ **by** *blast*
**qed**

**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** $P \leadsto{<}Rel{>}\ Q$
**using** *assms*
**by**(*auto simp add*: *weakStepSimDef*)

**lemma** *simActBoundCases*[*consumes 1*, *case-names Input BoundOutput*]:
**fixes** $P$ :: *pi*
**and** $a$ :: *subject*
**and** $x$ :: *name*
**and** $Q'$ :: *pi*

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

**assumes** *EqvtRel*: *eqvt Rel*
**and** *DerInput*: $\bigwedge b.\ a = InputS\ b \Longrightarrow (\exists P''.\ \forall u.\ \exists P'.\ (P \Longrightarrow_l u\ in\ P'' \rightarrow b<x> \prec P') \wedge (P',\ Q'[x::=u]) \in Rel)$
**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* $P$ $(a«x» \prec Q')$ $P$ *Rel*
**proof**(*simp add*: *weakStepSimAct-def fresh-prod*, *auto*)
**fix** $Q''$ $b$ $y$
**assume** *Eq*: $a«x» \prec Q' = b<\nu y> \prec Q''$
**assume** *yFreshP*: $y \sharp P$

**from** *Eq* **have** $a = BoundOutputS\ b$ **by**(*simp add*: *residual.inject*)

**from** *yFreshP DerBoundOutput*[*OF this*] *Eq* **show** $\exists P'.\ P \Longrightarrow_l b<\nu y> \prec P' \wedge (P',\ Q'') \in Rel$
**proof**(*cases x=y*, *auto simp add*: *residual.inject name-abs-eq*)
**fix** $P'$
**assume** *PTrans*: $P \Longrightarrow_l b<\nu x> \prec P'$
**assume** *P'RelQ'*: $(P',\ ([(x,\ y)] \cdot Q'')) \in Rel$
**assume** *xineqy*: $x \neq y$

**with** *PTrans yFreshP* **have** *yFreshP'*: $y \sharp P'$
**by**(*force intro*: *Weak-Late-Step-Semantics.freshTransition*)

**hence** $b<\nu x> \prec P' = b<\nu y> \prec [(x,\ y)] \cdot P'$ **by**(*rule alphaBoundResidual*)
**moreover have** $([(x,\ y)] \cdot P',\ Q'') \in Rel$
**proof** −
**from** *EqvtRel P'RelQ'* **have** $([(x,\ y)] \cdot P',\ [(x,\ y)] \cdot ([(x,\ y)] \cdot Q'')) \in Rel$
**by**(*rule eqvtRelI*)
**thus** *?thesis* **by**(*simp add*: *name-calc*)
**qed**

**ultimately show** $\exists P'.\ P \Longrightarrow_l b<\nu y> \prec P' \wedge (P',\ Q'') \in Rel$ **using** *PTrans* **by** *auto*
**qed**
**next**
**fix** $Q''$ $b$ $y$ $u$
**assume** *Eq*: $a«x» \prec Q' = b<y> \prec Q''$
**assume** *yFreshP*: $y \sharp P$

**from** *Eq* **have** $a = InputS\ b$ **by**(*simp add*: *residual.inject*)
**from** *DerInput*[*OF this*] **obtain** $P''$ **where** *L1*: $\forall u.\ \exists P'.\ P \Longrightarrow_l u\ in\ P'' \rightarrow b<x> \prec P' \wedge$
$(P',\ Q'[x::=u]) \in Rel$
**by** *blast*

**have** $\forall u.\ \exists P'.\ P \Longrightarrow_l u\ in\ ([(x,\ y)] \cdot P'') \rightarrow b<y> \prec P' \wedge (P',\ Q''[y::=u]) \in Rel$
**proof**(*rule allI*)
**fix** $u$
**from** *L1 Eq* **show** $\exists P'.\ P \Longrightarrow_l u\ in\ ([(x,\ y)] \cdot P'') \rightarrow b<y> \prec P' \wedge (P',$ $Q''[y::=u]) \in Rel$
**proof**(*cases* $x=y$, *auto simp add*: *residual.inject name-abs-eq*)
**assume** *Der*: $\forall u.\ \exists P'.\ P \Longrightarrow_l u\ in\ P'' \rightarrow b<x> \prec P' \wedge (P',\ ([(x,\ y)] \cdot$ $Q'')[x::=u]) \in Rel$
**assume** *xFreshQ''*: $x \sharp Q''$

**from** *Der* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l u\ in\ P'' \rightarrow b<x> \prec P'$
**and** *P'RelQ'*: $(P',\ ([(x,\ y)] \cdot Q'')[x::=u]) \in Rel$
**by** *force*

**from** *PTrans yFreshP* **have** $P \Longrightarrow_l u\ in\ ([(x,\ y)] \cdot P'') \rightarrow b<y> \prec P'$ **by**(*rule alphaInput*)
**moreover from** *xFreshQ'' P'RelQ'* **have** $(P',\ Q''[y::=u]) \in Rel$
**by**(*simp add*: *renaming*)
**ultimately show** *?thesis* **by** *force*
**qed**
**qed**
**thus** $\exists P''.\ \forall u.\ \exists P'.\ P \Longrightarrow_l u\ in\ P'' \rightarrow b<y> \prec P' \wedge (P',\ Q''[y::=u]) \in Rel$
**by** *blast*
**qed**

**lemma** *simActFreeCases*[*consumes 0*, *case-names Free*]:
**fixes** $P$ :: *pi*
**and** $\alpha$ :: *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*
**using** *assms*
**by**(*simp add*: *weakStepSimAct-def residual.inject*)

**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** $Q \longmapsto \alpha \prec Q' \Longrightarrow (\exists P'.\ P \Longrightarrow_l \alpha \prec P' \wedge (P', Q') \in Rel)$
**using** *assms* **by**(*simp add*: *weakStepSimDef*)+

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

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

**shows** $\exists P'.\ P \Longrightarrow_\tau P' \wedge (P', Q') \in Rel$
**proof** −
**from** *QChain* **show** *?thesis*
**proof**(*induct rule*: *tauChainInduct*)
**case** *id*
**have** $P \Longrightarrow_\tau P$ **by** *simp*
**with** *PRelQ* **show** *?case* **by** *blast*
**next**
**case**(*ih* $Q'$ $Q''$)
**have** *IH*: $\exists P'.\ P \Longrightarrow_\tau P' \wedge (P', Q') \in Rel$ **by** *fact*
**then obtain** $P'$ **where** *PChain*: $P \Longrightarrow_\tau P'$ **and** *P'RelQ'*: $(P', Q') \in Rel$ **by** *blast*
**from** *P'RelQ'* **have** $P' \leadsto{<}Rel{>}\ Q'$ **by**(*rule Sim*)
**moreover have** *Q'Trans*: $Q' \longmapsto \tau \prec Q''$ **by** *fact*
**ultimately have** $\exists P''.\ P' \Longrightarrow_l \tau \prec P'' \wedge (P'', Q'') \in Rel$ **by**(*rule simE*)
**then obtain** $P''$ **where** *P'Trans*: $P' \Longrightarrow_l \tau \prec P''$ **and** *P''RelQ''*: $(P'', Q'') \in Rel$ **by** *blast*
**from** *P'Trans* **have** $P' \Longrightarrow_\tau P''$ **by**(*rule Weak-Late-Step-Semantics.tauTransitionChain*)
**with** *PChain* **have** $P \Longrightarrow_\tau P''$ **by** *auto*
**with** *P''RelQ''* **show** *?case* **by** *blast*
**qed**
**qed**

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

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

**shows** $P \leadsto{<}Rel{>}\ Q$
**proof**(*auto simp add*: *weakStepSimDef*)

**fix** $Q'$ *a x*
**assume** $Q \longmapsto a{<}\nu x{>} \prec Q'$ **and** $x \sharp P$
**with** *PSimQ* **have** $\exists P'. \ P \longmapsto a{<}\nu x{>} \prec P' \wedge$ *derivative* $P'$ $Q'$ (*BoundOutputS a*) *x Rel*
**by**(*rule Strong-Late-Sim.simE*)
**then obtain** $P'$ **where** *PTrans*: $P \longmapsto a{<}\nu x{>} \prec P'$ **and** $P'RelQ'$: $(P', Q') \in$ *Rel*
**by**(*force simp add*: *derivative-def*)

**from** *PTrans* **have** $P \Longrightarrow_l a{<}\nu x{>} \prec P'$ **by**(*rule Weak-Late-Step-Semantics.singleActionChain*)
**thus** $\exists P'. \ P \Longrightarrow_l a{<}\nu x{>} \prec P' \wedge (P', Q') \in$ *Rel* **using** $P'RelQ'$ **by** *blast*
**next**
**fix** $Q'$ *a x u*
**assume** $Q \longmapsto a{<}x{>} \prec Q'$ **and** $x \sharp P$
**with** *PSimQ* **have** *L1*: $\exists P'. \ P \longmapsto a{<}x{>} \prec P' \wedge$ *derivative* $P'$ $Q'$ (*InputS a*) *x Rel*
**by**(*blast intro*: *Strong-Late-Sim.simE*)
**then obtain** $P'$ **where** *PTrans*: $P \longmapsto a{<}x{>} \prec P'$ **and** *PDer*: *derivative* $P'$ $Q'$ (*InputS a*) *x Rel*
**by** *blast*

**have** $\forall u. \ \exists P''. \ P \Longrightarrow_l u \ in \ P' \rightarrow a{<}x{>} \prec P'' \wedge (P'', Q'[x::=u]) \in$ *Rel*
**proof**(*rule allI*)
**fix** *u*
**from** *PTrans* **have** $P \Longrightarrow_l u \ in \ P' \rightarrow a{<}x{>} \prec P'[x::=u]$ **by**(*blast intro*: *Weak-Late-Step-Semantics.singleActio*
**moreover from** *PDer* **have** $(P'[x::=u], Q'[x::=u]) \in$ *Rel* **by**(*force simp add*: *derivative-def*)
**ultimately show** $\exists P''. \ P \Longrightarrow_l u \ in \ P' \rightarrow a{<}x{>} \prec P'' \wedge (P'', Q'[x::=u]) \in$ *Rel* **by** *auto*
**qed**
**thus** $\exists P''. \ \forall u. \ \exists P'. \ P \Longrightarrow_l u \ in \ P'' \rightarrow a{<}x{>} \prec P' \wedge (P', Q'[x::=u]) \in$ *Rel* **by** *blast*
**next**
**fix** $Q'$ $\alpha$
**assume** $Q \longmapsto \alpha \prec Q'$
**with** *PSimQ* **have** $\exists P'. \ P \longmapsto \alpha \prec P' \wedge (P', Q') \in$ *Rel* **by**(*rule Strong-Late-Sim.simE*)
**then obtain** $P'$ **where** *PTrans*: $P \longmapsto \alpha \prec P'$ **and** $P'RelQ'$: $(P', Q') \in$ *Rel* **by** *blast*

**from** *PTrans* **have** $P \Longrightarrow_l \alpha \prec P'$ **by**(*rule Weak-Late-Step-Semantics.singleActionChain*)
**thus** $\exists P'. \ P \Longrightarrow_l \alpha \prec P' \wedge (P', Q') \in$ *Rel* **using** $P'RelQ'$ **by** *blast*
**qed**

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

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

**shows** $P \leadsto\hat{}<Rel>\ Q$
**using** *assms*
**by**(*force simp add*: *weakStepSimDef simDef weakTransition-def*)

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

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

**shows** $(perm \bullet P) \leadsto<Rel'>\ (perm \bullet Q)$
**using** *EqvtRel'*
**proof**(*induct rule*: *simCasesCont*[*of* - $perm \bullet P$])
**case**(*Bound* $Q'$ $a$ $x$)
**have** *QTrans*: $(perm \bullet Q) \longmapsto a<\nu x> \prec Q'$ **by** *fact*
**have** *xFreshP*: $x \sharp perm \bullet P$ **by** *fact*

**from** *QTrans* **have** $(rev\ perm \bullet (perm \bullet Q)) \longmapsto rev\ perm \bullet (a<\nu x> \prec Q')$
**by**(*rule eqvts*)
**hence** $Q \longmapsto (rev\ perm \bullet a)<\nu(rev\ perm \bullet x)> \prec (rev\ perm \bullet Q')$
**by**(*simp add*: *name-rev-per*)
**moreover from** *xFreshP* **have** $(rev\ perm \bullet x) \sharp P$ **by**(*simp add*: *name-fresh-left*)
**ultimately obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l (rev\ perm \bullet a)<\nu(rev\ perm \bullet x)>$
$\prec P'$
**and** *P'RelQ'*: $(P', rev\ perm \bullet Q') \in Rel$ **using** *Sim*
**by**(*blast dest*: *simE*)

**from** *PTrans* **have** $(perm \bullet P) \Longrightarrow_l perm \bullet ((rev\ perm \bullet a)<\nu(rev\ perm \bullet x)> \prec$
$P')$
**by**(*rule Weak-Late-Step-Semantics.eqvtI*)
**hence** $(perm \bullet P) \Longrightarrow_l a<\nu x> \prec (perm \bullet P')$ **by**(*simp add*: *name-per-rev*)
**moreover have** $(perm \bullet P', Q') \in Rel'$
**proof** −
**from** *P'RelQ' RelRel'* **have** $(P', rev\ perm \bullet Q') \in Rel'$ **by** *blast*
**with** *EqvtRel'* **have** $(perm \bullet P', perm \bullet (rev\ perm \bullet Q')) \in Rel'$
**by**(*rule eqvtRelI*)
**thus** *?thesis* **by**(*simp add*: *name-per-rev*)
**qed**
**ultimately show** *?case* **by** *blast*
**next**
**case**(*Input* $Q'$ $a$ $x$)
**have** *QTrans*: $(perm \bullet Q) \longmapsto a<x> \prec Q'$ **by** *fact*
**have** *xFreshP*: $x \sharp perm \bullet P$ **by** *fact*

**from** *QTrans* **have** $(rev\ perm \cdot (perm \cdot Q)) \longmapsto rev\ perm \cdot (a{<}x{>} \prec Q')$
**by**(*rule eqvts*)
**hence** $Q \longmapsto (rev\ perm \cdot a){<}(rev\ perm \cdot x){>} \prec (rev\ perm \cdot Q')$
**by**(*simp add*: *name-rev-per*)
**moreover from** *xFreshP* **have** *xFreshP*: $(rev\ perm \cdot x) \sharp P$ **by**(*simp add*: *name-fresh-left*)
**ultimately obtain** $P''$
**where** *L1*: $\forall u.\ \exists P'.\ P \Longrightarrow_l u\ in\ P''{\rightarrow}(rev\ perm \cdot a){<}(rev\ perm \cdot x){>} \prec P' \wedge$
$(P', (rev\ perm \cdot Q')[(rev\ perm \cdot x)::=u]) \in Rel$ **using** *Sim*
**by**(*blast dest*: *simE*)
**have** $\forall u.\ \exists P'.\ (perm \cdot P) \Longrightarrow_l u\ in\ (perm \cdot P''){\rightarrow}a{<}x{>} \prec P' \wedge (P', Q'[x::=u]) \in Rel'$
**proof**(*rule allI*)
**fix** $u$
**from** *L1* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l (rev\ perm \cdot u)\ in\ P''{\rightarrow}(rev\ perm \cdot a){<}(rev\ perm \cdot x){>} \prec P'$
**and** *P'RelQ'*: $(P', (rev\ perm \cdot Q')[(rev\ perm \cdot x)::=(rev\ perm \cdot u)]) \in Rel$ **by** *blast*
**from** *PTrans* **have** $(perm \cdot P) \Longrightarrow_l (perm \cdot (rev\ perm \cdot u))\ in\ (perm \cdot P''){\rightarrow}(perm \cdot rev\ perm \cdot a){<}(perm \cdot rev\ perm \cdot x){>} \prec (perm \cdot P')$
**by**(*rule-tac Weak-Late-Step-Semantics.eqvtI*, *auto*)
**hence** $(perm \cdot P) \Longrightarrow_l u\ in\ (perm \cdot P''){\rightarrow}a{<}x{>} \prec (perm \cdot P')$ **by**(*simp add*: *name-per-rev*)
**moreover have** $(perm \cdot P', Q'[x::=u]) \in Rel'$
**proof** $-$
**from** *P'RelQ' RelRel'* **have** $(P', (rev\ perm \cdot Q')[(rev\ perm \cdot x)::=(rev\ perm \cdot u)]) \in Rel'$ **by** *blast*
**with** *EqvtRel'* **have** $(perm \cdot P', perm \cdot ((rev\ perm \cdot Q')[(rev\ perm \cdot x)::=(rev\ perm \cdot u)])) \in Rel'$
**by**(*rule eqvtRelI*)
**thus** *?thesis* **by**(*simp add*: *name-per-rev eqvt-subs*[*THEN sym*] *name-calc*)
**qed**
**ultimately show** $\exists P'.\ (perm \cdot P) \Longrightarrow_l u\ in\ (perm \cdot P''){\rightarrow}a{<}x{>} \prec P' \wedge (P', Q'[x::=u]) \in Rel'$ **by** *blast*
**qed**
**thus** *?case* **by** *blast*
**next**
**case**(*Free* $Q'$ $\alpha$)
**have** *QTrans*: $(perm \cdot Q) \longmapsto \alpha \prec Q'$ **by** *fact*

**from** *QTrans* **have** $(rev\ perm \cdot (perm \cdot Q)) \longmapsto rev\ perm \cdot (\alpha \prec Q')$
**by**(*rule eqvts*)
**hence** $Q \longmapsto (rev\ perm \cdot \alpha) \prec (rev\ perm \cdot Q')$
**by**(*simp add*: *name-rev-per*)
**with** *Sim* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l (rev\ perm \cdot \alpha) \prec P'$ **and** *PRel*: $(P', (rev\ perm \cdot Q')) \in Rel$
**by**(*blast dest*: *simE*)
**from** *PTrans* **have** $(perm \cdot P) \Longrightarrow_l perm \cdot ((rev\ perm \cdot \alpha) \prec P')$
**by**(*rule Weak-Late-Step-Semantics.eqvtI*)

**hence** $(perm \cdot P) \Longrightarrow_l \alpha \prec (perm \cdot P')$ **by**(*simp add*: *name-per-rev*)
**moreover have** $((perm \cdot P'), Q') \in Rel'$
**proof** $-$
**from** *PRel EqvtRel' RelRel'* **have** $((perm \cdot P'), (perm \cdot (rev\ perm \cdot Q'))) \in Rel'$
**by**(*force intro*: *eqvtRelI*)
**thus** *?thesis* **by**(*simp add*: *name-per-rev*)
**qed**
**ultimately show** *?case* **by** *blast*
**qed**

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

**assumes** *PSimQ*: $P \leadsto<Rel> Q$
**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** $Q \Longrightarrow_l a<\nu x> \prec Q' \Longrightarrow x \sharp P \Longrightarrow \exists P'.\ P \Longrightarrow_l a<\nu x> \prec P' \wedge (P', Q') \in Rel$
**and** $Q \Longrightarrow_l \alpha \prec Q' \Longrightarrow \exists P'.\ P \Longrightarrow_l \alpha \prec P' \wedge (P', Q') \in Rel$
**proof** $-$
**assume** *QTrans*: $Q \Longrightarrow_l a<\nu x> \prec Q'$
**assume** *xFreshP*: $x \sharp P$
**have** *Goal*: $\bigwedge P\ Q\ a\ x\ Q'.\ [\![P \leadsto<Rel> Q;\ Q \Longrightarrow_l a<\nu x> \prec Q';\ x \sharp P;\ x \sharp Q;\ (P, Q) \in Rel]\!] \Longrightarrow$
$\exists P'.\ P \Longrightarrow_l a<\nu x> \prec P' \wedge (P', Q') \in Rel$
**proof** $-$
**fix** $P\ Q\ a\ x\ Q'$
**assume** *PSimQ*: $P \leadsto<Rel> Q$
**assume** *QTrans*: $Q \Longrightarrow_l a<\nu x> \prec Q'$
**assume** *xFreshP*: $x \sharp P$
**assume** *xFreshQ*: $x \sharp Q$
**assume** *PRelQ*: $(P, Q) \in Rel$

**from** *QTrans xFreshQ* **obtain** $Q''\ Q'''$ **where** *QChain*: $Q \Longrightarrow_\tau Q''$
**and** $Q''$*Trans*: $Q'' \longmapsto a<\nu x> \prec Q'''$
**and** $Q'''$*Chain*: $Q''' \Longrightarrow_\tau Q'$
**by**(*force dest*: *transitionE simp add*: *weakTransition-def*)

**from** *QChain PRelQ Sim* **have** $\exists P''.\ P \Longrightarrow_\tau P'' \wedge (P'', Q'') \in Rel$
**by**(*rule Weak-Late-Sim.weakSimTauChain*)
**then obtain** $P''$ **where** *PChain*: $P \Longrightarrow_\tau P''$ **and** $P''$*RelQ*$''$: $(P'', Q'') \in Rel$

**by** *blast*
**from** *PChain xFreshP* **have** *xFreshP''*: $x \sharp P''$ **by**(*rule freshChain*)

**from** *P''RelQ''* **have** $P'' \leadsto\hat{}$ <*Rel*> $Q''$ **by**(*rule Sim*)
**hence** $\exists P'''.\ P'' \Longrightarrow_l\hat{}\ a{<}\nu x{>} \prec P''' \wedge (P''', Q''') \in Rel$ **using** *Q''Trans xFreshP''*
**by**(*rule Weak-Late-Sim.simE*)
**then obtain** $P'''$ **where** *P''Trans*: $P'' \Longrightarrow_l a{<}\nu x{>} \prec P'''$ **and** *P'''RelQ'''*: $(P''', Q''') \in Rel$
**by**(*force simp add*: *weakTransition-def*)

**have** $\exists P'.\ P''' \Longrightarrow_\tau P' \wedge (P', Q') \in Rel$ **using** *Q'''Chain P'''RelQ''' Sim*
**by**(*rule Weak-Late-Sim.weakSimTauChain*)
**then obtain** $P'$ **where** *P'''Chain*: $P''' \Longrightarrow_\tau P'$ **and** *P'RelQ'*: $(P', Q') \in Rel$ **by** *blast*

**from** *PChain P''Trans P'''Chain xFreshP''* **have** $P \Longrightarrow_l a{<}\nu x{>} \prec P'$
**by**(*blast dest*: *Weak-Late-Step-Semantics.chainTransitionAppend*)
**with** *P'RelQ'* **show** $\exists P'.\ P \Longrightarrow_l a{<}\nu x{>} \prec P' \wedge (P', Q') \in Rel$ **by** *blast*
**qed**

**have** $\exists c{::}name.\ c \sharp (Q, Q', P, x)$ **by**(*blast intro*: *name-exists-fresh*)
**then obtain** $c{::}name$ **where** *cFreshQ*: $c \sharp Q$ **and** *cFreshQ'*: $c \sharp Q'$ **and** *cFreshP*: $c \sharp P$
**and** *xineqc*: $x \neq c$
**by**(*force simp add*: *fresh-prod*)

**from** *QTrans cFreshQ'* **have** $Q \Longrightarrow_l a{<}\nu c{>} \prec ([(x, c)] \cdot Q')$ **by**(*simp add*: *alphaBoundResidual*)
**with** *PSimQ* **have** $\exists P'.\ P \Longrightarrow_l a{<}\nu c{>} \prec P' \wedge (P', [(x, c)] \cdot Q') \in Rel$ **using** *cFreshP cFreshQ PRelQ*
**by**(*rule Goal*)
**then obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l a{<}\nu c{>} \prec P'$ **and** *P'RelQ'*: $(P', [(x, c)] \cdot Q') \in Rel$
**by** *force*
**have** $P \Longrightarrow_l a{<}\nu x{>} \prec ([(x, c)] \cdot P')$
**proof** −
**from** *PTrans xFreshP xineqc* **have** $x \sharp P'$ **by**(*rule Weak-Late-Step-Semantics.freshTransition*)
**with** *PTrans* **show** *?thesis* **by**(*simp add*: *alphaBoundResidual name-swap*)
**qed**
**moreover have** $([(x, c)] \cdot P', Q') \in Rel$
**proof** −
**from** *Eqvt P'RelQ'* **have** $([(x, c)] \cdot P', [(x, c)] \cdot [(x, c)] \cdot Q') \in Rel$
**by**(*rule eqvtRelI*)
**thus** *?thesis* **by** *simp*
**qed**

**ultimately show** $\exists P'.\ P \Longrightarrow_l a{<}\nu x{>} \prec P' \wedge (P', Q') \in Rel$ **by** *blast*
**next**

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

**then obtain** $Q''$ $Q'''$ **where** *QChain*: $Q \Longrightarrow_\tau Q''$
**and** $Q''$*Trans*: $Q'' \longmapsto\alpha \prec Q'''$
**and** $Q'''$*Chain*: $Q''' \Longrightarrow_\tau Q'$
**by**(*blast dest*: *transitionE*)

**thus** $\exists P'.\ P \Longrightarrow_l \alpha \prec P' \wedge (P', Q') \in Rel$
**proof**(*induct arbitrary*: $\alpha$ $Q'''$ $Q'$ *rule*: *tauChainInduct*)
**case**(*id* $\alpha$ $Q'''$)
**from** *PSimQ* ‹$Q \longmapsto\alpha \prec Q'''$› **have** $\exists P'.\ P \Longrightarrow_l\alpha \prec P' \wedge (P', Q''') \in Rel$
**by**(*blast dest*: *simE*)
**then obtain** $P'''$ **where** *PTrans*: $P \Longrightarrow_l\alpha \prec P'''$ **and** $P'$*Rel*$Q'''$: $(P''', Q''')$ $\in Rel$
**by** *blast*

**have** $\exists P'.\ P''' \Longrightarrow_\tau P' \wedge (P', Q') \in Rel$ **using** ‹$Q''' \Longrightarrow_\tau Q'$› $P'$*Rel*$Q'''$ *Sim*
**by**(*rule Weak-Late-Sim.weakSimTauChain*)
**then obtain** $P'$ **where** $P'''$*Chain*: $P''' \Longrightarrow_\tau P'$ **and** $P'$*Rel*$Q'$: $(P', Q') \in Rel$ **by** *blast*

**from** $P'''$*Chain PTrans* **have** $P \Longrightarrow_l\alpha \prec P'$
**by**(*blast dest*: *Weak-Late-Step-Semantics.chainTransitionAppend*)

**with** $P'$*Rel*$Q'$ **show** *?case* **by** *blast*
**next**
**case**(*ih* $Q''''$ $Q'''$ $\alpha$ $Q''$ $Q'$)
**have** $Q''' \Longrightarrow_\tau Q'''$ **by** *simp*
**with** ‹$Q'''' \longmapsto\tau \prec Q'''$› **obtain** $P'''$ **where** *PTrans*: $P \Longrightarrow_l\tau \prec P'''$ **and** $P'''$*Rel*$Q'''$: $(P''', Q''') \in Rel$
**by**(*drule-tac ih*) *auto*
**from** $P'''$*Rel*$Q'''$ ‹$Q''' \longmapsto\alpha \prec Q''$› **obtain** $P''$ **where**
$P'''$*Trans*: $P''' \Longrightarrow_l\hat{}\alpha \prec P''$ **and** $P''$*Rel*$Q''$: $(P'', Q'') \in Rel$
**by**(*blast dest*: *Weak-Late-Sim.simE Sim*)
**from** $P''$*Rel*$Q''$ ‹$Q'' \Longrightarrow_\tau Q'$› *Sim* **obtain** $P'$ **where**
$P''$*Chain*: $P'' \Longrightarrow_\tau P'$ **and** $P'$*Rel*$Q'$: $(P', Q') \in Rel$
**by**(*drule-tac Weak-Late-Sim.weakSimTauChain*) *auto*

**from** *PTrans* $P'''$*Trans* $P''$*Chain* **have** $P \Longrightarrow_l\alpha \prec P'$
**apply**(*auto simp add*: *weakTransition-def residual.inject*)
**apply**(*drule-tac Weak-Late-Step-Semantics.tauTransitionChain*, *auto*)
**apply**(*drule-tac Weak-Late-Step-Semantics.chainTransitionAppend*, *simp*)
**apply**(*rule Weak-Late-Step-Semantics.chainTransitionAppend*, *auto*)
**by**(*drule-tac Weak-Late-Step-Semantics.chainTransitionAppend*, *auto*)
**with** ‹$(P', Q') \in Rel$› **show** *?case* **by** *blast*
**qed**
**qed**

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

  **assumes** *Id* ⊆ *Rel*

  **shows** $P \leadsto$<*Rel*> $P$
**using** *assms*
**by**(*auto intro*: *Weak-Late-Step-Semantics.singleActionChain simp add*: *weakStepSimDef*)

**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*: $P \leadsto$<*Rel*> $Q$
  **and** *QSimR*: $Q \leadsto$<*Rel'*> $R$
  **and** *Eqvt*: *eqvt Rel*
  **and** *Eqvt'*: *eqvt Rel''*
  **and** *Trans*: *Rel O Rel'* ⊆ *Rel''*
  **and** *Sim*: $\bigwedge P\ Q.\ (P, Q) \in Rel \Longrightarrow P \leadsto\hat{}$<*Rel*> $Q$
  **and** *PRelQ*: $(P, Q) \in Rel$

  **shows** $P \leadsto$<*Rel''*> $R$
**using** *Eqvt'*
**proof**(*induct rule*: *simCasesCont*[*of* - $(P, Q)$])
  **case**(*Bound* $R'$ $a$ $x$)
  **have** *RTrans*: $R \longmapsto a$<$\nu x$> $\prec R'$ **by** *fact*
  **have** $x \sharp (P, Q)$ **by** *fact*
  **hence** *xFreshP*: $x \sharp P$ **and** *xFreshQ*: $x \sharp Q$ **by**(*simp add*: *fresh-prod*)+

  **from** *QSimR RTrans xFreshQ* **obtain** $Q'$ **where** *QTrans*: $Q \Longrightarrow_l a$<$\nu x$> $\prec Q'$
    **and** *Q'RelR'*: $(Q', R') \in Rel'$
    **by**(*blast dest*: *simE*)
  **from** *PSimQ Sim Eqvt PRelQ QTrans xFreshP* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l a$<$\nu x$> $\prec P'$
    **and** *P'RelQ'*: $(P', Q') \in Rel$
    **by**(*blast dest*: *simE2*)
  **moreover from** *P'RelQ' Q'RelR' Trans* **have** $(P', R') \in Rel''$ **by** *blast*
  **ultimately show** *?case* **by** *blast*
**next**
  **case**(*Input* $R'$ $a$ $x$)
  **have** *RTrans*: $R \longmapsto a$<$x$> $\prec R'$ **by** *fact*
  **have** $x \sharp (P, Q)$ **by** *fact*

**hence** *xFreshP*: $x \sharp P$ **and** *xFreshQ*: $x \sharp Q$ **by**(*simp add*: *fresh-prod*)+

**from** *QSimR RTrans xFreshQ* **obtain** $Q''$
**where** $\forall u.\ \exists Q'.\ Q \Longrightarrow_l u\ in\ Q''{\rightarrow}a{<}x{>} \prec Q' \wedge (Q',\ R'[x{::=}u]) \in Rel'$
**by**(*blast dest*: *simE*)
**hence** $\exists Q'''.\ Q \Longrightarrow_\tau Q''' \wedge Q'''{\longmapsto}a{<}x{>} \prec Q'' \wedge (\forall u.\ \exists Q'.\ Q''[x{::=}u]{\Longrightarrow_\tau} Q' \wedge (Q',\ R'[x{::=}u]) \in Rel')$
**by**(*simp add*: *inputTransition-def*, *blast*)
**then obtain** $Q'''$ **where** *QChain*: $Q \Longrightarrow_\tau Q'''$
**and** $Q'''$*Trans*: $Q''' \longmapsto a{<}x{>} \prec Q''$
**and** *L1*: $\forall u.\ \exists Q'.\ Q''[x{::=}u]{\Longrightarrow_\tau} Q' \wedge (Q',\ R'[x{::=}u]) \in Rel'$
**by** *blast*
**from** *QChain PRelQ Sim* **obtain** $P'''$ **where** *PChain*: $P \Longrightarrow_\tau P'''$ **and** $P'''RelQ'''$: $(P''',\ Q''') \in Rel$
**by**(*drule-tac Weak-Late-Sim.weakSimTauChain*) *auto*
**from** *PChain xFreshP* **have** *xFreshP*$'''$: $x \sharp P'''$ **by**(*rule freshChain*)
**from** $P'''RelQ'''$ **have** $P''' \leadsto\hat{}{<}Rel{>}\ Q'''$ **by**(*rule Sim*)
**with** *xFreshP*$'''$ $Q'''$*Trans* **obtain** $P''''$ **where** *L2*: $\forall u.\ \exists P''.\ P''' \Longrightarrow_l u\ in\ P''''{\rightarrow}a{<}x{>} \prec P'' \wedge (P'',\ Q''[x{::=}u]) \in Rel$
**by**(*blast dest*: *Weak-Late-Sim.simE*)
**have** $\forall u.\ \exists P'\ Q'.\ P \Longrightarrow_l u\ in\ P''''{\rightarrow}a{<}x{>} \prec P' \wedge (P',\ R'[x{::=}u]) \in Rel''$
**proof**(*rule allI*)
**fix** $u$
**from** *L1* **obtain** $Q'$ **where** $Q''$*Chain*: $Q''[x{::=}u] \Longrightarrow_\tau Q'$ **and** $Q'RelR'$: $(Q',\ R'[x{::=}u]) \in Rel'$
**by** *blast*
**from** *L2* **obtain** $P''$ **where** $P'''$*Trans*: $P''' \Longrightarrow_l u\ in\ P''''{\rightarrow}a{<}x{>} \prec P''$
**and** $P''RelQ''$: $(P'',\ Q''[x{::=}u]) \in Rel$
**by** *blast*
**from** $P''RelQ''$ **have** $P'' \leadsto\hat{}{<}Rel{>}\ Q''[x{::=}u]$ **by**(*rule Sim*)
**have** $\exists P'.\ P'' \Longrightarrow_\tau P' \wedge (P',\ Q') \in Rel$ **using** $Q''$*Chain* $P''RelQ''$ *Sim*
**by**(*rule Weak-Late-Sim.weakSimTauChain*)
**then obtain** $P'$ **where** $P''$*Chain*: $P'' \Longrightarrow_\tau P'$ **and** $P'RelQ'$: $(P',\ Q') \in Rel$ **by** *blast*
**from** *PChain* $P'''$*Trans* $P''$*Chain* **have** $P \Longrightarrow_l u\ in\ P''''{\rightarrow}a{<}x{>} \prec P'$
**by**(*blast dest*: *Weak-Late-Step-Semantics.chainTransitionAppend*)
**moreover from** $P'RelQ'$ $Q'RelR'$ **have** $(P',\ R'[x{::=}u]) \in Rel''$ **by**(*insert Trans*, *auto*)
**ultimately show** $\exists P'\ Q'.\ P \Longrightarrow_l u\ in\ P''''{\rightarrow}a{<}x{>} \prec P' \wedge (P',\ R'[x{::=}u]) \in Rel''$ **by** *blast*
**qed**
**thus** *?case* **by** *force*
**next**
**case**(*Free* $R'$ $\alpha$)
**have** *RTrans*: $R \longmapsto \alpha \prec R'$ **by** *fact*
**with** *QSimR* **obtain** $Q'$ **where** *QTrans*: $Q \Longrightarrow_l \alpha \prec Q'$ **and** $Q'RelR'$: $(Q',\ R') \in Rel'$
**by**(*blast dest*: *simE*)
**from** *PSimQ Sim Eqvt PRelQ QTrans* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l \alpha \prec P'$

**and** $P'RelQ'$: $(P', Q') \in Rel$
**by**(*blast dest*: *simE2*)
**from** $P'RelQ'$ $Q'RelR'$ *Trans* **have** $(P', R') \in Rel''$ **by** *blast*
**with** *PTrans* **show** *?case* **by** *blast*
**qed**

**end**

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

**definition** *congruence* :: $(pi \times pi)$ *set* **where**
$congruence \equiv \{(P, Q) \mid P\ Q.\ P \leadsto<weakBisim> Q \wedge Q \leadsto<weakBisim> P\}$
**abbreviation** *congruenceJudge* (**infixr** ‹≃› *65*) **where** $P \simeq Q \equiv (P, Q) \in congruence$

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

**assumes** $P \simeq Q$

**shows** $P \leadsto<weakBisim> Q$
**and** $Q \leadsto<weakBisim> P$
**proof** −
**from** *assms* **show** $P \leadsto<weakBisim> Q$ **by**(*force simp add*: *congruence-def*)
**next**
**from** *assms* **show** $Q \leadsto<weakBisim> P$ **by**(*force simp add*: *congruence-def*)
**qed**

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

**assumes** $P \leadsto<weakBisim> Q$
**and** $Q \leadsto<weakBisim> P$

**shows** $P \simeq Q$
**using** *assms* **by**(*force simp add*: *congruence-def*)

**lemma** *eqvt*:
**shows** *eqvt congruence*
**proof** −
**have** $\bigwedge P\ Q\ (perm::name\ prm).\ P \leadsto<weakBisim> Q \Longrightarrow (perm \cdot P) \leadsto<weakBisim> (perm \cdot Q)$
**proof** −

**fix** *P Q perm*
**assume** $P \rightsquigarrow<weakBisim> Q$
**thus** $((perm::name\ prm) \cdot P) \rightsquigarrow<weakBisim> (perm \cdot Q)$
**apply** −
**by**(*blast intro*: *Weak-Late-Step-Sim.eqvtI Weak-Late-Bisim.eqvt*)
**qed**
**thus** *?thesis*
**by**(*simp add*: *congruence-def eqvt-def*)
**qed**

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

**assumes** $P \simeq Q$

**shows** $(perm \cdot P) \simeq (perm \cdot Q)$
**using** *assms*
**by**(*rule eqvtRelI*[*OF eqvt*])

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

**assumes** $P \sim Q$

**shows** $P \simeq Q$
**proof** −
**have** $\bigwedge P\ Q.\ P \rightsquigarrow[bisim]\ Q \Longrightarrow P \rightsquigarrow<weakBisim> Q$
**proof** −
**fix** *P Q*
**assume** $P \rightsquigarrow[bisim]\ Q$
**hence** $P \rightsquigarrow<bisim> Q$ **by**(*rule strongSimWeakEqSim*)
**moreover have** $bisim \subseteq weakBisim$
**by**(*auto intro*: *strongBisimWeakBisim*)
**ultimately show** $P \rightsquigarrow<weakBisim> Q$ **by**(*rule Weak-Late-Step-Sim.monotonic*)
**qed**
**with** *assms* **show** *?thesis*
**by**(*blast intro*: *unfoldI dest*: *Strong-Late-Bisim.bisimE Strong-Late-Bisim.symmetric*)
**qed**

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

**assumes** $P \simeq Q$

**shows** $P \approx Q$

```
proof -
  let ?X = {(P, Q) | P Q. P ≃ Q}
  from assms have (P, Q) ∈ ?X by auto
  thus ?thesis
  proof(coinduct rule: weakBisimCoinduct)
    case(cSim P Q)
    {
      fix P Q
      assume P ≃ Q
      hence P ⇝<weakBisim> Q by(simp add: congruence-def)
     hence P ⇝<(?X ∪ weakBisim)> Q by(rule-tac Weak-Late-Step-Sim.monotonic)
auto
     hence P ⇝^<(?X ∪ weakBisim)> Q by(rule Weak-Late-Step-Sim.weakSimWeakEqSim)
    }
    with ‹(P, Q) ∈ ?X› show ?case by auto
  next
    case(cSym P Q)
    thus ?case by(auto simp add: congruence-def)
  qed
qed

lemma congruenceSubsetWeakBisim:
  shows congruence ⊆ weakBisim
by(auto intro: congruenceWeakBisim)

lemma reflexive:
  fixes P :: pi

  shows P ≃ P
proof -
  from Weak-Late-Bisim.reflexive have ⋀P. P ⇝<weakBisim> P
    by(blast intro: Weak-Late-Step-Sim.reflexive)
  thus ?thesis
    by(force simp add: substClosed-def congruence-def)
qed

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

  assumes P ≃ Q

  shows Q ≃ P
using assms
by(force simp add: substClosed-def congruence-def)

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

**and** $R :: pi$

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

**shows** $P \simeq R$
**proof** −
**have** *Goal*: $\bigwedge P\ Q\ R.\ [\![P \rightsquigarrow<weakBisim>\ Q;\ Q \rightsquigarrow<weakBisim>\ R;\ P \approx Q]\!] \Longrightarrow P \rightsquigarrow<weakBisim>\ R$
**using** *Weak-Late-Bisim.eqvt Weak-Late-Bisim.unfoldE Weak-Late-Bisim.transitive*
**by**(*blast intro*: *Weak-Late-Step-Sim.transitive*)
**from** *assms* **show** *?thesis*
**apply**(*simp add*: *congruence-def*) **using** *assms*
**by**(*blast intro*: *Goal dest*: *congruenceWeakBisim symetric*)
**qed**

**end**

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

**consts** *weakBisimSubst* :: $(pi \times pi)$ *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** −
**from** *assms substClosedSubset* **show** *?thesis*
**by** *blast*
**qed**

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

**assumes** $P \sim^s Q$

**shows** $P \approx^s Q$
**using** *assms*
**by**(*auto simp add*: *substClosed-def intro*: *strongBisimWeakBisim*)

**lemma** *eqvt*:
  **shows** *eqvt* (*substClosed weakBisim*)
**by**(*rule eqvtSubstClosed*[*OF Weak-Late-Bisim.eqvt*])

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

  **assumes** $P \approx^s Q$

  **shows** $(perm \cdot P) \approx^s (perm \cdot Q)$
**using** *assms*
**by**(*rule-tac eqvtRelI*[*OF eqvt*])

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

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

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

  **assumes** $P \approx^s Q$

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

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

  **assumes** $P \approx^s Q$
  **and** $Q \approx^s R$

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

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

**assumes** $P \approx^s Q$

**shows** $P[<s>] \approx^s Q[<s>]$
**using** *assms*
**proof**(*auto simp add*: *substClosed-def*)
**fix** $s'$
**assume** $\forall s.\ P[<s>] \approx Q[<s>]$
**hence** $P[<(s@s')>] \approx Q[<(s@s')>]$ **by** *blast*
**moreover have** $P[<(s@s')>] = (P[<s>])[<s'>]$
**by**(*induct* $s'$, *auto*)
**moreover have** $Q[<(s@s')>] = (Q[<s>])[<s'>]$
**by**(*induct* $s'$, *auto*)

**ultimately show** $(P[<s>])[<s'>] \approx (Q[<s>])[<s'>]$
**by** *simp*
**qed**

**end**

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

**definition** *congruenceSubst* :: *pi* $\Rightarrow$ *pi* $\Rightarrow$ *bool* (**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* $\times$ *name*) *list*

**assumes** $P \simeq^s Q$

**shows** $P[<s>] \leadsto<weakBisim> Q[<s>]$
**and** $Q[<s>] \leadsto<weakBisim> P[<s>]$
**proof** −
**from** *assms* **show** $P[<s>] \leadsto<weakBisim> Q[<s>]$ **by**(*force simp add*: *congruenceSubstDef*)
**next**
**from** *assms* **show** $Q[<s>] \leadsto<weakBisim> P[<s>]$ **by**(*force simp add*: *congruenceSubstDef*)
**qed**

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

**assumes** $\forall s.\ P[<s>] \leadsto<weakBisim>\ Q[<s>] \wedge Q[<s>] \leadsto<weakBisim>\ P[<s>]$

**shows** $P \simeq^s Q$
**proof** −
**from** *assms* **show** *?thesis* **by**(*force simp add*: *congruenceSubstDef*)
**qed**

**lemma** *weakEqSubset*:
**shows** *substClosed congruence* $\subseteq$ *weakBisim*
**proof**(*auto simp add*: *substClosed-def*)
**fix** $P$ $Q$
**assume** $\forall s.\ P[<s>] \simeq Q[<s>]$
**hence** $P[<[]>] \simeq Q[<[]>]$ **by** *blast*
**thus** $P \approx Q$
**by**(*force dest*: *congruenceWeakBisim intro*: *Weak-Late-Bisim.unfoldI*)
**qed**

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

**assumes** $P \simeq^s Q$

**shows** $P \simeq Q$
**using** *assms*
**apply**(*auto simp add*: *substClosed-def congruenceSubst-def*)
**apply**(*erule-tac x*=[] **in** *allE*)
**by** *auto*

**lemma** *eqvt*:
**shows** *eqvt* (*substClosed congruence*)
**by**(*rule eqvtSubstClosed*[*OF Weak-Late-Cong.eqvt*])

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

**assumes** $P \simeq^s Q$

**shows** $(perm \cdot P) \simeq^s (perm \cdot Q)$
**using** *assms*
**by**(*simp add*: *congruenceSubst-def*) (*rule eqvtRelI*[*OF eqvt*])

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

**assumes** $P \sim^s Q$

**shows** $P \simeq^s Q$
**using** *assms*
**by**(*force intro*: *strongBisimWeakEq simp add*: *substClosed-def congruenceSubst-def*)

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

**assumes** $P \simeq^s Q$

**shows** $P \approx^s Q$
**using** *assms*
**by**(*force simp add*: *congruenceSubst-def substClosed-def intro*: *congruenceWeakBisim*)

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

**shows** $P \simeq^s P$
**proof** −
**from** *Weak-Late-Bisim.reflexive* **have** $\bigwedge P.\ P \rightsquigarrow$<*weakBisim*> $P$
**by**(*blast intro*: *Weak-Late-Step-Sim.reflexive*)
**thus** *?thesis*
**by**(*force simp add*: *congruenceSubstDef*)
**qed**

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

**assumes** $P \simeq^s Q$

**shows** $Q \simeq^s P$
**using** *assms*
**by**(*force simp add*: *congruenceSubstDef*)

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

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

**shows** $P \simeq^s R$
**using** *assms*

**by**(*force simp add*: *congruenceSubst-def substClosed-def intro*: *Weak-Late-Cong.transitive*)

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

  **assumes** $P \simeq^s Q$

  **shows** $P[<s>] \simeq^s Q[<s>]$
**using** *assms*
**proof**(*auto simp add*: *congruenceSubst-def substClosed-def*)
  **fix** $s'$
  **assume** $\forall s.\ (P[<s>],\ Q[<s>]) \in congruence$
  **hence** $(P[<(s@s')>],\ Q[<(s@s')>]) \in congruence$ **by** *blast*
  **moreover have** $P[<(s@s')>] = (P[<s>])[<s'>]$
    **by**(*induct* $s'$, *auto*)
  **moreover have** $Q[<(s@s')>] = (Q[<s>])[<s'>]$
    **by**(*induct* $s'$, *auto*)

  **ultimately show** $((P[<s>])[<s'>],\ (Q[<s>])[<s'>]) \in congruence$
    **by** *simp*
**qed**

**end**

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

**lemma** *nilSim*[*dest*]:
  **fixes** $a$ :: *name*
  **and** $b$ :: *name*
  **and** $x$ :: *name*
  **and** $P$ :: *pi*
  **and** $Q$ :: *pi*

  **shows** $\mathbf{0} \leadsto[Rel]\ \tau.(P) \Longrightarrow False$
  **and** $\mathbf{0} \leadsto[Rel]\ a<x>.P \Longrightarrow False$
  **and** $\mathbf{0} \leadsto[Rel]\ a\{b\}.P \Longrightarrow False$
**by**(*fastforce simp add*: *simulation-def intro*: *Tau Input Output*)+

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

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

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

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

  **assumes** $Id \subseteq Rel$

  **shows** $[a \frown a]P \leadsto[Rel]\ P$
**using** *assms*
**by**(*force simp add*: *simulation-def dest*: *Match derivativeReflexive*)

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

  **assumes** *IdRel*: $Id \subseteq Rel$

  **shows** $P \leadsto[Rel]\ [a \frown a]P$
**using** *assms*
**by**(*fastforce simp add*: *simulation-def elim*: *matchCases intro*: *derivativeReflexive*)

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

  **assumes** $a \neq b$

  **shows** $\mathbf{0} \leadsto[Rel]\ [a \frown b]P$
**using** *assms*
**by**(*auto simp add*: *simulation-def*)

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

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

  **shows** $[a \neq b]P \leadsto[Rel]\ P$

**using** *assms*
**by**(*fastforce simp add*: *simulation-def intro*: *Mismatch dest*: *derivativeReflexive*)

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

  **assumes** *IdRel*: *Id* ⊆ *Rel*
  **and** *aineqb*: $a \neq b$

  **shows** $P \leadsto[Rel]\ [a{\neq}b]P$
**using** *assms*
**by**(*fastforce simp add*: *simulation-def elim*: *mismatchCases intro*: *derivativeReflexive*)

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

  **shows** $\mathbf{0} \leadsto[Rel]\ [a{\neq}a]P$
**by**(*auto simp add*: *simulation-def*)

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

  **assumes** *Id*: *Id* ⊆ *Rel*

  **shows** $P \oplus Q \leadsto[Rel]\ Q \oplus P$
**using** *assms*
**by**(*fastforce simp add*: *simulation-def elim*: *sumCases intro*: *Sum1 Sum2 derivativeReflexive*)

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

  **assumes** *Id* ⊆ *Rel*

  **shows** $P \leadsto[Rel]\ P \oplus P$
**using** *assms*
**by**(*fastforce simp add*: *simulation-def elim*: *sumCases intro*: *derivativeReflexive*)

**lemma** *sumIdempRight*:

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

**assumes** $I$: *Id* $\subseteq$ *Rel*

**shows** $P \oplus P \leadsto$[*Rel*] $P$
**using** *assms*
**by**(*fastforce simp add*: *simulation-def intro*: *Sum1 derivativeReflexive*)

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

**assumes** *Id*: *Id* $\subseteq$ *Rel*

**shows** $(P \oplus Q) \oplus R \leadsto$[*Rel*] $P \oplus (Q \oplus R)$
**using** *assms*
**by**(*fastforce simp add*: *simulation-def elim*: *sumCases intro*: *Sum1 Sum2 derivativeReflexive*)

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

**assumes** *Id*: *Id* $\subseteq$ *Rel*

**shows** $P \oplus (Q \oplus R) \leadsto$[*Rel*] $(P \oplus Q) \oplus R$
**using** *assms*
**by**(*fastforce simp add*: *simulation-def elim*: *sumCases intro*: *Sum1 Sum2 derivativeReflexive*)

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

**assumes** *Id*: *Id* $\subseteq$ *Rel*

**shows** $P \oplus \mathbf{0} \leadsto$[*Rel*] $P$
**using** *assms*
**by**(*fastforce simp add*: *simulation-def intro*: *Sum1 derivativeReflexive*)

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

**assumes** *Id*: *Id* ⊆ *Rel*

**shows** *P* ⇝[*Rel*] *P* ⊕ **0**
**using** *assms*
**by**(*fastforce simp add*: *simulation-def elim*: *sumCases intro*: *derivativeReflexive*)

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

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

**shows** (<*νx*>*P*) ⊕ (<*νx*>*Q*) ⇝[*Rel*] <*νx*>(*P* ⊕ *Q*)
**using** *Eqvt*
**proof**(*induct rule*: *simCasesCont*[**where** *C*=(*x*, *P*, *Q*)])
**case**(*Bound a y PQ*)
**from** ‹*y* ♯ (*x*, *P*, *Q*)› **have** *y* ≠ *x* **and** *y* ♯ *P* **and** *y* ♯ *Q* **by**(*simp add*: *fresh-prod*)+
**hence** *y* ♯ *P* ⊕ *Q* **by** *simp*
**with** ‹<*νx*>(*P* ⊕ *Q*) ⟼*a*«*y*» ≺ *PQ*› ‹*y* ≠ *x*› **show** *?case*
**proof**(*induct rule*: *resCasesB*)
**case**(*cOpen a PQ*)
**from** ‹*P* ⊕ *Q* ⟼*a*[*x*] ≺ *PQ*› ‹*y* ♯ *P*› ‹*y* ♯ *Q*› **have** *y* ♯ *PQ* **by**(*force dest*: *freshFreeDerivative*)
**from** ‹*P* ⊕ *Q* ⟼*a*[*x*] ≺ *PQ*› **show** *?case*
**proof**(*induct rule*: *sumCases*)
**case** *cSum1*
**from** ‹*P* ⟼*a*[*x*] ≺ *PQ*› ‹*a* ≠ *x*› **have** <*νx*>*P* ⟼*a*<*νx*> ≺ *PQ* **by**(*rule Open*)
**hence** (<*νx*>*P*) ⊕ (<*νx*>*Q*) ⟼*a*<*νx*> ≺ *PQ* **by**(*rule Sum1*)
**with** ‹*y* ♯ *PQ*› **have** (<*νx*>*P*) ⊕ (<*νx*>*Q*) ⟼*a*<*νy*> ≺ ([(*y*, *x*)] • *PQ*)
**by**(*simp add*: *alphaBoundResidual*)
**moreover from** *Id* **have** *derivative* ([(*y*, *x*)] • *PQ*) ([(*y*, *x*)] • *PQ*) (*BoundOutputS a*) *y Rel*
**by**(*force simp add*: *derivative-def*)
**ultimately show** *?case* **by** *blast*
**next**
**case** *cSum2*
**from** ‹*Q* ⟼*a*[*x*] ≺ *PQ*› ‹*a* ≠ *x*› **have** <*νx*>*Q* ⟼*a*<*νx*> ≺ *PQ* **by**(*rule Open*)
**hence** (<*νx*>*P*) ⊕ (<*νx*>*Q*) ⟼*a*<*νx*> ≺ *PQ* **by**(*rule Sum2*)
**with** ‹*y* ♯ *PQ*› **have** (<*νx*>*P*) ⊕ (<*νx*>*Q*) ⟼*a*<*νy*> ≺ ([(*y*, *x*)] • *PQ*)
**by**(*simp add*: *alphaBoundResidual*)
**moreover from** *Id* **have** *derivative* ([(*y*, *x*)] • *PQ*) ([(*y*, *x*)] • *PQ*) (*BoundOutputS a*) *y Rel*
**by**(*force simp add*: *derivative-def*)
**ultimately show** *?case* **by** *blast*
**qed**

**next**
**case**(*cRes PQ*)
**from** $\langle P \oplus Q \longmapsto a«y» \prec PQ\rangle$ **show** *?case*
**proof**(*induct rule*: *sumCases*)
**case** *cSum1*
**from** $\langle P \longmapsto a«y» \prec PQ\rangle$ $\langle x \sharp a\rangle$ $\langle y \neq x\rangle$ **have** $<\nu x>P \longmapsto a«y» \prec <\nu x>PQ$
**by**(*rule-tac ResB*) *auto*
**hence** $(<\nu x>P) \oplus (<\nu x>Q) \longmapsto a«y» \prec <\nu x>PQ$ **by**(*rule Sum1*)
**moreover from** *Id* **have** *derivative* $(<\nu x>PQ)$ $(<\nu x>PQ)$ *a y Rel*
**by**(*cases a*) (*auto simp add*: *derivative-def*)
**ultimately show** *?case* **by** *blast*
**next**
**case** *cSum2*
**from** $\langle Q \longmapsto a«y» \prec PQ\rangle$ $\langle x \sharp a\rangle$ $\langle y \neq x\rangle$ **have** $<\nu x>Q \longmapsto a«y» \prec <\nu x>PQ$
**by**(*rule-tac ResB*) *auto*
**hence** $(<\nu x>P) \oplus (<\nu x>Q) \longmapsto a«y» \prec <\nu x>PQ$ **by**(*rule Sum2*)
**moreover from** *Id* **have** *derivative* $(<\nu x>PQ)$ $(<\nu x>PQ)$ *a y Rel*
**by**(*cases a*) (*auto simp add*: *derivative-def*)
**ultimately show** *?case* **by** *blast*
**qed**
**qed**
**next**
**case**(*Free* $\alpha$ *PQ*)
**from** $\langle <\nu x>(P \oplus Q) \longmapsto \alpha \prec PQ\rangle$ **show** *?case*
**proof**(*induct rule*: *resCasesF*)
**case**(*cRes PQ*)
**from** $\langle P \oplus Q \longmapsto \alpha \prec PQ\rangle$ **show** *?case*
**proof**(*induct rule*: *sumCases*)
**case** *cSum1*
**from** $\langle P \longmapsto \alpha \prec PQ\rangle$ $\langle x \sharp \alpha\rangle$ **have** $<\nu x>P \longmapsto \alpha \prec <\nu x>PQ$ **by**(*rule ResF*)
**hence** $(<\nu x>P) \oplus (<\nu x>Q) \longmapsto \alpha \prec <\nu x>PQ$ **by**(*rule Sum1*)
**with** *Id* **show** *?case* **by** *blast*
**next**
**case** *cSum2*
**from** $\langle Q \longmapsto \alpha \prec PQ\rangle$ $\langle x \sharp \alpha\rangle$ **have** $<\nu x>Q \longmapsto \alpha \prec <\nu x>PQ$ **by**(*rule ResF*)
**hence** $(<\nu x>P) \oplus (<\nu x>Q) \longmapsto \alpha \prec <\nu x>PQ$ **by**(*rule Sum2*)
**with** *Id* **show** *?case* **by** *blast*
**qed**
**qed**
**qed**

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

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

```
  shows <νx>(P ⊕ Q) ↝[Rel] (<νx>P) ⊕ (<νx>Q)
using ‹eqvt Rel›
proof(induct rule: simCasesCont[where C=(x, P, Q)])
  case(Bound a y PQ)
  from ‹y ♯ (x, P, Q)› have y ≠ x and y ♯ P and y ♯ Q by(simp add: fresh-prod)+
  from ‹(<νx>P) ⊕ (<νx>Q) ⟼a«y» ≺ PQ› show ?case
  proof(induct rule: sumCases)
    case cSum1
    from ‹<νx>P ⟼a«y» ≺ PQ› show ?case using ‹y ≠ x› ‹y ♯ P›
    proof(induct rule: resCasesB)
      case(cOpen a P')
      from ‹P ⟼a[x] ≺ P'› ‹y ♯ P› have y ♯ P' by(rule freshFreeDerivative)

      from ‹P ⟼a[x] ≺ P'› have P ⊕ Q ⟼a[x] ≺ P' by(rule Sum1)
      hence <νx>(P ⊕ Q) ⟼a<νx> ≺ P' using ‹a ≠ x› by(rule Open)
      with ‹y ♯ P'› have <νx>(P ⊕ Q) ⟼a<νy> ≺ [(y, x)] • P' by(simp add:
alphaBoundResidual)
      moreover from Id have derivative ([(y, x)] • P') ([(y, x)] • P') (BoundOutputS
a) y Rel
        by(force simp add: derivative-def)
      ultimately show ?case by blast
    next
      case(cRes P')
      from ‹P ⟼a«y» ≺ P'› have P ⊕ Q ⟼a«y» ≺ P' by(rule Sum1)
      hence <νx>(P ⊕ Q) ⟼a«y» ≺ <νx>P' using ‹x ♯ a› ‹y ≠ x› by(rule-tac
ResB) auto
      moreover from Id have derivative (<νx>P') (<νx>P') a y Rel
        by(cases a) (auto simp add: derivative-def)
      ultimately show ?case by blast
    qed
  next
    case cSum2
    from ‹<νx>Q ⟼a«y» ≺ PQ› show ?case using ‹y ≠ x› ‹y ♯ Q›
    proof(induct rule: resCasesB)
      case(cOpen a Q')
      from ‹Q ⟼a[x] ≺ Q'› ‹y ♯ Q› have y ♯ Q' by(rule freshFreeDerivative)

      from ‹Q ⟼a[x] ≺ Q'› have P ⊕ Q ⟼a[x] ≺ Q' by(rule Sum2)
      hence <νx>(P ⊕ Q) ⟼a<νx> ≺ Q' using ‹a ≠ x› by(rule Open)
      with ‹y ♯ Q'› have <νx>(P ⊕ Q) ⟼a<νy> ≺ [(y, x)] • Q' by(simp add:
alphaBoundResidual)
      moreover from Id have derivative ([(y, x)] • Q') ([(y, x)] • Q') (BoundOutputS
a) y Rel
        by(force simp add: derivative-def)
      ultimately show ?case by blast
    next
      case(cRes Q')
      from ‹Q ⟼a«y» ≺ Q'› have P ⊕ Q ⟼a«y» ≺ Q' by(rule Sum2)
      hence <νx>(P ⊕ Q) ⟼a«y» ≺ <νx>Q' using ‹x ♯ a› ‹y ≠ x› by(rule-tac
```

```
ResB) auto
      moreover from Id have derivative (<νx>Q′) (<νx>Q′) a y Rel
        by(cases a) (auto simp add: derivative-def)
      ultimately show ?case by blast
    qed
  qed
next
  case(Free α PQ)
  from ‹(<νx>P) ⊕ (<νx>Q) ⟼α ≺ PQ› show ?case
  proof(induct rule: sumCases)
    case cSum1
    from ‹<νx>P ⟼α ≺ PQ› show ?case
    proof(induct rule: resCasesF)
      case(cRes P′)
      from ‹P ⟼α ≺ P′› have P ⊕ Q ⟼α ≺ P′ by(rule Sum1)
      hence <νx>(P ⊕ Q) ⟼α ≺ <νx>P′ using ‹x ♯ α› by(rule ResF)
      with Id show ?case by blast
    qed
  next
    case cSum2
    from ‹<νx>Q ⟼α ≺ PQ› show ?case
    proof(induct rule: resCasesF)
      case(cRes Q′)
      from ‹Q ⟼α ≺ Q′› have P ⊕ Q ⟼α ≺ Q′ by(rule Sum2)
      hence <νx>(P ⊕ Q) ⟼α ≺ <νx>Q′ using ‹x ♯ α› by(rule ResF)
      with Id show ?case by blast
    qed
  qed
qed


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

  assumes ParZero: ⋀Q. (Q ∥ 0, Q) ∈ Rel

  shows P ∥ 0 ⇝[Rel] P
proof -
  {
    fix P Q a x
    from ParZero have derivative (P ∥ 0) P a x Rel
      by(case-tac a) (auto simp add: derivative-def)
  }
  thus ?thesis using assms
    by(fastforce simp add: simulation-def intro: Par1B Par1F)
qed
```

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

  assumes ParZero: ⋀Q. (Q, Q ∥ 0) ∈ Rel

  shows P ⇝[Rel] P ∥ 0
proof −
  {
    fix P Q a x
    from ParZero have derivative P (P ∥ 0) a x Rel
      by(case-tac a) (auto simp add: derivative-def)
  }
  thus ?thesis using assms
    by(fastforce simp add: simulation-def elim: parCasesF parCasesB)+
qed

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

  assumes Sym: ⋀R S. (R ∥ S, S ∥ R) ∈ Rel
  and     Res: ⋀R S x. (R, S) ∈ Rel ⟹ (<νx>R, <νx>S) ∈ Rel

  shows P ∥ Q ⇝[Rel] Q ∥ P
proof(induct rule: simCases)
  case(Bound a x QP)
  from ‹x ♯ (P ∥ Q)› have x ♯ Q and x ♯ P by simp+
  with ‹Q ∥ P ⟼ a«x» ≺ QP› show ?case
  proof(induct rule: parCasesB)
    case(cPar1 Q')
   from ‹Q ⟼a«x» ≺ Q'› have P ∥ Q ⟼a«x» ≺ P ∥ Q' using ‹x ♯ P› by(rule
Par2B)
    moreover have derivative (P ∥ Q') (Q' ∥ P) a x Rel
      by(cases a, auto simp add: derivative-def intro: Sym)
    ultimately show ?case by blast
  next
    case(cPar2 P')
   from ‹P ⟼a«x» ≺ P'› have P ∥ Q ⟼a«x» ≺ P' ∥ Q using ‹x ♯ Q› by(rule
Par1B)
    moreover have derivative (P' ∥ Q) (Q ∥ P') a x Rel
      by(cases a, auto simp add: derivative-def intro: Sym)
    ultimately show ?case by blast
  qed
next
  case(Free α QP)
  from ‹Q ∥ P ⟼ α ≺ QP› show ?case
  proof(induct rule: parCasesF[where C=()])
```

**case**(*cPar1 Q′*)
**from** ‹$Q \longmapsto \alpha \prec Q'$› **have** $P \parallel Q \longmapsto \alpha \prec P \parallel Q'$ **by**(*rule Par2F*)
**moreover have** $(P \parallel Q', Q' \parallel P) \in Rel$ **by**(*rule Sym*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*cPar2 P′*)
**from** ‹$P \longmapsto \alpha \prec P'$› **have** $P \parallel Q \longmapsto \alpha \prec P' \parallel Q$ **by**(*rule Par1F*)
**moreover have** $(P' \parallel Q, Q \parallel P') \in Rel$ **by**(*rule Sym*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*cComm1 Q′ P′ a b x*)
**from** ‹$P \longmapsto a[b] \prec P'$› ‹$Q \longmapsto a{<}x{>} \prec Q'$›
**have** $P \parallel Q \longmapsto \tau \prec P' \parallel (Q'[x{::=}b])$ **by**(*rule Comm2*)
**moreover have** $(P' \parallel Q'[x{::=}b], Q'[x{::=}b] \parallel P') \in Rel$ **by**(*rule Sym*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*cComm2 Q′ P′ a b x*)
**from** ‹$P \longmapsto a{<}x{>} \prec P'$› ‹$Q \longmapsto a[b] \prec Q'$›
**have** $P \parallel Q \longmapsto \tau \prec (P'[x{::=}b]) \parallel Q'$ **by**(*rule Comm1*)
**moreover have** $(P'[x{::=}b] \parallel Q', Q' \parallel P'[x{::=}b]) \in Rel$ **by**(*rule Sym*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*cClose1 Q′ P′ a x y*)
**from** ‹$P \longmapsto a{<}\nu y{>} \prec P'$› ‹$Q \longmapsto a{<}x{>} \prec Q'$› ‹$y \sharp Q$›
**have** $P \parallel Q \longmapsto \tau \prec {<}\nu y{>}(P' \parallel (Q'[x{::=}y]))$ **by**(*rule Close2*)
**moreover have** $({<}\nu y{>}(P' \parallel Q'[x{::=}y]), {<}\nu y{>}(Q'[x{::=}y] \parallel P')) \in Rel$ **by**(*metis Res Sym*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*cClose2 Q′ P′ a x y*)
**from** ‹$P \longmapsto a{<}x{>} \prec P'$› ‹$Q \longmapsto a{<}\nu y{>} \prec Q'$› ‹$y \sharp P$›
**have** $P \parallel Q \longmapsto \tau \prec {<}\nu y{>}((P'[x{::=}y]) \parallel Q')$ **by**(*rule Close1*)
**moreover have** $({<}\nu y{>}(P'[x{::=}y] \parallel Q'), {<}\nu y{>}(Q' \parallel P'[x{::=}y])) \in Rel$ **by**(*metis Res Sym*)
**ultimately show** *?case* **by** *blast*
**qed**
**qed**

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

**assumes** *Ass*: $\bigwedge S\ T\ U.\ ((S \parallel T) \parallel U, S \parallel (T \parallel U)) \in Rel$
**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** $(P \parallel Q) \parallel R \leadsto[Rel]\ P \parallel (Q \parallel R)$
**proof**(*induct rule*: *simCases*)
**case**(*Bound a x PQR*)
**from** ‹$x \sharp (P \parallel Q) \parallel R$› **have** $x \sharp P$ **and** $x \sharp Q$ **and** $x \sharp R$ **by** *simp+*
**hence** $x \sharp (Q \parallel R)$ **by** *simp*
**with** ‹$P \parallel (Q \parallel R) \longmapsto a«x» \prec PQR$› ‹$x \sharp P$› **show** *?case*
**proof**(*induct rule*: *parCasesB*)
**case**(*cPar1 P'*)
**from** ‹$P \longmapsto a«x» \prec P'$› **have** $P \parallel Q \longmapsto a«x» \prec P' \parallel Q$ **using** ‹$x \sharp Q$›
**by**(*rule Par1B*)
**hence** $(P \parallel Q) \parallel R \longmapsto a«x» \prec (P' \parallel Q) \parallel R$ **using** ‹$x \sharp R$› **by**(*rule Par1B*)
**moreover have** *derivative* $((P' \parallel Q) \parallel R)\ (P' \parallel (Q \parallel R))\ a\ x\ Rel$
**by**(*cases a*, *auto intro*: *Ass simp add*: *derivative-def*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*cPar2 QR*)
**from** ‹$Q \parallel R \longmapsto a«x» \prec QR$› ‹$x \sharp Q$› ‹$x \sharp R$› **show** *?case*
**proof**(*induct rule*: *parCasesB*)
**case**(*cPar1 Q'*)
**from** ‹$Q \longmapsto a«x» \prec Q'$› **have** $P \parallel Q \longmapsto a«x» \prec P \parallel Q'$ **using** ‹$x \sharp P$›
**by**(*rule Par2B*)
**hence** $(P \parallel Q) \parallel R \longmapsto a«x» \prec (P \parallel Q') \parallel R$ **using** ‹$x \sharp R$›**by**(*rule Par1B*)
**moreover have** *derivative* $((P \parallel Q') \parallel R)\ (P \parallel (Q' \parallel R))\ a\ x\ Rel$
**by**(*cases a*, *auto intro*: *Ass simp add*: *derivative-def*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*cPar2 R'*)
**from** ‹$R \longmapsto a«x» \prec R'$› **have** $(P \parallel Q) \parallel R \longmapsto a«x» \prec (P \parallel Q) \parallel R'$ **using**
‹$x \sharp P$› ‹$x \sharp Q$›
**by**(*rule-tac Par2B*) *auto*
**moreover have** *derivative* $((P \parallel Q) \parallel R')\ (P \parallel (Q \parallel R'))\ a\ x\ Rel$
**by**(*cases a*, *auto intro*: *Ass simp add*: *derivative-def*)
**ultimately show** *?case* **by** *blast*
**qed**
**qed**
**next**
**case**(*Free* $\alpha$ *PQR*)
**from** ‹$P \parallel (Q \parallel R) \longmapsto \alpha \prec PQR$› **show** *?case*
**proof**(*induct rule*: *parCasesF*[**where** $C=Q$])
**case**(*cPar1 P'*)
**from** ‹$P \longmapsto \alpha \prec P'$› **have** $P \parallel Q \longmapsto \alpha \prec P' \parallel Q$ **by**(*rule Par1F*)
**hence** $(P \parallel Q) \parallel R \longmapsto \alpha \prec (P' \parallel Q) \parallel R$ **by**(*rule Par1F*)
**moreover from** *Ass* **have** $((P' \parallel Q) \parallel R,\ P' \parallel (Q \parallel R)) \in Rel$ **by** *blast*
**ultimately show** *?case* **by** *blast*
**next**
**case**(*cPar2 QR*)
**from** ‹$Q \parallel R \longmapsto \alpha \prec QR$› **show** *?case*

**proof**(*induct rule*: *parCasesF*[**where** $C$=$P$])
**case**(*cPar1* $Q'$)
**from** ‹$Q \longmapsto \alpha \prec Q'$› **have** $(P \parallel Q) \longmapsto \alpha \prec P \parallel Q'$ **by**(*rule Par2F*)
**hence** $(P \parallel Q) \parallel R \longmapsto \alpha \prec (P \parallel Q') \parallel R$ **by**(*rule Par1F*)
**moreover from** *Ass* **have** $((P \parallel Q') \parallel R, P \parallel (Q' \parallel R)) \in Rel$ **by** *blast*
**ultimately show** *?case* **by** *blast*
**next**
**case**(*cPar2* $R'$)
**from** ‹$R \longmapsto \alpha \prec R'$› **have** $(P \parallel Q) \parallel R \longmapsto \alpha \prec (P \parallel Q) \parallel R'$ **by**(*rule Par2F*)
**moreover from** *Ass* **have** $((P \parallel Q) \parallel R', P \parallel (Q \parallel R')) \in Rel$ **by** *blast*
**ultimately show** *?case* **by** *blast*
**next**
**case**(*cComm1* $Q'$ $R'$ $a$ $b$ $x$)
**from** ‹$Q \longmapsto a{<}x{>} \prec Q'$› ‹$x \sharp P$› **have** $P \parallel Q \longmapsto a{<}x{>} \prec P \parallel Q'$ **by**(*rule Par2B*)
**hence** $(P \parallel Q) \parallel R \longmapsto \tau \prec (P \parallel Q')[x{::=}b] \parallel R'$ **using** ‹$R \longmapsto a[b] \prec R'$› **by**(*rule Comm1*)
**with** ‹$x \sharp P$› **have** $(P \parallel Q) \parallel R \longmapsto \tau \prec (P \parallel (Q'[x{::=}b])) \parallel R'$ **by**(*simp add*: *forget*)
**moreover from** *Ass* **have** $((P \parallel (Q'[x{::=}b])) \parallel R', P \parallel (Q'[x{::=}b] \parallel R')) \in Rel$ **by** *blast*
**ultimately show** *?case* **by** *blast*
**next**
**case**(*cComm2* $Q'$ $R'$ $a$ $b$ $x$)
**from** ‹$Q \longmapsto a[b] \prec Q'$› **have** $P \parallel Q \longmapsto a[b] \prec P \parallel Q'$ **by**(*rule Par2F*)
**with** ‹$x \sharp P$› ‹$x \sharp Q$› ‹$R \longmapsto a{<}x{>} \prec R'$› **have** $(P \parallel Q) \parallel R \longmapsto \tau \prec (P \parallel Q') \parallel R'[x{::=}b]$
**by**(*force intro*: *Comm2*)
**moreover from** *Ass* **have** $((P \parallel Q') \parallel R'[x{::=}b], P \parallel (Q' \parallel R'[x{::=}b])) \in Rel$ **by** *blast*
**ultimately show** *?case* **by** *blast*
**next**
**case**(*cClose1* $Q'$ $R'$ $a$ $x$ $y$)
**from** ‹$Q \longmapsto a{<}x{>} \prec Q'$› ‹$x \sharp P$› **have** $P \parallel Q \longmapsto a{<}x{>} \prec P \parallel Q'$ **by**(*rule Par2B*)
**with** ‹$y \sharp P$› ‹$y \sharp Q$› ‹$x \sharp P$› ‹$R \longmapsto a{<}\nu y{>} \prec R'$› **have** $(P \parallel Q) \parallel R \longmapsto \tau \prec {<}\nu y{>}((P \parallel Q')[x{::=}y] \parallel R')$
**by**(*rule-tac Close1*) *auto*
**with** ‹$x \sharp P$› **have** $(P \parallel Q) \parallel R \longmapsto \tau \prec {<}\nu y{>}((P \parallel (Q'[x{::=}y])) \parallel R')$ **by**(*simp add*: *forget*)
**moreover from** ‹$y \sharp P$› **have** $({<}\nu y{>}((P \parallel Q'[x{::=}y]) \parallel R'), P \parallel {<}\nu y{>}(Q'[x{::=}y] \parallel R')) \in Rel$
**by**(*rule FreshExt*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*cClose2* $Q'$ $R'$ $a$ $x$ $y$)
**from** ‹$Q \longmapsto a{<}\nu y{>} \prec Q'$› ‹$y \sharp P$› **have** $P \parallel Q \longmapsto a{<}\nu y{>} \prec P \parallel Q'$ **by**(*rule Par2B*)

**hence** *Act*: $(P \parallel Q) \parallel R \longmapsto \tau \prec <\nu y>((P \parallel Q') \parallel R'[x::=y])$ **using** $\langle R \longmapsto a<x> \prec R'\rangle$ $\langle y \sharp R\rangle$ **by**(*rule Close2*)
**moreover from** $\langle y \sharp P\rangle$ **have** $(<\nu y>((P \parallel Q') \parallel R'[x::=y]),\ P \parallel <\nu y>(Q' \parallel R'[x::=y])) \in Rel$
**by**(*rule FreshExt*)
**ultimately show** *?case* **by** *blast*
**qed**
**next**
**case**(*cComm1 P' QR a b x*)
**from** $\langle Q \parallel R \longmapsto a[b] \prec QR\rangle$ **show** *?case*
**proof**(*induct rule*: *parCasesF*[**where** $C$=()])
**case**(*cPar1* $Q'$)
**from** $\langle P \longmapsto a<x> \prec P'\rangle$ $\langle Q \longmapsto a[b] \prec Q'\rangle$ **have** $P \parallel Q \longmapsto \tau \prec P'[x::=b] \parallel Q'$ **by**(*rule Comm1*)
**hence** $(P \parallel Q) \parallel R \longmapsto \tau \prec (P'[x::=b] \parallel Q') \parallel R$ **by**(*rule Par1F*)
**moreover from** *Ass* **have** $((P'[x::=b] \parallel Q') \parallel R,\ P'[x::=b] \parallel (Q' \parallel R)) \in Rel$ **by** *blast*
**ultimately show** *?case* **by** *blast*
**next**
**case**(*cPar2* $R'$)
**from** $\langle P \longmapsto a<x> \prec P'\rangle$ $\langle x \sharp Q\rangle$ **have** $P \parallel Q \longmapsto a<x> \prec P' \parallel Q$ **by**(*rule Par1B*)
**hence** $(P \parallel Q) \parallel R \longmapsto \tau \prec (P' \parallel Q)[x::=b] \parallel R'$ **using** $\langle R \longmapsto a[b] \prec R'\rangle$ **by**(*rule Comm1*)
**with** $\langle x \sharp Q\rangle$ **have** $(P \parallel Q) \parallel R \longmapsto \tau \prec (P'[x::=b] \parallel Q) \parallel R'$ **by**(*simp add*: *forget*)
**moreover from** *Ass* **have** $((P'[x::=b] \parallel Q) \parallel R',\ P'[x::=b] \parallel (Q \parallel R')) \in Rel$ **by** *blast*
**ultimately show** *?case* **by** *blast*
**next**
**case**(*cComm1* $Q'$ $R'$)
**from** $\langle a[b] = \tau\rangle$ **have** *False* **by** *simp* **thus** *?case* **by** *simp*
**next**
**case**(*cComm2* $Q'$ $R'$)
**from** $\langle a[b] = \tau\rangle$ **have** *False* **by** *simp* **thus** *?case* **by** *simp*
**next**
**case**(*cClose1* $Q'$ $R'$)
**from** $\langle a[b] = \tau\rangle$ **have** *False* **by** *simp* **thus** *?case* **by** *simp*
**next**
**case**(*cClose2* $Q'$ $R'$)
**from** $\langle a[b] = \tau\rangle$ **have** *False* **by** *simp* **thus** *?case* **by** *simp*
**qed**
**next**
**case**(*cComm2 P' QR a b x*)
**from** $\langle x \sharp Q \parallel R\rangle$ **have** $x \sharp Q$ **and** $x \sharp R$ **by** *simp+*
**with** $\langle Q \parallel R \longmapsto a<x> \prec QR\rangle$ **show** *?case*
**proof**(*induct rule*: *parCasesB*)
**case**(*cPar1* $Q'$)
**from** $\langle P \longmapsto a[b] \prec P'\rangle$ $\langle Q \longmapsto a<x> \prec Q'\rangle$ **have** $P \parallel Q \longmapsto \tau \prec P' \parallel$

$(Q'[x::=b])$ **by**(*rule Comm2*)
**hence** $(P \parallel Q) \parallel R \longmapsto \tau \prec (P' \parallel Q'[x::=b]) \parallel R$ **by**(*rule Par1F*)
**moreover from** *Ass* **have** $((P' \parallel Q'[x::=b]) \parallel R,\ P' \parallel Q'[x::=b] \parallel R) \in Rel$ **by** *blast*
**with** $\langle x \sharp R\rangle$ **have** $((P' \parallel Q'[x::=b]) \parallel R,\ P' \parallel (Q' \parallel R)[x::=b]) \in Rel$ **by**(*force simp add*: *forget*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*cPar2* $R'$)
**from** $\langle P \longmapsto a[b] \prec P'\rangle$ **have** $P \parallel Q \longmapsto a[b] \prec P' \parallel Q$ **by**(*rule Par1F*)
**hence** $(P \parallel Q) \parallel R \longmapsto \tau \prec (P' \parallel Q) \parallel (R'[x::=b])$ **using** $\langle R \longmapsto a{<}x{>} \prec R'\rangle$ **by** (*rule Comm2*)
**moreover from** *Ass* **have** $((P' \parallel Q) \parallel R'[x::=b],\ P' \parallel Q \parallel (R'[x::=b])) \in Rel$ **by** *blast*
**hence** $((P' \parallel Q) \parallel R'[x::=b],\ P' \parallel (Q \parallel R')[x::=b]) \in Rel$ **using** $\langle x \sharp Q\rangle$ **by**(*force simp add*: *forget*)
**ultimately show** *?case* **by** *blast*
**qed**
**next**
**case**(*cClose1* $P'$ *QR a x y*)
**from** $\langle x \sharp Q \parallel R\rangle$ **have** $x \sharp Q$ **by** *simp*
**from** $\langle y \sharp Q \parallel R\rangle$ **have** $y \sharp Q$ **and** $y \sharp R$ **by** *simp+*
**from** $\langle Q \parallel R \longmapsto a{<}\nu y{>} \prec QR\rangle$ $\langle y \sharp Q\rangle$ $\langle y \sharp R\rangle$ **show** *?case*
**proof**(*induct rule*: *parCasesB*)
**case**(*cPar1* $Q'$)
**from** $\langle P \longmapsto a{<}x{>} \prec P'\rangle$ $\langle Q \longmapsto a{<}\nu y{>} \prec Q'\rangle$ $\langle y \sharp P\rangle$ **have** $P \parallel Q \longmapsto \tau \prec {<}\nu y{>}(P'[x::=y] \parallel Q')$ **by**(*rule Close1*)
**hence** $(P \parallel Q) \parallel R \longmapsto \tau \prec ({<}\nu y{>}(P'[x::=y] \parallel Q')) \parallel R$ **by**(*rule Par1F*)
**moreover from** $\langle y \sharp R\rangle$ **have** $(({<}\nu y{>}(P'[x::=y] \parallel Q')) \parallel R,\ {<}\nu y{>}(P'[x::=y] \parallel Q' \parallel R)) \in Rel$
**by**(*rule FreshExt'*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*cPar2* $R'$)
**from** $\langle P \longmapsto a{<}x{>} \prec P'\rangle$ $\langle x \sharp Q\rangle$ **have** $P \parallel Q \longmapsto a{<}x{>} \prec P' \parallel Q$ **by**(*rule Par1B*)
**with** $\langle R \longmapsto a{<}\nu y{>} \prec R'\rangle$ $\langle y \sharp P\rangle$ $\langle y \sharp Q\rangle$ **have** $(P \parallel Q) \parallel R \longmapsto \tau \prec {<}\nu y{>}((P' \parallel Q)[x::=y] \parallel R')$
**by**(*rule-tac Close1*) *auto*
**with** $\langle x \sharp Q\rangle$ **have** $(P \parallel Q) \parallel R \longmapsto \tau \prec {<}\nu y{>}((P'[x::=y] \parallel Q) \parallel R')$ **by**(*simp add*: *forget*)
**moreover have** $({<}\nu y{>}((P'[x::=y] \parallel Q) \parallel R'),\ {<}\nu y{>}(P'[x::=y] \parallel (Q \parallel R'))) \in Rel$ **by**(*metis Ass Res*)
**ultimately show** *?case* **by** *blast*
**qed**
**next**
**case**(*cClose2* $P'$ *QR a x y*)
**from** $\langle y \sharp Q \parallel R\rangle$ **have** $y \sharp Q$ **and** $y \sharp R$ **by** *simp+*
**from** $\langle x \sharp Q \parallel R\rangle$ **have** $x \sharp Q$ **and** $x \sharp R$ **by** *simp+*

**with** ‹$Q \parallel R \longmapsto a{<}x{>} \prec QR$› **show** *?case*
**proof**(*induct rule*: *parCasesB*)
**case**(*cPar1 Q'*)
**from** ‹$P \longmapsto a{<}\nu y{>} \prec P'$› ‹$Q \longmapsto a{<}x{>} \prec Q'$› **have** $P \parallel Q \longmapsto \tau \prec {<}\nu y{>}(P' \parallel Q'[x::=y])$ **using** ‹$y \sharp Q$›
**by**(*rule Close2*)
**hence** $(P \parallel Q) \parallel R \longmapsto \tau \prec ({<}\nu y{>}(P' \parallel Q'[x::=y])) \parallel R$ **by**(*rule Par1F*)
**moreover from** ‹$y \sharp R$› **have** $(({<}\nu y{>}(P' \parallel Q'[x::=y])) \parallel R, {<}\nu y{>}(P' \parallel (Q'[x::=y] \parallel R))) \in Rel$
**by**(*rule FreshExt'*)
**with** ‹$x \sharp R$› **have** $(({<}\nu y{>}(P' \parallel Q'[x::=y])) \parallel R, {<}\nu y{>}(P' \parallel (Q' \parallel R)[x::=y])) \in Rel$
**by**(*simp add*: *forget*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*cPar2 R'*)
**from** ‹$P \longmapsto a{<}\nu y{>} \prec P'$› ‹$y \sharp Q$› **have** $P \parallel Q \longmapsto a{<}\nu y{>} \prec P' \parallel Q$ **by**(*rule Par1B*)
**hence** $(P \parallel Q) \parallel R \longmapsto \tau \prec {<}\nu y{>}((P' \parallel Q) \parallel R'[x::=y])$ **using** ‹$R \longmapsto a{<}x{>} \prec R'$› ‹$y \sharp R$› **by**(*rule Close2*)
**moreover have** $((P' \parallel Q) \parallel R'[x::=y], P' \parallel (Q \parallel R'[x::=y])) \in Rel$ **by**(*rule Ass*)
**hence** $({<}\nu y{>}((P' \parallel Q) \parallel R'[x::=y]), {<}\nu y{>}(P' \parallel (Q \parallel R'[x::=y]))) \in Rel$ **by**(*rule Res*)
**hence** $({<}\nu y{>}((P' \parallel Q) \parallel R'[x::=y]), {<}\nu y{>}(P' \parallel (Q \parallel R')[x::=y])) \in Rel$ **using** ‹$x \sharp Q$›
**by**(*simp add*: *forget*)
**ultimately show** *?case* **by** *blast*
**qed**
**qed**
**qed**

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

**shows** $({<}\nu a{>}P)[x::=a] = {<}\nu x{>}([(x, a)] \cdot P)$
**proof** −
**have** $a \sharp {<}\nu a{>}P$ **by**(*simp add*: *name-fresh-abs*)
**hence** $({<}\nu a{>}P)[x::=a] = [(x, a)] \cdot {<}\nu a{>}P$ **by**(*rule injPermSubst*[*THEN sym*])
**thus** $({<}\nu a{>}P)[x::=a] = {<}\nu x{>}([(x, a)] \cdot P)$ **by**(*simp add*: *name-calc*)
**qed**

**lemma** *scopeExtParLeft*:
**fixes** $P$ :: *pi*
**and** $Q$ :: *pi*
**and** $a$ :: *name*
**and** *lst* :: *name list*

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

**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** $<\nu x>(P \parallel Q) \leadsto[Rel]\ P \parallel <\nu x>Q$
**using** ‹*eqvt Rel*›
**proof**(*induct rule*: *simCasesCont*[**where** $C=(x, P, Q)$])
**case**(*Bound a y PxQ*)
**from** ‹$y \sharp (x, P, Q)$› **have** $y \neq x$ **and** $y \sharp P$ **and** $y \sharp Q$ **by** *simp*+
**hence** $y \sharp P$ **and** $y \sharp <\nu x>Q$ **by**(*simp add*: *abs-fresh*)+
**with** ‹$P \parallel <\nu x>Q \longmapsto a«y» \prec PxQ$› **show** *?case*
**proof**(*induct rule*: *parCasesB*)
**case**(*cPar1 P'*)
**from** ‹$P \longmapsto a«y» \prec P'$› ‹$x \sharp P$› ‹$y \neq x$› **have** $x \sharp a$ **and** $x \sharp P'$
**by**(*force intro*: *freshBoundDerivative*)+

**from** ‹$P \longmapsto a«y» \prec P'$› ‹$y \sharp Q$› **have** $P \parallel Q \longmapsto a«y» \prec P' \parallel Q$ **by**(*rule Par1B*)
**with** ‹$x \sharp a$› ‹$y \neq x$› **have** $<\nu x>(P \parallel Q) \longmapsto a«y» \prec <\nu x>(P' \parallel Q)$ **by**(*rule-tac ResB*) *auto*
**moreover have** *derivative* $(<\nu x>(P' \parallel Q))\ (P' \parallel <\nu x>Q)\ a\ y\ Rel$
**proof**(*cases a*, *auto simp add*: *derivative-def*)
**fix** $u$

**show** $((<\nu x>(P' \parallel Q))[y::=u],\ P'[y::=u] \parallel ((<\nu x>Q)[y::=u])) \in Rel$
**proof**(*cases* $x=u$)
**case** *True*
**have** $(<\nu x>(P' \parallel Q))[y::=x] = <\nu y>(([(y, x)] \cdot P') \parallel ([(y, x)] \cdot Q))$
**by**(*simp add*: *substRes3*)
**moreover from** ‹$x \sharp P'$› **have** $P'[y::=x] = [(y, x)] \cdot P'$ **by**(*rule injPermSubst*[*THEN sym*])
**moreover have** $(<\nu x>Q)[y::=x] = <\nu y>([(y, x)] \cdot Q)$ **by**(*rule substRes3*)
**moreover from** ‹$x \sharp P'$› ‹$y \neq x$› **have** $y \sharp [(y, x)] \cdot P'$ **by**(*simp add*: *name-fresh-left name-calc*)
**ultimately show** *?thesis* **using** ‹$x = u$›**by**(*force intro*: *Res*)
**next**
**case** *False*
**with** ‹$y \neq x$› **have** $(<\nu x>(P' \parallel Q))[y::=u] = <\nu x>(P'[y::=u] \parallel Q[y::=u])$
**by**(*simp add*: *fresh-prod name-fresh*)
**moreover from** ‹$x \neq u$› ‹$y \neq x$› **have** $(<\nu x>Q)[y::=u] = <\nu x>(Q[y::=u])$
**by**(*simp add*: *fresh-prod name-fresh*)
**moreover from** ‹$x \sharp P'$› ‹$x \neq u$› **have** $x \sharp P'[y::=u]$ **by**(*simp add*: *fresh-fact1*)
**ultimately show** *?thesis* **by**(*force intro*: *Res*)
**qed**

**next**
**from** ‹x ♯ P′› **show** (<νx>(P′ ∥ Q), P′ ∥ <νx>Q) ∈ *Rel* **by**(*rule Res*)
**qed**

**ultimately show** *?case* **by** *blast*
**next**
**case**(*cPar2 xQ*)
**from** ‹<νx>Q ⟼a«y» ≺ xQ› ‹y ≠ x› ‹y ♯ Q› **show** *?case*
**proof**(*induct rule*: *resCasesB*)
**case**(*cOpen a Q′*)
**from** ‹Q ⟼a[x] ≺ Q′› ‹y ♯ Q› **have** *yFreshQ′*: y ♯ Q′ **by**(*force intro*: *freshFreeDerivative*)

**from** ‹Q ⟼ a[x] ≺ Q′› **have** P ∥ Q ⟼ a[x] ≺ P ∥ Q′ **by**(*rule Par2F*)
**hence** <νx>(P ∥ Q) ⟼ a<νx> ≺ P ∥ Q′ **using** ‹a ≠ x› **by**(*rule Open*)
**with** ‹y ♯ P› ‹y ♯ Q′› **have** <νx>(P ∥ Q) ⟼ a<νy> ≺ [(x, y)] • (P ∥ Q′)
**by**(*subst alphaBoundResidual*[**where** x′=x]) (*auto simp add*: *fresh-left calc-atm*)
**with** ‹y ♯ P› ‹x ♯ P› **have** <νx>(P ∥ Q) ⟼ a<νy> ≺ P ∥ ([(x, y)] • Q′)
**by**(*simp add*: *name-fresh-fresh*)

**moreover have** *derivative* (P ∥ ([(x, y)] • Q′)) (P ∥ ([(y, x)] • Q′)) (*BoundOutputS a*) *y Rel* **using** *Id*
**by**(*auto simp add*: *derivative-def name-swap*)

**ultimately show** *?case* **by** *blast*
**next**
**case**(*cRes Q′*)

**from** ‹Q ⟼ a«y» ≺ Q′› ‹y ♯ P› **have** P ∥ Q ⟼ a«y» ≺ P ∥ Q′ **by**(*rule Par2B*)
**hence** <νx>(P ∥ Q) ⟼ a«y» ≺ <νx>(P ∥ Q′) **using** ‹x ♯ a› ‹y ≠ x›
**by**(*rule-tac ResB*) *auto*
**moreover have** *derivative* (<νx>(P ∥ Q′)) (P ∥ <νx>Q′) *a y Rel*
**proof**(*cases a*, *auto simp add*: *derivative-def*)
**fix** *u*
**show** ((<νx>(P ∥ Q′))[y::=u], P[y::=u] ∥ (<νx>Q′)[y::=u]) ∈ *Rel*
**proof**(*cases x=u*)
**case** *True*
**from** ‹x ♯ P› ‹y ♯ P› **have** (<νx>(P ∥ Q′))[y::=x] = <νy>(P ∥ ([(y, x)] • Q′))
**by**(*simp add*: *substRes3 perm-fresh-fresh*)
**moreover from** ‹y ♯ P› **have** P[y::=x] = P **by**(*simp add*: *forget*)
**moreover have** (<νx>Q′)[y::=x] = <νy>([(y, x)] • Q′) **by**(*rule substRes3*)
**ultimately show** *?thesis* **using** ‹x=u› ‹y ♯ P› **by**(*force intro*: *Res*)
**next**
**case** *False*
**with** ‹y ≠ x› **have** (<νx>(P ∥ Q′))[y::=u] = <νx>((P ∥ Q′)[y::=u])
**by**(*simp add*: *fresh-prod name-fresh*)

**moreover from** ‹$y \neq x$› ‹$x \neq u$› **have** $(<\nu x>Q')[y::=u] = <\nu x>(Q'[y::=u])$
**by**(*simp add*: *fresh-prod name-fresh*)
**moreover from** ‹$x \sharp P$› ‹$x \neq u$› **have** $x \sharp P[y::=u]$ **by**(*force simp add*: *fresh-fact1*)
**ultimately show** *?thesis* **by**(*force intro*: *Res*)
**qed**
**next**
**from** ‹$x \sharp P$› **show** $(<\nu x>(P \parallel Q'), P \parallel <\nu x>Q') \in Rel$ **by**(*rule Res*)
**qed**
**ultimately show** *?case* **by** *blast*
**qed**
**qed**
**next**
**case**(*Free* $\alpha$ *PxQ*)
**from** ‹$P \parallel <\nu x>Q \longmapsto \alpha \prec PxQ$› **show** *?case*
**proof**(*induct rule*: *parCasesF*[**where** $C=x$])
**case**(*cPar1* $P'$)
**from** ‹$P \longmapsto \alpha \prec P'$› ‹$x \sharp P$›**have** $x \sharp \alpha$ **and** $x \sharp P'$ **by**(*force intro*: *freshFreeDerivative*)+
**from** ‹$P \longmapsto \alpha \prec P'$› **have** $P \parallel Q \longmapsto \alpha \prec P' \parallel Q$ **by**(*rule Par1F*)
**hence** $<\nu x>(P \parallel Q) \longmapsto \alpha \prec <\nu x>(P' \parallel Q)$ **using** ‹$x \sharp \alpha$› **by**(*rule ResF*)
**moreover from** ‹$x \sharp P'$› **have** $(<\nu x>(P' \parallel Q), P' \parallel <\nu x>Q) \in Rel$ **by**(*rule Res*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*cPar2* $Q'$)
**from** ‹$<\nu x>Q \longmapsto \alpha \prec Q'$› **show** *?case*
**proof**(*induct rule*: *resCasesF*)
**case**(*cRes* $Q'$)
**from** ‹$Q \longmapsto \alpha \prec Q'$› **have** $P \parallel Q \longmapsto \alpha \prec P \parallel Q'$ **by**(*rule Par2F*)
**hence** $<\nu x>(P \parallel Q) \longmapsto \alpha \prec <\nu x>(P \parallel Q')$ **using** ‹$x \sharp \alpha$› **by**(*rule ResF*)
**moreover from** ‹$x \sharp P$› **have** $(<\nu x>(P \parallel Q'), P \parallel <\nu x>Q') \in Rel$ **by**(*rule Res*)
**ultimately show** *?case* **by** *blast*
**qed**
**next**
**case**(*cComm1* $P'$ *xQ a b y*)
**from** ‹$y \sharp x$› **have** $y \neq x$ **by** *simp*
**from** ‹$P \longmapsto a<y> \prec P'$› ‹$x \sharp P$› ‹$y \neq x$› **have** $x \sharp P'$ **by**(*force intro*: *freshBoundDerivative*)
**from** ‹$<\nu x>Q \longmapsto a[b] \prec xQ$› **show** *?case*
**proof**(*induct rule*: *resCasesF*)
**case**(*cRes* $Q'$)
**from** ‹$x \sharp a[b]$› **have** $x \neq b$ **by** *simp*
**from** ‹$P \longmapsto a<y> \prec P'$› ‹$Q \longmapsto a[b] \prec Q'$› **have** $P \parallel Q \longmapsto \tau \prec P'[y::=b] \parallel Q'$ **by**(*rule Comm1*)
**hence** $<\nu x>(P \parallel Q) \longmapsto \tau \prec <\nu x>(P'[y::=b] \parallel Q')$ **by**(*rule-tac ResF*) *auto*
**moreover from** ‹$x \sharp P'$› ‹$x \neq b$› **have** $x \sharp P'[y::=b]$ **by**(*force intro*: *fresh-fact1*)
**hence** $(<\nu x>(P'[y::=b] \parallel Q'), P'[y::=b] \parallel <\nu x>Q') \in Rel$ **by**(*rule Res*)

**ultimately show** *?case* **by** *blast*
**qed**
**next**
**case**(*cComm2* $P'$ *xQ* $a$ $b$ $y$)
**from** ‹$y \sharp x$› ‹$y \sharp$ <$\nu x$>$Q$› **have** $y \neq x$ **and** $y \sharp Q$ **by**(*simp add*: *abs-fresh*)+
**with** ‹<$\nu x$>$Q \longmapsto a$<$y$> $\prec$ *xQ*› **show** *?case*
**proof**(*induct rule*: *resCasesB*)
**case**(*cOpen* $b$ $Q'$)
**from** ‹*InputS* $a$ = *BoundOutputS* $b$› **have** *False* **by** *simp*
**thus** *?case* **by** *simp*
**next**
**case**(*cRes* $Q'$)
**from** ‹$P \longmapsto a[b] \prec P'$› ‹$Q \longmapsto a$<$y$> $\prec Q'$› **have** $P \parallel Q \longmapsto \tau \prec P' \parallel Q'[y::=b]$ **by**(*rule Comm2*)
**hence** <$\nu x$>($P \parallel Q$) $\longmapsto \tau \prec$ <$\nu x$>($P' \parallel Q'[y::=b]$) **by**(*rule-tac ResF*) *auto*
**moreover from** ‹$P \longmapsto a[b] \prec P'$› ‹$x \sharp P$› **have** $x \sharp P'$ **and** $x \neq b$ **by**(*force dest*: *freshFreeDerivative*)+
**from** ‹$x \sharp P'$› **have** (<$\nu x$>($P' \parallel Q'[y::=b]$), $P' \parallel$ <$\nu x$>($Q'[y::=b]$)) $\in$ *Rel* **by**(*rule Res*)
**with** ‹$y \neq x$› ‹$x \neq b$› **have** (<$\nu x$>($P' \parallel Q'[y::=b]$), $P' \parallel$ (<$\nu x$>$Q'$)$[y::=b]$) $\in$ *Rel* **by** *simp*
**ultimately show** *?case* **by** *blast*
**qed**
**next**
**case**(*cClose1* $P'$ $Q'$ $a$ $y$ $z$)
**from** ‹$y \sharp x$› **have** $y \neq x$ **by** *simp*
**from** ‹$z \sharp x$› ‹$z \sharp$ <$\nu x$>$Q$› **have** $z \sharp Q$ **and** $z \neq x$ **by**(*simp add*: *abs-fresh*)+
**from** ‹$P \longmapsto a$<$y$> $\prec P'$› ‹$z \sharp P$› **have** $z \neq a$ **by**(*force dest*: *freshBoundDerivative*)
**from** ‹<$\nu x$>$Q \longmapsto a$<$\nu z$> $\prec Q'$› ‹$z \neq x$› ‹$z \sharp Q$› **show** *?case*
**proof**(*induct rule*: *resCasesB*)
**case**(*cOpen* $b$ $Q'$)
**from** ‹*BoundOutputS* $a$ = *BoundOutputS* $b$› **have** $a = b$ **by** *simp*
**with** ‹$Q \longmapsto b[x] \prec Q'$› **have** ($[(z, x)] \cdot Q$) $\longmapsto [(z, x)] \cdot (a[x] \prec Q')$
**by**(*rule-tac transitions.eqvt*) *simp*
**with** ‹$b \neq x$› ‹$z \neq a$› ‹$a = b$› ‹$z \neq x$› **have** ($[(z, x)] \cdot Q$) $\longmapsto a[z] \prec ([(z, x)] \cdot Q')$
**by**(*simp add*: *name-calc eqvts*)
**with** ‹$P \longmapsto a$<$y$> $\prec P'$› **have** $P \parallel ([(z, x)] \cdot Q) \longmapsto \tau \prec P'[y::=z] \parallel ([(z, x)] \cdot Q')$
**by**(*rule Comm1*)
**hence** <$\nu z$>($P \parallel ([(x, z)] \cdot Q)$) $\longmapsto \tau \prec$ <$\nu z$>($P'[y::=z] \parallel ([(z, x)] \cdot Q')$)
**by**(*rule-tac ResF*) *auto*
**hence** <$\nu x$>($P \parallel Q$) $\longmapsto \tau \prec$ <$\nu z$>($P'[y::=z] \parallel ([(z, x)] \cdot Q')$) **using** ‹$z \sharp P$› ‹$z \sharp Q$› ‹$x \sharp P$›
**by**(*subst alphaRes*[**where** $c$=$z$]) *auto*
**with** *Id* **show** *?case* **by** *force*
**next**
**case**(*cRes* $Q'$)

**from** $\langle P \longmapsto a{<}y{>} \prec P' \rangle$ $\langle Q \longmapsto a{<}\nu z{>} \prec Q' \rangle$ $\langle z \sharp P \rangle$ **have** $P \parallel Q \longmapsto \tau \prec {<}\nu z{>}(P'[y{::=}z] \parallel Q')$
**by**(*rule Close1*)
**hence** ${<}\nu x{>}(P \parallel Q) \longmapsto \tau \prec {<}\nu x{>}{<}\nu z{>}(P'[y{::=}z] \parallel Q')$ **by**(*rule-tac ResF*) *auto*
**moreover from** $\langle P \longmapsto a{<}y{>} \prec P' \rangle$ $\langle y \neq x \rangle$ $\langle x \sharp P \rangle$ **have** $x \sharp P'$
**by**(*force dest*: *freshBoundDerivative*)
**with** $\langle z \neq x \rangle$ **have** $x \sharp P'[y{::=}z]$ **by**(*simp add*: *fresh-fact1*)
**hence** $({<}\nu x{>}{<}\nu z{>}(P'[y{::=}z] \parallel Q'), {<}\nu z{>}(P'[y{::=}z] \parallel {<}\nu x{>}Q')) \in Rel$
**by**(*rule ScopeExt*)
**ultimately show** *?case* **by** *blast*
**qed**
**next**
**case**(*cClose2* $P'$ *xQ a y z*)
**from** $\langle z \sharp x \rangle$ $\langle z \sharp {<}\nu x{>}Q \rangle$ **have** $z \neq x$ **and** $z \sharp Q$ **by**(*auto simp add*: *abs-fresh*)
**from** $\langle y \sharp x \rangle$ $\langle y \sharp {<}\nu x{>}Q \rangle$ **have** $y \neq x$ **and** $y \sharp Q$ **by**(*auto simp add*: *abs-fresh*)
**with** $\langle {<}\nu x{>}Q \longmapsto a{<}y{>} \prec xQ \rangle$ **show** *?case*
**proof**(*induct rule*: *resCasesB*)
**case**(*cOpen b* $Q'$)
**from** $\langle InputS\ a = BoundOutputS\ b \rangle$ **have** *False* **by** *simp*
**thus** *?case* **by** *simp*
**next**
**case**(*cRes* $Q'$)
**from** $\langle P \longmapsto a{<}\nu z{>} \prec P' \rangle$ $\langle Q \longmapsto a{<}y{>} \prec Q' \rangle$ $\langle z \sharp Q \rangle$ **have** $P \parallel Q \longmapsto \tau \prec {<}\nu z{>}(P' \parallel Q'[y{::=}z])$
**by**(*rule Close2*)
**hence** ${<}\nu x{>}(P \parallel Q) \longmapsto \tau \prec {<}\nu x{>}{<}\nu z{>}(P' \parallel (Q'[y{::=}z]))$
**by**(*rule-tac ResF*) *auto*
**moreover from** $\langle P \longmapsto a{<}\nu z{>} \prec P' \rangle$ $\langle x \sharp P \rangle$ $\langle z \neq x \rangle$ **have** $x \sharp P'$ **by**(*force dest*: *freshBoundDerivative*)
**hence** $({<}\nu x{>}{<}\nu z{>}(P' \parallel (Q'[y{::=}z])), {<}\nu z{>}(P' \parallel ({<}\nu x{>}(Q'[y{::=}z])))) \in Rel$
**by**(*rule ScopeExt*)
**with** $\langle z \neq x \rangle$ $\langle y \neq x \rangle$ **have** $({<}\nu x{>}{<}\nu z{>}(P' \parallel (Q'[y{::=}z])), {<}\nu z{>}(P' \parallel ({<}\nu x{>}Q')[y{::=}z])) \in Rel$
**by** *simp*
**ultimately show** *?case* **by** *blast*
**qed**
**qed**
**qed**

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

**assumes** $x \sharp P$
**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** $P \parallel <\nu x>Q \rightsquigarrow[Rel]\ <\nu x>(P \parallel Q)$
**using** ‹*eqvt Rel*›
**proof**(*induct rule*: *simCasesCont*[**where** $C=(x, P, Q)$])
**case**(*Bound a y xPQ*)
**from** ‹$y \sharp (x, P, Q)$› **have** $y \neq x$ **and** $y \sharp P$ **and** $y \sharp Q$ **by** *simp*+
**hence** $y \neq x$ **and** $y \sharp P \parallel Q$ **by**(*auto simp add*: *abs-fresh*)
**with** ‹$<\nu x>(P \parallel Q) \longmapsto a«y» \prec xPQ$› **show** *?case*
**proof**(*induct rule*: *resCasesB*)
**case**(*cOpen a PQ*)
**from** ‹$P \parallel Q \longmapsto a[x] \prec PQ$› **show** *?case*
**proof**(*induct rule*: *parCasesF*[**where** $C=()$])
**case**(*cPar1* $P'$)
**from** ‹$P \longmapsto a[x] \prec P'$› ‹$x \sharp P$› **have** $x \neq x$ **by**(*force dest*: *freshFreeDerivative*)
**thus** *?case* **by** *simp*
**next**
**case**(*cPar2* $Q'$)
**from** ‹$Q \longmapsto a[x] \prec Q'$› ‹$y \sharp Q$› **have** $y \sharp Q'$ **by**(*force dest*: *freshFreeDerivative*)
**from** ‹$Q \longmapsto a[x] \prec Q'$› ‹$a \neq x$› **have** $<\nu x>Q \longmapsto a<\nu x> \prec Q'$ **by**(*rule Open*)
**hence** $P \parallel <\nu x>Q \longmapsto a<\nu x> \prec P \parallel Q'$ **using** ‹$x \sharp P$› **by**(*rule Par2B*)
**with** ‹$y \sharp P$› ‹$y \sharp Q'$› ‹$x \sharp P$› **have** $P \parallel <\nu x>Q \longmapsto a<\nu y> \prec ([(y, x)] \cdot (P \parallel Q'))$
**by**(*subst alphaBoundResidual*[**where** $x'=x$]) (*auto simp add*: *fresh-left calc-atm*)
**moreover with** *Id* **have** *derivative* $([(y, x)] \cdot (P \parallel Q'))$
$([(y, x)] \cdot (P \parallel Q'))$ (*BoundOutputS a*) *y Rel*
**by**(*auto simp add*: *derivative-def*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*cComm1* $P'$ $Q'$ *b c y*)
**from** ‹$a[x] = \tau$› **show** *?case* **by** *simp*
**next**
**case**(*cComm2* $P'$ $Q'$ *b c y*)
**from** ‹$a[x] = \tau$› **show** *?case* **by** *simp*
**next**
**case**(*cClose1* $P'$ $Q'$ *b y z*)
**from** ‹$a[x] = \tau$› **show** *?case* **by** *simp*
**next**
**case**(*cClose2* $P'$ $Q'$ *b y z*)
**from** ‹$a[x] = \tau$› **show** *?case* **by** *simp*
**qed**
**next**
**case**(*cRes PQ*)
**from** ‹$P \parallel Q \longmapsto a«y» \prec PQ$› ‹$y \sharp P$› ‹$y \sharp Q$›
**show** *?case*

**proof**(*induct rule*: *parCasesB*)
**case**(*cPar1 P'*)
**from** ‹$y \neq x$› ‹$x \sharp P$› ‹$P \longmapsto a«y» \prec P'$› **have** $x \sharp P'$ **by**(*force dest*: *freshBoundDerivative*)

**from** ‹$P \longmapsto a«y» \prec P'$› ‹$y \sharp Q$› **have** $P \parallel <\nu x>Q \longmapsto a«y» \prec P' \parallel <\nu x>Q$
**by**(*rule-tac Par1B*) (*auto simp add*: *abs-fresh*)
**moreover have** *derivative* $(P' \parallel <\nu x>Q)$ $(<\nu x>(P' \parallel Q))$ *a y Rel*
**proof**(*cases a*, *auto simp add*: *derivative-def*)
**fix** *u*::*name*
**obtain** *z*::*name* **where** $z \sharp Q$ **and** $y \neq z$ **and** $z \neq u$ **and** $z \sharp P$ **and** $z \sharp P'$
**by**(*generate-fresh name*) *auto*
**thus** $(P'[y::=u] \parallel (<\nu x>Q)[y::=u], (<\nu x>(P' \parallel Q))[y::=u]) \in Rel$ **using** ‹$x \sharp P'$›
**by**(*subst alphaRes*[**where** *c*=*z* **and** *a*=*x*], *auto*)
(*subst alphaRes*[**where** *c*=*z* **and** *a*=*x*], *auto intro*: *Res simp add*: *fresh-fact1*)
**next**
**from** ‹$x \sharp P'$› **show** $(P' \parallel <\nu x>Q, <\nu x>(P' \parallel Q)) \in Rel$
**by**(*rule Res*)
**qed**

**ultimately show** *?case* **by** *blast*
**next**
**case**(*cPar2 Q'*)
**from** ‹$Q \longmapsto a«y» \prec Q'$› **have** $<\nu x>Q \longmapsto a«y» \prec <\nu x>Q'$ **using** ‹$x \sharp a$› ‹$y \neq x$›
**by**(*rule-tac ResB*) *auto*
**hence** $P \parallel <\nu x>Q \longmapsto a«y» \prec P \parallel <\nu x>Q'$ **using** ‹$y \sharp P$› **by**(*rule Par2B*)

**moreover have** *derivative* $(P \parallel <\nu x>Q')$ $(<\nu x>(P \parallel Q'))$ *a y Rel*
**proof**(*cases a*, *auto simp add*: *derivative-def*)
**fix** *u*::*name*
**obtain** *z*::*name* **where** $z \sharp Q$ **and** $z \neq y$ **and** $z \neq u$ **and** $z \sharp P$ **and** $z \sharp Q'$
**by**(*generate-fresh name*) *auto*

**thus** $(P[y::=u] \parallel (<\nu x>Q')[y::=u], (<\nu x>(P \parallel Q'))[y::=u]) \in Rel$ **using** ‹$x \sharp P$›
**by**(*subst alphaRes*[**where** *a*=*x* **and** *c*=*z*], *auto*)
(*subst alphaRes*[**where** *a*=*x* **and** *c*=*z*], *auto intro*: *Res simp add*: *fresh-fact1*)
**next**
**from** ‹$x \sharp P$› **show** $(P \parallel <\nu x>Q', <\nu x>(P \parallel Q')) \in Rel$
**by**(*rule Res*)
**qed**

**ultimately show** *?case* **by** *blast*
**qed**
**qed**

**next**
**case**(*Free* $\alpha$ *xPQ*)
**from** ‹$<\nu x>(P \parallel Q) \longmapsto \alpha \prec xPQ$› **show** *?case*
**proof**(*induct rule*: *resCasesF*)
**case**(*cRes PQ*)
**from** ‹$P \parallel Q \longmapsto \alpha \prec PQ$› **show** *?case*
**proof**(*induct rule*: *parCasesF*[**where** $C{=}x$])
**case**(*cPar1* $P'$)
**from** ‹$P \longmapsto \alpha \prec P'$› **have** $P \parallel <\nu x>Q \longmapsto \alpha \prec P' \parallel <\nu x>Q$ **by**(*rule Par1F*)
**moreover from** ‹$P \longmapsto \alpha \prec P'$› ‹$x \sharp P$› **have** $x \sharp P'$ **by**(*rule freshFreeDerivative*)
**hence** $(P' \parallel <\nu x>Q, <\nu x>(P' \parallel Q)) \in Rel$ **by**(*rule Res*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*cPar2* $Q'$)
**from** ‹$Q \longmapsto \alpha \prec Q'$› ‹$x \sharp \alpha$› **have** $<\nu x>Q \longmapsto \alpha \prec <\nu x>Q'$ **by**(*rule ResF*)
**hence** $P \parallel <\nu x>Q \longmapsto \alpha \prec P \parallel <\nu x>Q'$ **by**(*rule Par2F*)
**moreover from** ‹$x \sharp P$› **have** $(P \parallel <\nu x>Q', <\nu x>(P \parallel Q')) \in Rel$ **by**(*rule Res*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*cComm1* $P'$ $Q'$ *a b y*)
**from** ‹$x \sharp P$› ‹$y \sharp x$› ‹$P \longmapsto a<y> \prec P'$› **have** $x \neq a$ **and** $x \sharp P'$ **by**(*force dest*: *freshBoundDerivative*)+
**show** *?case*
**proof**(*cases* $b{=}x$)
**case** *True*
**from** ‹$Q \longmapsto a[b] \prec Q'$› ‹$x \neq a$› ‹$b = x$› **have** $<\nu x>Q \longmapsto a<\nu x> \prec Q'$ **by**(*rule-tac Open*) *auto*
**with** ‹$P \longmapsto a<y> \prec P'$› **have** $P \parallel <\nu x>Q \longmapsto \tau \prec <\nu x>(P'[y{::=}x] \parallel Q')$ **using** ‹$x \sharp P$› **by**(*rule Close1*)
**moreover from** *Id* **have** $(<\nu x>(P'[y{::=}b] \parallel Q'), <\nu x>(P'[y{::=}b] \parallel Q')) \in Rel$ **by** *blast*
**ultimately show** *?thesis* **using** ‹$b{=}x$› **by** *blast*
**next**
**case** *False*
**from** ‹$Q \longmapsto a[b] \prec Q'$› ‹$x \neq a$› ‹$b \neq x$› **have** $<\nu x>Q \longmapsto a[b] \prec <\nu x>Q'$ **by**(*rule-tac ResF*) *auto*
**with** ‹$P \longmapsto a<y> \prec P'$› **have** $P \parallel <\nu x>Q \longmapsto \tau \prec (P'[y{::=}b] \parallel <\nu x>Q')$ **by**(*rule Comm1*)
**moreover from** ‹$x \sharp P'$› ‹$b \neq x$› **have** $(P'[y{::=}b] \parallel <\nu x>Q', <\nu x>(P'[y{::=}b] \parallel Q')) \in Rel$
**by**(*force intro*: *Res simp add*: *fresh-fact1*)
**ultimately show** *?thesis* **by** *blast*
**qed**
**next**
**case**(*cComm2* $P'$ $Q'$ *a b y*)
**from** ‹$P \longmapsto a[b] \prec P'$› ‹$x \sharp P$› **have** $x \neq a$ **and** $x \neq b$ **and** $x \sharp P'$ **by**(*force dest*: *freshFreeDerivative*)+

**from** ‹$Q \longmapsto a{<}y{>} \prec Q'$› ‹$y \sharp x$› ‹$x \neq a$› **have** $<\nu x>Q \longmapsto a{<}y{>} \prec <\nu x>Q'$ **by**(*rule-tac ResB*) *auto*
**with** ‹$P \longmapsto a[b] \prec P'$› **have** $P \parallel <\nu x>Q \longmapsto \tau \prec P' \parallel (<\nu x>Q')[y{::=}b]$ **by**(*rule Comm2*)
**moreover from** ‹$x \sharp P'$› **have** $(P' \parallel <\nu x>(Q'[y{::=}b]), <\nu x>(P' \parallel Q'[y{::=}b]))$ $\in$ *Rel* **by**(*rule Res*)
**ultimately show** *?case* **using** ‹$y \sharp x$› ‹$x \neq b$› **by** *force*
**next**
**case**(*cClose1* $P'$ $Q'$ *a y z*)
**from** ‹$P \longmapsto a{<}y{>} \prec P'$› ‹$x \sharp P$› ‹$y \sharp x$› **have** $x \neq a$ **and** $x \sharp P'$ **by**(*force dest*: *freshBoundDerivative*)+
**from** ‹$Q \longmapsto a{<}\nu z{>} \prec Q'$› ‹$z \sharp x$› ‹$x \neq a$› **have** $<\nu x>Q \longmapsto a{<}\nu z{>} \prec <\nu x>Q'$ **by**(*rule-tac ResB*) *auto*
**with** ‹$P \longmapsto a{<}y{>} \prec P'$› **have** $P \parallel <\nu x>Q \longmapsto \tau \prec <\nu z>(P'[y{::=}z] \parallel <\nu x>Q')$ **using** ‹$z \sharp P$› **by**(*rule Close1*)
**moreover from** ‹$x \sharp P'$› ‹$z \sharp x$› **have** $(<\nu z>(P'[y{::=}z] \parallel <\nu x>Q'), <\nu x>(<\nu z>(P'[y{::=}z] \parallel Q'))) \in$ *Rel*
**by**(*rule-tac ScopeExt*) (*auto simp add*: *fresh-fact1*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*cClose2* $P'$ $Q'$ *a y z*)
**from** ‹$P \longmapsto a{<}\nu z{>} \prec P'$› ‹$x \sharp P$› ‹$z \sharp x$› **have** $x \neq a$ **and** $x \sharp P'$ **by**(*force dest*: *freshBoundDerivative*)+
**from** ‹$Q \longmapsto a{<}y{>} \prec Q'$› ‹$y \sharp x$› ‹$x \neq a$› **have** $<\nu x>Q \longmapsto a{<}y{>} \prec <\nu x>Q'$ **by**(*rule-tac ResB*) *auto*
**with** ‹$P \longmapsto a{<}\nu z{>} \prec P'$› **have** $P \parallel <\nu x>Q \longmapsto \tau \prec <\nu z>(P' \parallel (<\nu x>Q')[y{::=}z])$ **using** ‹$z \sharp Q$›
**by**(*rule-tac Close2*) (*auto simp add*: *abs-fresh*)
**moreover from** ‹$x \sharp P'$› **have** $(<\nu z>(P' \parallel <\nu x>(Q'[y{::=}z])), <\nu x><\nu z>(P' \parallel Q'[y{::=}z])) \in$ *Rel* **by**(*rule ScopeExt*)
**ultimately show** *?case* **using** ‹$z \sharp x$› ‹$y \sharp x$› **by** *force*
**qed**
**qed**
**qed**

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

**shows** **0** $\leadsto$[*Rel*] $<\nu x>$**0**
**by**(*fastforce simp add*: *simulation-def pi.inject alpha'* *elim*: *resCasesB'* *resCasesF*)

**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* ⊆ *Rel*
**and** *EqvtRel*: *eqvt Rel*

**shows** $<\nu a><\nu b>P \leadsto[Rel]\ <\nu b><\nu a>P$
**proof**(*cases a=b*)
**assume** $a=b$
**with** *Id* **show** *?thesis* **by**(*force intro*: *Strong-Late-Sim.reflexive*)
**next**
**assume** *aineqb*: $a \neq b$
**from** *EqvtRel* **show** *?thesis*
**proof**(*induct rule*: *simCasesCont*[**where** $C=(a, b, P)$])
**case**(*Bound c x baP*)
**from** ‹$x \sharp (a, b, P)$› **have** $x \neq a$ **and** $x \neq b$ **and** $x \sharp P$ **by** *simp+*
**from** ‹$x \sharp P$› **have** $x \sharp <\nu a>P$ **by**(*simp add*: *abs-fresh*)
**with** ‹$<\nu b><\nu a>P \longmapsto c«x» \prec baP$› ‹$x \neq b$› **show** *?case*
**proof**(*induct rule*: *resCasesB*)
**case**(*cOpen c aP*)
**from** ‹$<\nu a>P \longmapsto c[b] \prec aP$›
**show** *?case*
**proof**(*induct rule*: *resCasesF*)
**case**(*cRes P′*)
**from** ‹$a \sharp c[b]$› **have** $a \neq c$ **and** $a \neq b$ **by** *simp+*
**from** ‹$x \sharp P$› ‹$P \longmapsto c[b] \prec P'$› **have** $x \neq c$ **and** $x \sharp P'$ **by**(*force dest*: *freshFreeDerivative*)+
**from** ‹$P \longmapsto c[b] \prec P'$› **have** $([(x, b)] \cdot P) \longmapsto [(x, b)] \cdot (c[b] \prec P')$ **by**(*rule transitions.eqvt*)
**with** ‹$x \neq c$› ‹$c \neq b$› ‹$x \neq b$› **have** $([(x, b)] \cdot P) \longmapsto c[x] \prec [(x, b)] \cdot P'$ **by**(*simp add*: *eqvts calc-atm*)
**hence** $<\nu x>([(x, b)] \cdot P) \longmapsto c<\nu x> \prec [(x, b)] \cdot P'$ **using** ‹$x \neq c$› **by**(*rule-tac Open*) *auto*
**with** ‹$x \sharp P$› **have** $<\nu b>P \longmapsto c<\nu x> \prec [(x, b)] \cdot P'$ **by**(*simp add*: *alphaRes*)
**hence** $<\nu a><\nu b>P \longmapsto c<\nu x> \prec <\nu a>([(x, b)] \cdot P')$ **using** ‹$a \neq c$› ‹$x \neq a$›
**by**(*rule-tac ResB*) *auto*
**moreover from** *Id* **have** *derivative* $(<\nu a>([(x, b)] \cdot P'))$ $(<\nu a>([(x, b)] \cdot P'))$ (*BoundOutputS c*) *x Rel*
**by**(*force simp add*: *derivative-def*)
**ultimately show** *?case* **using** ‹$a \neq b$› ‹$x \neq a$› ‹$a \neq c$› **by**(*force simp add*: *eqvts calc-atm*)
**qed**
**next**
**case**(*cRes aP*)
**from** ‹$<\nu a>P \longmapsto c«x» \prec aP$› ‹$x \neq a$› ‹$x \sharp P$› ‹$b \sharp c$› **show** *?case*
**proof**(*induct rule*: *resCasesB*)
**case**(*cOpen c P′*)
**from** ‹$P \longmapsto c[a] \prec P'$› ‹$x \sharp P$› **have** $x \sharp P'$ **by**(*force intro*: *freshFreeDerivative*)
**from** ‹$b \sharp$ *BoundOutputS c*› **have** $b \neq c$ **by** *simp*

**with** ‹$P \longmapsto c[a] \prec P'$› ‹$a \neq b$› **have** $<\nu b>P \longmapsto c[a] \prec <\nu b>P'$ **by**(*rule-tac ResF*) *auto*
**with** ‹$c \neq a$› **have** $<\nu a><\nu b>P \longmapsto c<\nu a> \prec <\nu b>P'$ **by**(*rule-tac Open*) *auto*
**hence** $<\nu a><\nu b>P \longmapsto c<\nu x> \prec <\nu b>([(x, a)] \cdot P')$ **using** ‹$x \neq b$› ‹$a \neq b$› ‹$x \sharp P'$›
**apply**(*subst alphaBoundResidual*[**where** $x'$=*a*]) **by**(*auto simp add*: *abs-fresh fresh-left calc-atm*)
**moreover have** *derivative* $(<\nu b>([(x, a)] \cdot P'))$ $(<\nu b>([(x, a)] \cdot P'))$ (*BoundOutputS c*) *x Rel* **using** *Id*
**by**(*force simp add*: *derivative-def*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*cRes P'*)
**from** ‹$P \longmapsto c«x» \prec P'$› ‹$b \sharp c$› ‹$x \neq b$› **have** $<\nu b>P \longmapsto c«x» \prec <\nu b>P'$ **by**(*rule-tac ResB*) *auto*
**hence** $<\nu a><\nu b>P \longmapsto c«x» \prec <\nu a><\nu b>P'$ **using** ‹$a \sharp c$› ‹$x \neq a$› **by**(*rule-tac ResB*) *auto*
**moreover have** *derivative* $(<\nu a><\nu b>P')$ $(<\nu b><\nu a>P')$ *c x Rel*
**proof**(*cases c*, *auto simp add*: *derivative-def*)
**fix** *u*::*name*
**show** $((<\nu a><\nu b>P')[x::=u], (<\nu b><\nu a>P')[x::=u]) \in Rel$
**proof**(*cases u*=*a*)
**case** *True*
**from** ‹$u = a$› ‹$a \neq b$› **show** *?thesis*
**by**(*subst injPermSubst*[*symmetric*], *auto simp add*: *abs-fresh*)
(*subst injPermSubst*[*symmetric*], *auto simp add*: *abs-fresh calc-atm intro*: *ResComm*)
**next**
**case** *False*
**show** *?thesis*
**proof**(*cases u*=*b*)
**case** *True*
**from** ‹$u = b$› ‹$u \neq a$› **show** *?thesis*
**by**(*subst injPermSubst*[*symmetric*], *auto simp add*: *abs-fresh*)
(*subst injPermSubst*[*symmetric*], *auto simp add*: *abs-fresh calc-atm intro*: *ResComm*)
**next**
**case** *False*
**from** ‹$u \neq a$› ‹$u \neq b$› ‹$x \neq a$› ‹$x \neq b$› **show** *?thesis* **by**(*auto intro*: *ResComm*)
**qed**
**qed**
**next**
**show** $(<\nu a><\nu b>P', <\nu b><\nu a>P') \in Rel$ **by**(*rule ResComm*)
**qed**
**ultimately show** *?case* **by** *blast*
**qed**
**qed**

**next**
**case**(*Free* $\alpha$ *baP*)
**from** $\langle <\nu b><\nu a>P \longmapsto \alpha \prec baP \rangle$ **show** *?case*
**proof**(*induct rule*: *resCasesF*)
**case**(*cRes aP*)
**from** $\langle <\nu a>P \longmapsto \alpha \prec aP \rangle$ **show** *?case*
**proof**(*induct rule*: *resCasesF*)
**case**(*cRes* $P'$)
**from** $\langle P \longmapsto \alpha \prec P' \rangle$ $\langle b \sharp \alpha \rangle$ **have** $<\nu b>P \longmapsto \alpha \prec <\nu b>P'$ **by**(*rule ResF*)
**hence** $<\nu a><\nu b>P \longmapsto \alpha \prec <\nu a><\nu b>P'$ **using** $\langle a \sharp \alpha \rangle$ **by**(*rule ResF*)
**moreover have** $(<\nu a><\nu b>P', <\nu b><\nu a>P') \in Rel$ **by**(*rule ResComm*)
**ultimately show** *?case* **by** *blast*
**qed**
**qed**
**qed**
**qed**

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

**assumes** $Id \subseteq Rel$

**shows** $!P \leadsto[Rel]\ P \parallel !P$
**using** *assms*
**by**(*force simp add*: *simulation-def dest*: *Bang derivativeReflexive*)

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

**assumes** *IdRel*: $Id \subseteq Rel$

**shows** $P \parallel !P \leadsto[Rel]\ !P$
**using** *assms*
**by**(*fastforce simp add*: *pi.inject simulation-def intro*: *derivativeReflexive elim*: *bang-Cases*)

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

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

```
by(auto simp add: simulation-def)

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

  assumes xineqa: x ≠ a
  and     xineqy: x ≠ y
  and     Eqvt: eqvt Rel
  and     Id: Id ⊆ Rel

  shows <νx>a<y>.P ⇝[Rel] a<y>.(<νx>P)
using Eqvt
proof(induct rule: simCasesCont[where C=(x, y, a, P)])
  case(Bound b z P′)
  from ‹z ♯ (x, y, a, P)› have z ≠ x and z ≠ y and z ♯ P and z ≠ a by simp+
  from ‹z ♯ P› have z ♯ <νx>P by(simp add: abs-fresh)
  with ‹a<y>.(<νx>P) ⟼b«z» ≺ P′› ‹z ≠ a› ‹z ≠ y› show ?case
  proof(induct rule: inputCases)
    case cInput
    have a<y>.P ⟼a<y> ≺ P by(rule Input)
     with ‹x ≠ y› ‹x ≠ a› have <νx>a<y>.P ⟼a<y> ≺ <νx>P by(rule-tac
ResB) auto
    hence <νx>a<y>.P ⟼a<z> ≺ [(y, z)] • <νx>P using ‹z ♯ P›
     by(subst alphaBoundResidual[where x′=y]) (auto simp add: abs-fresh fresh-left
calc-atm)
     moreover from Id have derivative ([(y, z)] • <νx>P) ([(y, z)] • <νx>P)
(InputS a) z Rel
      by(rule derivativeReflexive)
    ultimately show ?case by blast
  qed
next
  case(Free α P′)
  from ‹a<y>.(<νx>P) ⟼α ≺ P′› have False by auto
  thus ?case by simp
qed

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

  assumes xineqa: x ≠ a
  and     xineqy: x ≠ y
```

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

  **shows** $a{<}y{>}.({<}\nu x{>}P) \rightsquigarrow[Rel]\ {<}\nu x{>}a{<}y{>}.P$
  **using** *Eqvt*
**proof**(*induct rule*: *simCasesCont*[**where** $C=(x, y, a, P)$])
  **case**(*Bound b z xP*)
  **from** ‹$z \sharp (x, y, a, P)$› **have** $z \neq x$ **and** $z \neq y$ **and** $z \sharp P$ **and** $z \neq a$ **by** *simp+*
  **from** ‹$z \neq a$› ‹$z \sharp P$› **have** $z \sharp a{<}y{>}.P$ **by**(*simp add*: *abs-fresh*)
  **with** ‹${<}\nu x{>}a{<}y{>}.P \longmapsto b«z» \prec xP$› ‹$z \neq x$› **show** *?case*
  **proof**(*induct rule*: *resCasesB*)
    **case**(*cOpen b P'*)
    **from** ‹$a{<}y{>}.P \longmapsto b[x] \prec P'$› **have** *False* **by** *auto*
    **thus** *?case* **by** *simp*
  **next**
    **case**(*cRes P'*)
    **from** ‹$a{<}y{>}.P \longmapsto b«z» \prec P'$›‹$z \neq a$› ‹$z \neq y$› ‹$z \sharp P$› **show** *?case*
    **proof**(*induct rule*: *inputCases*)
      **case** *cInput*
      **have** $a{<}y{>}.({<}\nu x{>}P) \longmapsto a{<}y{>} \prec ({<}\nu x{>}P)$ **by**(*rule Input*)
      **with** ‹$z \sharp P$› ‹$x \neq y$› ‹$z \neq x$› **have** $a{<}y{>}.({<}\nu x{>}P) \longmapsto a{<}z{>} \prec ({<}\nu x{>}([(y, z)] \cdot P))$
        **by**(*subst alphaBoundResidual*[**where** $x'=y$]) (*auto simp add*: *abs-fresh calc-atm fresh-left*)
      **moreover from** *Id* **have** *derivative* $({<}\nu x{>}([(y, z)] \cdot P))$ $({<}\nu x{>}([(y, z)] \cdot P))$ (*InputS a*) *z Rel*
        **by**(*rule derivativeReflexive*)
      **ultimately show** *?case* **by** *blast*
    **qed**
  **qed**
**next**
  **case**(*Free* $\alpha$ *P'*)
  **from** ‹${<}\nu x{>}a{<}y{>}.P \longmapsto \alpha \prec P'$› **have** *False* **by** *auto*
  **thus** *?case* **by** *simp*
**qed**

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

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

  **shows** ${<}\nu x{>}a\{b\}.P \rightsquigarrow[Rel]\ a\{b\}.({<}\nu x{>}P)$
**using** *assms*

**by**(*fastforce simp add*: *simulation-def elim*: *outputCases intro*: *Output ResF*)

**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* ⊆ *Rel*
  **and** *Eqvt*: *eqvt Rel*

  **shows** $a\{b\}.(<\nu x>P) \rightsquigarrow[Rel]\ <\nu x>a\{b\}.P$
**using** *assms*
**by**(*erule-tac simCasesCont*[**where** *C*=*x*])
  (*force simp add*: *abs-fresh elim*: *resCasesB resCasesF outputCases intro*: *ResF Output*)+

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

  **assumes** *Id*: *Id* ⊆ *Rel*

  **shows** $<\nu x>(\tau.(P)) \rightsquigarrow[Rel]\ \tau.(<\nu x>P)$
**using** *assms*
**by**(*force simp add*: *simulation-def elim*: *tauCases resCasesF intro*: *Tau ResF*)

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

  **assumes** *Id*: *Id* ⊆ *Rel*

  **shows** $\tau.(<\nu x>P) \rightsquigarrow[Rel]\ <\nu x>(\tau.(P))$
**using** *assms*
**by**(*force simp add*: *simulation-def elim*: *tauCases resCasesF intro*: *Tau ResF*)

**end**

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

**lemma** *nilBisim*[*dest*]:

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

**shows** $\tau.(P) \sim \mathbf{0} \Longrightarrow$ *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*
**by**(*auto dest*: *bisimE symmetric*)

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

**shows** $[a{\frown}a]P \sim P$
**proof** −
**let** *?X* = $\{([a{\frown}a]P, P), (P, [a{\frown}a]P)\}$
**have** $([a{\frown}a]P, P) \in$ *?X* **by** *simp*
**thus** *?thesis*
**by**(*coinduct rule*: *bisimCoinduct*) (*auto intro*: *matchIdLeft matchIdRight reflexive*)
**qed**

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

**assumes** $a \neq b$

**shows** $[a{\frown}b]P \sim \mathbf{0}$
**proof** −
**let** *?X* = $\{([a{\frown}b]P, \mathbf{0}), (\mathbf{0}, [a{\frown}b]P)\}$
**have** $([a{\frown}b]P, \mathbf{0}) \in$ *?X* **by** *simp*
**thus** *?thesis* **using** ‹$a \neq b$›
**by**(*coinduct rule*: *bisimCoinduct*) (*auto intro*: *matchNilLeft nilSimRight reflexive*)
**qed**

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

**assumes** $a \neq b$

**shows** $[a \neq b]P \sim P$
**proof** −
**let** *?X* = $\{([a \neq b]P, P), (P, [a \neq b]P)\}$
**have** $([a \neq b]P, P) \in$ *?X* **by** *simp*
**thus** *?thesis* **using** ‹$a \neq b$›
**by**(*coinduct rule*: *bisimCoinduct*) (*auto intro*: *mismatchIdLeft mismatchIdRight reflexive*)
**qed**

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

**shows** $[a \neq a]P \sim \mathbf{0}$
**proof** −
**let** *?X* = $\{([a \neq a]P, \mathbf{0}), (\mathbf{0}, [a \neq a]P)\}$
**have** $([a \neq a]P, \mathbf{0}) \in$ *?X* **by** *simp*
**thus** *?thesis*
**by**(*coinduct rule*: *bisimCoinduct*) (*auto intro*: *mismatchNilLeft nilSimRight reflexive*)
**qed**

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

**shows** $<\nu x>\mathbf{0} \sim \mathbf{0}$
**proof** −
**let** *?X* = $\{(<\nu x>\mathbf{0}, \mathbf{0}), (\mathbf{0}, <\nu x>\mathbf{0})\}$
**have** $(<\nu x>\mathbf{0}, \mathbf{0}) \in$ *?X* **by** *simp*
**thus** *?thesis*
**by**(*coinduct rule*: *bisimCoinduct*) (*auto intro*: *nilSimRight resNilRight*)
**qed**

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

**shows** $<\nu x><\nu y>P \sim <\nu y><\nu x>P$
**proof** −
**let** *?X* = $\{(<\nu x><\nu y>P, <\nu y><\nu x>P) \mid x\ y\ P.\ True\}$
**have** $(<\nu x><\nu y>P, <\nu y><\nu x>P) \in$ *?X* **by** *auto*
**thus** *?thesis*
**proof**(*coinduct rule*: *bisimCoinduct*)
**case**(*cSim xyP yxP*)
**{**
**fix** $x\ y\ P$

**have** $\bigwedge x\ y\ P.\ (<\nu x><\nu y>P,\ <\nu y><\nu x>P) \in\ ?X \cup bisim$ **by** *auto*
**moreover have** $Id \subseteq\ ?X \cup bisim$ **by**(*auto intro*: *reflexive*)
**moreover have** *eqvt ?X* **by**(*force simp add*: *eqvt-def*)
**hence** $eqvt(?X \cup bisim)$ **by** *auto*
**ultimately have** $<\nu x><\nu y>P \leadsto[(?X \cup bisim)]\ <\nu y><\nu x>P$ **by**(*rule resComm*)
**}**
**with** ‹$(xyP,\ yxP) \in\ ?X$› **show** *?case* **by** *auto*
**next**
**case**(*cSym xyP yxP*)
**thus** *?case* **by** *auto*
**qed**
**qed**

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

**shows** $P \oplus Q \sim Q \oplus P$
**proof** −
**let** $?X = \{(P \oplus Q,\ Q \oplus P),\ (Q \oplus P,\ P \oplus Q)\}$
**have** $(P \oplus Q,\ Q \oplus P) \in\ ?X$ **by** *simp*
**thus** *?thesis*
**by**(*coinduct rule*: *bisimCoinduct*) (*auto intro*: *reflexive sumSym*)
**qed**

**lemma** *sumIdemp*:
**fixes** $P :: pi$

**shows** $P \oplus P \sim P$
**proof** −
**let** $?X = \{(P \oplus P,\ P),\ (P,\ P \oplus P)\}$
**have** $(P \oplus P,\ P) \in\ ?X$ **by** *simp*
**thus** *?thesis*
**by**(*coinduct rule*: *bisimCoinduct*) (*auto intro*: *reflexive sumIdempLeft sumIdempRight*)
**qed**

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

**shows** $(P \oplus Q) \oplus R \sim P \oplus (Q \oplus R)$
**proof** −
**let** $?X = \{((P \oplus Q) \oplus R,\ P \oplus (Q \oplus R)),\ (P \oplus (Q \oplus R),\ (P \oplus Q) \oplus R)\}$
**have** $((P \oplus Q) \oplus R,\ P \oplus (Q \oplus R)) \in\ ?X$ **by** *simp*

**thus** *?thesis*
**by**(*coinduct rule*: *bisimCoinduct*) (*auto intro*: *reflexive sumAssocLeft sumAssocRight*)
**qed**

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

**shows** $P \oplus \mathbf{0} \sim P$
**proof** −
**let** $?X = \{(P \oplus \mathbf{0}, P), (P, P \oplus \mathbf{0})\}$
**have** $(P \oplus \mathbf{0}, P) \in ?X$ **by** *simp*
**thus** *?thesis*
**by**(*coinduct rule*: *bisimCoinduct*) (*auto intro*: *reflexive sumZeroLeft sumZeroRight*)
**qed**

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

**shows** $P \parallel \mathbf{0} \sim P$
**proof** −
**let** $?X = \{(P \parallel \mathbf{0}, P) \mid P.\ True\} \cup \{(P, P \parallel \mathbf{0}) \mid P\ .\ True\}$
**have** $(P \parallel \mathbf{0}, P) \in ?X$ **by** *blast*
**thus** *?thesis*
**by**(*coinduct rule*: *bisimCoinduct*, *auto intro*: *parZeroRight parZeroLeft*)
**qed**

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

**shows** $P \parallel Q \sim Q \parallel P$
**proof** −
**let** $?X = \{(resChain\ lst\ (P \parallel Q), resChain\ lst\ (Q \parallel P)) \mid lst\ P\ Q.\ True\}$
**have** $(P \parallel Q, Q \parallel P) \in ?X$ **by**(*blast intro*: *resChain.base*[*THEN sym*])
**thus** *?thesis*
**proof**(*coinduct rule*: *bisimCoinduct*)
**case**(*cSim PQ QP*)
**{**
**fix** *lst P Q*
**have** $\bigwedge P\ Q.\ (P \parallel Q, Q \parallel P) \in ?X \cup bisim$ **by**(*blast intro*: *resChain.base*[*THEN sym*])
**moreover have** *Res*: $\bigwedge x\ P\ Q.\ (P, Q) \in ?X \cup bisim \Longrightarrow (<\nu x>P, <\nu x>Q) \in ?X \cup bisim$
**by**(*auto intro*: *resPres resChain.step*[*THEN sym*])
**ultimately have** $P \parallel Q \rightsquigarrow[(?X \cup bisim)]\ Q \parallel P$ **by**(*rule parSym*)

**moreover have** *eqvt ?X* **by**(*force simp add*: *eqvt-def*)
**hence** *eqvt*(*?X* ∪ *bisim*) **by** *auto*
**ultimately have** *resChain lst* ($P \parallel Q$) ⇝[(*?X* ∪ *bisim*)] *resChain lst* ($Q \parallel P$) **using** *Res*
**by**(*rule resChainI*)
**}**
**with** ‹($PQ$, $QP$) ∈ *?X*› **show** *?case* **by** *auto*
**next**
**case**(*cSym PQ QP*)
**thus** *?case* **by** *auto*
**qed**
**qed**

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

**assumes** $x \sharp P$

**shows** <$\nu x$>($P \parallel Q$) ∼ $P \parallel$ <$\nu x$>$Q$
**proof** −
**let** *?X* = {(*resChain lst* (<$\nu x$>($P \parallel Q$)), *resChain lst* ($P \parallel$ <$\nu x$>$Q$)) | *lst x P Q*. $x \sharp P$} ∪
{(*resChain lst* ($P \parallel$ <$\nu x$>$Q$), *resChain lst* (<$\nu x$>($P \parallel Q$))) | *lst x P Q*. $x \sharp P$}
**let** *?Y* = *bisim O* (*?X* ∪ *bisim*) *O bisim*

**have** *Res*: $\bigwedge P\ Q\ x$. ($P$, $Q$) ∈ *?X* ⟹ (<$\nu x$>$P$, <$\nu x$>$Q$) ∈ *?X* **by**(*blast intro*: *resChain.step*[*THEN sym*])

**from** ‹$x \sharp P$› **have** (<$\nu x$>($P \parallel Q$), $P \parallel$ <$\nu x$>$Q$) ∈ *?X* **by**(*blast intro*: *resChain.base*[*THEN sym*])
**moreover have** *EqvtX*: *eqvt ?X* **by**(*fastforce simp add*: *eqvt-def name-fresh-left name-rev-per*)
**ultimately show** *?thesis*
**proof**(*coinduct rule*: *bisimTransitiveCoinduct*)
**case**(*cSim P Q*)
**{**
**fix** *P Q lst x*
**assume** ($x$::*name*) ♯ ($P$::*pi*)
**moreover have** *Id* ⊆ *?Y* **by**(*blast intro*: *reflexive*)
**moreover from** ‹*eqvt ?X*› *bisimEqvt* **have** *eqvt ?Y* **by** *blast*
**moreover have** $\bigwedge P\ Q\ x$. $x \sharp P$ ⟹ (<$\nu x$>($P \parallel Q$), $P \parallel$ <$\nu x$>$Q$) ∈ *?Y*
**by**(*blast intro*: *resChain.base*[*THEN sym*] *reflexive*)
**moreover {**
**fix** *P Q x y*
**have** <$\nu x$><$\nu y$>($P \parallel Q$) ∼ <$\nu y$><$\nu x$>($P \parallel Q$) **by**(*rule resComm*)
**moreover assume** $x \sharp P$

```
      hence (<νx>(P ∥ Q), P ∥ <νx>Q) ∈ ?X by(fastforce intro: resChain.base[THEN
sym])
        hence (<νy><νx>(P ∥ Q), <νy>(P ∥ <νx>Q)) ∈ ?X by(rule Res)
       ultimately have (<νx><νy>(P ∥ Q), <νy>(P ∥ <νx>Q)) ∈ ?Y by(blast
intro: reflexive)
      }
      ultimately have <νx>(P ∥ Q) ⇝[?Y] (P ∥ <νx>Q) by(rule scopeExtPar-
Left)
      moreover note ‹eqvt ?Y›
      moreover from Res have ⋀P Q x. (P, Q) ∈ ?Y ⟹ (<νx>P, <νx>Q) ∈
?Y
        by(blast intro: resChain.step[THEN sym] dest: resPres)
        ultimately have resChain lst (<νx>(P ∥ Q)) ⇝[?Y] resChain lst (P ∥
<νx>Q)
        by(rule resChainI)
    }
    moreover {
      fix P Q lst x
      assume (x::name) ♯ (P::pi)
      moreover have Id ⊆ ?Y by(blast intro: reflexive)
      moreover from ‹eqvt ?X› bisimEqvt have eqvt ?Y by blast
      moreover have ⋀P Q x. x ♯ P ⟹ (P ∥ <νx>Q, <νx>(P ∥ Q)) ∈ ?Y
        by(blast intro: resChain.base[THEN sym] reflexive)
      moreover {
        fix P Q x y
        have <νy><νx>(P ∥ Q) ∼ <νx><νy>(P ∥ Q) by(rule resComm)
        moreover assume x ♯ P
      hence (P ∥ <νx>Q, <νx>(P ∥ Q)) ∈ ?X by(fastforce intro: resChain.base[THEN
sym])
        hence (<νy>(P ∥ <νx>Q), <νy><νx>(P ∥ Q)) ∈ ?X by(rule Res)
       ultimately have (<νy>(P ∥ <νx>Q), <νx><νy>(P ∥ Q)) ∈ ?Y by(blast
intro: reflexive)
      }
      ultimately have (P ∥ <νx>Q) ⇝[?Y] <νx>(P ∥ Q)
        by(rule scopeExtParRight)
      moreover note ‹eqvt ?Y›
      moreover from Res have ⋀P Q x. (P, Q) ∈ ?Y ⟹ (<νx>P, <νx>Q) ∈
?Y
        by(blast intro: resChain.step[THEN sym] dest: resPres)
       ultimately have resChain lst (P ∥ <νx>Q) ⇝[?Y] resChain lst (<νx>(P
∥ Q))
        by(rule resChainI)
    }
    ultimately show ?case using ‹(P, Q) ∈ ?X› by auto
  next
    case(cSym P Q)
    thus ?case
      by auto (blast dest: symmetric transitive intro: resChain.base[THEN sym]
reflexive)+
```

```
  qed
qed

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

  assumes xFreshQ: x ♯ Q

  shows <νx>(P ∥ Q) ∼ (<νx>P) ∥ Q
proof −
  have <νx>(P ∥ Q) ∼ <νx>(Q ∥ P)
  proof −
    have P ∥ Q ∼ Q ∥ P by(rule parSym)
    thus ?thesis by(rule resPres)
  qed
  moreover from xFreshQ have <νx>(Q ∥ P) ∼ Q ∥ (<νx>P) by(rule scope-
ExtPar)
  moreover have Q ∥ <νx>P ∼ (<νx>P) ∥ Q by(rule parSym)
  ultimately show ?thesis by(blast intro: transitive)
qed

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

  shows (P ∥ Q) ∥ R ∼ P ∥ (Q ∥ R)
proof −
  let ?X = {(resChain lst ((P ∥ Q) ∥ R), resChain lst (P ∥ (Q ∥ R))) | lst P Q R.
True}
  let ?Y = bisim O (?X ∪ bisim) O bisim

  have ResX: ⋀P Q x. (P, Q) ∈ ?X ⟹ (<νx>P, <νx>Q) ∈ ?X
    by(blast intro: resChain.step[symmetric])
  hence ResY: ⋀P Q x. (P, Q) ∈ ?Y ⟹ (<νx>P, <νx>Q) ∈ ?Y
    by(blast intro: resChain.step[symmetric] dest: resPres)

  have ((P ∥ Q) ∥ R, P ∥ (Q ∥ R)) ∈ ?X by(blast intro: resChain.base[symmetric])
  moreover have eqvt ?X by(fastforce simp add: eqvt-def)
  ultimately show ?thesis
  proof(coinduct rule: bisimTransitiveCoinduct)
    case(cSim P Q)
    {
      fix P Q R lst
      have ⋀P Q R. ((P ∥ Q) ∥ R, P ∥ (Q ∥ R)) ∈ ?Y by(blast intro: reflexive
resChain.base[symmetric])
      moreover have ⋀P Q x. (P, Q) ∈ ?Y ⟹ (<νx>P, <νx>Q) ∈ ?Y by(blast
```

*intro*: *resChain.step*[*symmetric*] *resPres*)
**moreover {**
**fix** $P\ Q\ R\ x$
**have** $(<\nu x>((P \parallel Q) \parallel R), <\nu x>(P \parallel (Q \parallel R))) \in ?X$ **by**(*rule-tac ResX*)
(*blast intro*: *resChain.base*[*symmetric*])
**moreover assume** $x \sharp P$
**hence** $<\nu x>(P \parallel (Q \parallel R)) \sim P \parallel <\nu x>(Q \parallel R)$ **by**(*rule scopeExtPar*)
**ultimately have** $(<\nu x>((P \parallel Q) \parallel R), P \parallel <\nu x>(Q \parallel R)) \in ?Y$ **by**(*blast*
*intro*: *reflexive*)
**}**
**moreover {**
**fix** $P\ Q\ R\ x$
**have** $(<\nu x>((P \parallel Q) \parallel R), <\nu x>(P \parallel (Q \parallel R))) \in ?X$ **by**(*rule-tac ResX*)
(*blast intro*: *resChain.base*[*symmetric*])
**moreover assume** $x \sharp R$
**hence** $<\nu x>(P \parallel Q) \parallel R \sim <\nu x>((P \parallel Q) \parallel R)$ **by**(*metis scopeExtPar'*
*symmetric*)
**ultimately have** $(<\nu x>(P \parallel Q) \parallel R, <\nu x>(P \parallel (Q \parallel R))) \in ?Y$ **by**(*blast*
*intro*: *reflexive*)
**}**
**ultimately have** $(P \parallel Q) \parallel R \rightsquigarrow[?Y]\ P \parallel (Q \parallel R)$ **by**(*rule parAssocLeft*)
**moreover from** ‹*eqvt ?X*› *bisimEqvt* **have** *eqvt ?Y* **by** *blast*
**ultimately have** *resChain lst* $((P \parallel Q) \parallel R) \rightsquigarrow[?Y]$ *resChain lst* $(P \parallel (Q \parallel$
$R))$ **using** *ResY*
**by**(*rule resChainI*)
**}**
**with** ‹$(P, Q) \in ?X$› **show** *?case* **by** *auto*
**next**
**case**(*cSym P Q*)
**{**
**fix** $P\ Q\ R$ *lst*
**have** $P \parallel (Q \parallel R) \sim (R \parallel Q) \parallel P$ **by**(*metis parPres parSym transitive*)
**moreover have** $((R \parallel Q) \parallel P, R \parallel (Q \parallel P)) \in ?X$ **by**(*blast intro*: *resChain.base*[*symmetric*])
**moreover have** $R \parallel (Q \parallel P) \sim (P \parallel Q) \parallel R$ **by**(*metis parPres parSym*
*transitive*)
**ultimately have** $(P \parallel (Q \parallel R), (P \parallel Q) \parallel R) \in ?Y$ **by** *blast*
**hence** (*resChain lst* $(P \parallel (Q \parallel R))$, *resChain lst* $((P \parallel Q) \parallel R)) \in ?Y$ **using**
*ResY*
**by**(*induct lst*) *auto*
**}**
**with** ‹$(P, Q) \in ?X$› **show** *?case* **by** *blast*
**qed**
**qed**

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

**assumes** $x \sharp P$

**shows** $<\nu x>P \sim P$
**proof** −
**have** $<\nu x>P \sim <\nu x>P \parallel \mathbf{0}$ **by**(*rule parZero*[*THEN symmetric*])

**moreover have** $<\nu x>P \parallel \mathbf{0} \sim \mathbf{0} \parallel <\nu x>P$ **by**(*rule parSym*)
**moreover have** $\mathbf{0} \parallel <\nu x>P \sim <\nu x>(\mathbf{0} \parallel P)$ **by**(*rule scopeExtPar*[*THEN symmetric*]) *auto*
**moreover have** $<\nu x>(\mathbf{0} \parallel P) \sim <\nu x>(P \parallel \mathbf{0})$ **by**(*rule resPres*[*OF parSym*])
**moreover from** ‹$x \sharp P$› **have** $<\nu x>(P \parallel \mathbf{0}) \sim P \parallel <\nu x>\mathbf{0}$ **by**(*rule scopeExtPar*)
**moreover have** $P \parallel <\nu x>\mathbf{0} \sim <\nu x>\mathbf{0} \parallel P$ **by**(*rule parSym*)
**moreover have** $<\nu x>\mathbf{0} \parallel P \sim \mathbf{0} \parallel P$ **by**(*rule parPres*[*OF nilRes*])
**moreover have** $\mathbf{0} \parallel P \sim P \parallel \mathbf{0}$ **by**(*rule parSym*)
**moreover have** $P \parallel \mathbf{0} \sim P$ **by**(*rule parZero*)
**ultimately show** *?thesis* **by**(*metis transitive*)
**qed**

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

**shows** $<\nu x>(P \oplus Q) \sim (<\nu x>P) \oplus (<\nu x>Q)$
**proof** −
**let** $?X = \{(<\nu x>(P \oplus Q), <\nu x>P \oplus <\nu x>Q) \mid x\ P\ Q.\ True\} \cup$
$\{(<\nu x>P \oplus <\nu x>Q, <\nu x>(P \oplus Q)) \mid x\ P\ Q.\ True\}$
**have** $(<\nu x>(P \oplus Q), <\nu x>P \oplus <\nu x>Q) \in ?X$ **by** *auto*
**moreover have** *eqvt ?X* **by**(*fastforce simp add*: *eqvt-def*)
**ultimately show** *?thesis*
**by**(*coinduct rule*: *bisimCoinduct*) (*fastforce intro*: *sumResLeft sumResRight reflexive*)+
**qed**

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

**assumes** $x \sharp P$

**shows** $<\nu x>(P \oplus Q) \sim P \oplus <\nu x>Q$
**proof** −
**have** $<\nu x>(P \oplus Q) \sim <\nu x>P \oplus <\nu x>Q$ **by**(*rule sumRes*)
**moreover from** ‹$x \sharp P$› **have** $<\nu x>P \oplus <\nu x>Q \sim P \oplus <\nu x>Q$
**by**(*rule sumPres*[*OF scopeFresh*])
**ultimately show** *?thesis* **by**(*rule transitive*)
**qed**

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

 **shows** $!P \sim P \parallel !P$
**proof** −
 **let** $?X = \{(!P,\ P \parallel !P),\ (P \parallel !P,\ !P)\}$
 **have** $(!P,\ P \parallel !P) \in ?X$ **by** *simp*
 **thus** *?thesis*
  **by**(*coinduct rule*: *bisimCoinduct*) (*auto intro*: *bangLeftSC bangRightSC reflexive*)
**qed**

**lemma** *resNil*:
 **fixes** $x :: name$
 **and** $y :: name$
 **and** $P :: pi$
 **and** $b :: name$

 **shows** $<\nu x>x<y>.P \sim \mathbf{0}$
 **and** $<\nu x>x\{b\}.P \sim \mathbf{0}$
**proof** −
 **let** $?X = \{(<\nu x>x<y>.P,\ \mathbf{0}),\ (\mathbf{0},\ <\nu x>x<y>.P)\}$
 **have** $(<\nu x>x<y>.P,\ \mathbf{0}) \in ?X$ **by** *simp*
 **thus** $<\nu x>x<y>.P \sim \mathbf{0}$
  **by**(*coinduct rule*: *bisimCoinduct*) (*auto simp add*: *simulation-def*)
**next**
 **let** $?X = \{(<\nu x>x\{b\}.P,\ \mathbf{0}),\ (\mathbf{0},\ <\nu x>x\{b\}.P)\}$
 **have** $(<\nu x>x\{b\}.P,\ \mathbf{0}) \in ?X$ **by** *simp*
 **thus** $<\nu x>x\{b\}.P \sim \mathbf{0}$
  **by**(*coinduct rule*: *bisimCoinduct*) (*auto simp add*: *simulation-def*)
**qed**

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

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

 **shows** $<\nu x>a<y>.P \sim a<y>.(<\nu x>P)$
**proof** −
 **let** $?X = \{(<\nu x>a<y>.P,\ a<y>.(<\nu x>P)) \mid x\ a\ y\ P.\ x \neq a \wedge x \neq y\} \cup$
  $\{(a<y>.(<\nu x>P),\ <\nu x>a<y>.P) \mid x\ a\ y\ P.\ x \neq a \wedge x \neq y\}$
 **from** *assms* **have** $(<\nu x>a<y>.P,\ a<y>.(<\nu x>P)) \in ?X$ **by** *auto*
 **moreover have** *eqvt ?X* **by**(*fastforce simp add*: *eqvt-def pt-bij*[*OF pt-name-inst*, *OF at-name-inst*])
 **ultimately show** *?thesis*
  **by**(*coinduct rule*: *bisimCoinduct*) (*fastforce intro*: *resInputLeft reflexive resIn-*

*putRight*)+
**qed**

**lemma** *resOutput*:
  **fixes** $x$ :: *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** −
  **let** $?X = \{(<\nu x>a\{b\}.P,\ a\{b\}.(<\nu x>P)) \mid x\ a\ b\ P.\ x \neq a \wedge x \neq b\} \cup$
  $\{(a\{b\}.(<\nu x>P),\ <\nu x>a\{b\}.P) \mid x\ a\ b\ P.\ x \neq a \wedge x \neq b\}$
  **from** *assms* **have** $(<\nu x>a\{b\}.P,\ a\{b\}.(<\nu x>P)) \in ?X$ **by** *blast*
  **moreover have** *eqvt ?X* **by**(*fastforce simp add*: *eqvt-def pt-bij*[*OF pt-name-inst*, *OF at-name-inst*])
  **ultimately show** *?thesis*
  **by**(*coinduct rule*: *bisimCoinduct*) (*fastforce intro*: *resOutputLeft resOutputRight reflexive*)+
**qed**

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

  **shows** $<\nu x>\tau.(P) \sim \tau.(<\nu x>P)$
**proof** −
  **let** $?X = \{(<\nu x>\tau.(P),\ \tau.(<\nu x>P)),\ (\tau.(<\nu x>P),\ <\nu x>\tau.(P))\}$
  **have** $(<\nu x>\tau.(P),\ \tau.(<\nu x>P)) \in ?X$ **by** *auto*
  **thus** *?thesis*
  **by**(*coinduct rule*: *bisimCoinduct*) (*fastforce intro*: *resTauLeft resTauRight reflexive*)+
**qed**

**inductive** *structCong* :: *pi* ⇒ *pi* ⇒ *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** $P \sim Q$
**using** *assms*
**by**(*induct rule*: *structCong.induct*)
  (*auto intro*: *reflexive symmetric transitive sumSym sumAssoc sumZero parSym parAssoc parZero*
    *nilRes resComm resInput resOutput resTau sumRes scopeExtPar bangSC matchId mismatchId*)

**end**

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

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

  **shows** $[a \frown a]P \sim^s P$
**by**(*auto simp add*: *substClosed-def intro*: *Strong-Late-Bisim-SC.matchId*)

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

  **shows** $[a \neq a]P \sim^s \mathbf{0}$
**by**(*auto simp add*: *substClosed-def intro*: *Strong-Late-Bisim-SC.mismatchNil*)

**lemma** *scopeFresh*:

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

**assumes** *xFreshP*: *x* ♯ *P*

**shows** <ν*x*>*P* ∼$^s$ *P*
**proof**(*auto simp add*: *substClosed-def*)
**fix** *s* :: (*name* × *name*) *list*

**have** ∃ *c*::*name*. *c* ♯ (*P*, *s*) **by**(*blast intro*: *name-exists-fresh*)
**then obtain** *c*::*name* **where** *cFreshP*: *c* ♯ *P* **and** *cFreshs*: *c* ♯ *s* **by**(*force simp add*: *fresh-prod*)

**have** <ν*x*>*P* = <ν*c*>*P*
**proof** −
**from** *cFreshP* **have** <ν*x*>*P* = <ν*c*>([(*x*, *c*)] • *P*) **by**(*simp add*: *alphaRes*)
**with** *cFreshP xFreshP* **show** *?thesis* **by**(*simp add*: *name-fresh-fresh*)
**qed**

**with** *cFreshP cFreshs* **show** (<ν*x*>*P*)[<*s*>] ∼ *P*[<*s*>]
**by**(*force intro*: *Strong-Late-Bisim-SC.scopeFresh*)
**qed**

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

**shows** <ν*x*><ν*y*>*P* ∼$^s$ <ν*y*><ν*x*>*P*
**proof**(*cases x*=*y*)
**assume** *xeqy*: *x*=*y*
**have** *P* ∼$^s$ *P* **by**(*rule Strong-Late-Bisim-Subst.reflexive*)
**hence** <ν*x*>*P* ∼$^s$ <ν*x*>*P* **by**(*rule resPres*)
**hence** <ν*x*><ν*x*>*P* ∼$^s$ <ν*x*><ν*x*>*P* **by**(*rule resPres*)
**with** *xeqy* **show** *?thesis* **by** *simp*
**next**
**assume** *xineqy*: *x* ≠ *y*
**show** *?thesis*
**proof**(*auto simp add*: *substClosed-def*)
**fix** *s*::(*name* × *name*) *list*

**have** ∃ *c*::*name*. *c* ♯ (*P*, *s*, *y*) **by**(*blast intro*: *name-exists-fresh*)
**then obtain** *c*::*name* **where** *cFreshP*: *c* ♯ *P* **and** *cFreshs*: *c* ♯ *s* **and** *cineqy*: *c* ≠ *y*
**by**(*force simp add*: *fresh-prod*)

**have** ∃ *d*::*name*. *d* ♯ (*P*, *s*, *c*, *x*, *y*) **by** (*blast intro*: *name-exists-fresh*)
**then obtain** *d*::*name* **where** *dFreshP*: *d* ♯ *P* **and** *dFreshs*: *d* ♯ *s* **and** *dineqc*: *d* ≠ *c*

**and** *dineqx*: $d \neq x$ **and** *dineqy*: $d \neq y$
**by**(*force simp add*: *fresh-prod*)

**have** $<\nu x><\nu y>P = <\nu c><\nu d>([(x, c)] \cdot [(y, d)] \cdot P)$
**proof** −
**from** *cineqy cFreshP* **have** *cFreshyP*: $c \sharp <\nu y>P$ **by**(*simp add*: *name-fresh-abs*)
**from** *dFreshP* **have** $<\nu y>P = <\nu d>([(y, d)] \cdot P)$ **by**(*rule alphaRes*)
**moreover from** *cFreshyP* **have** $<\nu x><\nu y>P = <\nu c>([(x, c)] \cdot (<\nu y>P))$
**by**(*rule alphaRes*)
**ultimately show** *?thesis* **using** *dineqc dineqx* **by**(*simp add*: *name-calc*)
**qed**
**moreover have** $<\nu y><\nu x>P = <\nu d><\nu c>([(x, c)] \cdot [(y, d)] \cdot P)$
**proof** −
**from** *dineqx dFreshP* **have** *dFreshxP*: $d \sharp <\nu x>P$ **by**(*simp add*: *name-fresh-abs*)
**from** *cFreshP* **have** $<\nu x>P = <\nu c>([(x, c)] \cdot P)$ **by**(*rule alphaRes*)
**moreover from** *dFreshxP* **have** $<\nu y><\nu x>P = <\nu d>([(y, d)] \cdot (<\nu x>P))$
**by**(*rule alphaRes*)
**ultimately have** $<\nu y><\nu x>P = <\nu d><\nu c>([(y, d)] \cdot [(x, c)] \cdot P)$ **using**
*dineqc cineqy*
**by**(*simp add*: *name-calc*)
**thus** *?thesis* **using** *dineqx dineqc cineqy xineqy*
**by**(*subst name-perm-compose*, *simp add*: *name-calc*)
**qed**

**ultimately show** $(<\nu x><\nu y>P)[<s>] \sim (<\nu y><\nu x>P)[<s>]$ **using** *cFreshs*
*dFreshs*
**by**(*force intro*: *Strong-Late-Bisim-SC.resComm*)
**qed**
**qed**

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

**shows** $P \oplus \mathbf{0} \sim^s P$
**by**(*force simp add*: *substClosed-def intro*: *Strong-Late-Bisim-SC.sumZero*)

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

**shows** $P \oplus Q \sim^s Q \oplus P$
**by**(*force simp add*: *substClosed-def intro*: *Strong-Late-Bisim-SC.sumSym*)

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

**shows** $(P \oplus Q) \oplus R \sim^s P \oplus (Q \oplus R)$

**by**(*force simp add*: *substClosed-def intro*: *Strong-Late-Bisim-SC.sumAssoc*)

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

  **shows** $<\nu x>(P \oplus Q) \sim^s <\nu x>P \oplus <\nu x>Q$
**proof**(*auto simp add*: *substClosed-def*)
  **fix** $s :: (name \times name)\ list$

  **have** $\exists c::name.\ c \sharp (P, Q, s)$ **by**(*blast intro*: *name-exists-fresh*)
  **then obtain** $c::name$ **where** *cFreshP*: $c \sharp P$ **and** *cFreshQ*: $c \sharp Q$ **and** *cFreshs*: $c \sharp s$
    **by**(*force simp add*: *fresh-prod*)

  **have** $<\nu x>(P \oplus Q) = <\nu c>(([(x, c)] \cdot P) \oplus ([(x, c)] \cdot Q))$
  **proof** −
    **from** *cFreshP cFreshQ* **have** $c \sharp P \oplus Q$ **by** *simp*
    **hence** $<\nu x>(P \oplus Q) = <\nu c>([(x, c)] \cdot (P \oplus Q))$ **by**(*simp add*: *alphaRes*)
    **thus** *?thesis* **by**(*simp add*: *name-fresh-fresh*)
  **qed**

  **moreover from** *cFreshP* **have** $<\nu x>P = <\nu c>([(x, c)] \cdot P)$ **by**(*simp add*: *alphaRes*)
  **moreover from** *cFreshQ* **have** $<\nu x>Q = <\nu c>([(x, c)] \cdot Q)$ **by**(*simp add*: *alphaRes*)

  **ultimately show** $(<\nu x>(P \oplus Q))[<s>] \sim (<\nu x>P)[<s>] \oplus (<\nu x>Q)[<s>]$ **using** *cFreshs*
    **by**(*force intro*: *Strong-Late-Bisim-SC.sumRes*)
**qed**

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

  **assumes** *xFreshP*: $x \sharp P$

  **shows** $<\nu x>(P \oplus Q) \sim^s P \oplus <\nu x>Q$
**proof**(*auto simp add*: *substClosed-def*)
  **fix** $s :: (name \times name)\ list$

  **have** $\exists c::name.\ c \sharp (P, Q, s)$ **by**(*blast intro*: *name-exists-fresh*)
  **then obtain** $c::name$ **where** *cFreshP*: $c \sharp P$ **and** *cFreshQ*: $c \sharp Q$ **and** *cFreshs*: $c \sharp s$
    **by**(*force simp add*: *fresh-prod*)

**have** $<\nu x>(P \oplus Q) = <\nu c>(P \oplus ([(x, c)] \cdot Q))$
**proof** −
**from** *cFreshP cFreshQ* **have** $c \sharp P \oplus Q$ **by** *simp*
**hence** $<\nu x>(P \oplus Q) = <\nu c>([(x, c)] \cdot (P \oplus Q))$ **by**(*simp add*: *alphaRes*)
**with** *xFreshP cFreshP* **show** *?thesis* **by**(*simp add*: *name-fresh-fresh*)
**qed**

**moreover from** *cFreshQ* **have** $<\nu x>Q = <\nu c>([(x, c)] \cdot Q)$ **by**(*simp add*: *alphaRes*)

**ultimately show** $(<\nu x>(P \oplus Q))[<s>] \sim P[<s>] \oplus (<\nu x>Q)[<s>]$ **using** *cFreshs cFreshP*
**by**(*force intro*: *Strong-Late-Bisim-SC.scopeExtSum*)
**qed**

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

**shows** $P \parallel \mathbf{0} \sim^s P$
**by**(*force simp add*: *substClosed-def intro*: *Strong-Late-Bisim-SC.parZero*)

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

**shows** $P \parallel Q \sim^s Q \parallel P$
**by**(*force simp add*: *substClosed-def intro*: *Strong-Late-Bisim-SC.parSym*)

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

**shows** $(P \parallel Q) \parallel R \sim^s P \parallel (Q \parallel R)$
**by**(*force simp add*: *substClosed-def intro*: *Strong-Late-Bisim-SC.parAssoc*)

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

**assumes** *xFreshP*: $x \sharp P$

**shows** $<\nu x>(P \parallel Q) \sim^s P \parallel <\nu x>Q$
**proof**(*auto simp add*: *substClosed-def*)
**fix** $s :: (name \times name)\ list$

**have** $\exists c{::}name.\ c \sharp (P, Q, s)$ **by**(*blast intro*: *name-exists-fresh*)
**then obtain** $c{::}name$ **where** *cFreshP*: $c \sharp P$ **and** *cFreshQ*: $c \sharp Q$ **and** *cFreshs*:

$c \sharp s$
  **by**(*force simp add*: *fresh-prod*)

**have** $<\nu x>(P \parallel Q) = <\nu c>(P \parallel ([(x, c)] \cdot Q))$
**proof** −
  **from** *cFreshP cFreshQ* **have** $c \sharp P \parallel Q$ **by** *simp*
  **hence** $<\nu x>(P \parallel Q) = <\nu c>([(x, c)] \cdot (P \parallel Q))$ **by**(*simp add*: *alphaRes*)
  **with** *xFreshP cFreshP* **show** *?thesis* **by**(*simp add*: *name-fresh-fresh*)
**qed**

**moreover from** *cFreshQ* **have** $<\nu x>Q = <\nu c>([(x, c)] \cdot Q)$ **by**(*simp add*: *alphaRes*)

**ultimately show** $(<\nu x>(P \parallel Q))[<s>] \sim P[<s>] \parallel (<\nu x>Q)[<s>]$ **using** *cFreshs cFreshP*
  **by**(*force intro*: *Strong-Late-Bisim-SC.scopeExtPar*)
**qed**

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

  **assumes** *xFreshP*: $x \sharp Q$

  **shows** $<\nu x>(P \parallel Q) \sim^s (<\nu x>P) \parallel Q$
**proof** −
  **have** $<\nu x>(P \parallel Q) \sim^s <\nu x>(Q \parallel P)$ **by**(*blast intro*: *parSym resPres*)
  **moreover from** *xFreshP* **have** $<\nu x>(Q \parallel P) \sim^s Q \parallel <\nu x>P$ **by**(*rule scopeExtPar*)
  **moreover have** $Q \parallel <\nu x>P \sim^s (<\nu x>P) \parallel Q$ **by**(*rule parSym*)
  **ultimately show** *?thesis* **by** (*blast intro*: *transitive*)
**qed**

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

  **shows** $!P \sim^s P \parallel !P$
**by**(*auto simp add*: *substClosed-def intro*: *Strong-Late-Bisim-SC.bangSC*)

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

  **shows** $<\nu x>\mathbf{0} \sim^s \mathbf{0}$
**proof**(*auto simp add*: *substClosed-def*)
  **fix** $\sigma$::(*name* × *name*) *list*
  **obtain** $y$::*name* **where** $y \sharp \sigma$
    **by**(*generate-fresh name*) *auto*
  **have** $<\nu y>\mathbf{0} \sim \mathbf{0}$ **by** (*rule Strong-Late-Bisim-SC.nilRes*)

**with** $\langle y \sharp \sigma\rangle$ **have** $(<\nu y>\mathbf{0})[<\sigma>] \sim \mathbf{0}$ **by** *simp*
**thus** $(<\nu x>\mathbf{0})[<\sigma>] \sim \mathbf{0}$
**by**(*subst alphaRes*[**where** $c=y$]) *auto*
**qed**

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

**shows** $<\nu x>(\tau.(P)) \sim^s \tau.(<\nu x>P)$
**proof**(*auto simp add*: *substClosed-def*)
**fix** $\sigma$::(*name* $\times$ *name*) *list*
**obtain** $y$::*name* **where** $y \sharp P$ **and** $y \sharp \sigma$
**by**(*generate-fresh name*, *auto*)
**have** $<\nu y>(\tau.((([(x, y)] \cdot P)[<\sigma>])) \sim \tau.(<\nu y>((([(x, y)] \cdot P)[<\sigma>]))$
**by**(*rule resTau*)
**with** $\langle y \sharp \sigma\rangle$ **have** $(<\nu y>(\tau.([(x, y)] \cdot P)))[<\sigma>] \sim (\tau.(<\nu y>([(x, y)] \cdot P)))[<\sigma>]$
**by** *simp*
**with** $\langle y \sharp P\rangle$ **show** $(<\nu x>\tau.(P))[<\sigma>] \sim \tau.((<\nu x>P)[<\sigma>])$
**apply**(*subst alphaRes*[**where** $c=y$])
**apply** *simp*
**apply**(*subst alphaRes*[**where** $c=y$ **and** $a=x$])
**by** *simp+*
**qed**

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

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

**shows** $<\nu x>(a\{b\}.(P)) \sim^s a\{b\}.(<\nu x>P)$
**proof**(*auto simp add*: *substClosed-def*)
**fix** $\sigma$::(*name* $\times$ *name*) *list*
**obtain** $y$::*name* **where** $y \sharp P$ **and** $y \sharp \sigma$ **and** $y \neq a$ **and** $y \neq b$
**by**(*generate-fresh name*, *auto*)
**have** $<\nu y>((\textit{seq-subst-name}\ a\ \sigma)\{\textit{seq-subst-name}\ b\ \sigma\}.((([(x, y)] \cdot P)[<\sigma>])) \sim \textit{seq-subst-name}\ a\ \sigma\{\textit{seq-subst-name}\ b\ \sigma\}.(<\nu y>((([(x, y)] \cdot P)[<\sigma>]))$
**using** $\langle y \neq a\rangle$ $\langle y \neq b\rangle$ $\langle y \sharp \sigma\rangle$ *freshSeqSubstName*
**by**(*rule-tac resOutput*) *auto*
**with** $\langle y \sharp \sigma\rangle$ **have** $(<\nu y>(a\{b\}.([(x, y)] \cdot P)))[<\sigma>] \sim (a\{b\}.(<\nu y>([(x, y)] \cdot P)))[<\sigma>]$
**by** *simp*
**with** $\langle y \sharp P\rangle$ $\langle y \neq a\rangle$ $\langle y \neq b\rangle$ $\langle x \neq a\rangle$ $\langle x \neq b\rangle$ **show** $(<\nu x>a\{b\}.(P))[<\sigma>] \sim \textit{seq-subst-name}\ a\ \sigma\{\textit{seq-subst-name}\ b\ \sigma\}.((<\nu x>P)[<\sigma>])$
**apply**(*subst alphaRes*[**where** $c=y$])

**apply** *simp*
**apply**(*subst alphaRes*[**where** $c=y$ **and** $a=x$])
**by** *simp*+
**qed**

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

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

**shows** $<\nu x>(a<y>.(P)) \sim^s a<y>.(<\nu x>P)$
**proof**(*auto simp add*: *substClosed-def*)
**fix** $\sigma$::(*name* $\times$ *name*) *list*
**obtain** $x'$::*name* **where** $x' \sharp P$ **and** $x' \sharp \sigma$ **and** $x' \neq a$ **and** $x' \neq x$ **and** $x' \neq y$
**by**(*generate-fresh name*, *auto*)
**obtain** $y'$::*name* **where** $y' \sharp P$ **and** $y' \sharp \sigma$ **and** $y' \neq a$ **and** $y' \neq x$ **and** $y' \neq y$ **and** $x' \neq y'$
**by**(*generate-fresh name*, *auto*)
**have** $<\nu x'>((\textit{seq-subst-name}\ a\ \sigma)<y'>.(([(y, y')] \cdot [(x, x')] \cdot P)[<\sigma>])) \sim \textit{seq-subst-name}\ a\ \sigma<y'>.(<\nu x'>(([(y, y')] \cdot [(x, x')] \cdot P)[<\sigma>]))$
**using** ‹$x' \neq a$› ‹$x' \neq y'$› ‹$x' \sharp \sigma$› ‹$y' \sharp \sigma$› *freshSeqSubstName*
**by**(*rule-tac resInput*) *auto*
**with** ‹$x' \sharp \sigma$› ‹$y' \sharp \sigma$› **have** $(<\nu x'>(a<y'>.([(y, y')] \cdot [(x, x')] \cdot P)))[<\sigma>] \sim (a<y'>.(<\nu x'>([(y, y')] \cdot [(x, x')] \cdot P)))[<\sigma>]$
**by** *simp*
**with** ‹$x' \sharp P$› ‹$y' \neq x$› ‹$x' \neq y$› ‹$y' \sharp P$› ‹$x' \neq y'$› ‹$x' \neq a$› ‹$y' \neq a$› ‹$x \neq a$› ‹$x \neq y$› **show** $(<\nu x>a<y>.(P))[<\sigma>] \sim a<y>.(<\nu x>P)[<\sigma>]$
**apply**(*subst alphaInput*[**where** $c=y'$])
**apply** *simp*
**apply**(*subst alphaRes*[**where** $c=x'$])
**apply**(*simp add*: *abs-fresh fresh-left calc-atm*)
**apply**(*simp add*: *eqvts calc-atm*)
**apply**(*subst alphaRes*[**where** $c=x'$ **and** $a=x$])
**apply** *simp*
**apply**(*subst alphaInput*[**where** $c=y'$ **and** $x=y$])
**apply**(*simp add*: *abs-fresh fresh-left calc-atm*)
**apply**(*simp add*: *eqvts calc-atm*)
**apply**(*subst perm-compose*)
**by**(*simp add*: *eqvts calc-atm*)
**qed**

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

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

**using** *assms*
**apply**(*induct rule*: *structCong.induct*)
**by**(*auto intro*: *reflexive symmetric transitive sumSym sumAssoc sumZero parSym parAssoc parZero*
*nilRes resComm resInput resOutput resTau sumRes scopeExtPar bangSC matchId mismatchId*)

**end**

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

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

**shows** $<\nu a><\nu b>P \simeq^s <\nu b><\nu a>P$
**proof** −
**have** $<\nu a><\nu b>P \sim^s <\nu b><\nu a>P$
**by**(*rule Strong-Late-Bisim-Subst-SC.resComm*)
**thus** *?thesis* **by**(*rule strongEqWeakCong*)
**qed**

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

**shows** $[a\frown a]P \simeq^s P$
**proof** −
**have** $[a\frown a]P \sim^s P$ **by**(*rule Strong-Late-Bisim-Subst-SC.matchId*)
**thus** *?thesis* **by**(*rule strongEqWeakCong*)
**qed**

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

**shows** $[a{\neq}a]P \simeq^s \mathbf{0}$
**proof** −
**have** $[a{\neq}a]P \sim^s \mathbf{0}$ **by**(*rule Strong-Late-Bisim-Subst-SC.mismatchNil*)
**thus** *?thesis* **by**(*rule strongEqWeakCong*)
**qed**

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

**shows** $P \oplus Q \simeq^s Q \oplus P$
**proof** −
**have** $P \oplus Q \sim^s Q \oplus P$ **by**(*rule Strong-Late-Bisim-Subst-SC.sumSym*)
**thus** *?thesis* **by**(*rule strongEqWeakCong*)
**qed**

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

**shows** $(P \oplus Q) \oplus R \simeq^s P \oplus (Q \oplus R)$
**proof** −
**have** $(P \oplus Q) \oplus R \sim^s P \oplus (Q \oplus R)$ **by**(*rule Strong-Late-Bisim-Subst-SC.sumAssoc*)
**thus** *?thesis* **by**(*rule strongEqWeakCong*)
**qed**

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

**shows** $P \oplus \mathbf{0} \simeq^s P$
**proof** −
**have** $P \oplus \mathbf{0} \sim^s P$ **by**(*rule Strong-Late-Bisim-Subst-SC.sumZero*)
**thus** *?thesis* **by**(*rule strongEqWeakCong*)
**qed**

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

**shows** $P \parallel \mathbf{0} \simeq^s P$
**proof** −
**have** $P \parallel \mathbf{0} \sim^s P$ **by**(*rule Strong-Late-Bisim-Subst-SC.parZero*)
**thus** *?thesis* **by**(*rule strongEqWeakCong*)
**qed**

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

**shows** $P \parallel Q \simeq^s Q \parallel P$
**proof** −
**have** $P \parallel Q \sim^s Q \parallel P$ **by**(*rule Strong-Late-Bisim-Subst-SC.parSym*)
**thus** *?thesis* **by**(*rule strongEqWeakCong*)
**qed**

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

**assumes** $x \sharp P$

**shows** $<\nu x>(P \parallel Q) \simeq^s P \parallel <\nu x>Q$
**proof** −
**from** *assms* **have** $<\nu x>(P \parallel Q) \sim^s P \parallel <\nu x>Q$ **by**(*rule Strong-Late-Bisim-Subst-SC.scopeExtPar*)
**thus** *?thesis* **by**(*rule strongEqWeakCong*)
**qed**

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

**assumes** *xFreshQ*: $x \sharp Q$

**shows** $<\nu x>(P \parallel Q) \simeq^s (<\nu x>P) \parallel Q$
**proof** −
**from** *assms* **have** $<\nu x>(P \parallel Q) \sim^s (<\nu x>P) \parallel Q$ **by**(*rule Strong-Late-Bisim-Subst-SC.scopeExtPar'*)
**thus** *?thesis* **by**(*rule strongEqWeakCong*)
**qed**

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

**shows** $(P \parallel Q) \parallel R \simeq^s P \parallel (Q \parallel R)$
**proof** −
**have** $(P \parallel Q) \parallel R \sim^s P \parallel (Q \parallel R)$ **by**(*rule Strong-Late-Bisim-Subst-SC.parAssoc*)
**thus** *?thesis* **by**(*rule strongEqWeakCong*)
**qed**

**lemma** *scopeFresh*:
**fixes** $P$ :: *pi*

**and** $a$ :: *name*

**assumes** *aFreshP*: $a \sharp P$

**shows** $<\nu a>P \simeq^s P$
**proof** $-$
**from** *assms* **have** $<\nu a>P \sim^s P$ **by**(*rule Strong-Late-Bisim-Subst-SC.scopeFresh*)
**thus** *?thesis* **by**(*rule strongEqWeakCong*)
**qed**

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

**assumes** $x \sharp P$

**shows** $<\nu x>(P \oplus Q) \simeq^s P \oplus <\nu x>Q$
**proof** $-$
**from** *assms* **have** $<\nu x>(P \oplus Q) \sim^s P \oplus <\nu x>Q$ **by**(*rule Strong-Late-Bisim-Subst-SC.scopeExtSum*)
**thus** *?thesis* **by**(*rule strongEqWeakCong*)
**qed**

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

**shows** $!P \simeq^s P \parallel !P$
**proof** $-$
**have** $!P \sim^s P \parallel !P$ **by**(*rule Strong-Late-Bisim-Subst-SC.bangSC*)
**thus** *?thesis* **by**(*rule strongEqWeakCong*)
**qed**

**end**

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

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

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

**shows** $\tau.(P) \rightsquigarrow<Rel> \tau.(Q)$
**proof**(*induct rule*: *simCases*)
**case**(*Bound Q' a y*)

**have** $\tau.(Q) \longmapsto a{<}\nu y{>} \prec Q'$ **by** *fact*
**hence** *False* **by** *auto*
**thus** *?case* **by** *simp*
**next**
**case**(*Input* $Q'$ *a x*)
**have** $\tau.(Q) \longmapsto a{<}x{>} \prec Q'$ **by** *fact*
**hence** *False* **by** *auto*
**thus** *?case* **by** *simp*
**next**
**case**(*Free* $Q'$ $\alpha$)
**have** $\tau.(Q) \longmapsto \alpha \prec Q'$ **by** *fact*
**thus** *?case* **using** *PRelQ*
**proof**(*induct rule*: *tauCases*, *auto simp add*: *pi.inject residual.inject*)
**have** $\tau.(P) \Longrightarrow_l \tau \prec P$ **by**(*rule Weak-Late-Step-Semantics.Tau*)
**moreover assume** $(P,\ Q') \in Rel$
**ultimately show** $\exists P'.\ \tau.(P) \Longrightarrow_l \tau \prec P' \wedge (P',\ Q') \in Rel$ **by** *blast*
**qed**
**qed**

**lemma** *inputPres*:
**fixes** $P$ :: *pi*
**and** $Q$ :: *pi*
**and** $a$ :: *name*
**and** $x$ :: *name*
**and** *Rel* :: $(pi \times pi)$ *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** −
**show** *?thesis* **using** *Eqvt*
**proof**(*induct rule*: *simCasesCont*[*of* - $(P,\ a,\ x,\ Q)$])
**case**(*Bound* $Q'$ *b y*)
**have** $a{<}x{>}.Q \longmapsto b{<}\nu y{>} \prec Q'$ **by** *fact*
**hence** *False* **by** *auto*
**thus** *?case* **by** *simp*
**next**
**case**(*Input* $Q'$ *b y*)
**have** $y \sharp (P,\ a,\ x,\ Q)$ **by** *fact*
**hence** *yFreshP*: $(y{::}name) \sharp P$ **and** *yineqx*: $y \neq x$ **and** $y \neq a$ **and** $y \sharp Q$
**by**(*simp add*: *fresh-prod*)+
**have** $a{<}x{>}.Q \longmapsto b{<}y{>} \prec Q'$ **by** *fact*
**thus** *?case* **using** ‹$y \neq a$› ‹$y \neq x$› ‹$y \sharp Q$›
**proof**(*induct rule*: *inputCases*, *auto simp add*: *subject.inject*)
**have** $\forall u.\ \exists P'.\ a{<}x{>}.P \Longrightarrow_l u\ in\ ([(x,\ y)] \cdot P) \rightarrow a{<}y{>} \prec P' \wedge (P',\ ([(x,\ y)] \cdot Q)[y{::=}u]) \in Rel$
**proof**(*rule allI*)
**fix** $u$

**have** $a{<}x{>}.P \Longrightarrow_l u$ *in* $([(x, y)] \cdot P) \rightarrow a{<}y{>} \prec ([(x, y)] \cdot P)[y::=u]$ (**is** *?goal*)
**proof** −
**from** *yFreshP* **have** $a{<}x{>}.P = a{<}y{>}.([(x, y)] \cdot P)$ **by**(*rule Agent.alphaInput*)
**moreover have** $a{<}y{>}.([(x, y)] \cdot P) \Longrightarrow_l u$ *in* $([(x, y)] \cdot P) \rightarrow a{<}y{>} \prec ([(x, y)] \cdot P)[y::=u]$
**by**(*rule Weak-Late-Step-Semantics.Input*)
**ultimately show** *?goal* **by**(*simp add*: *name-swap*)
**qed**

**moreover have** $(([(x, y)] \cdot P)[y::=u], ([(x, y)] \cdot Q)[y::=u]) \in Rel$
**proof** −
**from** *PRelQ* **have** $(P[x::=u], Q[x::=u]) \in Rel$ **by** *auto*
**with** $\langle y \sharp P\rangle$ $\langle y \sharp Q\rangle$ **show** *?thesis* **by**(*simp add*: *renaming*)
**qed**

**ultimately show** $\exists P'. a{<}x{>}.P \Longrightarrow_l u$ *in* $([(x, y)] \cdot P) \rightarrow a{<}y{>} \prec P' \wedge (P', ([(x, y)] \cdot Q)[y::=u]) \in Rel$
**by** *blast*
**qed**

**thus** $\exists P''. \forall u. \exists P'. a{<}x{>}.P \Longrightarrow_l u$ *in* $P'' \rightarrow a{<}y{>} \prec P' \wedge (P', ([(x, y)] \cdot Q)[y::=u]) \in Rel$ **by** *blast*
**qed**
**next**
**case**(*Free* $Q'$ $\alpha$)
**have** $a{<}x{>}.Q \longmapsto \alpha \prec Q'$ **by** *fact*
**hence** *False* **by** *auto*
**thus** *?case* **by** *simp*
**qed**
**qed**

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

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

**shows** $a\{b\}.P \leadsto{<}Rel{>}\ a\{b\}.Q$
**proof**(*induct rule*: *simCases*)
**case**(*Bound* $Q'$ $c$ $x$)
**have** $a\{b\}.Q \longmapsto c{<}\nu x{>} \prec Q'$ **by** *fact*
**hence** *False* **by** *auto*
**thus** *?case* **by** *simp*
**next**

**case**(*Input* $Q'$ *c x*)
**have** $a\{b\}.Q \longmapsto c{<}x{>} \prec Q'$ **by** *fact*
**hence** *False* **by** *auto*
**thus** *?case* **by** *simp*
**next**
**case**(*Free* $Q'$ $\alpha$)
**have** $a\{b\}.Q \longmapsto \alpha \prec Q'$ **by** *fact*
**thus** *?case* **using** *PRelQ*
**proof**(*induct rule*: *outputCases*, *auto simp add*: *pi.inject residual.inject*)
**have** $a\{b\}.P \Longrightarrow_l a[b] \prec P$ **by**(*rule Weak-Late-Step-Semantics.Output*)
**moreover assume** $(P,\ Q') \in Rel$
**ultimately show** $\exists P'.\ a\{b\}.P \Longrightarrow_l a[b] \prec P' \wedge (P',\ Q') \in Rel$ **by** *blast*
**qed**
**qed**

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

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

**shows** $[a\frown b]P \leadsto{<}Rel'{>}\ [a\frown b]Q$
**proof**(*induct rule*: *simCases*)
**case**(*Bound* $Q'$ *c x*)
**have** $x \sharp [a\frown b]P$ **by** *fact*
**hence** *xFreshP*: $(x{::}name) \sharp P$ **by** *simp*
**have** $[a\frown b]Q \longmapsto c{<}\nu x{>} \prec Q'$ **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *matchCases*)
**case** *cMatch*
**have** $Q \longmapsto c{<}\nu x{>} \prec Q'$ **by** *fact*
**with** *PSimQ xFreshP* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l c{<}\nu x{>} \prec P'$
**and** *P'RelQ'*: $(P',\ Q') \in Rel$
**by**(*blast dest*: *simE*)
**from** *PTrans* **have** $[a\frown a]P \Longrightarrow_l c{<}\nu x{>} \prec P'$ **by**(*rule Weak-Late-Step-Semantics.Match*)
**moreover from** *P'RelQ' RelRel'* **have** $(P',\ Q') \in Rel'$ **by** *blast*
**ultimately show** *?case* **by** *blast*
**qed**
**next**
**case**(*Input* $Q'$ *c x*)
**have** $x \sharp [a\frown b]P$ **by** *fact*
**hence** *xFreshP*: $(x{::}name) \sharp P$ **by** *simp*
**have** $[a\frown b]Q \longmapsto c{<}x{>} \prec Q'$ **by** *fact*
**thus** *?case*

**proof**(*induct rule*: *matchCases*)
**case** *cMatch*
**have** $Q \longmapsto c{<}x{>} \prec Q'$ **by** *fact*
**with** *PSimQ xFreshP* **obtain** $P''$ **where** *L1*: $\forall u.\ \exists P'.\ P \Longrightarrow_l u\ in\ P''{\rightarrow}c{<}x{>} \prec P' \wedge (P',\ Q'[x{::=}u]) \in Rel$
**by**(*blast dest*: *simE*)
**have** $\forall u.\ \exists P'.\ [a{\frown}a]P \Longrightarrow_l u\ in\ P''{\rightarrow}c{<}x{>} \prec P' \wedge (P',\ Q'[x{::=}u]) \in Rel'$
**proof**(*rule allI*)
**fix** $u$
**from** *L1* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l u\ in\ P''{\rightarrow}c{<}x{>} \prec P'$ **and** *P'RelQ'*: $(P',\ Q'[x{::=}u]) \in Rel$
**by** *blast*
**from** *PTrans* **have** $[a{\frown}a]P \Longrightarrow_l u\ in\ P''{\rightarrow}c{<}x{>} \prec P'$ **by**(*rule Weak-Late-Step-Semantics.Match*)
**with** *P'RelQ' RelRel'* **show** $\exists P'.\ [a{\frown}a]P \Longrightarrow_l u\ in\ P''{\rightarrow}c{<}x{>} \prec P' \wedge (P',\ Q'[x{::=}u]) \in Rel'$
**by** *blast*
**qed**
**thus** *?case* **by** *blast*
**qed**
**next**
**case**(*Free* $Q'$ $\alpha$)
**have** $[a{\frown}b]Q \longmapsto \alpha \prec Q'$ **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *matchCases*)
**case** *cMatch*
**have** $Q \longmapsto \alpha \prec Q'$ **by** *fact*
**with** *PSimQ* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l \alpha \prec P'$ **and** *PRel*: $(P',\ Q') \in Rel$
**by**(*blast dest*: *simE*)
**from** *PTrans* **have** $[a{\frown}a]P \Longrightarrow_l \alpha \prec P'$ **by**(*rule Weak-Late-Step-Semantics.Match*)
**with** *PRel RelRel'* **show** *?case* **by** *blast*
**qed**
**qed**

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

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

**shows** $[a{\neq}b]P \leadsto{<}Rel'{>}\ [a{\neq}b]Q$
**proof**(*cases* $a{=}b$)
**assume** $a{=}b$
**thus** *?thesis*

**by**(*auto simp add*: *weakStepSimDef*)
**next**
**assume** *aineqb*: $a \neq b$
**show** *?thesis*
**proof**(*induct rule*: *simCases*)
**case**(*Bound* $Q'$ *c x*)
**have** $x \sharp [a \neq b]P$ **by** *fact*
**hence** *xFreshP*: $(x::name) \sharp P$ **by** *simp*
**have** $[a \neq b]Q \longmapsto c{<}\nu x{>} \prec Q'$ **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *mismatchCases*)
**case** *cMismatch*
**have** $Q \longmapsto c{<}\nu x{>} \prec Q'$ **by** *fact*
**with** *PSimQ xFreshP* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l c{<}\nu x{>} \prec P'$
**and** $P'RelQ'$: $(P', Q') \in Rel$
**by**(*blast dest*: *simE*)
**from** *PTrans aineqb* **have** $[a \neq b]P \Longrightarrow_l c{<}\nu x{>} \prec P'$ **by**(*rule Weak-Late-Step-Semantics.Mismatch*)
**moreover from** $P'RelQ'$ $RelRel'$ **have** $(P', Q') \in Rel'$ **by** *blast*
**ultimately show** *?case* **by** *blast*
**qed**
**next**
**case**(*Input* $Q'$ *c x*)
**have** $x \sharp [a \neq b]P$ **by** *fact*
**hence** *xFreshP*: $(x::name) \sharp P$ **by** *simp*
**have** $[a \neq b]Q \longmapsto c{<}x{>} \prec Q'$ **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *mismatchCases*)
**case** *cMismatch*
**have** $Q \longmapsto c{<}x{>} \prec Q'$ **by** *fact*
**with** *PSimQ xFreshP* **obtain** $P''$ **where** *L1*: $\forall u.\ \exists P'.\ P \Longrightarrow_l u\ in\ P'' \rightarrow c{<}x{>} \prec P' \wedge (P', Q'[x::=u]) \in Rel$
**by**(*blast dest*: *simE*)
**have** $\forall u.\ \exists P'.\ [a \neq b]P \Longrightarrow_l u\ in\ P'' \rightarrow c{<}x{>} \prec P' \wedge (P', Q'[x::=u]) \in Rel'$
**proof**(*rule allI*)
**fix** *u*
**from** *L1* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l u\ in\ P'' \rightarrow c{<}x{>} \prec P'$ **and** $P'RelQ'$: $(P', Q'[x::=u]) \in Rel$
**by** *blast*
**from** *PTrans aineqb* **have** $[a \neq b]P \Longrightarrow_l u\ in\ P'' \rightarrow c{<}x{>} \prec P'$ **by**(*rule Weak-Late-Step-Semantics.Mismatch*)
**with** $P'RelQ'$ $RelRel'$ **show** $\exists P'.\ [a \neq b]P \Longrightarrow_l u\ in\ P'' \rightarrow c{<}x{>} \prec P' \wedge (P', Q'[x::=u]) \in Rel'$
**by** *blast*
**qed**
**thus** *?case* **by** *blast*
**qed**
**next**
**case**(*Free* $Q'$ $\alpha$)
**have** $[a \neq b]Q \longmapsto \alpha \prec Q'$ **by** *fact*

**thus** *?case*
**proof**(*induct rule*: *mismatchCases*)
**case** *cMismatch*
**have** $Q \longmapsto \alpha \prec Q'$ **by** *fact*
**with** *PSimQ* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l \alpha \prec P'$ **and** *PRel*: $(P', Q')$ $\in$ *Rel*
**by**(*blast dest*: *simE*)
**from** *PTrans* ‹$a \neq b$› **have** $[a{\neq}b]P \Longrightarrow_l \alpha \prec P'$ **by**(*rule Weak-Late-Step-Semantics.Mismatch*)
**with** *PRel RelRel′* **show** *?case* **by** *blast*
**qed**
**qed**
**qed**

**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**(*induct rule*: *simCases*)
**case**(*Bound* $Q'$ *a x*)
**have** $x \sharp P \oplus R$ **by** *fact*
**hence** *xFreshP*: $(x{::}name) \sharp P$ **and** *xFreshR*: $x \sharp R$ **by** *simp+*
**have** $Q \oplus T \longmapsto a<\nu x> \prec Q'$ **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *sumCases*)
**case** *cSum1*
**have** $Q \longmapsto a<\nu x> \prec Q'$ **by** *fact*
**with** *xFreshP PSimQ* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l a<\nu x> \prec P'$ **and** *P′RelQ′*: $(P', Q') \in Rel$
**by**(*blast dest*: *simE*)
**from** *PTrans* **have** $P \oplus R \Longrightarrow_l a<\nu x> \prec P'$ **by**(*rule Weak-Late-Step-Semantics.Sum1*)
**moreover from** *P′RelQ′ RelRel′* **have** $(P', Q') \in Rel'$ **by** *blast*
**ultimately show** *?case* **by** *blast*
**next**
**case** *cSum2*
**have** $T \longmapsto a<\nu x> \prec Q'$ **by** *fact*
**with** *xFreshR RSimT* **obtain** $R'$ **where** *RTrans*: $R \Longrightarrow_l a<\nu x> \prec R'$ **and** *R′RelQ′*: $(R', Q') \in Rel$
**by**(*blast dest*: *simE*)
**from** *RTrans* **have** $P \oplus R \Longrightarrow_l a<\nu x> \prec R'$ **by**(*rule Weak-Late-Step-Semantics.Sum2*)
**moreover from** *R′RelQ′ RelRel′* **have** $(R', Q') \in Rel'$ **by** *blast*
**ultimately show** *?thesis* **by** *blast*
**qed**

**next**
**case**(*Input* $Q'$ *a x*)
**have** $x \sharp P \oplus R$ **by** *fact*
**hence** *xFreshP*: $(x::name) \sharp P$ **and** *xFreshR*: $x \sharp R$ **by** *simp+*
**have** $Q \oplus T \longmapsto a{<}x{>} \prec Q'$ **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *sumCases*)
**case** *cSum1*
**have** $Q \longmapsto a{<}x{>} \prec Q'$ **by** *fact*
**with** *xFreshP PSimQ* **obtain** $P''$ **where** *L1*: $\forall u. \exists P'. P \Longrightarrow_l u$ *in* $P''{\rightarrow}a{<}x{>}$
$\prec P' \wedge (P', Q'[x::=u]) \in Rel$
**by**(*blast dest*: *simE*)
**have** $\forall u. \exists P'. P \oplus R \Longrightarrow_l u$ *in* $P''{\rightarrow}a{<}x{>} \prec P' \wedge (P', Q'[x::=u]) \in Rel'$
**proof**(*rule allI*)
**fix** $u$
**from** *L1* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l u$ *in* $P''{\rightarrow}a{<}x{>} \prec P'$
**and** *P'RelQ'*: $(P', Q'[x::=u]) \in Rel$ **by** *blast*
**from** *PTrans* **have** $P \oplus R \Longrightarrow_l u$ *in* $P''{\rightarrow}a{<}x{>} \prec P'$ **by**(*rule Weak-Late-Step-Semantics.Sum1*)
**with** *P'RelQ' RelRel'* **show** $\exists P'. P \oplus R \Longrightarrow_l u$ *in* $P''{\rightarrow}a{<}x{>} \prec P' \wedge (P',$
$Q'[x::=u]) \in Rel'$ **by** *blast*
**qed**
**thus** *?case* **by** *blast*
**next**
**case** *cSum2*
**have** $T \longmapsto a{<}x{>} \prec Q'$ **by** *fact*
**with** *xFreshR RSimT* **obtain** $R''$ **where** *L1*: $\forall u. \exists R'. R \Longrightarrow_l u$ *in* $R''{\rightarrow}a{<}x{>}$
$\prec R' \wedge (R', Q'[x::=u]) \in Rel$
**by**(*blast dest*: *simE*)
**have** $\forall u. \exists P'. P \oplus R \Longrightarrow_l u$ *in* $R''{\rightarrow}a{<}x{>} \prec P' \wedge (P', Q'[x::=u]) \in Rel'$
**proof**(*rule allI*)
**fix** $u$
**from** *L1* **obtain** $R'$ **where** *RTrans*: $R \Longrightarrow_l u$ *in* $R''{\rightarrow}a{<}x{>} \prec R'$
**and** *R'RelQ'*: $(R', Q'[x::=u]) \in Rel$ **by** *blast*
**from** *RTrans* **have** $P \oplus R \Longrightarrow_l u$ *in* $R''{\rightarrow}a{<}x{>} \prec R'$ **by**(*rule Weak-Late-Step-Semantics.Sum2*)
**with** *R'RelQ' RelRel'* **show** $\exists P'. P \oplus R \Longrightarrow_l u$ *in* $R''{\rightarrow}a{<}x{>} \prec P' \wedge (P',$
$Q'[x::=u]) \in Rel'$ **by** *blast*
**qed**
**thus** *?case* **by** *blast*
**qed**
**next**
**case**(*Free* $Q'$ $\alpha$)
**have** $Q \oplus T \longmapsto \alpha \prec Q'$ **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *sumCases*)
**case** *cSum1*
**have** $Q \longmapsto \alpha \prec Q'$ **by** *fact*
**with** *PSimQ* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l \alpha \prec P'$ **and** *PRel*: $(P', Q') \in$
*Rel*
**by**(*blast dest*: *simE*)

**from** *PTrans* **have** $P \oplus R \Longrightarrow_l \alpha \prec P'$ **by**(*rule Weak-Late-Step-Semantics.Sum1*)
**with** *RelRel′ PRel* **show** *?case* **by** *blast*
**next**
**case** *cSum2*
**have** $T \longmapsto \alpha \prec Q'$ **by** *fact*
**with** *RSimT* **obtain** $R'$ **where** *RTrans*: $R \Longrightarrow_l \alpha \prec R'$ **and** *RRel*: $(R', Q') \in Rel$
**by**(*blast dest*: *simE*)
**from** *RTrans* **have** $P \oplus R \Longrightarrow_l \alpha \prec R'$ **by**(*rule Weak-Late-Step-Semantics.Sum2*)
**with** *RelRel′ RRel* **show** *?case* **by** *blast*
**qed**
**qed**

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

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

**shows** $P \oplus R \leadsto\!<Rel'>\ Q \oplus R$
**proof** −
**from** *Id* **have** *Refl*: $R \leadsto\!<Rel>\ R$ **by**(*rule reflexive*)
**from** *PSimQ Refl RelRel′* **show** *?thesis* **by**(*rule sumCompose*)
**qed**

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

**assumes** *PSimQ*: $P \leadsto\!<Rel>\ Q$
**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** $P \parallel R \leadsto\!<Rel'>\ Q \parallel R$
**using** *EqvtRel′*
**proof**(*induct rule*: *simCasesCont*[**where** $C=(P, Q, R)$])
**case**(*Bound* $Q'$ *a x*)
**have** $x \sharp (P, Q, R)$ **by** *fact*
**hence** *xFreshP*: $x \sharp P$ **and** *xFreshR*: $x \sharp R$ **and** $x \sharp Q$ **by** *simp+*
**from** ‹$Q \parallel R \longmapsto a<\nu x> \prec Q'$› ‹$x \sharp Q$› ‹$x \sharp R$› **show** *?case*

**proof**(*induct rule*: *parCasesB*)
**case**(*cPar1* $Q'$)
**have** *QTrans*: $Q \longmapsto a{<}\nu x{>} \prec Q'$ **by** *fact*

**from** *xFreshP PSimQ QTrans* **obtain** $P'$ **where** *PTrans*:$P \Longrightarrow_l a{<}\nu x{>} \prec P'$
**and** $P'RelQ'$: $(P', Q') \in Rel$
**by**(*blast dest*: *simE*)
**from** *PTrans xFreshR* **have** $P \parallel R \Longrightarrow_l a{<}\nu x{>} \prec (P' \parallel R)$ **by**(*rule Weak-Late-Step-Semantics.Par1B*)
**moreover from** $P'RelQ'$ **have** $(P' \parallel R, Q' \parallel R) \in Rel'$ **by**(*rule Par*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*cPar2* $R'$)
**have** *RTrans*: $R \longmapsto a{<}\nu x{>} \prec R'$ **by** *fact*
**hence** $R \Longrightarrow_l a{<}\nu x{>} \prec R'$
**by**(*auto simp add*: *weakTransition-def dest*: *Weak-Late-Step-Semantics.singleActionChain*)
**with** *xFreshP xFreshR* **have** *ParTrans*: $P \parallel R \Longrightarrow_l a{<}\nu x{>} \prec P \parallel R'$
**by**(*blast intro*: *Weak-Late-Step-Semantics.Par2B*)
**moreover from** *PRelQ* **have** $(P \parallel R', Q \parallel R') \in Rel'$ **by**(*rule Par*)
**ultimately show** *?case* **by** *blast*
**qed**
**next**
**case**(*Input* $Q'$ *a x*)
**have** $x \sharp (P, Q, R)$ **by** *fact*
**hence** *xFreshP*: $x \sharp P$ **and** *xFreshR*: $x \sharp R$ **and** $x \sharp Q$ **by** *simp+*
**from** $\langle Q \parallel R \longmapsto a{<}x{>} \prec Q' \rangle$ $\langle x \sharp Q \rangle$ $\langle x \sharp R \rangle$
**show** *?case*
**proof**(*induct rule*: *parCasesB*)
**case**(*cPar1* $Q'$)
**have** *QTrans*: $Q \longmapsto a{<}x{>} \prec Q'$ **by** *fact*
**from** *xFreshP PSimQ QTrans* **obtain** $P''$
**where** *L1*: $\forall u. \exists P'. P \Longrightarrow_l u \ in \ P'' \rightarrow a{<}x{>} \prec P' \wedge (P', Q'[x::=u]) \in Rel$
**by**(*blast dest*: *simE*)
**have** $\forall u. \exists P'. P \parallel R \Longrightarrow_l u \ in \ (P'' \parallel R) \rightarrow a{<}x{>} \prec P' \wedge (P', Q'[x::=u] \parallel R[x::=u]) \in Rel'$
**proof**(*rule allI*)
**fix** *u*
**from** *L1* **obtain** $P'$ **where** *PTrans*:$P \Longrightarrow_l u \ in \ P'' \rightarrow a{<}x{>} \prec P'$
**and** $P'RelQ'$: $(P', Q'[x::=u]) \in Rel$ **by** *blast*
**from** *PTrans xFreshR* **have** $P \parallel R \Longrightarrow_l u \ in \ (P'' \parallel R) \rightarrow a{<}x{>} \prec (P' \parallel R)$
**by**(*rule Weak-Late-Step-Semantics.Par1B*)
**moreover from** $P'RelQ'$ **have** $(P' \parallel R, Q'[x::=u] \parallel R) \in Rel'$
**by**(*rule Par*)
**ultimately show** $\exists P'. P \parallel R \Longrightarrow_l u \ in \ (P'' \parallel R) \rightarrow a{<}x{>} \prec P' \wedge (P', Q'[x::=u] \parallel (R[x::=u])) \in Rel'$
**using** *xFreshR*
**by**(*force simp add*: *forget*)
**qed**
**thus** *?case* **by** *force*
**next**

**case**$(\mathit{cPar2}\ R')$
**have** $\mathit{RTrans}$: $R \longmapsto a{<}x{>} \prec R'$ **by** *fact*
**have** $\forall\, u.\ \exists\, P'.\ P \parallel R \Longrightarrow_l u\ \mathit{in}\ (P \parallel R') \rightarrow a{<}x{>} \prec P' \wedge (P',\ Q \parallel R'[x::=u]) \in \mathit{Rel}'$
**proof**
**fix** $u$
**from** *RTrans* **have** $R \Longrightarrow_l u\ \mathit{in}\ R' \rightarrow a{<}x{>} \prec R'[x::=u]$
**by**(*rule Weak-Late-Step-Semantics.singleActionChain*)
**hence** $P \parallel R \Longrightarrow_l u\ \mathit{in}\ P \parallel R' \rightarrow a{<}x{>} \prec P \parallel R'[x::=u]$ **using** $\langle x \sharp P \rangle$
**by**(*rule Weak-Late-Step-Semantics.Par2B*)
**moreover from** *PRelQ* **have** $(P \parallel R'[x::=u],\ Q \parallel R'[x::=u]) \in \mathit{Rel}'$ **by**(*rule Par*)
**ultimately show** $\exists\, P'.\ P \parallel R \Longrightarrow_l u\ \mathit{in}\ (P \parallel R') \rightarrow a{<}x{>} \prec P' \wedge$
$(P',\ Q \parallel R'[x::=u]) \in \mathit{Rel}'$ **by** *blast*
**qed**
**thus** *?case* **using** $\langle x \sharp Q \rangle$ **by**(*fastforce simp add*: *forget*)
**qed**
**next**
**case**$(\mathit{Free}\ \mathit{QR}'\ \alpha)$
**have** $Q \parallel R \longmapsto \alpha \prec \mathit{QR}'$ **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *parCasesF*[*of* - - - - - $(P, R)$])
**case**$(\mathit{cPar1}\ Q')$
**have** $Q \longmapsto \alpha \prec Q'$ **by** *fact*
**with** *PSimQ* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l \alpha \prec P'$ **and** *PRel*: $(P',\ Q') \in \mathit{Rel}$
**by**(*blast dest*: *simE*)
**from** *PTrans* **have** *Trans*: $P \parallel R \Longrightarrow_l \alpha \prec P' \parallel R$ **by**(*rule Weak-Late-Step-Semantics.Par1F*)
**moreover from** *PRel* **have** $(P' \parallel R,\ Q' \parallel R) \in \mathit{Rel}'$ **by**(*blast intro*: *Par*)
**ultimately show** *?case* **by** *blast*
**next**
**case**$(\mathit{cPar2}\ R')$
**have** $R \longmapsto \alpha \prec R'$ **by** *fact*
**hence** $R \Longrightarrow_l \alpha \prec R'$
**by**(*rule Weak-Late-Step-Semantics.singleActionChain*)
**hence** $P \parallel R \Longrightarrow_l \alpha \prec (P \parallel R')$ **by**(*rule Weak-Late-Step-Semantics.Par2F*)
**moreover from** *PRelQ* **have** $(P \parallel R',\ Q \parallel R') \in \mathit{Rel}'$ **by**(*blast intro*: *Par*)
**ultimately show** *?case* **by** *blast*
**next**
**case**$(\mathit{cComm1}\ Q'\ R'\ a\ b\ x)$
**have** *QTrans*: $Q \longmapsto a{<}x{>} \prec Q'$ **and** *RTrans*: $R \longmapsto a[b] \prec R'$ **by** *fact+*
**have** $x \sharp (P, R)$ **by** *fact*
**hence** *xFreshP*: $x \sharp P$ **by**(*simp add*: *fresh-prod*)

**from** *PSimQ QTrans xFreshP* **obtain** $P'\ P''$ **where** *PTrans*: $P \Longrightarrow_l b\ \mathit{in}\ P'' \rightarrow a{<}x{>} \prec P'$
**and** $P'\mathit{RelQ}'$: $(P',\ Q'[x::=b]) \in \mathit{Rel}$
**by**(*blast dest*: *simE*)

**from** *RTrans* **have** $R \Longrightarrow_l a[b] \prec R'$
**by**(*rule Weak-Late-Step-Semantics.singleActionChain*)

**with** *PTrans* **have** $P \parallel R \Longrightarrow_l \tau \prec P' \parallel R'$ **by**(*rule Weak-Late-Step-Semantics.Comm1*)
**moreover from** $P'RelQ'$ **have** $(P' \parallel R', Q'[x::=b] \parallel R') \in Rel'$ **by**(*rule Par*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*cComm2* $Q'$ $R'$ *a b x*)
**have** *QTrans*: $Q \longmapsto a[b] \prec Q'$ **and** *RTrans*: $R \longmapsto a<x> \prec R'$ **by** *fact+*
**have** $x \sharp (P, R)$ **by** *fact*
**hence** *xFreshR*: $x \sharp R$ **by**(*simp add*: *fresh-prod*)

**from** *PSimQ QTrans* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l a[b] \prec P'$
**and** *PRel*: $(P', Q') \in Rel$
**by**(*blast dest*: *simE*)
**from** *RTrans* **have** $R \Longrightarrow_l b \ in \ R' \rightarrow a<x> \prec R'[x::=b]$
**by**(*rule Weak-Late-Step-Semantics.singleActionChain*)
**with** *PTrans* **have** $P \parallel R \Longrightarrow_l \tau \prec P' \parallel R'[x::=b]$ **by**(*rule Weak-Late-Step-Semantics.Comm2*)
**moreover from** *PRel* **have** $(P' \parallel R'[x::=b], Q' \parallel R'[x::=b]) \in Rel'$ **by**(*rule Par*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*cClose1* $Q'$ $R'$ *a x y*)
**have** *QTrans*: $Q \longmapsto a<x> \prec Q'$ **and** *RTrans*: $R \longmapsto a<\nu y> \prec R'$ **by** *fact+*
**have** $x \sharp (P, R)$ **and** $y \sharp (P, R)$ **by** *fact+*
**hence** *xFreshP*: $x \sharp P$ **and** *yFreshR*: $y \sharp R$ **and** *yFreshP*: $y \sharp P$ **by**(*simp add*: *fresh-prod*)+

**from** *PSimQ QTrans xFreshP* **obtain** $P'$ $P''$ **where** *PTrans*: $P \Longrightarrow_l y \ in \ P'' \rightarrow a<x> \prec P'$
**and** $P'RelQ'$: $(P', Q'[x::=y]) \in Rel$
**by**(*blast dest*: *simE*)
**from** *RTrans* **have** $R \Longrightarrow_l a<\nu y> \prec R'$
**by**(*auto simp add*: *weakTransition-def dest*: *Weak-Late-Step-Semantics.singleActionChain*)
**with** *PTrans* **have** *Trans*: $P \parallel R \Longrightarrow_l \tau \prec <\nu y>(P' \parallel R')$ **using** *yFreshP yFreshR*
**by**(*rule Weak-Late-Step-Semantics.Close1*)
**moreover from** $P'RelQ'$ **have** $(<\nu y>(P' \parallel R'), <\nu y>(Q'[x::=y] \parallel R')) \in Rel'$
**by**(*blast intro*: *Par Res*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*cClose2* $Q'$ $R'$ *a x y*)
**have** *QTrans*: $Q \longmapsto a<\nu y> \prec Q'$ **and** *RTrans*: $R \longmapsto a<x> \prec R'$ **by** *fact+*
**have** $x \sharp (P, R)$ **and** $y \sharp (P, R)$ **by** *fact+*
**hence** *xFreshR*: $x \sharp R$ **and** *yFreshP*: $y \sharp P$ **and** *yFreshR*: $y \sharp R$ **by**(*simp add*: *fresh-prod*)+

**from** *PSimQ QTrans yFreshP* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l a<\nu y> \prec P'$
**and** $P'RelQ'$: $(P', Q') \in Rel$

**by**(*blast dest*: *simE*)

**from** *RTrans* **have** $R \Longrightarrow_l y$ *in* $R' \rightarrow a$<x> $\prec R'[x::=y]$
**by**(*rule Weak-Late-Step-Semantics.singleActionChain*)
**with** *PTrans* **have** $P \parallel R \Longrightarrow_l \tau \prec$ <$\nu y$>$(P' \parallel R'[x::=y])$ **using** *yFreshP yFreshR*
**by**(*rule Weak-Late-Step-Semantics.Close2*)
**moreover from** *P'RelQ'* **have** (<$\nu y$>$(P' \parallel R'[x::=y])$, <$\nu y$>$(Q' \parallel R'[x::=y])$) $\in$ *Rel'*
**by**(*blast intro*: *Par Res*)
**ultimately show** *?case* **by** *blast*
**qed**
**qed**

**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** –
**from** *EqvtRel'* **show** *?thesis*
**proof**(*induct rule*: *simCasesCont*[*of* - (*P*, *Q*, *x*)])
**case**(*Bound Q' a y*)
**have** *Trans*: <$\nu x$>$Q \longmapsto a$<$\nu y$> $\prec Q'$ **by** *fact*
**have** $y \sharp (P, Q, x)$ **by** *fact*
**hence** *yineqx*: $y \neq x$ **and** *yFreshP*: $y \sharp P$ **and** $y \sharp Q$ **by**(*simp add*: *fresh-prod*)+
**from** *Trans* ‹$y \neq x$› ‹$y \sharp Q$› **show** *?case*
**proof**(*induct rule*: *resCasesB*)
**case**(*cOpen a Q'*)
**have** *QTrans*: $Q \longmapsto a[x] \prec Q'$ **and** *aineqx*: $a \neq x$ **by** *fact*+

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

**have** <$\nu x$>$P \Longrightarrow_l a$<$\nu y$> $\prec ([(y, x)] \cdot P')$
**proof** –
**from** *PTrans aineqx* **have** <$\nu x$>$P \Longrightarrow_l a$<$\nu x$> $\prec P'$ **by**(*rule Weak-Late-Step-Semantics.Open*)
**moreover from** *PTrans yFreshP* **have** $y \sharp P'$ **by**(*force intro*: *Weak-Late-Step-Semantics.freshTransition*)

**ultimately show** *?thesis* **by**(*simp add*: *alphaBoundResidual name-swap*)
**qed**
**moreover from** *EqvtRel P′RelQ′ RelRel′* **have** $([(y, x)] \cdot P', [(y, x)] \cdot Q') \in$ *Rel′*
**by**(*blast intro*: *eqvtRelI*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*cRes Q′*)
**have** *QTrans*: $Q \longmapsto a{<}\nu y{>} \prec Q'$ **by** *fact*
**from** ‹$x \sharp$ *BoundOutputS a*› **have** $x \neq a$ **by** *simp*

**from** *PSimQ yFreshP QTrans* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l a{<}\nu y{>} \prec P'$
**and** *P′RelQ′*: $(P', Q') \in Rel$
**by**(*blast dest*: *simE*)
**from** *PTrans* ‹$x \neq a$› *yineqx yFreshP* **have** *ResTrans*: $<\nu x>P \Longrightarrow_l a{<}\nu y{>} \prec (<\nu x>P')$
**by**(*blast intro*: *Weak-Late-Step-Semantics.ResB*)
**moreover from** *P′RelQ′* **have** $((<\nu x>P'), (<\nu x>Q')) \in Rel'$
**by**(*rule ResRel*)
**ultimately show** *?case* **by** *blast*
**qed**
**next**
**case**(*Input Q′ a y*)
**have** $y \sharp (P, Q, x)$ **by** *fact*
**hence** *yineqx*: $y \neq x$ **and** *yFreshP*: $y \sharp P$ **and** $y \sharp Q$ **by**(*simp add*: *fresh-prod*)+

**have** $<\nu x>Q \longmapsto a{<}y{>} \prec Q'$ **by** *fact*
**thus** *?case* **using** *yineqx* ‹$y \sharp Q$›
**proof**(*induct rule*: *resCasesB*)
**case**(*cOpen a Q′*)
**thus** *?case* **by** *simp*
**next**
**case**(*cRes Q′*)
**have** *QTrans*: $Q \longmapsto a{<}y{>} \prec Q'$ **by** *fact*
**from** ‹$x \sharp$ *InputS a*› **have** $x \neq a$ **by** *simp*

**from** *PSimQ QTrans yFreshP* **obtain** $P''$
**where** *L1*: $\forall u. \exists P'. P \Longrightarrow_l u \ in \ P'' \rightarrow a{<}y{>} \prec P' \wedge (P', Q'[y::=u]) \in Rel$
**by**(*blast dest*: *simE*)
**have** $\forall u. \exists P'. <\nu x>P \Longrightarrow_l u \ in \ (<\nu x>P'') \rightarrow a{<}y{>} \prec P' \wedge (P', (<\nu x>Q')[y::=u]) \in Rel'$
**proof**(*rule allI*)
**fix** *u*
**show** $\exists P'. <\nu x>P \Longrightarrow_l u \ in \ <\nu x>P'' \rightarrow a{<}y{>} \prec P' \wedge (P', (<\nu x>Q')[y::=u]) \in Rel'$
**proof**(*cases x=u*)
**assume** *xequ*: *x=u*

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

**from** *L1* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l c\ in\ P''{\rightarrow}a{<}y{>} \prec P'$
**and** *P'RelQ'*: $(P',\ Q'[y{::=}c]) \in Rel$
**by** *blast*
**have** $<\nu x>P \Longrightarrow_l u\ in\ (<\nu x>P''){\rightarrow}a{<}y{>} \prec <\nu c>([(x,\ c)] \cdot P')$
**proof** −
**from** *PTrans yineqx* ‹$x \neq a$› *cineqx* **have** $<\nu x>P \Longrightarrow_l c\ in\ (<\nu x>P''){\rightarrow}a{<}y{>} \prec <\nu x>P'$
**by**(*blast intro*: *Weak-Late-Step-Semantics.ResB*)
**hence** $([(x,\ c)] \cdot <\nu x>P) \Longrightarrow_l([(x,\ c)] \cdot c)\ in\ ([(x,\ c)] \cdot <\nu x>P''){\rightarrow}([(x,\ c)] \cdot a){<}([(x,\ c)] \cdot y){>} \prec [(x,\ c)] \cdot <\nu x>P'$
**by**(*rule Weak-Late-Step-Semantics.eqvtI*)
**moreover from** *cFreshP* **have** $<\nu c>([(x,\ c)] \cdot P) = <\nu x>P$ **by**(*simp add*: *alphaRes*)
**moreover from** *cFreshP''* **have** $<\nu c>([(x,\ c)] \cdot P'') = <\nu x>P''$ **by**(*simp add*: *alphaRes*)
**ultimately show** *?thesis* **using** ‹$x \neq a$› *cineqa yineqx cineqy cineqx xequ* **by**(*simp add*: *name-calc*)
**qed**
**moreover have** $(<\nu c>([(x,\ c)] \cdot P'),\ (<\nu x>Q')[y{::=}u]) \in Rel'$
**proof** −
**from** *P'RelQ'* **have** $(<\nu x>P',\ <\nu x>(Q'[y{::=}c])) \in Rel'$ **by**(*rule ResRel*)
**with** *EqvtRel'* **have** $([(x,\ c)] \cdot <\nu x>P',\ [(x,\ c)] \cdot <\nu x>(Q'[y{::=}c])) \in Rel'$ **by**(*rule eqvtRelI*)
**with** *cineqy yineqx cineqx* **have** $(<\nu c>([(x,\ c)] \cdot P'),\ (<\nu c>([(x,\ c)] \cdot Q'))[y{::=}x]) \in Rel'$
**by**(*simp add*: *name-calc eqvt-subs*)
**with** *cFreshQ' xequ* **show** *?thesis* **by**(*simp add*: *alphaRes*)
**qed**
**ultimately show** *?thesis* **by** *blast*
**next**
**assume** *xinequ*: $x \neq u$
**from** *L1* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l u\ in\ P''{\rightarrow}a{<}y{>} \prec P'$
**and** *P'RelQ'*: $(P',\ Q'[y{::=}u]) \in Rel$ **by** *blast*

**from** *PTrans* ‹$x \neq a$› *yineqx xinequ* **have** $<\nu x>P \Longrightarrow_l u\ in\ (<\nu x>P''){\rightarrow}a{<}y{>} \prec <\nu x>P'$
**by**(*blast intro*: *Weak-Late-Step-Semantics.ResB*)
**moreover from** *P'RelQ' xinequ yineqx* **have** $(<\nu x>P',\ (<\nu x>Q')[y{::=}u]) \in Rel'$
**by**(*force intro*: *ResRel*)
**ultimately show** *?thesis* **by** *blast*
**qed**
**qed**

**thus** *?case* **by** *blast*
**qed**
**next**
**case**(*Free* $Q'$ $\alpha$)
**have** $<\nu x>Q \longmapsto \alpha \prec Q'$ **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *resCasesF*)
**case**(*cRes* $Q'$)
**have** $Q \longmapsto\alpha \prec Q'$ **by** *fact*
**with** *PSimQ* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l\alpha \prec P'$
**and** $P'RelQ'$: $(P', Q') \in Rel$
**by**(*blast dest*: *simE*)

**have** $<\nu x>P \Longrightarrow_l\alpha \prec <\nu x>P'$
**proof** −
**have** *xFreshAlpha*: $x \sharp \alpha$ **by** *fact*
**with** *PTrans* **show** *?thesis* **by**(*rule Weak-Late-Step-Semantics.ResF*)
**qed**
**moreover from** $P'RelQ'$ **have** $(<\nu x>P', <\nu x>Q') \in Rel'$ **by**(*rule ResRel*)
**ultimately show** *?case* **by** *blast*
**qed**
**qed**
**qed**

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

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

**shows** $!P \leadsto<bangRel\ Rel'> !Q$
**proof** −
**from** $eqvtRel'$ **have** $EqvtBangRel'$: *eqvt*(*bangRel* $Rel'$) **by**(*rule eqvtBangRel*)
**from** $RelRel'$ **have** $BRelRel'$: $\bigwedge P\ Q.\ (P, Q) \in bangRel\ Rel \Longrightarrow (P, Q) \in bangRel\ Rel'$
**by**(*auto intro*: *bangRelSubset*)

**have** $\bigwedge Rs\ P.\ [\![!Q \longmapsto Rs; (P, !Q) \in bangRel\ Rel]\!] \Longrightarrow weakStepSimAct\ P\ Rs\ P\ (bangRel\ Rel')$
**proof** −
**fix** *Rs P*
**assume** $!Q \longmapsto Rs$ **and** $(P, !Q) \in bangRel\ Rel$
**thus** *weakStepSimAct P Rs P* (*bangRel* $Rel'$)
**proof**(*nominal-induct avoiding*: *P rule*: *bangInduct*)

**case**(*cPar1B aa x Q′ P*)
**have** *QTrans*: $Q \longmapsto aa«x» \prec Q'$ **and** *xFreshQ*: $x \sharp Q$ **by** *fact+*
**have** $(P, Q \parallel !Q) \in bangRel\ Rel$ **and** $x \sharp P$ **by** *fact+*
**thus** *?case*
**proof**(*induct rule*: *BRParCases*)
**case**(*BRPar P R*)
**have** *PRelQ*: $(P, Q) \in Rel$ **and** *RBangRelQ*: $(R, !Q) \in bangRel\ Rel$ **by** *fact+*
**have** $x \sharp P \parallel R$ **by** *fact*
**hence** *xFreshP*: $x \sharp P$ **and** *xFreshR*: $x \sharp R$ **by** *simp+*
**from** *PRelQ* **have** *PSimQ*: $P \rightsquigarrow<Rel'> Q$ **by**(*rule Sim*)
**from** *EqvtBangRel′* **show** *?case*
**proof**(*induct rule*: *simActBoundCases*)
**case**(*Input a*)
**have** $aa = InputS\ a$ **by** *fact*
**with** *PSimQ QTrans xFreshP* **obtain** $P''$
**where** *L1*: $\forall u.\ \exists P'.\ P \Longrightarrow_l u\ in\ P''\rightarrow a<x> \prec P' \wedge (P', Q'[x::=u]) \in Rel'$
**by**(*blast dest*: *simE*)
**have** $\forall u.\ \exists P'.\ P \parallel R \Longrightarrow_l u\ in\ (P'' \parallel R)\rightarrow a<x> \prec P' \wedge (P', (Q' \parallel !Q)[x::=u]) \in bangRel\ Rel'$
**proof**(*rule allI*)
**fix** *u*
**from** *L1* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l u\ in\ P''\rightarrow a<x> \prec P'$
**and** *P′RelQ′*: $(P', Q'[x::=u]) \in Rel'$
**by** *blast*
**from** *PTrans xFreshR* **have** $P \parallel R \Longrightarrow_l u\ in\ (P'' \parallel R)\rightarrow a<x>\prec P' \parallel R$
**by**(*rule Weak-Late-Step-Semantics.Par1B*)
**moreover have** $(P' \parallel R, (Q' \parallel !Q)[x::=u]) \in bangRel\ Rel'$
**proof** −
**from** *P′RelQ′ RBangRelQ* **have** $(P' \parallel R, Q'[x::=u] \parallel !Q) \in bangRel\ Rel'$
**by**(*blast intro*: *BRelRel′ Rel.BRPar*)
**with** *xFreshQ* **show** *?thesis* **by**(*force simp add*: *forget*)
**qed**
**ultimately show** $\exists P'.\ P \parallel R \Longrightarrow_l u\ in\ (P'' \parallel R)\rightarrow a<x> \prec P' \wedge$
$(P', (Q' \parallel !Q)[x::=u]) \in bangRel\ Rel'$
**by** *blast*
**qed**
**thus** *?case* **by** *blast*
**next**
**case**(*BoundOutput a*)
**have** $aa = BoundOutputS\ a$ **by** *fact*
**with** *PSimQ QTrans xFreshP* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l a<\nu x> \prec P'$ **and** *P′RelQ′*: $(P', Q') \in Rel'$
**by**(*force dest*: *simE*)
**from** *PTrans xFreshR* **have** $P \parallel R \Longrightarrow_l a<\nu x>\prec P' \parallel R$
**by**(*rule Weak-Late-Step-Semantics.Par1B*)
**moreover from** *P′RelQ′ RBangRelQ* **have** $(P' \parallel R, Q' \parallel !Q) \in bangRel$

```
Rel'
          by(blast intro: Rel.BRPar BRelRel')
        ultimately show ?case by blast
      qed
    qed
  next
    case(cPar1F α Q' P)
    have QTrans: Q ⟼ α ≺ Q' by fact
    have (P, Q ∥ !Q) ∈ bangRel Rel by fact
    thus ?case
    proof(induct rule: BRParCases)
      case(BRPar P R)
       have PRelQ: (P, Q) ∈ Rel and RBangRelQ: (R, !Q) ∈ bangRel Rel by
fact+
      show ?case
      proof(induct rule: simActFreeCases)
        case Free
        from PRelQ have P ⇝<Rel'> Q by(rule Sim)
       with QTrans obtain P' where PTrans: P ⟹ₗα ≺ P' and P'RelQ': (P',
Q') ∈ Rel'
          by(blast dest: simE)

      from PTrans have P ∥ R ⟹ₗα ≺ P' ∥ R by(rule Weak-Late-Step-Semantics.Par1F)
        moreover from P'RelQ' RBangRelQ have (P' ∥ R, Q' ∥ !Q) ∈ bangRel
Rel'
          by(blast intro: BRelRel' Rel.BRPar)
        ultimately show ?case by blast
      qed
    qed
  next
    case(cPar2B aa x Q' P)
    have IH: ⋀P. (P, !Q) ∈ bangRel Rel ⟹ weakStepSimAct P (aa«x» ≺ Q')
P (bangRel Rel') by fact
    have xFreshQ: x ♯ Q by fact
    have (P, Q ∥ !Q) ∈ bangRel Rel and x ♯ P by fact+
    thus ?case
    proof(induct rule: BRParCases)
      case(BRPar P R)
       have PRelQ: (P, Q) ∈ Rel and RBangRelQ: (R, !Q) ∈ bangRel Rel by
fact+
      have x ♯ P ∥ R by fact
      hence xFreshP: x ♯ P and xFreshR: x ♯ R by simp+
       from RBangRelQ have IH: weakStepSimAct R (aa«x» ≺ Q') R (bangRel
Rel') by(rule IH)
      from EqvtBangRel' show ?case
      proof(induct rule: simActBoundCases)
        case(Input a)
        have aa = InputS a by fact
       with xFreshR IH obtain  R'' where L1: ∀ u. ∃ R'. R ⟹ₗu in R''→a<x>
```

$\prec R' \wedge$

$(R',\ Q'[x::=u]) \in bangRel\ Rel'$

**by**(*simp add*: *weakStepSimAct-def*, *blast*)

**have** $\forall u.\ \exists P'.\ P \parallel R \Longrightarrow_l u\ in\ (P \parallel R'')\rightarrow a{<}x{>} \prec P' \wedge (P',\ (Q \parallel Q')[x::=u]) \in bangRel\ Rel'$

**proof**(*rule allI*)

**fix** $u$

**from** *L1* **obtain** $R'$ **where** *RTrans*: $R \Longrightarrow_l u\ in\ R''\rightarrow a{<}x{>} \prec R'$

**and** $R'BangRelT'$: $(R',\ Q'[x::=u]) \in bangRel\ Rel'$

**by** *blast*

**from** *RTrans xFreshP* **have** $P \parallel R \Longrightarrow_l u\ in\ (P \parallel R'')\rightarrow a{<}x{>} \prec P \parallel R'$

**by**(*rule Weak-Late-Step-Semantics.Par2B*)

**moreover have** $(P \parallel R',\ (Q \parallel Q')[x::=u]) \in bangRel\ Rel'$

**proof** $-$

**from** *PRelQ* $R'BangRelT'$ **have** $(P \parallel R',\ Q \parallel Q'[x::=u]) \in bangRel\ Rel'$

**by**(*blast intro*: $RelRel'$ *Rel.BRPar*)

**with** *xFreshQ* **show** *?thesis* **by**(*simp add*: *forget*)

**qed**

**ultimately show** $\exists P'.\ P \parallel R \Longrightarrow_l u\ in\ (P \parallel R'')\rightarrow a{<}x{>} \prec P' \wedge (P',\ (Q \parallel Q')[x::=u]) \in bangRel\ Rel'$

**by** *blast*

**qed**

**thus** *?case* **by** *blast*

**next**

**case**(*BoundOutput a*)

**have** $aa = BoundOutputS\ a$ **by** *fact*

**with** *IH xFreshR* **obtain** $R'$ **where** *RTrans*: $R \Longrightarrow_l a{<}\nu x{>} \prec R'$

**and** $R'BangRelT'$: $(R',\ Q') \in bangRel\ Rel'$

**by**(*simp add*: *weakStepSimAct-def*, *blast*)

**from** *RTrans xFreshP* **have** $P \parallel R \Longrightarrow_l a{<}\nu x{>} \prec P \parallel R'$

**by**(*auto intro*: *Weak-Late-Step-Semantics.Par2B*)

**moreover from** *PRelQ* $R'BangRelT'$ **have** $(P \parallel R',\ Q \parallel Q') \in bangRel\ Rel'$

**by**(*blast intro*: $RelRel'$ *Rel.BRPar*)

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

**qed**

**qed**

**next**

**case**(*cPar2F* $\alpha$ $Q'$)

**have** *IH*: $\bigwedge P.\ (P,\ !Q) \in bangRel\ Rel \Longrightarrow weakStepSimAct\ P\ (\alpha \prec Q')\ P\ (bangRel\ Rel')$ **by** *fact+*

**have** $(P,\ Q \parallel !Q) \in bangRel\ Rel$ **by** *fact*

**thus** *?case*

**proof**(*induct rule*: *BRParCases*)

**case**(*BRPar P R*)

**have** *PRelQ*: $(P,\ Q) \in Rel$ **and** *RBangRelQ*: $(R,\ !Q) \in bangRel\ Rel$ **by**

*fact+*
**show** *?case*
**proof**(*induct rule*: *simActFreeCases*)
**case** *Free*
**from** *RBangRelQ* **have** *weakStepSimAct* $R$ $(\alpha \prec Q')$ $R$ (*bangRel* $Rel'$) **by**(*rule IH*)
**then obtain** $R'$ **where** *RTrans*: $R \Longrightarrow_l \alpha \prec R'$ **and** $R'BangRelQ'$: $(R', Q') \in$ *bangRel* $Rel'$
**by**(*simp add*: *weakStepSimAct-def*, *blast*)

**from** *RTrans* **have** $P \parallel R \Longrightarrow_l \alpha \prec P \parallel R'$ **by**(*rule Weak-Late-Step-Semantics.Par2F*)
**moreover from** *PRelQ* $R'BangRelQ'$ **have** $(P \parallel R', Q \parallel Q') \in$ *bangRel* $Rel'$
**by**(*blast intro*: *RelRel'* *Rel.BRPar*)
**ultimately show** *?case* **by** *blast*
**qed**
**qed**
**next**
**case**(*cComm1* $a$ $x$ $Q'$ $b$ $Q''$ $P$)
**have** *QTrans*: $Q \longmapsto a<x> \prec Q'$ **by** *fact*
**have** *IH*: $\bigwedge P.\ (P, !Q) \in$ *bangRel Rel* $\Longrightarrow$ *weakStepSimAct* $P$ $(a[b] \prec Q'')$ $P$ (*bangRel* $Rel'$) **by** *fact+*
**have** $(P, Q \parallel !Q) \in$ *bangRel Rel* **and** $x \sharp P$ **by** *fact+*
**thus** *?case*
**proof**(*induct rule*: *BRParCases*)
**case**(*BRPar* $P$ $R$)
**have** *PRelQ*: $(P, Q) \in$ *Rel* **and** *RBangRelQ*: $(R, !Q) \in$ *bangRel Rel* **by** *fact+*
**have** $x \sharp P \parallel R$ **by** *fact*
**hence** *xFreshP*: $x \sharp P$ **by** *simp*
**show** *?case*
**proof**(*induct rule*: *simActFreeCases*)
**case** *Free*
**from** *PRelQ* **have** $P \leadsto<Rel'> Q$ **by**(*rule Sim*)
**with** *QTrans xFreshP* **obtain** $P'$ $P''$ **where** *PTrans*: $P \Longrightarrow_l b\ in\ P'' \rightarrow a<x> \prec P'$
**and** $P'RelQ'$: $(P', Q'[x::=b]) \in Rel'$
**by**(*blast dest*: *simE*)

**from** *RBangRelQ* **have** *weakStepSimAct* $R$ $(a[b] \prec Q'')$ $R$ (*bangRel* $Rel'$) **by**(*rule IH*)
**then obtain** $R'$ **where** *RTrans*: $R \Longrightarrow_l a[b] \prec R'$
**and** $R'RelT'$: $(R', Q'') \in$ *bangRel* $Rel'$
**by**(*simp add*: *weakStepSimAct-def*, *blast*)
**from** *PTrans RTrans* **have** $P \parallel R \Longrightarrow_l \tau \prec (P' \parallel R')$
**by**(*rule Weak-Late-Step-Semantics.Comm1*)
**moreover from** $P'RelQ'$ $R'RelT'$ **have** $(P' \parallel R', Q'[x::=b] \parallel Q'') \in$ *bangRel* $Rel'$
**by**(*blast intro*: *RelRel'* *Rel.BRPar*)

**ultimately show** *?case* **by** *blast*
**qed**
**qed**
**next**
**case**(*cComm2 a b Q′ x Q′′ P*)
**have** *QTrans*: $Q \longmapsto a[b] \prec Q'$ **by** *fact*
**have** *IH*: $\bigwedge P.\ (P,\ !Q) \in$ *bangRel Rel* $\Longrightarrow$ *weakStepSimAct P* $(a{<}x{>} \prec Q'')$ *P* (*bangRel Rel′*)
**by** *fact*
**have** $(P,\ Q \parallel !Q) \in$ *bangRel Rel* **and** $x \sharp P$ **by** *fact+*
**thus** *?case*
**proof**(*induct rule*: *BRParCases*)
**case**(*BRPar P R*)
**have** *PRelQ*: $(P,\ Q) \in$ *Rel* **and** *RBangRelQ*: $(R,\ !Q) \in$ *bangRel Rel* **by** *fact+*
**have** $x \sharp P \parallel R$ **by** *fact*
**hence** *xFreshR*: $x \sharp R$ **by** *simp*
**show** *?case*
**proof**(*induct rule*: *simActFreeCases*)
**case** *Free*

**from** *PRelQ* **have** $P \leadsto{<}Rel'{>}\ Q$ **by**(*rule Sim*)
**with** *QTrans* **obtain** *P′* **where** *PTrans*: $P \Longrightarrow_l a[b] \prec P'$
**and** *P′RelQ′*: $(P',\ Q') \in Rel'$
**by**(*blast dest*: *simE*)

**from** *RBangRelQ* **have** *weakStepSimAct R* $(a{<}x{>} \prec Q'')$ *R* (*bangRel Rel′*)
**by**(*rule IH*)
**with** *xFreshR* **obtain** *R′ R′′* **where** *RTrans*: $R \Longrightarrow_l b\ in\ R'' {\rightarrow} a{<}x{>} \prec R'$
**and** *R′BangRelQ′′*: $(R',\ Q''[x{::=}b]) \in$ *bangRel Rel′*
**by**(*simp add*: *weakStepSimAct-def*, *blast*)

**from** *PTrans RTrans* **have** $P \parallel R \Longrightarrow_l \tau \prec (P' \parallel R')$
**by**(*rule Weak-Late-Step-Semantics.Comm2*)
**moreover from** *P′RelQ′ R′BangRelQ′′* **have** $(P' \parallel R',\ Q' \parallel Q''[x{::=}b]) \in$ *bangRel Rel′*
**by**(*rule Rel.BRPar*)
**ultimately show** *?case* **by** *blast*
**qed**
**qed**
**next**
**case**(*cClose1 a x Q′ y Q′′ P*)
**have** *QTrans*: $Q \longmapsto a{<}x{>} \prec Q'$ **by** *fact*
**have** *IH*: $\bigwedge P.\ (P,\ !Q) \in$ *bangRel Rel* $\Longrightarrow$ *weakStepSimAct P* $(a{<}\nu y{>} \prec Q'')$ *P* (*bangRel Rel′*)
**by** *fact*
**have** $(P,\ Q \parallel !Q) \in$ *bangRel Rel* **and** $x \sharp P$ **and** $y \sharp P$ **by** *fact+*
**thus** *?case*
**proof**(*induct rule*: *BRParCases*)

**case**(*BRPar P R*)
**have** *PRelQ*: $(P,\ Q) \in$ *Rel* **and** *RBangRelQ*: $(R,\ !Q) \in$ *bangRel Rel* **by** *fact+*
**have** $x \sharp P \parallel R$ **and** $y \sharp P \parallel R$ **by** *fact+*
**hence** *xFreshP*: $x \sharp P$ **and** *yFreshR*: $y \sharp R$ **and** *yFreshP*: $y \sharp P$ **by** *simp+*
**show** *?case*
**proof**(*induct rule*: *simActFreeCases*)
**case** *Free*
**from** *PRelQ* **have** $P \leadsto<Rel'>\ Q$ **by**(*rule Sim*)
**with** *QTrans xFreshP* **obtain** $P'$ $P''$ **where** *PTrans*: $P \Longrightarrow_l y\ in\ P''\rightarrow a<x> \prec P'$
**and** *P'RelQ'*: $(P',\ Q'[x::=y]) \in Rel'$
**by**(*blast dest*: *simE*)

**from** *RBangRelQ* **have** *weakStepSimAct R* $(a<\nu y> \prec Q'')$ *R* (*bangRel Rel'*) **by**(*rule IH*)
**with** *yFreshR* **obtain** $R'$ **where** *RTrans*: $R \Longrightarrow_l a<\nu y> \prec R'$
**and** *R'BangRelQ''*: $(R',\ Q'') \in$ *bangRel Rel'*
**by**(*simp add*: *weakStepSimAct-def*, *blast*)
**from** *PTrans RTrans yFreshP yFreshR* **have** $P \parallel R \Longrightarrow_l \tau \prec <\nu y>(P' \parallel R')$
**by**(*rule Weak-Late-Step-Semantics.Close1*)
**moreover from** *P'RelQ' R'BangRelQ''* **have** $(<\nu y>(P' \parallel R'),\ <\nu y>(Q'[x::=y] \parallel Q'')) \in$ *bangRel Rel'*
**by**(*force intro*: *Rel.BRPar Rel.BRRes*)
**ultimately show** *?case* **by** *blast*
**qed**
**qed**
**next**
**case**(*cClose2 a y Q' x Q''*)
**have** *QTrans*: $Q \longmapsto a<\nu y> \prec Q'$ **by** *fact*
**have** *IH*: $\bigwedge P.\ (P,\ !Q) \in$ *bangRel Rel* $\Longrightarrow$ *weakStepSimAct P* $(a<x> \prec Q'')$ *P* (*bangRel Rel'*)
**by** *fact*
**have** $(P,\ Q \parallel !Q) \in$ *bangRel Rel* **and** $x \sharp P$ **and** $y \sharp P$ **by** *fact+*
**thus** *?case*
**proof**(*induct rule*: *BRParCases*)
**case**(*BRPar P R*)
**have** *PRelQ*: $(P,\ Q) \in$ *Rel* **and** *RBangRelQ*: $(R,\ !Q) \in$ *bangRel Rel* **by** *fact+*
**have** $x \sharp P \parallel R$ **and** $y \sharp P \parallel R$ **by** *fact+*
**hence** *xFreshR*: $x \sharp R$ **and** *yFreshR*: $y \sharp R$ **and** *yFreshP*: $y \sharp P$ **by** *simp+*
**show** *?case*
**proof**(*induct rule*: *simActFreeCases*)
**case** *Free*
**from** *PRelQ* **have** $P \leadsto<Rel'>\ Q$ **by**(*rule Sim*)
**with** *QTrans yFreshP* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l a<\nu y> \prec P'$
**and** *P'RelQ'*: $(P',\ Q') \in Rel'$
**by**(*blast dest*: *simE*)

**from** *RBangRelQ* **have** $\mathit{weakStepSimAct}\ R\ (a{<}x{>} \prec Q'')\ R\ (\mathit{bangRel}\ \mathit{Rel}')$
**by**(*rule IH*)
**with** *xFreshR* **obtain** $R'\ R''$ **where** $\mathit{RTrans}$: $R \Longrightarrow_l y\ \mathit{in}\ R''{\rightarrow}a{<}x{>} \prec R'$
**and** $R'\mathit{BangRelT}'$: $(R',\ Q''[x{::=}y]) \in \mathit{bangRel}\ \mathit{Rel}'$
**by**(*simp add*: *weakStepSimAct-def*, *blast*)

**from** *PTrans RTrans yFreshP yFreshR* **have** $P \parallel R \Longrightarrow_l \tau \prec {<}\nu y{>}(P' \parallel R')$
**by**(*rule Weak-Late-Step-Semantics.Close2*)
**moreover from** $P'\mathit{RelQ}'\ R'\mathit{BangRelT}'$ **have** $({<}\nu y{>}(P' \parallel R'),\ {<}\nu y{>}(Q' \parallel Q''[x{::=}y])) \in \mathit{bangRel}\ \mathit{Rel}'$
**by**(*force intro*: *Rel.BRPar Rel.BRRes*)
**ultimately show** *?case* **by** *blast*
**qed**
**qed**
**next**
**case**(*cBang Rs*)
**have** *IH*: $\bigwedge P.\ (P,\ Q \parallel !Q) \in \mathit{bangRel}\ \mathit{Rel} \Longrightarrow \mathit{weakStepSimAct}\ P\ \mathit{Rs}\ P\ (\mathit{bangRel}\ \mathit{Rel}')$
**by** *fact*
**have** $(P,\ !Q) \in \mathit{bangRel}\ \mathit{Rel}$ **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *BRBangCases*)
**case**(*BRBang P*)
**have** *PRelQ*: $(P,\ Q) \in \mathit{Rel}$ **by** *fact*
**hence** $(!P,\ !Q) \in \mathit{bangRel}\ \mathit{Rel}$ **by**(*rule Rel.BRBang*)
**with** *PRelQ* **have** $(P \parallel !P,\ Q \parallel !Q) \in \mathit{bangRel}\ \mathit{Rel}$ **by**(*rule Rel.BRPar*)
**hence** $\mathit{weakStepSimAct}\ (P \parallel !P)\ \mathit{Rs}\ (P \parallel !P)\ (\mathit{bangRel}\ \mathit{Rel}')$ **by**(*rule IH*)
**thus** *?case*
**proof**(*simp* (*no-asm*) *add*: *weakStepSimAct-def*, *auto*)
**fix** $Q'\ a\ x$
**assume** $\mathit{weakStepSimAct}\ (P \parallel !P)\ (a{<}\nu x{>} \prec Q')\ (P \parallel !P)\ (\mathit{bangRel}\ \mathit{Rel}')$ **and** $x \sharp P$
**then obtain** $P'$ **where** *PTrans*: $(P \parallel !P) \Longrightarrow_l a{<}\nu x{>} \prec P'$
**and** $P'\mathit{RelQ}'$: $(P',\ Q') \in (\mathit{bangRel}\ \mathit{Rel}')$
**by**(*simp add*: *weakStepSimAct-def*, *blast*)
**from** *PTrans* **have** $!P \Longrightarrow_l a{<}\nu x{>} \prec P'$
**by**(*rule Weak-Late-Step-Semantics.Bang*)
**with** $P'\mathit{RelQ}'$ **show** $\exists P'.\ !P \Longrightarrow_l a{<}\nu x{>} \prec P' \wedge (P',\ Q') \in \mathit{bangRel}\ \mathit{Rel}'$ **by** *blast*
**next**
**fix** $Q'\ a\ x$
**assume** $\mathit{weakStepSimAct}\ (P \parallel !P)\ (a{<}x{>} \prec Q')\ (P \parallel !P)\ (\mathit{bangRel}\ \mathit{Rel}')$ **and** $x \sharp P$
**then obtain** $P''$ **where** *L1*: $\forall u.\ \exists P'.\ P \parallel !P \Longrightarrow_l u\ \mathit{in}\ P''{\rightarrow}a{<}x{>} \prec P' \wedge (P',\ Q'[x{::=}u]) \in (\mathit{bangRel}\ \mathit{Rel}')$
**by**(*simp add*: *weakStepSimAct-def*, *blast*)
**have** $\forall u.\ \exists P'.\ !P \Longrightarrow_l u\ \mathit{in}\ P''{\rightarrow}a{<}x{>} \prec P' \wedge (P',\ Q'[x{::=}u]) \in (\mathit{bangRel}$

*Rel'*)
**proof**(*rule allI*)
**fix** *u*
**from** *L1* **obtain** $P'$ **where** *PTrans*: $P \parallel !P \Longrightarrow_l u$ *in* $P''\rightarrow a$<$x$> $\prec P'$
**and** *P'RelQ'*: $(P', Q'[x::=u]) \in$ (*bangRel Rel'*)
**by** *blast*
**from** *PTrans* **have** $!P \Longrightarrow_l u$ *in* $P''\rightarrow a$<$x$> $\prec P'$ **by**(*rule Weak-Late-Step-Semantics.Bang*)
**with** *P'RelQ'* **show** $\exists P'.\ !P \Longrightarrow_l u$ *in* $P''\rightarrow a$<$x$> $\prec P' \wedge (P', Q'[x::=u])$
$\in$ (*bangRel Rel'*) **by** *blast*
**qed**
**thus** $\exists P''.\ \forall u.\ \exists P'.\ !P \Longrightarrow_l u$ *in* $P''\rightarrow a$<$x$> $\prec P' \wedge (P', Q'[x::=u]) \in$
(*bangRel Rel'*) **by** *blast*
**next**
**fix** $Q'$ $\alpha$
**assume** *weakStepSimAct* $(P \parallel !P)$ $(\alpha \prec Q')$ $(P \parallel !P)$ (*bangRel Rel'*)
**then obtain** $P'$ **where** *PTrans*: $(P \parallel !P) \Longrightarrow_l \alpha \prec P'$
**and** *P'RelQ'*: $(P', Q') \in$ (*bangRel Rel'*)
**by**(*simp add*: *weakStepSimAct-def*, *blast*)
**from** *PTrans* **have** $!P \Longrightarrow_l \alpha \prec P'$
**by**(*rule Weak-Late-Step-Semantics.Bang*)
**with** *P'RelQ'* **show** $\exists P'.\ !P \Longrightarrow_l \alpha \prec P' \wedge (P', Q') \in$ (*bangRel Rel'*) **by**
*blast*
**qed**
**qed**
**qed**
**qed**
**moreover from** *PRelQ* **have** $(!P, !Q) \in$ *bangRel Rel* **by**(*rule Rel.BRBang*)
**ultimately show** *?thesis* **by**(*simp add*: *weakStepSim-def*)
**qed**

**end**

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

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

**shows** <$\nu a$><$\nu b$>$P \approx$ <$\nu b$><$\nu a$>$P$
**proof** −
**have** <$\nu a$><$\nu b$>$P \sim$ <$\nu b$><$\nu a$>$P$ **by**(*rule Strong-Late-Bisim-SC.resComm*)
**thus** *?thesis* **by**(*rule strongBisimWeakBisim*)
**qed**

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

  **shows** $[a \frown a]P \approx P$
**proof** −
  **have** $[a \frown a]P \sim P$ **by**(*rule Strong-Late-Bisim-SC.matchId*)
  **thus** *?thesis* **by**(*rule strongBisimWeakBisim*)
**qed**

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

  **assumes** $a \neq b$

  **shows** $[a \neq b]P \approx P$
**proof** −
  **from** *assms* **have** $[a \neq b]P \sim P$ **by**(*rule Strong-Late-Bisim-SC.mismatchId*)
  **thus** *?thesis* **by**(*rule strongBisimWeakBisim*)
**qed**

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

  **shows** $[a \neq a]P \approx \mathbf{0}$
**proof** −
  **have** $[a \neq a]P \sim \mathbf{0}$ **by**(*rule Strong-Late-Bisim-SC.mismatchNil*)
  **thus** *?thesis* **by**(*rule strongBisimWeakBisim*)
**qed**

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

  **shows** $P \oplus Q \approx Q \oplus P$
**proof** −
  **have** $P \oplus Q \sim Q \oplus P$ **by**(*rule Strong-Late-Bisim-SC.sumSym*)
  **thus** *?thesis* **by**(*rule strongBisimWeakBisim*)
**qed**

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

  **shows** $(P \oplus Q) \oplus R \approx P \oplus (Q \oplus R)$
**proof** −
  **have** $(P \oplus Q) \oplus R \sim P \oplus (Q \oplus R)$ **by**(*rule Strong-Late-Bisim-SC.sumAssoc*)
  **thus** *?thesis* **by**(*rule strongBisimWeakBisim*)
**qed**

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

  **shows** $P \oplus \mathbf{0} \approx P$
**proof** −
  **have** $P \oplus \mathbf{0} \sim P$ **by**(*rule Strong-Late-Bisim-SC.sumZero*)
  **thus** *?thesis* **by**(*rule strongBisimWeakBisim*)
**qed**

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

  **shows** $P \parallel \mathbf{0} \approx P$
**proof** −
  **have** $P \parallel \mathbf{0} \sim P$ **by**(*rule Strong-Late-Bisim-SC.parZero*)
  **thus** *?thesis* **by**(*rule strongBisimWeakBisim*)
**qed**

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

  **shows** $P \parallel Q \approx Q \parallel P$
**proof** −
  **have** $P \parallel Q \sim Q \parallel P$ **by**(*rule Strong-Late-Bisim-SC.parSym*)
  **thus** *?thesis* **by**(*rule strongBisimWeakBisim*)
**qed**

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

  **assumes** $x \sharp P$

  **shows** $<\nu x>(P \parallel Q) \approx P \parallel <\nu x>Q$

**proof** −
**from** *assms* **have** $<\nu x>(P \parallel Q) \sim P \parallel <\nu x>Q$ **by**(*rule Strong-Late-Bisim-SC.scopeExtPar*)
**thus** *?thesis* **by**(*rule strongBisimWeakBisim*)
**qed**

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

**assumes** *xFreshQ*: $x \sharp Q$

**shows** $<\nu x>(P \parallel Q) \approx (<\nu x>P) \parallel Q$
**proof** −
**from** *assms* **have** $<\nu x>(P \parallel Q) \sim (<\nu x>P) \parallel Q$ **by**(*rule Strong-Late-Bisim-SC.scopeExtPar'*)
**thus** *?thesis* **by**(*rule strongBisimWeakBisim*)
**qed**

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

**shows** $(P \parallel Q) \parallel R \approx P \parallel (Q \parallel R)$
**proof** −
**have** $(P \parallel Q) \parallel R \sim P \parallel (Q \parallel R)$ **by**(*rule Strong-Late-Bisim-SC.parAssoc*)
**thus** *?thesis* **by**(*rule strongBisimWeakBisim*)
**qed**

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

**assumes** *aFreshP*: $a \sharp P$

**shows** $<\nu a>P \approx P$
**proof** −
**from** *assms* **have** $<\nu a>P \sim P$ **by**(*rule Strong-Late-Bisim-SC.scopeFresh*)
**thus** *?thesis* **by**(*rule strongBisimWeakBisim*)
**qed**

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

**assumes** $x \sharp P$

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

**proof** −
**from** *assms* **have** $<\nu x>(P \oplus Q) \sim P \oplus <\nu x>Q$ **by**(*rule Strong-Late-Bisim-SC.scopeExtSum*)
**thus** *?thesis* **by**(*rule strongBisimWeakBisim*)
**qed**

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

**shows** $!P \approx P \parallel !P$
**proof** −
**have** $!P \sim P \parallel !P$ **by**(*rule Strong-Late-Bisim-SC.bangSC*)
**thus** *?thesis* **by**(*rule strongBisimWeakBisim*)
**qed**

**end**

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

**lemma** *tauPres*:
**fixes** $P$ :: *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**(*induct rule*: *simCases*)
**case**(*Bound Q' a x*)
**have** $\tau.(Q) \longmapsto a<\nu x> \prec Q'$ **by** *fact*
**hence** *False* **by** *auto*
**thus** *?case* **by** *simp*
**next**
**case**(*Input Q' a x*)
**have** $\tau.(Q) \longmapsto a<x> \prec Q'$ **by** *fact*
**hence** *False* **by** *auto*
**thus** *?case* **by** *simp*
**next**
**case**(*Free Q' α*)
**have** $\tau.(Q) \longmapsto(\alpha \prec Q')$ **by** *fact*
**thus** *?case* **using** *PRelQ*
**proof**(*induct rule*: *tauCases*, *auto simp add*: *pi.inject residual.inject*)
**have** $\tau.(P) \Longrightarrow_l\hat{}\ \tau \prec P$ **by**(*rule Tau*)
**moreover assume** $(P, Q') \in Rel$
**ultimately show** $\exists P'.\ \tau.(P) \Longrightarrow_l\hat{}\ \tau \prec P' \wedge (P', Q') \in Rel$ **by** *blast*
**qed**
**qed**

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

  **assumes** *PRelQ*: ∀ *y*. (*P*[*x*::=*y*], *Q*[*x*::=*y*]) ∈ *Rel*
  **and** *Eqvt*: *eqvt Rel*

  **shows** *a*<*x*>.*P* ⇝^ <*Rel*> *a*<*x*>.*Q*
**proof** −
  **show** *?thesis* **using** *Eqvt*
  **proof**(*induct rule*: *simCasesCont*[*of* - (*P*, *a*, *x*, *Q*)])
    **case**(*Bound Q′ b y*)
    **have** *a*<*x*>.*Q* ⟼*b*<ν*y*> ≺ *Q′* **by** *fact*
    **hence** *False* **by** *auto*
    **thus** *?case* **by** *simp*
  **next**
    **case**(*Input Q′ b y*)
    **have** *y* ♯ (*P*, *a*, *x*, *Q*) **by** *fact*
    **hence** *yFreshP*: (*y*::*name*) ♯ *P* **and** *yineqx*: *y* ≠ *x* **and** *y* ≠ *a* **and** *y* ♯ *Q*
      **by**(*simp add*: *fresh-prod*)+
    **have** *a*<*x*>.*Q* ⟼*b*<*y*> ≺ *Q′* **by** *fact*
    **thus** *?case* **using** ‹*y* ≠ *a*› ‹*y* ≠ *x*› ‹*y* ♯ *Q*›
    **proof**(*induct rule*: *inputCases*, *auto simp add*: *subject.inject*)
      **have** ∀ *u*. ∃ *P′*. *a*<*x*>.*P* ⟹$_l$*u in* ([(*x*, *y*)] • *P*)→*a*<*y*> ≺ *P′* ∧ (*P′*, ([(*x*, *y*)] • *Q*)[*y*::=*u*]) ∈ *Rel*
      **proof**(*rule allI*)
        **fix** *u*
          **have** *a*<*x*>.*P* ⟹$_l$*u in* ([(*x*, *y*)] • *P*)→*a*<*y*> ≺ ([(*x*, *y*)] • *P*)[*y*::=*u*] (**is** *?goal*)
        **proof** −
        **from** *yFreshP* **have** *a*<*x*>.*P* = *a*<*y*>.([(*x*, *y*)] • *P*) **by**(*rule Agent.alphaInput*)
          **moreover have** *a*<*y*>.([(*x*, *y*)] • *P*) ⟹$_l$*u in* ([(*x*, *y*)] • *P*)→*a*<*y*> ≺ ([(*x*, *y*)] • *P*)[*y*::=*u*]
            **by**(*rule Weak-Late-Step-Semantics.Input*)
          **ultimately show** *?goal* **by**(*simp add*: *name-swap*)
        **qed**

        **moreover have** (([(*x*, *y*)] • *P*)[*y*::=*u*], ([(*x*, *y*)] • *Q*)[*y*::=*u*]) ∈ *Rel*
        **proof** −
          **from** *PRelQ* **have** (*P*[*x*::=*u*], *Q*[*x*::=*u*]) ∈ *Rel* **by** *auto*
          **with** ‹*y* ♯ *P*› ‹*y* ♯ *Q*› **show** *?thesis* **by**(*simp add*: *renaming*)
        **qed**

        **ultimately show** ∃ *P′*. *a*<*x*>.*P* ⟹$_l$*u in* ([(*x*, *y*)] • *P*)→*a*<*y*> ≺ *P′* ∧ (*P′*, ([(*x*, *y*)] • *Q*)[*y*::=*u*]) ∈ *Rel*

**by** *blast*
**qed**

**thus** $\exists P''.\ \forall u.\ \exists P'.\ a{<}x{>}.P \Longrightarrow_l u\ in\ P''{\rightarrow}a{<}y{>} \prec P' \wedge (P',\ ([(x,\ y)] \bullet Q)[y{::=}u]) \in Rel$ **by** *blast*
**qed**
**next**
**case**(*Free* $Q'$ $\alpha$)
**have** $a{<}x{>}.Q \longmapsto \alpha \prec Q'$ **by** *fact*
**hence** *False* **by** *auto*
**thus** *?case* **by** *simp*
**qed**
**qed**

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

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

**shows** $a\{b\}.P \leadsto\hat{}{<}Rel{>}\ a\{b\}.Q$
**proof**(*induct rule*: *simCases*)
**case**(*Bound* $Q'$ $c$ $x$)
**have** $a\{b\}.Q \longmapsto c{<}\nu x{>} \prec Q'$ **by** *fact*
**hence** *False* **by** *auto*
**thus** *?case* **by** *simp*
**next**
**case**(*Input* $Q'$ $c$ $x$)
**have** $a\{b\}.Q \longmapsto c{<}x{>} \prec Q'$ **by** *fact*
**hence** *False* **by** *auto*
**thus** *?case* **by** *simp*
**next**
**case**(*Free* $Q'$ $\alpha$)
**have** $a\{b\}.Q \longmapsto \alpha \prec Q'$ **by** *fact*
**thus** $\exists P'.\ a\{b\}.P \Longrightarrow_l\hat{}\ \alpha \prec P' \wedge (P',\ Q') \in Rel$ **using** *PRelQ*
**proof**(*induct rule*: *outputCases*, *auto simp add*: *pi.inject residual.inject*)
**have** $a\{b\}.P \Longrightarrow_l\hat{}\ a[b] \prec P$ **by**(*rule Output*)
**moreover assume** $(P,\ Q') \in Rel$
**ultimately show** $\exists P'.\ a\{b\}.P \Longrightarrow_l\hat{}\ a[b] \prec P' \wedge (P',\ Q') \in Rel$ **by** *blast*
**qed**
**qed**

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

**and** $a$ :: *name*
**and** $b$ :: *name*
**and** *Rel* :: (*pi* × *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**(*induct rule*: *simCases*)
**case**(*Bound* $Q'$ *c x*)
**have** $x \sharp [a\frown b]P$ **by** *fact*
**hence** *xFreshP*: $(x::name) \sharp P$ **by** *simp*
**have** $[a\frown b]Q \longmapsto c<\nu x> \prec Q'$ **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *matchCases*)
**case** *cMatch*
**have** $Q \longmapsto c<\nu x> \prec Q'$ **by** *fact*
**with** *PSimQ xFreshP* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l\hat{}\ c<\nu x> \prec P'$
**and** *P'RelQ'*: $(P',\ Q') \in Rel$
**by**(*blast dest*: *simE*)
**from** *PTrans* **have** $[a\frown a]P \Longrightarrow_l\hat{}\ c<\nu x> \prec P'$ **by**(*rule Weak-Late-Semantics.Match*)
**with** *P'RelQ' RelRel'* **show** *?case* **by** *blast*
**qed**
**next**
**case**(*Input* $Q'$ *c x*)
**have** $x \sharp [a\frown b]P$ **by** *fact*
**hence** *xFreshP*: $x \sharp P$ **by** *simp*
**have** $[a\frown b]Q \longmapsto c<x> \prec Q'$ **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *matchCases*)
**case** *cMatch*
**have** $Q \longmapsto c<x> \prec Q'$ **by** *fact*
**with** *PSimQ xFreshP* **obtain** $P''$ **where** *L1*: $\forall u.\ \exists P'.\ P \Longrightarrow_l u\ in\ P''\rightarrow c<x>$
$\prec P' \wedge (P',\ Q'[x::=u]) \in Rel$
**by**(*force intro*: *simE*)
**have** $\forall u.\ \exists P'.\ [a\frown a]P \Longrightarrow_l u\ in\ P''\rightarrow c<x> \prec P' \wedge (P',\ Q'[x::=u]) \in Rel'$
**proof**(*rule allI*)
**fix** $u$
**from** *L1* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l u\ in\ P''\rightarrow c<x> \prec P'$ **and** *P'RelQ'*:
$(P',\ Q'[x::=u]) \in Rel$
**by** *blast*
**from** *PTrans* **have** $[a\frown a]P \Longrightarrow_l u\ in\ P''\rightarrow c<x> \prec P'$ **by**(*rule Weak-Late-Step-Semantics.Match*)
**with** *P'RelQ' RelRel'* **show** $\exists P'.\ [a\frown a]P \Longrightarrow_l u\ in\ P''\rightarrow c<x> \prec P' \wedge (P',$
$Q'[x::=u]) \in Rel'$
**by** *blast*
**qed**
**thus** *?case* **by** *blast*

**qed**
**next**
**case**(*Free* $Q'$ $\alpha$)
**have** $[a \frown b]Q \longmapsto \alpha \prec Q'$ **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *matchCases*)
**case** *cMatch*
**have** $Q \longmapsto \alpha \prec Q'$ **by** *fact*
**with** *PSimQ* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l\hat{} \alpha \prec P'$ **and** *PRel*: $(P', Q') \in$ *Rel*
**by**(*blast dest*: *simE*)
**from** *PTrans* **show** *?case*
**proof**(*induct rule*: *transitionCases*)
**case** *Step*
**have** $P \Longrightarrow_l \alpha \prec P'$ **by** *fact*
**hence** $[a \frown a]P \Longrightarrow_l \alpha \prec P'$ **by**(*rule Weak-Late-Step-Semantics.Match*)
**with** *PRel RelRel'* **show** *?case* **by**(*force simp add*: *weakTransition-def*)
**next**
**case** *Stay*
**have** $\alpha \prec P' = \tau \prec P$ **by** *fact*
**hence** *alphaEqTau*: $\alpha = \tau$ **and** *PeqP'*: $P = P'$ **by**(*simp add*: *residual.inject*)+
**have** $[a \frown a]P \Longrightarrow_l\hat{} \tau \prec [a \frown a]P$ **by**(*simp add*: *weakTransition-def*)
**moreover from** *PeqP' PRel* **have** $([a \frown a]P, Q') \in$ *Rel* **by**(*blast intro*: *RelStay*)
**ultimately show** *?case* **using** *RelRel' alphaEqTau* **by** *blast*
**qed**
**qed**
**qed**

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

**assumes** *PSimQ*: $P \leadsto\hat{}$ <*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* ⊆ *Rel'*

**shows** $[a \neq b]P \leadsto\hat{}$ <*Rel'*> $[a \neq b]Q$
**proof**(*cases* $a = b$)
**assume** $a = b$
**thus** *?thesis* **by**(*auto simp add*: *weakSimulation-def*)
**next**
**assume** *aineqb*: $a \neq b$
**show** *?thesis*
**proof**(*induct rule*: *simCases*)
**case**(*Bound* $Q'$ $c$ $x$)

**have** $x \sharp [a{\neq}b]P$ **by** *fact*
**hence** *xFreshP*: $(x{::}name) \sharp P$ **by** *simp*
**have** $[a{\neq}b]Q \longmapsto c{<}\nu x{>} \prec Q'$ **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *mismatchCases*)
**case** *cMismatch*
**have** $Q \longmapsto c{<}\nu x{>} \prec Q'$ **by** *fact*
**with** *PSimQ xFreshP* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l\hat{} c{<}\nu x{>} \prec P'$
**and** $P'RelQ'$: $(P', Q') \in Rel$
**by**(*blast dest*: *simE*)
**from** *PTrans aineqb* **have** $[a{\neq}b]P \Longrightarrow_l\hat{} c{<}\nu x{>} \prec P'$ **by**(*rule Weak-Late-Semantics.Mismatch*)
**with** $P'RelQ'$ $RelRel'$ **show** *?case* **by** *blast*
**qed**
**next**
**case**(*Input* $Q'$ *c x*)
**have** $x \sharp [a{\neq}b]P$ **by** *fact*
**hence** *xFreshP*: $x \sharp P$ **by** *simp*
**have** $[a{\neq}b]Q \longmapsto c{<}x{>} \prec Q'$ **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *mismatchCases*)
**case** *cMismatch*
**have** $Q \longmapsto c{<}x{>} \prec Q'$ **by** *fact*
**with** *PSimQ xFreshP* **obtain** $P''$ **where** *L1*: $\forall u.\ \exists P'.\ P \Longrightarrow_l u\ in\ P''{\rightarrow}c{<}x{>} \prec P' \wedge (P', Q'[x{::=}u]) \in Rel$
**by**(*force intro*: *simE*)
**have** $\forall u.\ \exists P'.\ [a{\neq}b]P \Longrightarrow_l u\ in\ P''{\rightarrow}c{<}x{>} \prec P' \wedge (P', Q'[x{::=}u]) \in Rel'$
**proof**(*rule allI*)
**fix** *u*
**from** *L1* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l u\ in\ P''{\rightarrow}c{<}x{>} \prec P'$ **and** $P'RelQ'$: $(P', Q'[x{::=}u]) \in Rel$
**by** *blast*
**from** *PTrans aineqb* **have** $[a{\neq}b]P \Longrightarrow_l u\ in\ P''{\rightarrow}c{<}x{>} \prec P'$ **by**(*rule Weak-Late-Step-Semantics.Mismatch*)
**with** $P'RelQ'$ $RelRel'$ **show** $\exists P'.\ [a{\neq}b]P \Longrightarrow_l u\ in\ P''{\rightarrow}c{<}x{>} \prec P' \wedge (P', Q'[x{::=}u]) \in Rel'$
**by** *blast*
**qed**
**thus** *?case* **by** *blast*
**qed**
**next**
**case**(*Free* $Q'$ $\alpha$)
**have** $[a{\neq}b]Q \longmapsto \alpha \prec Q'$ **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *mismatchCases*)
**case** *cMismatch*
**have** $a \neq b$ **by** *fact*
**have** $Q \longmapsto \alpha \prec Q'$ **by** *fact*
**with** *PSimQ* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l\hat{} \alpha \prec P'$ **and** *PRel*: $(P', Q') \in Rel$

**by**(*blast dest*: *simE*)
**from** *PTrans* **show** *?case*
**proof**(*induct rule*: *transitionCases*)
**case** *Step*
**have** $P \Longrightarrow_l \alpha \prec P'$ **by** *fact*
**hence** $[a{\neq}b]P \Longrightarrow_l \alpha \prec P'$ **using** $\langle a \neq b \rangle$ **by**(*rule Weak-Late-Step-Semantics.Mismatch*)
**with** *PRel RelRel′* **show** *?case* **by**(*force simp add*: *weakTransition-def*)
**next**
**case** *Stay*
**have** $\alpha \prec P' = \tau \prec P$ **by** *fact*
**hence** *alphaEqTau*: $\alpha = \tau$ **and** *PeqP′*: $P = P'$ **by**(*simp add*: *residual.inject*)+
**have** $[a{\neq}b]P \Longrightarrow_l\hat{}\ \tau \prec [a{\neq}b]P$ **by**(*simp add*: *weakTransition-def*)
**moreover from** *PeqP′ PRel aineqb* **have** $([a{\neq}b]P,\ Q') \in Rel$ **by**(*blast intro*: *RelStay*)
**ultimately show** *?case* **using** *alphaEqTau RelRel′* **by** *blast*
**qed**
**qed**
**qed**
**qed**

**lemma** *parCompose*:
**fixes** $P$ :: *pi*
**and** $Q$ :: *pi*
**and** $R$ :: *pi*
**and** $T$ :: *pi*
**and** *Rel* :: (*pi* × *pi*) *set*
**and** *Rel′* :: (*pi* × *pi*) *set*
**and** *Rel″* :: (*pi* × *pi*) *set*

**assumes** *PSimQ*: $P \leadsto\hat{}<Rel> Q$
**and** *RSimT*: $R \leadsto\hat{}<Rel'> T$
**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** $P \parallel R \leadsto\hat{}<Rel''> Q \parallel T$
**using** ‹*eqvt Rel″*›
**proof**(*induct rule*: *simCasesCont*[**where** $C{=}(P,\ Q,\ R,\ T)$])
**case**(*Bound* $Q'$ *a x*)
**from** $\langle x \sharp (P,\ Q,\ R,\ T) \rangle$ **have** $x \sharp P$ **and** $x \sharp R$ **and** $x \sharp Q$ **and** $x \sharp T$ **by** *simp*+
**from** $\langle Q \parallel T \longmapsto a<\nu x> \prec Q' \rangle$ $\langle x \sharp Q \rangle$ $\langle x \sharp T \rangle$
**show** *?case*
**proof**(*induct rule*: *parCasesB*)
**case**(*cPar1* $Q'$)

**from** *PSimQ* $\langle Q \longmapsto a{<}\nu x{>} \prec Q'\rangle$ $\langle x \sharp P\rangle$ **obtain** $P'$ **where** *PTrans*:$P \Longrightarrow_l\hat{}$ $a{<}\nu x{>} \prec P'$
**and** $P'RelQ'$: $(P', Q') \in Rel$
**by**(*blast dest*: *simE*)
**from** *PTrans* $\langle x \sharp R\rangle$ **have** $P \parallel R \Longrightarrow_l\hat{}\ a{<}\nu x{>} \prec (P' \parallel R)$ **by**(*rule Weak-Late-Semantics.Par1B*)
**moreover from** $P'RelQ'$ $RRel'T$ **have** $(P' \parallel R,\ Q' \parallel T) \in Rel''$ **by**(*rule Par*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*cPar2* $T'$)
**from** *RSimT* $\langle T \longmapsto a{<}\nu x{>} \prec T'\rangle$ $\langle x \sharp R\rangle$ **obtain** $R'$ **where** *RTrans*:$R \Longrightarrow_l\hat{}$ $a{<}\nu x{>} \prec R'$
**and** $R'Rel'T'$: $(R', T') \in Rel'$
**by**(*blast dest*: *simE*)
**from** *RTrans* $\langle x \sharp P\rangle$ $\langle x \sharp R\rangle$ **have** *ParTrans*: $P \parallel R \Longrightarrow_l\hat{}\ a{<}\nu x{>} \prec (P \parallel R')$
**by**(*blast intro*: *Weak-Late-Semantics.Par2B*)
**moreover from** *PRelQ* $R'Rel'T'$ **have** $(P \parallel R',\ Q \parallel T') \in Rel''$ **by**(*rule Par*)
**ultimately show** *?case* **by** *blast*
**qed**
**next**
**case**(*Input* $Q'$ $a$ $x$)
**from** $\langle x \sharp (P, Q, R, T)\rangle$ **have** $x \sharp P$ **and** $x \sharp R$ **and** $x \sharp Q$ **and** $x \sharp T$ **by** *simp+*
**from** $\langle Q \parallel T \longmapsto a{<}x{>} \prec Q'\rangle$ $\langle x \sharp Q\rangle$ $\langle x \sharp T\rangle$
**show** *?case*
**proof**(*induct rule*: *parCasesB*)
**case**(*cPar1* $Q'$)
**from** *PSimQ* $\langle Q \longmapsto a{<}x{>} \prec Q'\rangle$ $\langle x \sharp P\rangle$ **obtain** $P''$
**where** *L1*: $\forall u.\ \exists P'.\ P \Longrightarrow_l u\ in\ P'' \rightarrow a{<}x{>} \prec P' \wedge (P', Q'[x::=u]) \in Rel$
**by**(*blast dest*: *simE*)
**have** $\forall u.\ \exists P'.\ P \parallel R \Longrightarrow_l u\ in\ (P'' \parallel R) \rightarrow a{<}x{>} \prec P' \wedge (P', Q'[x::=u] \parallel T[x::=u]) \in Rel''$
**proof**(*rule allI*)
**fix** $u$
**from** *L1* **obtain** $P'$ **where** *PTrans*:$P \Longrightarrow_l u\ in\ P'' \rightarrow a{<}x{>} \prec P'$
**and** $P'RelQ'$: $(P', Q'[x::=u]) \in Rel$ **by** *blast*
**from** *PTrans* $\langle x \sharp R\rangle$ **have** $P \parallel R \Longrightarrow_l u\ in\ (P'' \parallel R) \rightarrow a{<}x{>} \prec (P' \parallel R)$
**by**(*rule Weak-Late-Step-Semantics.Par1B*)
**moreover from** $P'RelQ'$ $RRel'T$ **have** $(P' \parallel R,\ Q'[x::=u] \parallel T) \in Rel''$ **by**(*rule Par*)
**ultimately show** $\exists P'.\ P \parallel R \Longrightarrow_l u\ in\ (P'' \parallel R) \rightarrow a{<}x{>} \prec P' \wedge$
$(P', Q'[x::=u] \parallel (T[x::=u])) \in Rel''$ **using** $\langle x \sharp T\rangle$
**by**(*force simp add*: *forget*)
**qed**
**thus** *?case* **by** *force*
**next**
**case**(*cPar2* $T'$)
**from** *RSimT* $\langle T \longmapsto a{<}x{>} \prec T'\rangle$ $\langle x \sharp R\rangle$ **obtain** $R''$
**where** *L1*: $\forall u.\ \exists R'.\ R \Longrightarrow_l u\ in\ R'' \rightarrow a{<}x{>} \prec R' \wedge (R', T'[x::=u]) \in Rel'$
**by**(*blast dest*: *simE*)
**have** $\forall u.\ \exists P'.\ P \parallel R \Longrightarrow_l u\ in\ (P \parallel R'') \rightarrow a{<}x{>} \prec P' \wedge (P', Q[x::=u] \parallel$

$T'[x::=u]) \in Rel''$
  **proof**(*rule allI*)
   **fix** *u*
   **from** *L1* **obtain** $R'$ **where** *RTrans*:$R \Longrightarrow_l u$ *in* $R''\rightarrow a<x> \prec R'$
          **and** $R'Rel'T'$: $(R', T'[x::=u]) \in Rel'$ **by** *blast*
   **from** *RTrans* $\langle x \sharp P\rangle$ **have** *ParTrans*: $P \parallel R \Longrightarrow_l u$ *in* $(P \parallel R'')\rightarrow a<x> \prec (P \parallel R')$
    **by**(*rule Weak-Late-Step-Semantics.Par2B*)

   **moreover from** *PRelQ* $R'Rel'T'$ **have** $(P \parallel R', Q \parallel T'[x::=u]) \in Rel''$ **by**(*rule Par*)

   **ultimately show** $\exists P'. P \parallel R \Longrightarrow_l u$ *in* $(P \parallel R'')\rightarrow a<x> \prec P' \wedge$
          $(P', Q[x::=u] \parallel T'[x::=u]) \in Rel''$ **using** $\langle x \sharp Q\rangle$
    **by**(*force simp add*: *forget*)
  **qed**
  **thus** *?case* **by** *force*
 **qed**
**next**
 **case**(*Free QT'* $\alpha$)
 **have** $Q \parallel T \longmapsto \alpha \prec QT'$ **by** *fact*
 **thus** *?case*
 **proof**(*induct rule*: *parCasesF*[*of* - - - - - $(P, R)$])
  **case**(*cPar1* $Q'$)
  **have** $Q \longmapsto \alpha \prec Q'$ **by** *fact*
  **with** *PSimQ* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l\hat{} \alpha \prec P'$ **and** *PRel*: $(P', Q') \in Rel$
   **by**(*blast dest*: *simE*)
  **from** *PTrans* **have** *Trans*: $P \parallel R \Longrightarrow_l\hat{} \alpha \prec P' \parallel R$ **by**(*rule Weak-Late-Semantics.Par1F*)
  **moreover from** *PRel RRel'T* **have** $(P' \parallel R, Q' \parallel T) \in Rel''$ **by**(*blast intro*: *Par*)
  **ultimately show** *?case* **by** *blast*
 **next**
  **case**(*cPar2* $T'$)
  **have** $T \longmapsto \alpha \prec T'$ **by** *fact*
  **with** *RSimT* **obtain** $R'$ **where** *RTrans*: $R \Longrightarrow_l\hat{} \alpha \prec R'$ **and** *RRel*: $(R', T') \in Rel'$
   **by**(*blast dest*: *simE*)
  **from** *RTrans* **have** *Trans*: $P \parallel R \Longrightarrow_l\hat{} \alpha \prec P \parallel R'$ **by**(*rule Weak-Late-Semantics.Par2F*)
  **moreover from** *PRelQ RRel* **have** $(P \parallel R', Q \parallel T') \in Rel''$ **by**(*blast intro*: *Par*)
  **ultimately show** *?case* **by** *blast*
 **next**
  **case**(*cComm1* $Q'$ $T'$ *a b x*)
  **have** *QTrans*: $Q \longmapsto a<x> \prec Q'$ **and** *TTrans*: $T \longmapsto a[b] \prec T'$ **by** *fact+*
  **have** $x \sharp (P, R)$ **by** *fact*
  **hence** *xFreshP*: $x \sharp P$ **by**(*simp add*: *fresh-prod*)

  **from** *PSimQ QTrans xFreshP* **obtain** $P'$ $P''$ **where** *PTrans*: $P \Longrightarrow_l b$ *in*

$P''{\rightarrow}a{<}x{>} \prec P'$
**and** $P'RelQ'$: $(P', Q'[x{::=}b]) \in Rel$
**by**(*blast dest*: *simE*)

**from** *RSimT TTrans* **obtain** $R'$ **where** *RTrans*: $R \Longrightarrow_l\hat{} a[b] \prec R'$
**and** *RRel*: $(R', T') \in Rel'$
**by**(*blast dest*: *simE*)

**from** *PTrans RTrans* **have** $P \parallel R \Longrightarrow_l\hat{}\ \tau \prec P' \parallel R'$ **by**(*rule Weak-Late-Semantics.Comm1*)
**moreover from** $P'RelQ'$ *RRel* **have** $(P' \parallel R', Q'[x{::=}b] \parallel T') \in Rel''$ **by**(*rule Par*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*cComm2* $Q'$ $T'$ $a$ $b$ $x$)
**have** *QTrans*: $Q \longmapsto a[b] \prec Q'$ **and** *TTrans*: $T \longmapsto a{<}x{>} \prec T'$ **by** *fact+*
**have** $x \sharp (P, R)$ **by** *fact*
**hence** *xFreshR*: $x \sharp R$ **by**(*simp add*: *fresh-prod*)

**from** *PSimQ QTrans* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l\hat{} a[b] \prec P'$
**and** *PRel*: $(P', Q') \in Rel$
**by**(*blast dest*: *simE*)

**from** *RSimT TTrans xFreshR* **obtain** $R'$ $R''$ **where** *RTrans*: $R \Longrightarrow_l b$ *in* $R''{\rightarrow}a{<}x{>} \prec R'$
**and** $R'Rel'T'$: $(R', T'[x{::=}b]) \in Rel'$
**by**(*blast dest*: *simE*)

**from** *PTrans RTrans* **have** $P \parallel R \Longrightarrow_l\hat{}\ \tau \prec P' \parallel R'$ **by**(*rule Weak-Late-Semantics.Comm2*)
**moreover from** *PRel* $R'Rel'T'$ **have** $(P' \parallel R', Q' \parallel T'[x{::=}b]) \in Rel''$ **by**(*rule Par*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*cClose1* $Q'$ $T'$ $a$ $x$ $y$)
**have** *QTrans*: $Q \longmapsto a{<}x{>} \prec Q'$ **and** *TTrans*: $T \longmapsto a{<}\nu y{>} \prec T'$ **by** *fact+*
**have** $x \sharp (P, R)$ **and** $y \sharp (P, R)$ **by** *fact+*
**hence** *xFreshP*: $x \sharp P$ **and** *yFreshR*: $y \sharp R$ **and** *yFreshP*: $y \sharp P$ **by**(*simp add*: *fresh-prod*)+

**from** *PSimQ QTrans xFreshP* **obtain** $P'$ $P''$ **where** *PTrans*: $P \Longrightarrow_l y$ *in* $P''{\rightarrow}a{<}x{>} \prec P'$
**and** $P'RelQ'$: $(P', Q'[x{::=}y]) \in Rel$
**by**(*blast dest*: *simE*)

**from** *RSimT TTrans yFreshR* **obtain** $R'$ **where** *RTrans*: $R \Longrightarrow_l\hat{} a{<}\nu y{>} \prec R'$

**and** $R'Rel'T'$: $(R', T') \in Rel'$
**by**(*blast dest*: *simE*)

**from** *PTrans RTrans yFreshP yFreshR* **have** *Trans*: $P \parallel R \Longrightarrow_l\hat{}\ \tau \prec {<}\nu y{>}(P'$

$\parallel R')$
**by**(*rule Weak-Late-Semantics.Close1*)
**moreover from** $P'RelQ'$ $R'Rel'T'$ **have** $(<\nu y>(P' \parallel R'),\ <\nu y>(Q'[x::=y] \parallel T')) \in Rel''$
**by**(*blast intro*: *Par Res*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*cClose2* $Q'$ $T'$ *a x y*)
**have** *QTrans*: $Q \longmapsto a<\nu y> \prec Q'$ **and** *TTrans*: $T \longmapsto a<x> \prec T'$ **by** *fact*+
**have** $x \sharp (P, R)$ **and** $y \sharp (P, R)$ **by** *fact*+
**hence** *xFreshR*: $x \sharp R$ **and** *yFreshP*: $y \sharp P$ **and** *yFreshR*: $y \sharp R$ **by**(*simp add*: *fresh-prod*)+

**from** *PSimQ QTrans yFreshP* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l\hat{}\ a<\nu y> \prec P'$
**and** $P'RelQ'$: $(P', Q') \in Rel$
**by**(*blast dest*: *simE*)

**from** *RSimT TTrans xFreshR* **obtain** $R'$ $R''$ **where** *RTrans*: $R \Longrightarrow_l y$ *in* $R''{\rightarrow}a<x> \prec R'$
**and** $R'Rel'T'$: $(R', T'[x::=y]) \in Rel'$
**by**(*blast dest*: *simE*)

**from** *PTrans RTrans yFreshP yFreshR* **have** *Trans*: $P \parallel R \Longrightarrow_l\hat{}\ \tau \prec <\nu y>(P' \parallel R')$
**by**(*rule Weak-Late-Semantics.Close2*)
**moreover from** $P'RelQ'$ $R'Rel'T'$ **have** $(<\nu y>(P' \parallel R'),\ <\nu y>(Q' \parallel T'[x::=y])) \in Rel''$
**by**(*blast intro*: *Par Res*)
**ultimately show** *?case* **by** *blast*
**qed**
**qed**

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

**assumes** *PSimQ*: $P \leadsto\hat{}<Rel>\ Q$
**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** $P \parallel R \leadsto\hat{}<Rel'>\ Q \parallel R$

**proof** −
  **note** *PSimQ*
  **moreover have** *RSimR*: $R \leadsto\hat{}<Id> R$ **by**(*auto intro*: *reflexive*)
  **moreover note** *PRelQ* **moreover have** $(R, R) \in Id$ **by** *auto*
  **moreover from** *Par* **have** $\bigwedge P\ Q\ R\ T.\ [\![(P, Q) \in Rel; (R, T) \in Id]\!] \Longrightarrow (P \parallel$
$R, Q \parallel T) \in Rel'$
    **by** *auto*
  **moreover note** *Res* ‹*eqvt Rel*›
  **moreover have** *eqvt Id* **by**(*auto simp add*: *eqvt-def*)
  **ultimately show** *?thesis* **using** *EqvtRel′* **by**(*rule parCompose*)
**qed**

**lemma** *resPres*:
  **fixes** $P$ :: *pi*
  **and** $Q$ :: *pi*
  **and** *Rel* :: $(pi \times pi)$ *set*
  **and** $x$ :: *name*
  **and** *Rel′* :: $(pi \times pi)$ *set*

  **assumes** *PSimQ*: $P \leadsto\hat{}<Rel> Q$
  **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** $<\nu x>P \leadsto\hat{}<Rel'> <\nu x>Q$
**proof** −
  **from** *EqvtRel′* **show** *?thesis*
  **proof**(*induct rule*: *simCasesCont*[*of* - $(P, Q, x)$])
    **case**(*Bound* $Q'$ $a$ $y$)
    **have** *Trans*: $<\nu x>Q \longmapsto a<\nu y> \prec Q'$ **by** *fact*
    **have** $y \sharp (P, Q, x)$ **by** *fact*
    **hence** *yineqx*: $y \neq x$ **and** *yFreshP*: $y \sharp P$ **and** $y \sharp Q$ **by**(*simp add*: *fresh-prod*)+
    **from** *Trans* ‹$y \neq x$› ‹$y \sharp Q$› **show** *?case*
    **proof**(*induct rule*: *resCasesB*)
      **case**(*cOpen* $a$ $Q'$)
      **have** *QTrans*: $Q \longmapsto a[x] \prec Q'$ **and** *aineqx*: $a \neq x$ **by** *fact*+

      **from** *PSimQ QTrans* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l\hat{}\ a[x] \prec P'$
                **and** *P′RelQ′*: $(P', Q') \in Rel$
        **by**(*blast dest*: *simE*)

      **have** $<\nu x>P \Longrightarrow_l\hat{}\ a<\nu y> \prec ([(y, x)] \cdot P')$
      **proof** −
      **from** *PTrans aineqx* **have** $<\nu x>P \Longrightarrow_l\hat{}\ a<\nu x> \prec P'$ **by**(*rule Weak-Late-Semantics.Open*)
      **moreover from** *PTrans yFreshP* **have** $y \sharp P'$ **by**(*force intro*: *freshTransition*)
        **ultimately show** *?thesis* **by**(*simp add*: *alphaBoundResidual name-swap*)
      **qed**

**moreover from** *EqvtRel P′RelQ′ RelRel′* **have** $([(y, x)] \cdot P', [(y, x)] \cdot Q') \in$ *Rel′*
**by**(*blast intro*: *eqvtRelI*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*cRes Q′*)
**have** *QTrans*: $Q \longmapsto a<\nu y> \prec Q'$ **by** *fact*
**from** $\langle x \sharp$ *BoundOutputS a*$\rangle$ **have** $x \neq a$ **by** *simp*

**from** *PSimQ yFreshP QTrans* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l \hat{}\, a<\nu y> \prec P'$
**and** *P′RelQ′*: $(P', Q') \in Rel$
**by**(*blast dest*: *simE*)
**from** *PTrans* $\langle x \neq a\rangle$ *yineqx yFreshP* **have** *ResTrans*: $<\nu x>P \Longrightarrow_l \hat{}\, a<\nu y> \prec (<\nu x>P')$
**by**(*blast intro*: *Weak-Late-Semantics.ResB*)
**moreover from** *P′RelQ′* **have** $((<\nu x>P'), (<\nu x>Q')) \in Rel'$
**by**(*rule ResRel*)
**ultimately show** *?case* **by** *blast*
**qed**
**next**
**case**(*Input Q′ a y*)
**have** $y \sharp (P, Q, x)$ **by** *fact*
**hence** *yineqx*: $y \neq x$ **and** *yFreshP*: $y \sharp P$ **and** $y \sharp Q$ **by**(*simp add*: *fresh-prod*)+

**have** $<\nu x>Q \longmapsto a<y> \prec Q'$ **by** *fact*
**thus** *?case* **using** *yineqx* $\langle y \sharp Q\rangle$
**proof**(*induct rule*: *resCasesB*)
**case**(*cOpen a Q′*)
**thus** *?case* **by** *simp*
**next**
**case**(*cRes Q′*)
**have** *QTrans*: $Q \longmapsto a<y> \prec Q'$ **by** *fact*
**from** $\langle x \sharp$ *InputS a*$\rangle$ **have** $x \neq a$ **by** *simp*

**from** *PSimQ QTrans yFreshP* **obtain** $P''$
**where** *L1*: $\forall u.\ \exists P'.\ P \Longrightarrow_l u\ in\ P'' \rightarrow a<y> \prec P' \wedge (P', Q'[y::=u]) \in Rel$
**by**(*blast dest*: *simE*)
**have** $\forall u.\ \exists P'.\ <\nu x>P \Longrightarrow_l u\ in\ (<\nu x>P'') \rightarrow a<y> \prec P' \wedge (P', (<\nu x>Q')[y::=u]) \in Rel'$
**proof**(*rule allI*)
**fix** *u*
**show** $\exists P'.\ <\nu x>P \Longrightarrow_l u\ in\ <\nu x>P'' \rightarrow a<y> \prec P' \wedge (P', (<\nu x>Q')[y::=u]) \in Rel'$
**proof**(*cases x=u*)
**assume** *xequ*: *x=u*

**have** $\exists c::name.\ c \sharp (P, P'', Q', x, y, a)$ **by**(*blast intro*: *name-exists-fresh*)
**then obtain** *c::name* **where** *cFreshP*: $c \sharp P$ **and** *cFreshP′′*: $c \sharp P''$ **and**

*cFreshQ'*: $c \sharp Q'$
**and** *cineqx*: $c \neq x$ **and** *cineqy*: $c \neq y$ **and** *cineqa*: $c \neq a$
**by**(*force simp add*: *fresh-prod*)

**from** *L1* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l c$ *in* $P''\to a{<}y{>} \prec P'$
**and** *P'RelQ'*: $(P', Q'[y::=c]) \in Rel$
**by** *blast*
**have** $<\nu x>P \Longrightarrow_l u$ *in* $(<\nu x>P'')\to a{<}y{>} \prec <\nu c>([(x, c)] \cdot P')$
**proof** −
**from** *PTrans yineqx* ‹$x \neq a$› *cineqx* **have** $<\nu x>P \Longrightarrow_l c$ *in* $(<\nu x>P'')\to a{<}y{>}$
$\prec <\nu x>P'$
**by**(*blast intro*: *Weak-Late-Step-Semantics.ResB*)
**hence** $([(x, c)] \cdot <\nu x>P) \Longrightarrow_l([(x, c)] \cdot c)$ *in* $([(x, c)] \cdot <\nu x>P'')\to([(x,$
$c)] \cdot a){<}([(x, c)] \cdot y){>} \prec [(x, c)] \cdot <\nu x>P'$
**by**(*rule Weak-Late-Step-Semantics.eqvtI*)
**moreover from** *cFreshP* **have** $<\nu c>([(x, c)] \cdot P) = <\nu x>P$ **by**(*simp*
*add*: *alphaRes*)
**moreover from** *cFreshP''* **have** $<\nu c>([(x, c)] \cdot P'') = <\nu x>P''$ **by**(*simp*
*add*: *alphaRes*)
**ultimately show** *?thesis* **using** ‹$x \neq a$› *cineqa yineqx cineqy cineqx xequ*
**by**(*simp add*: *name-calc*)
**qed**
**moreover have** $(<\nu c>([(x, c)] \cdot P'), (<\nu x>Q')[y::=u]) \in Rel'$
**proof** −
**from** *P'RelQ'* **have** $(<\nu x>P', <\nu x>(Q'[y::=c])) \in Rel'$ **by**(*rule ResRel*)
**with** *EqvtRel'* **have** $([(x, c)] \cdot <\nu x>P', [(x, c)] \cdot <\nu x>(Q'[y::=c])) \in$
$Rel'$ **by**(*rule eqvtRelI*)
**with** *cineqy yineqx cineqx* **have** $(<\nu c>([(x, c)] \cdot P'), (<\nu c>([(x, c)] \cdot$
$Q'))[y::=x]) \in Rel'$
**by**(*simp add*: *name-calc eqvt-subs*)
**with** *cFreshQ' xequ* **show** *?thesis* **by**(*simp add*: *alphaRes*)
**qed**
**ultimately show** *?thesis* **by** *blast*
**next**
**assume** *xinequ*: $x \neq u$
**from** *L1* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l u$ *in* $P''\to a{<}y{>} \prec P'$
**and** *P'RelQ'*: $(P', Q'[y::=u]) \in Rel$ **by** *blast*

**from** *PTrans* ‹$x \neq a$› *yineqx xinequ* **have** $<\nu x>P \Longrightarrow_l u$ *in* $(<\nu x>P'')\to a{<}y{>}$
$\prec <\nu x>P'$
**by**(*blast intro*: *Weak-Late-Step-Semantics.ResB*)
**moreover from** *P'RelQ' xinequ yineqx* **have** $(<\nu x>P', (<\nu x>Q')[y::=u])$
$\in Rel'$
**by**(*force intro*: *ResRel*)
**ultimately show** *?thesis* **by** *blast*
**qed**
**qed**
**thus** *?case* **by** *blast*
**qed**

**next**
**case**(*Free* $Q'$ $\alpha$)
**have** $<\nu x>Q \longmapsto \alpha \prec Q'$ **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *resCasesF*)
**case**(*cRes* $Q'$)
**have** $Q \longmapsto\alpha \prec Q'$ **by** *fact*
**with** *PSimQ* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l\hat{}\ \alpha \prec P'$
**and** $P'RelQ'$: $(P', Q') \in Rel$
**by**(*blast dest*: *simE*)

**have** $<\nu x>P \Longrightarrow_l\hat{}\ \alpha \prec <\nu x>P'$
**proof** −
**have** *xFreshAlpha*: $x \sharp \alpha$ **by** *fact*
**with** *PTrans* **show** *?thesis* **by**(*rule ResF*)
**qed**
**moreover from** $P'RelQ'$ **have** $(<\nu x>P', <\nu x>Q') \in Rel'$ **by**(*rule ResRel*)
**ultimately show** *?case* **by** *blast*
**qed**
**qed**
**qed**

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

**assumes** *eqvtRel*: *eqvt Rel*
**and** *Res*: $\bigwedge P\ Q\ a.\ (P, Q) \in Rel \Longrightarrow (<\nu a>P, <\nu a>Q) \in Rel$
**and** *PRelQ*: $P \rightsquigarrow\hat{}<Rel>\ Q$

**shows** $(resChain\ lst)\ P \rightsquigarrow\hat{}<Rel>\ (resChain\ lst)\ Q$
**proof** −
**show** *?thesis*
**proof**(*induct lst*)
**from** *PRelQ* **show** $resChain\ []\ P \rightsquigarrow\hat{}<Rel>\ resChain\ []\ Q$ **by** *simp*
**next**
**fix** *a lst*
**assume** *IH*: $(resChain\ lst\ P) \rightsquigarrow\hat{}<Rel>\ (resChain\ lst\ Q)$
**moreover from** *Res* **have** $\bigwedge P\ Q\ a.\ (P, Q) \in Rel \Longrightarrow (<\nu a>P, <\nu a>Q) \in Rel$
**by** *simp*
**moreover have** $Rel \subseteq Rel$ **by** *simp*
**ultimately have** $<\nu a>(resChain\ lst\ P) \rightsquigarrow\hat{}<Rel>\ <\nu a>(resChain\ lst\ Q)$ **using** *eqvtRel*
**by**(*rule-tac resPres*)
**thus** $resChain\ (a\ \#\ lst)\ P \rightsquigarrow\hat{}<Rel>\ resChain\ (a\ \#\ lst)\ Q$
**by** *simp*

**qed**
**qed**

**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** −
**have** $\bigwedge Rs\ P.\ [\![!Q \longmapsto Rs; (P, !Q) \in$ *bangRel Rel*$]\!] \Longrightarrow$ *weakSimAct P Rs P Rel'*
**proof** −
**fix** *Rs P*
**assume** $!Q \longmapsto Rs$ **and** $(P, !Q) \in$ *bangRel Rel*
**thus** *weakSimAct P Rs P Rel'*
**proof**(*nominal-induct avoiding*: *P rule*: *bangInduct*)
**case**(*cPar1B aa x Q'*)
**have** *QTrans*: $Q \longmapsto aa«x» \prec Q'$ **and** *xFreshQ*: $x \sharp Q$ **by** *fact+*
**have** $(P, Q \parallel !Q) \in$ *bangRel Rel* **and** $x \sharp P$ **by** *fact+*
**thus** *?case*
**proof**(*induct rule*: *BRParCases*)
**case**(*BRPar P R*)
**have** *PRelQ*: $(P, Q) \in Rel$ **and** *RBangRelT*: $(R, !Q) \in$ *bangRel Rel* **by** *fact+*
**have** $x \sharp P \parallel R$ **by** *fact*
**hence** *xFreshP*: $x \sharp P$ **and** *xFreshR*: $x \sharp R$ **by** *simp+*
**from** *PRelQ* **have** *PSimQ*: $P \leadsto\hat{}$ <*Rel*> $Q$ **by**(*rule Sim*)
**from** *eqvtRel'* **show** *?case*
**proof**(*induct rule*: *simActBoundCases*)
**case**(*Input a*)
**have** $aa = InputS\ a$ **by** *fact*
**with** *PSimQ QTrans xFreshP* **obtain** $P''$
**where** *L1*: $\forall u.\ \exists P'.\ P \Longrightarrow_l u$ *in* $P''$→$a$<$x$> $\prec P' \wedge (P', Q'[x::=u]) \in Rel$
**by**(*blast dest*: *simE*)
**have** $\forall u.\ \exists P'.\ P \parallel R \Longrightarrow_l u$ *in* $(P'' \parallel R)$→$a$<$x$> $\prec P' \wedge (P', (Q' \parallel !Q)[x::=u]) \in Rel'$

**proof**(*rule allI*)
**fix** *u*
**from** *L1* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l u \ in \ P''{\rightarrow}a{<}x{>} \prec P'$
**and** $P'RelQ'$: $(P', Q'[x{::=}u]) \in Rel$
**by** *blast*

**from** *PTrans xFreshR* **have** $P \parallel R \Longrightarrow_l u \ in \ (P'' \parallel R){\rightarrow}a{<}x{>}{\prec} P' \parallel R$
**by**(*rule Weak-Late-Step-Semantics.Par1B*)
**moreover have** $(P' \parallel R, (Q' \parallel !Q)[x{::=}u]) \in Rel'$
**proof** −
**from** $P'RelQ'$ *RBangRelT* **have** $(P' \parallel R, Q'[x{::=}u] \parallel !Q) \in bangRel$
*Rel*
**by**(*rule Rel.BRPar*)
**with** *xFreshQ* $BangRelRel'$ **show** *?thesis* **by**(*auto simp add*: *forget*)
**qed**
**ultimately show** $\exists P'. \ P \parallel R \Longrightarrow_l u \ in \ (P'' \parallel R){\rightarrow}a{<}x{>} \prec P' \land$
$(P', (Q' \parallel !Q)[x{::=}u]) \in Rel'$ **by** *blast*
**qed**
**thus** *?case* **by** *blast*
**next**
**case**(*BoundOutput a*)
**have** $aa = BoundOutputS \ a$ **by** *fact*
**with** *PSimQ QTrans xFreshP* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l\hat{} \ a{<}\nu x{>}$
$\prec P'$
**and** $P'RelQ'$: $(P', Q') \in Rel$
**by**(*blast dest*: *simE*)
**from** *PTrans xFreshR* **have** $P \parallel R \Longrightarrow_l\hat{} \ a{<}\nu x{>}{\prec} P' \parallel R$
**by**(*rule Weak-Late-Semantics.Par1B*)
**moreover from** $P'RelQ'$ *RBangRelT* $BangRelRel'$ **have** $(P' \parallel R, Q' \parallel$
$!Q) \in Rel'$
**by**(*blast intro*: *Rel.BRPar*)
**ultimately show** *?case* **by** *blast*
**qed**
**qed**
**next**
**case**($cPar1F \ \alpha \ Q' \ P$)
**have** *QTrans*: $Q \longmapsto \alpha \prec Q'$ **by** *fact*
**have** $(P, Q \parallel !Q) \in bangRel \ Rel$ **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *BRParCases*)
**case**(*BRPar P R*)
**have** *PRelQ*: $(P, Q) \in Rel$ **and** *RBangRelQ*: $(R, !Q) \in bangRel \ Rel$ **by**
*fact+*
**show** *?case*
**proof**(*induct rule*: *simActFreeCases*)
**case** *Der*
**from** *PRelQ* **have** $P \leadsto\hat{} {<}Rel{>} \ Q$ **by**(*rule Sim*)
**with** *QTrans* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l\hat{} \ \alpha \prec P'$ **and** $P'RelQ'$:
$(P', Q') \in Rel$

**by**(*blast dest*: *simE*)

**from** *PTrans* **have** $P \parallel R \Longrightarrow_l\hat{}\ \alpha \prec P' \parallel R$ **by**(*rule Weak-Late-Semantics.Par1F*)
**moreover from** *P'RelQ' RBangRelQ* **have** $(P' \parallel R,\ Q' \parallel !Q) \in$ *bangRel Rel*
**by**(*rule Rel.BRPar*)
**ultimately show** *?case* **using** *BangRelRel'* **by** *blast*
**qed**
**qed**
**next**
**case**(*cPar2B aa x Q' P*)
**have** *IH*: $\bigwedge P.\ (P,\ !Q) \in$ *bangRel Rel* $\Longrightarrow$ *weakSimAct P* (*aa«x»* $\prec Q'$) *P Rel'* **by** *fact*
**have** *xFreshQ*: $x \sharp Q$ **by** *fact*
**have** $(P,\ Q \parallel !Q) \in$ *bangRel Rel* **and** $x \sharp P$ **by** *fact+*
**thus** *?case*
**proof**(*induct rule*: *BRParCases*)
**case**(*BRPar P R*)
**have** *PRelQ*: $(P,\ Q) \in$ *Rel* **and** *RBangRelQ*: $(R,\ !Q) \in$ *bangRel Rel* **by** *fact+*
**have** $x \sharp P \parallel R$ **by** *fact*
**hence** *xFreshP*: $x \sharp P$ **and** *xFreshR*: $x \sharp R$ **by** *simp+*
**from** *eqvtRel'* **show** *?case*
**proof**(*induct rule*: *simActBoundCases*)
**case**(*Input a*)
**have** *aa* = *InputS a* **by** *fact*
**with** *RBangRelQ IH* **have** *weakSimAct R* (*a<x>* $\prec Q'$) *R Rel'* **by** *blast*
**with** *xFreshR* **obtain** $R''$ **where** *L1*: $\forall u.\ \exists R'.\ R \Longrightarrow_l u\ in\ R''\rightarrow a$<x> $\prec R' \wedge (R',\ Q'[x::=u]) \in Rel'$
**by**(*force simp add*: *weakSimAct-def*)
**have** $\forall u.\ \exists P'.\ P \parallel R \Longrightarrow_l u\ in\ (P \parallel R'')\rightarrow a$<x> $\prec P' \wedge (P',\ (Q \parallel Q')[x::=u]) \in Rel'$
**proof**(*rule allI*)
**fix** *u*
**from** *L1* **obtain** $R'$ **where** *RTrans*: $R \Longrightarrow_l u\ in\ R''\rightarrow a$<x> $\prec R'$
**and** *R'Rel'Q'*: $(R',\ Q'[x::=u]) \in Rel'$
**by** *blast*

**from** *RTrans xFreshP* **have** $P \parallel R \Longrightarrow_l u\ in\ (P \parallel R'')\rightarrow a$<x> $\prec P \parallel R'$
**by**(*rule Weak-Late-Step-Semantics.Par2B*)
**moreover have** $(P \parallel R',\ (Q \parallel Q')[x::=u]) \in Rel'$
**proof** −
**from** *PRelQ R'Rel'Q'* **have** $(P \parallel R',\ Q \parallel Q'[x::=u]) \in Rel'$
**by**(*rule ParComp*)
**with** *xFreshQ* **show** *?thesis* **by**(*simp add*: *forget*)
**qed**
**ultimately show** $\exists P'.\ P \parallel R \Longrightarrow_l u\ in\ (P \parallel R'')\rightarrow a$<x> $\prec P' \wedge (P',\ (Q \parallel Q')[x::=u]) \in Rel'$
**by** *blast*

**qed**
**thus** *?case* **by** *blast*
**next**
**case**(*BoundOutput a*)
**have** $aa = BoundOutputS\ a$ **by** *fact*
**with** *IH RBangRelQ* **have** *weakSimAct R* ($a{<}\nu x{>} \prec Q'$) *R Rel'* **by** *blast*
**with** *xFreshR* **obtain** $R'$ **where** *RTrans*: $R \Longrightarrow_l\hat{}\ a{<}\nu x{>} \prec R'$ **and** $R'BangRelQ'$: $(R', Q') \in Rel'$
**by**(*simp add*: *weakSimAct-def*, *blast*)

**from** *RTrans xFreshP* **have** $P \parallel R \Longrightarrow_l\hat{}\ a{<}\nu x{>} \prec P \parallel R'$
**by**(*auto intro*: *Weak-Late-Semantics.Par2B*)
**moreover from** *PRelQ* $R'BangRelQ'$ **have** $(P \parallel R', Q \parallel Q') \in Rel'$
**by**(*rule ParComp*)
**ultimately show** *?case* **by** *blast*
**qed**
**qed**
**next**
**case**(*cPar2F* $\alpha$ $Q'$ *P*)
**have** *IH*: $\bigwedge P.\ (P, !Q) \in bangRel\ Rel \Longrightarrow weakSimAct\ P\ (\alpha \prec Q')\ P\ Rel'$ **by** *fact*
**have** $(P, Q \parallel !Q) \in bangRel\ Rel$ **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *BRParCases*)
**case**(*BRPar P R*)
**have** *PRelQ*: $(P, Q) \in Rel$ **and** *RBangRelQ*: $(R, !Q) \in bangRel\ Rel$ **by** *fact+*
**show** *?case*
**proof**(*induct rule*: *simActFreeCases*)
**case** *Der*
**from** *RBangRelQ* **have** *weakSimAct R* ($\alpha \prec Q'$) *R Rel'* **by**(*rule IH*)
**then obtain** $R'$ **where** *RTrans*: $R \Longrightarrow_l\hat{}\ \alpha \prec R'$ **and** $R'RelQ'$: $(R', Q') \in Rel'$
**by**(*simp add*: *weakSimAct-def*, *blast*)

**from** *RTrans* **have** $P \parallel R \Longrightarrow_l\hat{}\ \alpha \prec P \parallel R'$ **by**(*rule Weak-Late-Semantics.Par2F*)
**moreover from** *PRelQ* $R'RelQ'$ **have** $(P \parallel R', Q \parallel Q') \in Rel'$ **by**(*rule ParComp*)
**ultimately show** *?case* **by** *blast*
**qed**
**qed**
**next**
**case**(*cComm1 a x* $Q'$ *b* $Q''$ *P*)
**have** *QTrans*: $Q \longmapsto a{<}x{>} \prec Q'$ **by** *fact*
**have** *IH*: $\bigwedge P.\ (P, !Q) \in bangRel\ Rel \Longrightarrow weakSimAct\ P\ (a[b] \prec Q'')\ P\ Rel'$ **by** *fact*
**have** $(P, Q \parallel !Q) \in bangRel\ Rel$ **and** $x \sharp P$ **by** *fact+*
**thus** *?case*
**proof**(*induct rule*: *BRParCases*)

**case**(*BRPar P R*)
**have** *PRelQ*: $(P,\ Q) \in Rel$ **and** *RBangRelQ*: $(R,\ !Q) \in bangRel\ Rel$ **by** *fact+*
**have** $x \sharp P \parallel R$ **by** *fact*
**hence** *xFreshP*: $x \sharp P$ **by** *simp*
**show** *?case*
**proof**(*induct rule*: *simActFreeCases*)
**case** *Der*
**from** *PRelQ* **have** $P \leadsto\hat{}<Rel>\ Q$ **by**(*rule Sim*)
**with** *QTrans xFreshP* **obtain** $P'\ P''$ **where** *PTrans*: $P \Longrightarrow_l b\ in\ P''\rightarrow a<x> \prec P'$
**and** $P'RelQ'$: $(P',\ Q'[x{::=}b]) \in Rel$
**by**(*blast dest*: *simE*)

**from** *RBangRelQ* **have** *weakSimAct* $R\ (a[b] \prec Q'')\ R\ Rel'$ **by**(*rule IH*)
**then obtain** $R'$ **where** *RTrans*: $R \Longrightarrow_l\hat{}\ a[b] \prec R'$
**and** $R'RelQ''$: $(R',\ Q'') \in Rel'$
**by**(*simp add*: *weakSimAct-def*, *blast*)

**from** *PTrans RTrans* **have** $P \parallel R \Longrightarrow_l\hat{}\ \tau \prec (P' \parallel R')$
**by**(*rule Weak-Late-Semantics.Comm1*)
**moreover from** $P'RelQ'\ R'RelQ''$ **have** $(P' \parallel R',\ Q'[x{::=}b] \parallel Q'') \in Rel'$
**by**(*rule ParComp*)
**ultimately show** *?case* **by** *blast*
**qed**
**qed**
**next**
**case**(*cComm2 a b* $Q'$ *x* $Q''$ *P*)
**have** *QTrans*: $Q \longmapsto a[b] \prec Q'$ **by** *fact*
**have** *IH*: $\bigwedge P.\ (P,\ !Q) \in bangRel\ Rel \Longrightarrow weakSimAct\ P\ (a<x> \prec Q'')\ P\ Rel'$ **by** *fact*
**have** $(P,\ Q \parallel\ !Q) \in bangRel\ Rel$ **and** $x \sharp P$ **by** *fact+*
**thus** *?case*
**proof**(*induct rule*: *BRParCases*)
**case**(*BRPar P R*)
**have** *PRelQ*: $(P,\ Q) \in Rel$ **and** *RBangRelQ*: $(R,\ !Q) \in bangRel\ Rel$ **by** *fact+*
**have** $x \sharp P \parallel R$ **by** *fact*
**hence** *xFreshR*: $x \sharp R$ **by** *simp*
**show** *?case*
**proof**(*induct rule*: *simActFreeCases*)
**case** *Der*
**from** *PRelQ* **have** $P \leadsto\hat{}<Rel>\ Q$ **by**(*rule Sim*)
**with** *QTrans* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l\hat{}\ a[b] \prec P'$ **and** $P'RelQ'$: $(P',\ Q') \in Rel$
**by**(*blast dest*: *simE*)

**from** *RBangRelQ* **have** *weakSimAct* $R\ (a<x> \prec Q'')\ R\ Rel'$ **by**(*rule IH*)
**with** *xFreshR* **obtain** $R'\ R''$ **where** *RTrans*: $R \Longrightarrow_l b\ in\ R''\rightarrow a<x> \prec R'$

**and** $R'BangRelQ''$: $(R', Q''[x::=b]) \in Rel'$
**by**(*simp add*: *weakSimAct-def*, *blast*)

**from** *PTrans RTrans* **have** $P \parallel R \Longrightarrow_l\hat{}\ \tau \prec (P' \parallel R')$
**by**(*rule Weak-Late-Semantics.Comm2*)
**moreover from** $P'RelQ'$ $R'BangRelQ''$ **have** $(P' \parallel R', Q' \parallel Q''[x::=b]) \in Rel'$
**by**(*rule ParComp*)
**ultimately show** *?case* **by** *blast*
**qed**
**qed**
**next**
**case**(*cClose1 a x* $Q'$ *y* $Q''$ *P*)
**have** *QTrans*: $Q \longmapsto a{<}x{>} \prec Q'$ **by** *fact*
**have** *IH*: $\bigwedge P.\ (P, !Q) \in bangRel\ Rel \Longrightarrow weakSimAct\ P\ (a{<}\nu y{>} \prec Q'')\ P\ Rel'$ **by** *fact*
**have** $(P, Q \parallel !Q) \in bangRel\ Rel$ **and** $x \sharp P$ **and** $y \sharp P$ **by** *fact+*
**thus** *?case*
**proof**(*induct rule*: *BRParCases*)
**case**(*BRPar P R*)
**have** *PRelQ*: $(P, Q) \in Rel$ **and** *RBangRelQ*: $(R, !Q) \in bangRel\ Rel$ **by** *fact+*
**have** $x \sharp P \parallel R$ **by** *fact*
**hence** *xFreshP*: $x \sharp P$ **by** *simp*
**have** $y \sharp P \parallel R$ **by** *fact*
**hence** *yFreshR*: $y \sharp R$ **and** *yFreshP*: $y \sharp P$ **by** *simp+*
**show** *?case*
**proof**(*induct rule*: *simActFreeCases*)
**case** *Der*
**from** *PRelQ* **have** $P \leadsto\hat{}{<}Rel{>}\ Q$ **by**(*rule Sim*)
**with** *QTrans xFreshP* **obtain** $P'$ $P''$ **where** *PTrans*: $P \Longrightarrow_l y\ in\ P'' {\rightarrow} a{<}x{>} \prec P'$
**and** $P'RelQ'$: $(P', Q'[x::=y]) \in Rel$
**by**(*blast dest*: *simE*)

**from** *RBangRelQ* **have** *weakSimAct R* $(a{<}\nu y{>} \prec Q'')$ *R* $Rel'$ **by**(*rule IH*)
**with** *yFreshR* **obtain** $R'$ **where** *RTrans*: $R \Longrightarrow_l\hat{}\ a{<}\nu y{>} \prec R'$
**and** $R'RelQ''$: $(R', Q'') \in Rel'$
**by**(*simp add*: *weakSimAct-def*, *blast*)

**from** *PTrans RTrans yFreshP yFreshR* **have** $P \parallel R \Longrightarrow_l\hat{}\ \tau \prec {<}\nu y{>}(P' \parallel R')$
**by**(*rule Weak-Late-Semantics.Close1*)
**moreover from** $P'RelQ'$ $R'RelQ''$ **have** $({<}\nu y{>}(P' \parallel R'), {<}\nu y{>}(Q'[x::=y] \parallel Q'')) \in Rel'$
**by**(*force intro*: *ParComp Res*)
**ultimately show** *?case* **by** *blast*
**qed**
**qed**

**next**
**case**(*cClose2 a y Q′ x Q′′ P*)
**have** *QTrans*: $Q \longmapsto a{<}\nu y{>} \prec Q'$ **by** *fact*
**have** *IH*: $\bigwedge P.\ (P,\ !Q) \in$ *bangRel Rel* $\Longrightarrow$ *weakSimAct P* $(a{<}x{>} \prec Q'')$ *P Rel′* **by** *fact*
**have** $(P,\ Q \parallel !Q) \in$ *bangRel Rel* **and** $x \sharp P$ **and** $y \sharp P$ **by** *fact+*
**thus** *?case*
**proof**(*induct rule*: *BRParCases*)
**case**(*BRPar P R*)
**have** *PRelQ*: $(P,\ Q) \in$ *Rel* **and** *RBangRelQ*: $(R,\ !Q) \in$ *bangRel Rel* **by** *fact+*
**have** $x \sharp P \parallel R$ **by** *fact*
**hence** *xFreshR*: $x \sharp R$ **by** *simp*
**have** $y \sharp P \parallel R$ **by** *fact*
**hence** *yFreshP*: $y \sharp P$ **and** *yFreshR*: $y \sharp R$ **by** *simp+*
**show** *?case*
**proof**(*induct rule*: *simActFreeCases*)
**case** *Der*
**from** *PRelQ* **have** $P \rightsquigarrow\hat{}{<}Rel{>}\ Q$ **by**(*rule Sim*)
**with** *QTrans yFreshP* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow_l\hat{}\ a{<}\nu y{>} \prec P'$
**and** *P′RelQ′*: $(P',\ Q') \in$ *Rel*
**by**(*blast dest*: *simE*)

**from** *RBangRelQ* **have** *weakSimAct R* $(a{<}x{>} \prec Q'')$ *R Rel′* **by**(*rule IH*)
**with** *xFreshR* **obtain** $R'\ R''$ **where** *RTrans*: $R \Longrightarrow_l y$ *in* $R''{\rightarrow}a{<}x{>} \prec R'$
**and** *R′RelQ′′*: $(R',\ Q''[x{::=}y]) \in$ *Rel′*
**by**(*simp add*: *weakSimAct-def*, *blast*)

**from** *PTrans RTrans yFreshP yFreshR* **have** $P \parallel R \Longrightarrow_l\hat{}\ \tau \prec {<}\nu y{>}(P' \parallel R')$
**by**(*rule Weak-Late-Semantics.Close2*)
**moreover from** *P′RelQ′ R′RelQ′′* **have** $({<}\nu y{>}(P' \parallel R'),\ {<}\nu y{>}(Q' \parallel Q''[x{::=}y])) \in$ *Rel′*
**by**(*force intro*: *ParComp Res*)
**ultimately show** *?case* **by** *blast*
**qed**
**qed**
**next**
**case**(*cBang Rs*)
**have** *IH*: $\bigwedge P.\ (P,\ Q \parallel !Q) \in$ *bangRel Rel* $\Longrightarrow$ *weakSimAct P Rs P Rel′* **by** *fact*
**have** $(P,\ !Q) \in$ *bangRel Rel* **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *BRBangCases*)
**case**(*BRBang P*)
**have** *PRelQ*: $(P,\ Q) \in$ *Rel* **by** *fact*
**hence** $(!P,\ !Q) \in$ *bangRel Rel* **by**(*rule Rel.BRBang*)
**with** *PRelQ* **have** $(P \parallel !P,\ Q \parallel !Q) \in$ *bangRel Rel* **by**(*rule Rel.BRPar*)
**hence** *weakSimAct* $(P \parallel !P)$ *Rs* $(P \parallel !P)$ *Rel′* **by**(*rule IH*)

**thus** *?case*
**proof**(*simp* (*no-asm*) *add*: *weakSimAct-def*, *auto*)
**fix** $Q'$ $a$ $x$
**assume** *weakSimAct* $(P \parallel !P)$ $(a{<}\nu x{>} \prec Q')$ $(P \parallel !P)$ $Rel'$ **and** $x \sharp P$
**then obtain** $P'$ **where** *PTrans*: $(P \parallel !P) \Longrightarrow_l\hat{}\ a{<}\nu x{>} \prec P'$
**and** $P'RelQ'$: $(P', Q') \in Rel'$
**by**(*simp add*: *weakSimAct-def*, *blast*)
**from** *PTrans* **have** $!P \Longrightarrow_l\hat{}\ a{<}\nu x{>} \prec P'$
**by**(*force intro*: *Weak-Late-Step-Semantics.Bang simp add*: *weakTransition-def*)
**with** $P'RelQ'$ **show** $\exists P'.\ !P \Longrightarrow_l\hat{}\ a{<}\nu x{>} \prec P' \wedge (P', Q') \in Rel'$ **by** *blast*
**next**
**fix** $Q'$ $a$ $x$
**assume** *weakSimAct* $(P \parallel !P)$ $(a{<}x{>} \prec Q')$ $(P \parallel !P)$ $Rel'$ **and** $x \sharp P$
**then obtain** $P''$ **where** *L1*: $\forall u.\ \exists P'.\ P \parallel !P \Longrightarrow_l u\ in\ P'' \rightarrow a{<}x{>} \prec P' \wedge (P', Q'[x::=u]) \in Rel'$
**by**(*simp add*: *weakSimAct-def*, *blast*)
**have** $\forall u.\ \exists P'.\ !P \Longrightarrow_l u\ in\ P'' \rightarrow a{<}x{>} \prec P' \wedge (P', Q'[x::=u]) \in Rel'$
**proof**(*rule allI*)
**fix** $u$
**from** *L1* **obtain** $P'$ **where** *PTrans*: $P \parallel !P \Longrightarrow_l u\ in\ P'' \rightarrow a{<}x{>} \prec P'$
**and** $P'RelQ'$: $(P', Q'[x::=u]) \in Rel'$
**by** *blast*
**from** *PTrans* **have** $!P \Longrightarrow_l u\ in\ P'' \rightarrow a{<}x{>} \prec P'$ **by**(*rule Weak-Late-Step-Semantics.Bang*)
**with** $P'RelQ'$ **show** $\exists P'.\ !P \Longrightarrow_l u\ in\ P'' \rightarrow a{<}x{>} \prec P' \wedge (P', Q'[x::=u]) \in Rel'$ **by** *blast*
**qed**
**thus** $\exists P''.\ \forall u.\ \exists P'.\ !P \Longrightarrow_l u\ in\ P'' \rightarrow a{<}x{>} \prec P' \wedge (P', Q'[x::=u]) \in Rel'$ **by** *blast*
**next**
**fix** $Q'$ $\alpha$
**assume** *weakSimAct* $(P \parallel !P)$ $(\alpha \prec Q')$ $(P \parallel !P)$ $Rel'$
**then obtain** $P'$ **where** *PTrans*: $(P \parallel !P) \Longrightarrow_l\hat{}\ \alpha \prec P'$
**and** $P'RelQ'$: $(P', Q') \in Rel'$
**by**(*simp add*: *weakSimAct-def*, *blast*)
**from** *PTrans* **show** $\exists P'.\ !P \Longrightarrow_l\hat{}\ \alpha \prec P' \wedge (P', Q') \in Rel'$
**proof**(*induct rule*: *transitionCases*)
**case** *Step*
**have** $P \parallel !P \Longrightarrow_l \alpha \prec P'$ **by** *fact*
**hence** $!P \Longrightarrow_l \alpha \prec P'$ **by**(*rule Weak-Late-Step-Semantics.Bang*)
**with** $P'RelQ'$ **show** *?case* **by**(*force simp add*: *weakTransition-def*)
**next**
**case** *Stay*
**have** $\alpha \prec P' = \tau \prec P \parallel !P$ **by** *fact*
**hence** $\alpha eq\tau$: $\alpha = \tau$ **and** $P'eqP$: $P' = P \parallel !P$ **by**(*simp add*: *residual.inject*)+
**have** $!P \Longrightarrow_l\hat{}\ \tau \prec !P$ **by**(*simp add*: *weakTransition-def*)
**moreover from** $P'eqP$ $P'RelQ'$ **have** $(!P, Q') \in Rel'$ **by**(*blast intro*: *RelStay*)
**ultimately show** *?case* **using** $\alpha eq\tau$ **by** *blast*

**qed**
**qed**
**qed**
**qed**
**qed**
**moreover from** *PRelQ* **have** $(!P, !Q) \in$ *bangRel Rel* **by**(*rule Rel.BRBang*)
**ultimately show** *?thesis* **by**(*simp add*: *simDef*)
**qed**

**end**

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

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

**assumes** $P \approx Q$

**shows** $\tau.(P) \approx \tau.(Q)$
**proof** −
**let** $?X = \{(\tau.(P), \tau.(Q)) \mid P\ Q.\ P \approx Q\}$
**from** *assms* **have** $(\tau.(P), \tau.(Q)) \in ?X$ **by** *auto*
**thus** *?thesis*
**by**(*coinduct rule*: *weakBisimCoinduct*)
(*auto simp add*: *pi.inject intro*: *Weak-Late-Sim-Pres.tauPres symmetric*)
**qed**

**lemma** *inputPres*:
**fixes** $P :: pi$
**and** $Q :: pi$
**and** $a :: name$
**and** $x :: name$

**assumes** *PSimQ*: $\forall y.\ P[x::=y] \approx Q[x::=y]$

**shows** $a<x>.P \approx a<x>.Q$
**proof** −
**let** $?X = \{(a<x>.P, a<x>.Q) \mid a\ x\ P\ Q.\ \forall y.\ P[x::=y] \approx Q[x::=y]\}$
**{**
**fix** *axP axQ p*
**assume** $(axP, axQ) \in ?X$
**then obtain** $a\ x\ P\ Q$ **where** $A$: $\forall y.\ P[x::=y] \approx Q[x::=y]$ **and** $B$: $axP = a<x>.P$ **and** $C$: $axQ = a<x>.Q$
**by** *auto*
**have** $\bigwedge y.\ ((p::name\ prm) \cdot P)[(p \cdot x)::=y] \approx (p \cdot Q)[(p \cdot x)::=y]$
**proof** −

```
      fix y
      from A have P[x::=(rev p • y)] ≈ Q[x::=(rev p • y)]
        by blast
      hence (p • (P[x::=(rev p • y)])) ≈ p • (Q[x::=(rev p • y)])
        by(rule eqvtI)
      thus (p • P)[(p • x)::=y] ≈ (p • Q)[(p • x)::=y]
        by(simp add: eqvts pt-pi-rev[OF pt-name-inst, OF at-name-inst])
    qed
    hence ((p::name prm) • axP, p • axQ) ∈ ?X using B C
      by auto
  }
  hence eqvt ?X by(simp add: eqvt-def)

  from PSimQ have (a<x>.P, a<x>.Q) ∈ ?X by auto
  thus ?thesis
  proof(coinduct rule: weakBisimCoinduct)
    case(cSim P Q)
    thus ?case using ‹eqvt ?X›
      by(force intro: inputPres)
  next
    case(cSym P Q)
    thus ?case
      by(blast dest: symmetric)
  qed
qed

lemma outputPres:
  fixes P :: pi
  and   Q :: pi
  and   a :: name
  and   b :: name

  assumes P ≈ Q

  shows a{b}.(P) ≈ a{b}.(Q)
proof -
  let ?X = {(a{b}.(P), a{b}.(Q)) | a b P Q. P ≈ Q}
  from assms have (a{b}.(P), a{b}.(Q)) ∈ ?X by auto
  thus ?thesis
    by(coinduct rule: weakBisimCoinduct)
      (auto simp add: pi.inject intro:  Weak-Late-Sim-Pres.outputPres symmetric)
qed

lemma resPres:
  fixes P :: pi
  and   Q :: pi
  and   x :: name

  assumes PBiSimQ: P ≈ Q
```

**shows** $<\nu x>P \approx <\nu x>Q$
**proof** $-$
**let** $?X = \{x.\ \exists P\ Q.\ P \approx Q \wedge (\exists a.\ x = (<\nu a>P,\ <\nu a>Q))\}$
**from** *PBiSimQ* **have** $(<\nu x>P,\ <\nu x>Q) \in\ ?X$ **by** *blast*
**moreover have** $\bigwedge P\ Q\ a.\ P \leadsto\hat{}<\mathit{weakBisim}>\ Q \Longrightarrow <\nu a>P \leadsto\hat{}<(?X \cup \mathit{weak}$-$\mathit{Bisim})>\ <\nu a>Q$
**proof** $-$
**fix** $P\ Q\ a$
**assume** *PSimQ*: $P \leadsto\hat{}<\mathit{weakBisim}>\ Q$
**moreover have** $\bigwedge P\ Q\ a.\ P \approx Q \Longrightarrow (<\nu a>P,\ <\nu a>Q) \in\ ?X \cup \mathit{weakBisim}$ **by** *blast*
**moreover have** $\mathit{weakBisim} \subseteq\ ?X \cup \mathit{weakBisim}$ **by** *blast*
**moreover have** *eqvt weakBisim* **by**(*rule eqvt*)
**moreover have** *eqvt* $(?X \cup \mathit{weakBisim})$
**by**(*auto simp add*: *eqvt-def dest*: *eqvtI*)+
**ultimately show** $<\nu a>P \leadsto\hat{}<(?X \cup \mathit{weakBisim})>\ <\nu a>Q$
**by**(*rule Weak-Late-Sim-Pres.resPres*)
**qed**

**ultimately show** *?thesis* **using** *PBiSimQ*
**by**(*coinduct rule*: *weakBisimCoinductAux*, *blast dest*: *unfoldE*)
**qed**

**lemma** *matchPres*:
**fixes** $P :: \mathit{pi}$
**and** $Q :: \mathit{pi}$
**and** $a :: \mathit{name}$
**and** $b :: \mathit{name}$

**assumes** $P \approx Q$

**shows** $[a \frown b]P \approx [a \frown b]Q$
**proof** $-$
**let** $?X = \{([a \frown b]P,\ [a \frown b]Q) \mid a\ b\ P\ Q.\ P \approx Q\}$
**from** *assms* **have** $([a \frown b]P,\ [a \frown b]Q) \in\ ?X$ **by** *auto*
**thus** *?thesis*
**proof**(*coinduct rule*: *weakBisimCoinduct*)
**case**(*cSim P Q*)
**{**
**fix** $P\ Q\ a\ b$
**assume** $P \approx Q$
**hence** $P \leadsto\hat{}<\mathit{weakBisim}>\ Q$ **by**(*rule unfoldE*)
**moreover {**
**fix** $P\ Q\ a$
**assume** $P \approx Q$
**moreover have** $[a \frown a]P \approx P$ **by**(*rule matchId*)
**ultimately have** $[a \frown a]P \approx Q$ **by**(*blast intro*: *transitive*)
**}**

```
      moreover have weakBisim ⊆ ?X ∪ weakBisim by blast
      ultimately have [a⌢b]P ↝^<(?X ∪ weakBisim)> [a⌢b]Q
        by(rule matchPres)
    }
    with ‹(P, Q) ∈ ?X› show ?case by auto
  next
    case(cSym P Q)
    thus ?case by(auto simp add: pi.inject dest: symmetric)
  qed
qed

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

  assumes P ≈ Q

  shows [a≠b]P ≈ [a≠b]Q
proof −
  let ?X = {([a≠b]P, [a≠b]Q) | a b P Q. P ≈ Q}
  from assms have ([a≠b]P, [a≠b]Q) ∈ ?X by auto
  thus ?thesis
  proof(coinduct rule: weakBisimCoinduct)
    case(cSim P Q)
    {
      fix P Q a b
      assume P ≈ Q
      hence P ↝^<weakBisim> Q by(rule unfoldE)
      moreover {
        fix P Q a b
        assume P ≈ Q and (a::name) ≠ b
        note ‹P ≈ Q›
        moreover from ‹a ≠ b› have [a≠b]P ≈ P by(rule mismatchId)
        ultimately have [a≠b]P ≈ Q by(blast intro: transitive)
      }
      moreover have weakBisim ⊆ ?X ∪ weakBisim by blast
      ultimately have [a≠b]P ↝^<(?X ∪ weakBisim)> [a≠b]Q
        by(rule mismatchPres)
    }
    with ‹(P, Q) ∈ ?X› show ?case by auto
  next
    case(cSym P Q)
    thus ?case by(auto simp add: pi.inject dest: symmetric)
  qed
qed

lemma parPres:
```

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

**assumes** $P \approx Q$

**shows** $P \parallel R \approx Q \parallel R$
**proof** −
**let** *?ParSet* = $\{(resChain\ lst\ (P \parallel R),\ resChain\ lst\ (Q \parallel R)) \mid lst\ P\ Q\ R.\ P \approx Q\}$
**have** *BC*: $\bigwedge P\ Q.\ P \parallel Q = resChain\ []\ (P \parallel Q)$ **by** *auto*
**from** *assms* **have** $(P \parallel R,\ Q \parallel R) \in$ *?ParSet* **by**(*blast intro*: *BC*)
**thus** *?thesis*
**proof**(*coinduct rule*: *weakBisimCoinduct*)
**case**(*cSim PR QR*)
**{**
**fix** *P Q R lst*
**assume** $P \approx Q$

**from** *eqvtI* **have** *eqvt* (*?ParSet* ∪ *weakBisim*)
**by**(*auto simp add*: *eqvt-def*, *blast*)
**moreover have** $\bigwedge P\ Q\ a.$ $(P,\ Q) \in$ *?ParSet* ∪ *weakBisim* $\Longrightarrow$ (<$\nu a$>$P$, <$\nu a$>$Q$) ∈ *?ParSet* ∪ *weakBisim*
**by**(*blast intro*: *resChain.step*[*THEN sym*] *resPres*)
**moreover {**
**from** ‹$P \approx Q$› **have** $P \leadsto\hat{}$ <*weakBisim*> $Q$ **by**(*rule unfoldE*)
**moreover note** ‹$P \approx Q$›
**moreover {**
**fix** *P Q R*
**assume** $P \approx Q$
**moreover have** $P \parallel R = resChain\ []\ (P \parallel R)$ **by** *simp*
**moreover have** $Q \parallel R = resChain\ []\ (Q \parallel R)$ **by** *simp*
**ultimately have** $(P \parallel R,\ Q \parallel R) \in$ *?ParSet* ∪ *weakBisim* **by** *blast*
**}**
**moreover {**
**fix** *P Q a*
**assume** *A*: $(P,\ Q) \in$ *?ParSet* ∪ *weakBisim*
**hence** (<$\nu a$>$P$, <$\nu a$>$Q$) ∈ *?ParSet* ∪ *weakBisim* (**is** *?goal*)
**apply**(*auto intro*: *resPres*)
**by**(*rule-tac x=a#lst* **in** *exI*) *auto*
**}**
**ultimately have** $(P \parallel R) \leadsto\hat{}$ <(*?ParSet* ∪ *weakBisim*)> $(Q \parallel R)$ **using** *eqvt* ‹*eqvt*(*?ParSet* ∪ *weakBisim*)›
**by**(*rule Weak-Late-Sim-Pres.parPres*)
**}**

**ultimately have** *resChain lst* $(P \parallel R) \leadsto\hat{}$ <(*?ParSet* ∪ *weakBisim*)> *resChain lst* $(Q \parallel R)$
**by**(*rule resChainI*)

```
    }
    with ‹(PR, QR) ∈ ?ParSet› show ?case by blast
  next
    case(cSym PR QR)
    thus ?case by(auto dest: symmetric)
  qed
qed

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

  assumes PBisimQ: P ≈ Q

  shows !P ≈ !Q
proof -
  let ?X = (bangRel weakBisim)
 let ?Y = Strong-Late-Bisim.bisim O (bangRel weakBisim) O Strong-Late-Bisim.bisim

 from eqvt Strong-Late-Bisim.bisimEqvt have eqvtY: eqvt ?Y by(blast intro: eqvt-
BangRel)
  have XsubY: ?X ⊆ ?Y by(auto intro: Strong-Late-Bisim.reflexive)

  have RelStay: ⋀P Q. (P ∥ !P, Q) ∈ ?Y ⟹ (!P, Q) ∈ ?Y
  proof(auto)
    fix P Q R T
    assume PBisimQ: P ∥ !P ∼ Q
       and QBRR: (Q, R) ∈ bangRel weakBisim
       and RBisimT: R ∼ T
    have !P ∼ Q
    proof -
      have !P ∼ P ∥ !P by(rule Strong-Late-Bisim-SC.bangSC)
      thus ?thesis using PBisimQ by(rule Strong-Late-Bisim.transitive)
    qed
    with QBRR RBisimT show (!P, T) ∈ ?Y by blast
  qed

  have ParCompose: ⋀P Q R T. ⟦P ≈ Q; (R, T) ∈ ?Y⟧ ⟹ (P ∥ R, Q ∥ T) ∈
?Y
  proof -
    fix P Q R T
    assume PBisimQ: P ≈ Q
       and RYT:      (R, T) ∈ ?Y
    thus (P ∥ R, Q ∥ T) ∈ ?Y
    proof(auto)
      fix T' R'
      assume T'BisimT: T' ∼ T and RBisimR': R ∼ R'
         and R'BRT': (R', T') ∈ bangRel weakBisim
      have P ∥ R ∼ P ∥ R'
```

**proof** −
**from** *RBisimR'* **have** $R \parallel P \sim R' \parallel P$ **by**(*rule Strong-Late-Bisim-Pres.parPres*)
**moreover have** $P \parallel R \sim R \parallel P$ **and** $R' \parallel P \sim P \parallel R'$ **by**(*rule Strong-Late-Bisim-SC.parSym*)+
**ultimately show** *?thesis* **by**(*blast intro*: *Strong-Late-Bisim.transitive*)
**qed**
**moreover from** *PBisimQ R'BRT'* **have** $(P \parallel R', Q \parallel T') \in$ *bangRel weakBisim* **by**(*rule BRPar*)
**moreover have** $Q \parallel T' \sim Q \parallel T$
**proof** −
**from** *T'BisimT* **have** $T' \parallel Q \sim T \parallel Q$ **by**(*rule Strong-Late-Bisim-Pres.parPres*)
**moreover have** $Q \parallel T' \sim T' \parallel Q$ **and** $T \parallel Q \sim Q \parallel T$ **by**(*rule Strong-Late-Bisim-SC.parSym*)+
**ultimately show** *?thesis* **by**(*blast intro*: *Strong-Late-Bisim.transitive*)
**qed**
**ultimately show** *?thesis* **by** *blast*
**qed**
**qed**

**have** *ResCong*: $\bigwedge P\ Q\ x.\ (P, Q) \in\ ?Y \Longrightarrow (<\nu x>P, <\nu x>Q) \in\ ?Y$
**by**(*auto intro*: *BRRes Strong-Late-Bisim-Pres.resPres transitive*)

**from** *PBisimQ* **have** $(!P, !Q) \in\ ?X$ **by**(*rule BRBang*)
**moreover from** *eqvt* **have** *eqvt* (*bangRel weakBisim*) **by**(*rule eqvtBangRel*)
**ultimately show** *?thesis*
**proof**(*coinduct rule*: *weakBisimTransitiveCoinduct*)
**case**(*cSim P Q*)
**from** ‹$(P, Q) \in\ ?X$›
**show** $P \leadsto\hat{}<?Y> Q$
**proof**(*induct*)
**case**(*BRBang P Q*)
**have** $P \approx Q$ **by** *fact*
**moreover hence** $P \leadsto\hat{}<weakBisim> Q$ **by**(*blast dest*: *unfoldE*)
**moreover have** $\bigwedge P\ Q.\ P \approx Q \Longrightarrow P \leadsto\hat{}<weakBisim> Q$ **by**(*blast dest*: *unfoldE*)
**moreover from** *Strong-Late-Bisim.bisimEqvt eqvt* **have** *eqvt ?Y* **by**(*blast intro*: *eqvtBangRel*)

**ultimately show** $!P \leadsto\hat{}<?Y>\ !Q$ **using** *ParCompose ResCong RelStay XsubY*
**by**(*rule-tac Weak-Late-Sim-Pres.bangPres*, *simp-all*)
**next**
**case**(*BRPar P Q R T*)
**have** *PBiSimQ*: $P \approx Q$ **by** *fact*
**have** *RBangRelT*: $(R, T) \in\ ?X$ **by** *fact*
**have** *RSimT*: $R \leadsto\hat{}<?Y> T$ **by** *fact*
**moreover from** *PBiSimQ* **have** $P \leadsto\hat{}<weakBisim> Q$ **by**(*blast dest*: *unfoldE*)
**moreover from** *RBangRelT* **have** $(R, T) \in\ ?Y$ **by**(*blast intro*: *Strong-Late-Bisim.reflexive*)
**ultimately show** $P \parallel R \leadsto\hat{}<?Y> Q \parallel T$ **using** *ParCompose ResCong eqvt*

*eqvtY* ‹$P \approx Q$›
**by**(*rule-tac Weak-Late-Sim-Pres.parCompose*)
**next**
**case**(*BRRes P Q x*)
**have** $P \rightsquigarrow\hat{}<?Y> Q$ **by** *fact*
**thus** $<\nu x>P \rightsquigarrow\hat{}<?Y> <\nu x>Q$ **using** *ResCong eqvtY XsubY*
**by**(*rule-tac Weak-Late-Sim-Pres.resPres*, *simp-all*)
**qed**
**next**
**case**(*cSym P Q*)
**thus** *?case* **by**(*metis symmetric bangRelSymetric*)
**qed**
**qed**

**end**

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

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

**assumes** $P \simeq Q$

**shows** $\tau.(P) \simeq \tau.(Q)$
**using** *assms*
**by**(*blast intro*: *unfoldI Weak-Late-Step-Sim-Pres.tauPres dest*: *congruenceWeakBisim symetric*)

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

**assumes** $P \simeq Q$

**shows** $a\{b\}.P \simeq a\{b\}.Q$
**using** *assms*
**by**(*blast intro*: *unfoldI Weak-Late-Step-Sim-Pres.outputPres dest*: *congruenceWeakBisim symetric*)

**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** $a<x>.P \simeq a<x>.Q$
**using** *assms*
**apply**(*rule-tac unfoldI*)
**apply**(*rule-tac Weak-Late-Step-Sim-Pres.inputPres*, *auto intro*: *congruenceWeak-Bisim*)
**by**(*rule-tac Weak-Late-Step-Sim-Pres.inputPres*, *auto intro*: *congruenceWeakBisim Weak-Late-Bisim.symmetric*)

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

**assumes** $P \simeq Q$

**shows** $[a\frown b]P \simeq [a\frown b]Q$
**using** *assms*
**by**(*blast intro*: *unfoldI Weak-Late-Step-Sim-Pres.matchPres dest*: *unfoldE symetric*)

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

**assumes** $P \simeq Q$

**shows** $[a\neq b]P \simeq [a\neq b]Q$
**using** *assms*
**by**(*blast intro*: *unfoldI Weak-Late-Step-Sim-Pres.mismatchPres dest*: *unfoldE symetric*)

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

**assumes** $P \simeq Q$

**shows** $P \oplus R \simeq Q \oplus R$
**using** *assms*
**by**(*blast intro*: *Weak-Late-Bisim.reflexive unfoldI Weak-Late-Step-Sim-Pres.sumPres dest*: *unfoldE symetric*)

**lemma** *parPres*:
**fixes** $P :: pi$

**and** $Q :: pi$
**and** $R :: pi$

**assumes** $P \simeq Q$

**shows** $P \parallel R \simeq Q \parallel R$
**proof** $-$
**have** $\bigwedge P\ Q\ R.\ [\![P \leadsto\!<weakBisim>\ Q;\ P \approx Q]\!] \Longrightarrow P \parallel R \leadsto\!<weakBisim>\ Q \parallel R$
**proof** $-$
**fix** $P\ Q\ R$
**assume** $P \leadsto\!<weakBisim>\ Q$ **and** $P \approx Q$
**thus** $P \parallel R \leadsto\!<weakBisim>\ Q \parallel R$
**using** *Weak-Late-Bisim-Pres.parPres Weak-Late-Bisim-Pres.resPres Weak-Late-Bisim.reflexive Weak-Late-Bisim.eqvt*
**by**(*blast intro*: *Weak-Late-Step-Sim-Pres.parPres*)
**qed**
**with** *assms* **show** *?thesis*
**by**(*blast intro*: *unfoldI dest*: *congruenceWeakBisim unfoldE symetric*)
**qed**

**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** $-$
**have** $\bigwedge P\ Q\ x.\ P \leadsto\!<weakBisim>\ Q \Longrightarrow <\nu x>P \leadsto\!<weakBisim>\ <\nu x>Q$
**proof** $-$
**fix** $P\ Q\ x$
**assume** $P \leadsto\!<weakBisim>\ Q$
**with** *Weak-Late-Bisim.eqvt Weak-Late-Bisim-Pres.resPres* **show** $<\nu x>P \leadsto\!<weakBisim>\ <\nu x>Q$
**by**(*blast intro*: *Weak-Late-Step-Sim-Pres.resPres*)
**qed**
**with** *assms* **show** *?thesis*
**by**(*blast intro*: *unfoldI dest*: *congruenceWeakBisim unfoldE symetric*)
**qed**

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

**assumes** $P \simeq Q$

**shows** $!P \simeq !Q$

**proof** −
**have** $\bigwedge P\ Q.\ [\![P \rightsquigarrow$<*weakBisim*> $Q;\ P \simeq Q]\!] \Longrightarrow\ !P \rightsquigarrow$<*weakBisim*> $!Q$
**proof** −
**fix** $P\ Q$
**assume** $P \rightsquigarrow$<*weakBisim*> $Q$ **and** $P \simeq Q$
**hence** $!P \rightsquigarrow$<*bangRel weakBisim*> $!Q$ **using** *unfoldE*(*1*) *congruenceWeakBisim* *Weak-Late-Bisim.eqvt*
**by**(*rule Weak-Late-Step-Sim-Pres.bangPres*)
**moreover have** *bangRel weakBisim* $\subseteq$ *weakBisim*
**proof** *auto*
**fix** $a\ b$
**assume** $(a,\ b) \in$ *bangRel weakBisim*
**thus** $a \approx b$
**apply**(*induct rule*: *bangRel.induct*)
**apply** (*metis Weak-Late-Bisim-Pres.bangPres*)
**apply** (*metis Weak-Late-Bisim.reflexive Weak-Late-Bisim.symmetric* *Weak-Late-Bisim.transitive Weak-Late-Bisim-Pres.parPres Weak-Late-Bisim-SC.parSym*)
**by** (*metis Weak-Late-Bisim-Pres.resPres*)
**qed**
**ultimately show**$!P \rightsquigarrow$<*weakBisim*> $!Q$
**by**(*rule Weak-Late-Step-Sim.monotonic*)
**qed**

**with** *assms* **show** *?thesis*
**by**(*blast intro*: *unfoldI dest*: *unfoldE symetric congruenceWeakBisim*)
**qed**

**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* (‹$\tau$› *110*)

**nominal-datatype** *residual* = *BoundOutputR name* «*name*» *pi* (‹-<$\nu$-> $\prec$ -› [*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** $a{<}\nu x{>} \prec P = a{<}\nu x'{>} \prec ([(x, x')] \cdot P)$
**proof**(*cases* $x{=}x'$)
**assume** $x{=}x'$
**thus** *?thesis* **by** *simp*
**next**
**assume** $x \neq x'$
**with** *A1* **show** *?thesis*
**by**(*simp add*: *residual.inject alpha name-fresh-left name-calc*)
**qed**

**declare** *name-fresh*[*simp*]

**abbreviation** *Transitions-Freejudge* (‹- ≺ -› [*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*
**by**(*auto simp add*: *abs-fresh fresh-fact2*)

**lemmas** [*simp*] = *freeRes.inject*

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

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

  **shows** $c \neq a$ **and** $c \neq b$ **and** $c \sharp P'$
**proof** −
  **from** *assms* **have** $c \neq a \wedge c \neq b \wedge c \sharp P'$
   **by**(*nominal-induct x2*==$a[b] \prec P'$ *arbitrary*: $P'$ *rule*: *TransitionsEarly.strong-induct*)
(*fastforce simp add*: *residual.inject abs-fresh freeRes.inject*)+
  **thus** $c \neq a$ **and** $c \neq b$ **and** $c \sharp P'$
    **by** *blast*+
**qed**

**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** $c \neq a$
**using** *assms*
**by**(*nominal-induct x2*==$a{<}b{>} \prec P'$ *arbitrary*: $P'$ *rule*: *TransitionsEarly.strong-induct*)
(*auto simp add*: *residual.inject abs-fresh*)

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

**and** $c$ :: *name*

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

**shows** $c \neq a$
**using** *assms*
**by**(*nominal-induct x2*==$a{<}\nu x{>} \prec P'$ *avoiding*: $x$ *arbitrary*: $P'$ *rule*: *TransitionsEarly.strong-induct*) (*auto simp add*: *residual.inject abs-fresh fresh-left calc-atm dest*: *freshOutputAction*)

**lemmas** *freshAction* = *freshOutputAction freshInputAction freshBoundOutputAction*

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

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

**shows** $c \sharp P'$
**using** *assms*
**by**(*nominal-induct x2*==$a{<}u{>} \prec P'$ *arbitrary*: $P'$ *rule*: *TransitionsEarly.strong-induct*)
(*fastforce simp add*: *residual.inject name-fresh-abs fresh-fact1 fresh-fact2*)+

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

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

**shows** $c \sharp P'$
**using** *assms*
**apply**(*nominal-induct x2*==$a{<}\nu x{>} \prec P'$ *avoiding*: $x$ *arbitrary*: $P'$ *rule*: *TransitionsEarly.strong-induct*)
**apply**(*fastforce simp add*: *residual.inject name-fresh-abs alpha'* *fresh-left calc-atm dest*: *freshOutputAction* | *simp* | *auto simp add*: *abs-fresh residual.inject alpha'* *calc-atm*)
**apply**(*fastforce simp add*: *residual.inject name-fresh-abs alpha'* *fresh-left calc-atm*

*dest*: *freshOutputAction* | *simp* | *auto simp add*: *abs-fresh residual.inject alpha′ calc-atm*)
**apply**(*fastforce simp add*: *residual.inject name-fresh-abs alpha′ fresh-left calc-atm dest*: *freshOutputAction* | *simp* | *auto simp add*: *abs-fresh residual.inject alpha′ calc-atm*)
**apply**(*force simp add*: *residual.inject name-fresh-abs alpha′ fresh-left calc-atm dest*: *freshOutputAction* | *simp* | *auto simp add*: *abs-fresh residual.inject alpha′ calc-atm*)
**apply**(*force simp add*: *residual.inject name-fresh-abs alpha′ fresh-left calc-atm dest*: *freshOutputAction* | *simp* | *auto simp add*: *abs-fresh residual.inject alpha′ calc-atm*)
**apply**(*force simp add*: *residual.inject name-fresh-abs alpha′ fresh-left calc-atm dest*: *freshOutputAction* | *simp* | *auto simp add*: *abs-fresh residual.inject alpha′ calc-atm*)
**apply**(*force simp add*: *residual.inject name-fresh-abs alpha′ fresh-left calc-atm dest*: *freshOutputAction* | *simp* | *auto simp add*: *abs-fresh residual.inject alpha′ calc-atm*)
**apply**(*force simp add*: *residual.inject name-fresh-abs alpha′ fresh-left calc-atm dest*: *freshOutputAction* | *simp* | *auto simp add*: *abs-fresh residual.inject alpha′ calc-atm*)
**apply**(*force simp add*: *residual.inject name-fresh-abs alpha′ fresh-left calc-atm dest*: *freshOutputAction* | *simp* | *auto simp add*: *abs-fresh residual.inject alpha′ calc-atm*)
**apply**(*force simp add*: *residual.inject name-fresh-abs alpha′ fresh-left calc-atm dest*: *freshOutputAction* | *simp* | *auto simp add*: *abs-fresh residual.inject alpha′ calc-atm*)
**apply**(*force simp add*: *residual.inject name-fresh-abs alpha′ fresh-left calc-atm dest*: *freshOutputAction* | *simp* | *auto simp add*: *abs-fresh residual.inject alpha′ calc-atm*)
**apply**(*force simp add*: *residual.inject name-fresh-abs alpha′ fresh-left calc-atm dest*: *freshOutputAction* | *simp* | *auto simp add*: *abs-fresh residual.inject alpha′ calc-atm*)
**apply**(*fastforce simp add*: *residual.inject name-fresh-abs alpha′ fresh-left calc-atm dest*: *freshOutputAction*)
**apply**(*fastforce simp add*: *residual.inject name-fresh-abs alpha′ fresh-left calc-atm dest*: *freshOutputAction*)
**apply**(*fastforce simp add*: *residual.inject name-fresh-abs alpha′ fresh-left calc-atm dest*: *freshOutputAction*)
**apply**(*fastforce simp add*: *residual.inject name-fresh-abs alpha′ fresh-left calc-atm dest*: *freshOutputAction*)
**apply**(*auto simp add*: *residual.inject name-fresh-abs alpha′ fresh-left calc-atm dest*: *freshOutputAction*)
**done**

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

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

  **shows** $c \sharp P'$
**using** *assms*
**apply**(*nominal-induct* $x2 == \tau \prec P'$ *arbitrary*: $P'$ *rule*: *TransitionsEarly.strong-induct*)
**by**(*fastforce simp add*: *residual.inject abs-fresh dest*: *freshOutputAction freshInputTransition freshBoundOutputTransition*)+

**lemma** *freshFreeTransition*:
**fixes** $P$ :: *pi*
**and** $\alpha$ :: *freeRes*
**and** $P'$ :: *pi*
**and** $c$ :: *name*

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

**shows** $c \sharp P'$
**using** *assms*
**by**(*nominal-induct* $\alpha$ *rule*: *freeRes.strong-inducts*)
(*auto dest*: *freshInputTransition freshOutputAction freshTauTransition*)

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

**lemma** *substTrans*[*simp*]: $b \sharp P \Longrightarrow ((P::pi)[a::=b])[b::=c] = P[a::=c]$
**apply**(*simp add*: *injPermSubst*[*THEN sym*])
**apply**(*simp add*: *renaming*)
**by**(*simp add*: *pt-swap*[*OF pt-name-inst*, *OF at-name-inst*])

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

**shows** $a{<}x{>}.P \longmapsto a{<}u{>} \prec P[x::=u]$
**proof** −
**obtain** $y$::*name* **where** $y \neq a$ **and** $y \neq u$ **and** $y \sharp P$
**by**(*generate-fresh name*) (*auto simp add*: *fresh-prod*)
**from** ‹$y \neq a$› ‹$y \neq u$› **have** $a{<}y{>}.([(x, y)] \cdot P) \longmapsto a{<}u{>} \prec ([(x, y)] \cdot P)[y::=u]$
**by**(*rule Input*)
**with** ‹$y \sharp P$› **show** *?thesis* **by**(*simp add*: *alphaInput renaming name-swap*)
**qed**

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

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

**shows** $P \parallel Q \longmapsto a{<}\nu x{>} \prec (P' \parallel Q)$
**proof** $-$
**obtain** $y$::*name* **where** $y \sharp P$ **and** $y \sharp Q$ **and** $y \neq a$ **and** $y \sharp P'$
**by**(*generate-fresh name*) (*auto simp add*: *fresh-prod*)
**from** ‹$P \longmapsto a{<}\nu x{>} \prec P'$› ‹$y \sharp P'$› **have** $P \longmapsto a{<}\nu y{>} \prec ([(x, y)] \cdot P')$
**by**(*simp add*: *alphaBoundOutput*)
**hence** $P \parallel Q \longmapsto a{<}\nu y{>} \prec (([(x, y)] \cdot P') \parallel Q)$ **using** ‹$y \sharp P$› ‹$y \sharp Q$› ‹$y \neq a$›
**by**(*rule Par1B*)
**with** ‹$x \sharp Q$› ‹$y \sharp Q$› ‹$y \sharp P'$› **show** *?thesis*
**by**(*subst alphaBoundOutput*) (*auto simp add*: *name-fresh-fresh*)
**qed**

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

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

**shows** $P \parallel Q \longmapsto a{<}\nu x{>} \prec (P \parallel Q')$
**proof** $-$
**obtain** $y$::*name* **where** $y \sharp P$ **and** $y \sharp Q$ **and** $y \neq a$ **and** $y \sharp Q'$
**by**(*generate-fresh name*) (*auto simp add*: *fresh-prod*)
**from** ‹$Q \longmapsto a{<}\nu x{>} \prec Q'$› ‹$y \sharp Q'$› **have** $Q \longmapsto a{<}\nu y{>} \prec ([(x, y)] \cdot Q')$
**by**(*simp add*: *alphaBoundOutput*)
**hence** $P \parallel Q \longmapsto a{<}\nu y{>} \prec (P \parallel ([(x, y)] \cdot Q'))$ **using** ‹$y \sharp P$› ‹$y \sharp Q$› ‹$y \neq a$›
**by**(*rule Par2B*)
**with** ‹$x \sharp P$› ‹$y \sharp P$› ‹$y \sharp Q'$› **show** *?thesis*
**by**(*subst alphaBoundOutput*[*of y*]) (*auto simp add*: *name-fresh-fresh*)
**qed**

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

**assumes** *Trans*: $P \longmapsto a{<}u{>} \prec P'$
**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** $F\ C\ P\ a\ u\ P'$
**using** *assms*
**by**(*nominal-induct* $x2 == a{<}u{>} \prec P'$ *avoiding*: $C$ *arbitrary*: $P'$ *rule*: *TransitionsEarly.strong-induct*)
(*auto simp add*: *residual.inject*)

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

**shows** $P \longmapsto a{<}r{>} \prec ([(u, r)] \cdot P')$
**using** *assms*
**proof**(*nominal-induct avoiding*: *r rule*: *inputInduct*)
**case**(*cInput a x P u r*)
**from** ‹$x \neq u$› ‹$u \sharp a{<}x{>}.P$› **have** $u \neq a$ **and** $u \sharp P$ **by**(*simp add*: *abs-fresh*)+
**have** $a{<}x{>}.P \longmapsto a{<}r{>} \prec P[x{::=}r]$
**by**(*rule Input*)
**thus** *?case* **using** ‹$r \sharp P[x{::=}u]$› ‹$u \sharp P$›
**by**(*simp add*: *injPermSubst substTrans*)
**next**
**case**(*cMatch P a u P' b r*)
**thus** *?case* **by**(*force intro*: *Match*)
**next**
**case**(*cMismatch P a u P' b c r*)
**thus** *?case* **by**(*force intro*: *Mismatch*)
**next**
**case**(*cSum1 P a u P' Q r*)
**thus** *?case* **by**(*force intro*: *Sum1*)
**next**
**case**(*cSum2 Q a u Q' P r*)
**thus** *?case* **by**(*force intro*: *Sum2*)
**next**
**case**(*cPar1 P a u P' Q r*)
**thus** *?case* **by**(*force intro*: *Par1F simp add*: *eqvts name-fresh-fresh*)
**next**

**case**(*cPar2 Q a u Q′ P r*)
**thus** *?case* **by**(*force intro*: *Par2F simp add*: *eqvts name-fresh-fresh*)
**next**
**case**(*cRes P a u P′ x r*)
**thus** *?case* **by**(*force intro*: *ResF simp add*: *eqvts calc-atm abs-fresh*)
**next**
**case**(*cBang P a u P′ R*)
**thus** *?case* **by**(*force intro*: *Bang*)
**qed**

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

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

**shows** $P \parallel Q \longmapsto \tau \prec {<}\nu x{>}(P' \parallel Q')$
**proof** −
**obtain** *y*::*name* **where** $y \sharp P$ **and** $y \sharp Q$ **and** $y \neq a$ **and** $y \sharp Q'$ **and** $y \sharp P'$
**by**(*generate-fresh name*) (*auto simp add*: *fresh-prod*)
**from** ‹$P \longmapsto a{<}x{>} \prec P'$› ‹$x \sharp P$› ‹$y \sharp P'$› **have** $P \longmapsto a{<}y{>} \prec ([(x, y)] \cdot P')$
**by**(*rule inputAlpha*)
**moreover from** ‹$Q \longmapsto a{<}\nu x{>} \prec Q'$› ‹$y \sharp Q'$› **have** $Q \longmapsto a{<}\nu y{>} \prec ([(x, y)] \cdot Q')$
**by**(*simp add*: *alphaBoundOutput*)

**ultimately have** $P \parallel Q \longmapsto \tau \prec {<}\nu y{>}(([(x, y)] \cdot P') \parallel ([(x, y)] \cdot Q'))$ **using** ‹$y \sharp P$› ‹$y \sharp Q$› ‹$y \neq a$›
**by**(*rule Close1*)
**with** ‹$y \sharp P'$› ‹$y \sharp Q'$› **show** *?thesis* **by**(*subst alphaRes*) (*auto simp add*: *name-fresh-fresh*)
**qed**

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

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

**shows** $P \parallel Q \longmapsto \tau \prec <\nu x>(P' \parallel Q')$
**proof** $-$
**obtain** $y$::*name* **where** $y \sharp P$ **and** $y \sharp Q$ **and** $y \neq a$ **and** $y \sharp Q'$ **and** $y \sharp P'$
**by**(*generate-fresh name*) (*auto simp add*: *fresh-prod*)
**from** ‹$P \longmapsto a<\nu x> \prec P'$› ‹$y \sharp P'$› **have** $P \longmapsto a<\nu y> \prec ([(x, y)] \cdot P')$
**by**(*simp add*: *alphaBoundOutput*)
**moreover from** ‹$Q \longmapsto a<x> \prec Q'$› ‹$x \sharp Q$› ‹$y \sharp Q'$› **have** $Q \longmapsto a<y> \prec ([(x, y)] \cdot Q')$
**by**(*rule inputAlpha*)

**ultimately have** $P \parallel Q \longmapsto \tau \prec <\nu y>(([(x, y)] \cdot P') \parallel ([(x, y)] \cdot Q'))$ **using** ‹$y \sharp P$› ‹$y \sharp Q$› ‹$y \neq a$›
**by**(*rule Close2*)
**with** ‹$y \sharp P'$› ‹$y \sharp Q'$› **show** *?thesis* **by**(*subst alphaRes*) (*auto simp add*: *name-fresh-fresh*)
**qed**

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

**assumes** $P \longmapsto a<\nu x> \prec P'$
**and** $y \neq a$
**and** $y \neq x$

**shows** $<\nu y>P \longmapsto a<\nu x> \prec (<\nu y>P')$
**proof** $-$
**obtain** $z$ :: *name* **where** $z \sharp P$ **and** $z \sharp P'$ **and** $z \neq a$ **and** $z \neq y$
**by**(*generate-fresh name*) (*auto simp add*: *fresh-prod*)
**from** ‹$P \longmapsto a<\nu x> \prec P'$› ‹$z \sharp P'$› **have** $P \longmapsto a<\nu z> \prec ([(x, z)] \cdot P')$
**by**(*simp add*: *alphaBoundOutput*)
**hence** $<\nu y>P \longmapsto a<\nu z> \prec (<\nu y>([(x, z)] \cdot P'))$ **using** ‹$y \neq a$› ‹$z \neq y$› ‹$z \sharp P$› ‹$z \neq a$›
**by**(*rule-tac ResB*) *auto*
**thus** *?thesis* **using** ‹$z \neq y$› ‹$y \neq x$› ‹$z \sharp P'$›
**by**(*subst alphaBoundOutput*[**where** $x'=z$]) (*auto simp add*: *eqvts calc-atm abs-fresh*)
**qed**

**lemma** *outputInduct*[*consumes 1*, *case-names Output Match Mismatch Sum1 Sum2 Par1 Par2 Res Bang*]:
**fixes** $P$ :: *pi*
**and** $a$ :: *name*
**and** $b$ :: *name*
**and** $P'$ :: *pi*
**and** $F$ :: *'a::fs-name* $\Rightarrow$ *pi* $\Rightarrow$ *name* $\Rightarrow$ *name* $\Rightarrow$ *pi* $\Rightarrow$ *bool*
**and** $C$ :: *'a::fs-name*

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

**shows** $F\ C\ P\ a\ b\ P'$
**using** *assms*
**by**(*nominal-induct* $x2{=}{=}a[b] \prec P'$ *avoiding*: $C$ *arbitrary*: $P'$ *rule*: *TransitionsEarly.strong-induct*)
(*auto simp add*: *residual.inject*)

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

**assumes** *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*: $\bigwedge P\ a\ x\ P'\ C.\ [\![P \longmapsto (\mathit{OutputR}\ a\ x) \prec P';\ a \neq x]\!] \Longrightarrow F\ C\ (<\nu x>P)\ a\ x\ P'$
**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** $F\ C\ P\ a\ x\ P'$
**using** *assms*
**proof** −
**have** *Goal*: $\bigwedge P\ Rs\ a\ x\ P'\ C.\ [\![P \longmapsto Rs;\ Rs = a{<}\nu x{>} \prec P';\ x \sharp P]\!] \Longrightarrow F\ C\ P\ a\ x\ P'$
**proof** −
**fix** $P\ Rs\ a\ x\ P'\ C$
**assume** $P \longmapsto Rs$ **and** $Rs = a{<}\nu x{>} \prec P'$ **and** $x \sharp P$
**thus** $F\ C\ P\ a\ x\ P'$
**proof**(*nominal-induct avoiding*: $C\ a\ x\ P'$ *rule*: *TransitionsEarly.strong-induct*)
**case**(*Tau P*)
**thus** *?case* **by**(*simp add*: *residual.inject*)
**next**
**case**(*Input P a x*)
**thus** *?case* **by**(*simp add*: *residual.inject*)
**next**
**case**(*Output P a b*)
**thus** *?case* **by**(*simp add*: *residual.inject*)
**next**
**case**(*Match P Rs b C a x P'*)
**thus** *?case*
**by**(*force intro*: *cMatch simp add*: *residual.inject*)
**next**
**case**(*Mismatch P Rs b c C a x P'*)
**thus** *?case*
**by**(*force intro*: *cMismatch simp add*: *residual.inject*)
**next**
**case**(*Sum1 P Q Rs C*)
**thus** *?case* **by**(*force intro*: *cSum1*)
**next**
**case**(*Sum2 P Q Rs C*)
**thus** *?case* **by**(*force intro*: *cSum2*)
**next**
**case**(*Open P a b P' C a' x P''*)
**have** $b \sharp x$ **by** *fact* **hence** *bineqx*: $b \neq x$ **by** *simp*
**moreover have** $a{<}\nu b{>} \prec P' = a'{<}\nu x{>} \prec P''$ **by** *fact*
**ultimately have** *aeqa'*: $a{=}a'$ **and** *P'eqP''*: $P'' = [(b, x)] \cdot P'$
**by**(*simp add*: *residual.inject name-abs-eq*)+
**have** $x \sharp {<}\nu b{>}P$ **by** *fact*
**with** *bineqx* **have** *xFreshP*: $x \sharp P$ **by**(*simp add*: *name-fresh-abs*)

**have** *aineqb*: $a \neq b$ **by** *fact*

**have** *PTrans*: $P \longmapsto a[b] \prec P'$ **by** *fact*
**with** *xFreshP* **have** *xineqa*: $x \neq a$ **by**(*force dest*: *freshAction*)
**from** *PTrans* **have** $([(b, x)] \cdot P) \longmapsto [(b, x)] \cdot (a[b] \prec P')$ **by**(*rule Transition-sEarly.eqvt*)
**with** *P′eqP′′ xineqa aineqb* **have** *Trans*: $([(b, x)] \cdot P) \longmapsto a[x] \prec P''$
**by**(*auto simp add*: *name-calc*)
**hence** $F\ C\ (<\nu x>([(b, x)] \cdot P))\ a\ x\ P''$ **using** *xineqa* **by**(*blast intro*: *cOpen*)
**with** *xFreshP aeqa′* **show** *?case* **by**(*simp add*: *alphaRes*)
**next**
**case**(*Par1B P a x P′ Q C a′ x′ P′′*)
**have** $x \sharp x'$ **by** *fact* **hence** *xineqx′*: $x \neq x'$ **by** *simp*
**moreover have** *Eq*: $a<\nu x> \prec (P' \parallel Q) = a'<\nu x'> \prec P''$ **by** *fact*
**hence** *aeqa′*: $a = a'$ **by**(*simp add*: *residual.inject*)
**have** *xFreshQ*: $x \sharp Q$ **by** *fact*
**have** $x' \sharp P \parallel Q$ **by** *fact*
**hence** *x′FreshP*: $x' \sharp P$ **and** *x′FreshQ*: $x' \sharp Q$ **by** *simp+*
**have** *P′′eq*: $P'' = ([(x, x')] \cdot P') \parallel Q$
**proof** −
**from** *Eq xineqx′* **have** $(P' \parallel Q) = [(x, x')] \cdot P''$
**by**(*simp add*: *residual.inject name-abs-eq*)
**hence** $([(x, x')] \cdot (P' \parallel Q)) = P''$ **by** *simp*
**with** *x′FreshQ xFreshQ* **show** *?thesis* **by**(*simp add*: *name-fresh-fresh*)
**qed**

**have** $x \sharp P''$ **by** *fact*
**with** *P′′eq* **have** *x′FreshP′*: $x' \sharp P'$ **by**(*simp add*: *name-fresh-left name-calc*)

**have** $P \longmapsto a<\nu x> \prec P'$ **by** *fact*
**with** *x′FreshP′ aeqa′* **have** $P \longmapsto a'<\nu x'> \prec ([(x, x')] \cdot P')$
**by**(*simp add*: *alphaBoundOutput*)
**moreover have** $\bigwedge C.\ F\ C\ P\ a\ x'\ ([(x, x')] \cdot P')$
**proof** −
**fix** $C$
**have** $\bigwedge C\ a'\ x'\ P''.\ [\![a<\nu x> \prec P' = a'<\nu x'> \prec P''; x' \sharp P]\!] \Longrightarrow F\ C\ P\ a'\ x'\ P''$ **by** *fact*
**moreover with** *aeqa′ xineqx′ x′FreshP′* **have** $a<\nu x> \prec P' = a'<\nu x'> \prec ([(x, x')] \cdot P')$
**by**(*simp add*: *residual.inject name-abs-eq name-fresh-left name-calc*)
**ultimately show** $F\ C\ P\ a\ x'\ ([(x, x')] \cdot P')$ **using** *x′FreshP aeqa′* **by** *blast*
**qed**
**ultimately have** $F\ C\ (P \parallel Q)\ a'\ x'\ (([(x, x')] \cdot P') \parallel Q)$ **using** *x′FreshQ aeqa′*
**by**(*blast intro*: *cPar1B*)
**with** *P′′eq* **show** *?case* **by** *simp*
**next**
**case**(*Par1F P P′ Q α*)
**thus** *?case* **by**(*simp add*: *residual.inject*)

**next**
  **case**(*Par2B* $Q\ a\ x\ Q'\ P\ C\ a'\ x'\ Q''$)
  **have** $x \sharp x'$ **by** *fact* **hence** *xineqx'*: $x \neq x'$ **by** *simp*
  **moreover have** *Eq*: $a{<}\nu x{>} \prec (P \parallel Q') = a'{<}\nu x'{>} \prec Q''$ **by** *fact*
  **hence** *aeqa'*: $a = a'$ **by**(*simp add*: *residual.inject*)
  **have** *xFreshP*: $x \sharp P$ **by** *fact*
  **have** $x' \sharp P \parallel Q$ **by** *fact*
  **hence** *x'FreshP*: $x' \sharp P$ **and** *x'FreshQ*: $x' \sharp Q$ **by** *simp+*
  **have** *Q''eq*: $Q'' = P \parallel ([(x, x')] \cdot Q')$
  **proof** −
    **from** *Eq xineqx'* **have** $(P \parallel Q') = [(x, x')] \cdot Q''$
      **by**(*simp add*: *residual.inject name-abs-eq*)
    **hence** $([(x, x')] \cdot (P \parallel Q')) = Q''$ **by** *simp*
    **with** *x'FreshP xFreshP* **show** *?thesis* **by**(*simp add*: *name-fresh-fresh*)
  **qed**

  **have** $x \sharp Q''$ **by** *fact*
  **with** *Q''eq* **have** *x'FreshQ'*: $x' \sharp Q'$ **by**(*simp add*: *name-fresh-left name-calc*)

  **have** $Q \longmapsto a{<}\nu x{>} \prec Q'$ **by** *fact*
  **with** *x'FreshQ' aeqa'* **have** $Q \longmapsto a'{<}\nu x'{>} \prec ([(x, x')] \cdot Q')$
    **by**(*simp add*: *alphaBoundOutput*)
  **moreover have** $\bigwedge C.\ F\ C\ Q\ a\ x'\ ([(x, x')] \cdot Q')$
  **proof** −
    **fix** $C$
    **have** $\bigwedge C\ a'\ x'\ Q''.\ [\![a{<}\nu x{>} \prec Q' = a'{<}\nu x'{>} \prec Q''; x' \sharp Q]\!] \Longrightarrow F\ C\ Q\ a'\ x'\ Q''$ **by** *fact*
    **moreover with** *aeqa' xineqx' x'FreshQ'* **have** $a{<}\nu x{>} \prec Q' = a'{<}\nu x'{>} \prec ([(x, x')] \cdot Q')$
      **by**(*simp add*: *residual.inject name-abs-eq name-fresh-left name-calc*)
    **ultimately show** $F\ C\ Q\ a\ x'\ ([(x, x')] \cdot Q')$ **using** *x'FreshQ aeqa'* **by** *blast*
  **qed**
  **ultimately have** $F\ C\ (P \parallel Q)\ a'\ x'\ (P \parallel ([(x, x')] \cdot Q'))$ **using** *x'FreshP aeqa'*
    **by**(*blast intro*: *cPar2B*)
  **with** *Q''eq* **show** *?case* **by** *simp*
**next**
  **case**(*Par2F* $P\ P'\ Q\ \alpha$)
  **thus** *?case* **by**(*simp add*: *residual.inject*)
**next**
  **case**(*Comm1* $P\ P'\ Q\ Q'\ a\ b\ x$)
  **thus** *?case* **by**(*simp add*: *residual.inject*)
**next**
  **case**(*Comm2* $P\ P'\ Q\ Q'\ a\ b\ x$)
  **thus** *?case* **by**(*simp add*: *residual.inject*)
**next**
  **case**(*Close1* $P\ P'\ Q\ Q'\ a\ x\ y$)
  **thus** *?case* **by**(*simp add*: *residual.inject*)
**next**

**case**(*Close2 P P′ Q Q′ a x y*)
**thus** *?case* **by**(*simp add*: *residual.inject*)
**next**
**case**(*ResB P a x P′ y C a′ x′ P″*)
**have** $x \sharp x'$ **by** *fact* **hence** *xineqx′*: $x \neq x'$ **by** *simp*
**moreover have** *Eq*: $a{<}\nu x{>} \prec ({<}\nu y{>}P') = a'{<}\nu x'{>} \prec P''$ **by** *fact*
**hence** *aeqa′*: $a = a'$ **by**(*simp add*: *residual.inject*)
**have** $y \sharp x'$ **by** *fact* **hence** *yineqx′*: $y \neq x'$ **by** *simp*
**moreover have** $x' \sharp {<}\nu y{>}P$ **by** *fact*
**ultimately have** *x′FreshP*: $x' \sharp P$ **by**(*simp add*: *name-fresh-abs*)
**have** *yineqx*: $y \neq x$ **and** *yineqa*: $y \neq a$ **and** *yFreshC*: $y \sharp C$ **by** *fact+*

**have** *P″eq*: $P'' = {<}\nu y{>}([(x, x')] \cdot P')$
**proof** −
**from** *Eq xineqx′* **have** ${<}\nu y{>}P' = [(x, x')] \cdot P''$
**by**(*simp add*: *residual.inject name-abs-eq*)
**hence** $([(x, x')] \cdot ({<}\nu y{>}P')) = P''$ **by** *simp*
**with** *yineqx′ yineqx* **show** *?thesis* **by**(*simp add*: *name-fresh-fresh*)
**qed**

**have** $x \sharp P''$ **by** *fact*
**with** *P″eq yineqx* **have** *x′FreshP′*: $x' \sharp P'$ **by**(*simp add*: *name-fresh-left name-calc name-fresh-abs*)

**have** $P \longmapsto a{<}\nu x{>} \prec P'$ **by** *fact*
**with** *x′FreshP′ aeqa′* **have** $P \longmapsto a'{<}\nu x'{>} \prec ([(x, x')] \cdot P')$
**by**(*simp add*: *alphaBoundOutput*)
**moreover have** $\bigwedge C.\ F\ C\ P\ a\ x'\ ([(x, x')] \cdot P')$
**proof** −
**fix** $C$
**have** $\bigwedge C\ a'\ x'\ P''.\ [\![a{<}\nu x{>} \prec P' = a'{<}\nu x'{>} \prec P''; x' \sharp P]\!] \Longrightarrow F\ C\ P\ a'\ x'\ P''$ **by** *fact*
**moreover with** *aeqa′ xineqx′ x′FreshP′* **have** $a{<}\nu x{>} \prec P' = a'{<}\nu x'{>} \prec ([(x, x')] \cdot P')$
**by**(*simp add*: *residual.inject name-abs-eq name-fresh-left name-calc*)
**ultimately show** $F\ C\ P\ a\ x'\ ([(x, x')] \cdot P')$ **using** *x′FreshP aeqa′* **by** *blast*
**qed**
**ultimately have** $F\ C\ ({<}\nu y{>}P)\ a'\ x'\ ({<}\nu y{>}([(x, x')] \cdot P'))$ **using** *yineqx′ yineqa yFreshC aeqa′*
**by**(*force intro*: *cResB*)
**with** *P″eq* **show** *?case* **by** *simp*
**next**
**case**(*ResF P P′ α y*)
**thus** *?case* **by**(*simp add*: *residual.inject*)
**next**
**case**(*Bang P Rs*)
**thus** *?case* **by**(*force intro*: *cBang*)
**qed**
**qed**

**with** *a xFreshP* **show** *?thesis* **by** *simp*
**qed**

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

**assumes** *Trans*: $P \longmapsto \tau \prec P'$
**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** $\bigwedge P\ a\ b\ P'\ Q\ Q'\ C.\ [\![P \longmapsto a{<}b{>} \prec P';\ Q \longmapsto OutputR\ a\ b \prec Q']\!] \Longrightarrow F\ C\ (P \parallel Q)\ (P' \parallel Q')$
**and** $\bigwedge P\ a\ b\ P'\ Q\ Q'\ C.\ [\![P \longmapsto OutputR\ a\ b \prec P';\ Q \longmapsto a{<}b{>} \prec Q']\!] \Longrightarrow F\ C\ (P \parallel Q)\ (P' \parallel Q')$
**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** $F\ C\ P\ P'$
**using** ‹$P \longmapsto \tau \prec P'$›
**by**(*nominal-induct x2==*$\tau \prec P'$ *avoiding*: $C$ *arbitrary*: $P'$ *rule*: *TransitionsEarly.strong-induct*)
(*auto simp add*: *residual.inject intro*: *assms*)+

**inductive** *bangPred* :: *pi* ⇒ *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** $\alpha$ :: *freeRes*
  **and** $P'$ :: *pi*

  **assumes** $\tau.(P) \longmapsto \alpha \prec P'$
  **and** *Prop* $(\tau)$ $P$

  **shows** *Prop* $\alpha$ $P'$
**using** *assms*
**by**(*cases rule*: *tauCases'*) (*auto simp add*: *pi.inject residual.inject*)

**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* $\alpha$ $P'$
**proof** −
  **{**
    **fix** $x$ $P$
    **assume** $a{<}x{>}.P \longmapsto \alpha \prec P'$
    **moreover assume** $(x{::}name) \sharp \alpha$ **and** $x \sharp P'$ **and** $x \neq a$
    **moreover assume** $\bigwedge u.$ *Prop* $(a{<}u{>})$ $(P[x{::=}u])$
    **moreover obtain** $z{::}name$ **where** $z \neq x$ **and** $z \sharp P$ **and** $z \sharp \alpha$ **and** $z \sharp P'$ **and** $z \neq a$
      **by**(*generate-fresh name*, *auto simp add*: *fresh-prod*)
    **moreover obtain** $z'{::}name$ **where** $z' \neq x$ **and** $z' \neq z$ **and** $z' \sharp P$ **and** $z' \sharp \alpha$ **and** $z' \sharp P'$ **and** $z' \neq a$
      **by**(*generate-fresh name*, *auto simp add*: *fresh-prod*)
    **ultimately have** *Prop* $\alpha$ $P'$
      **by**(*cases rule*: *TransitionsEarly.strong-cases*[**where** $x{=}x$ **and** $b{=}z$ **and** $xa{=}z$ **and** $xb{=}z$ **and** $xc{=}z$ **and** $xd{=}z$ **and** $xe{=}z$
                                   **and** $y{=}z'$ **and** $ya{=}z'$])
        (*auto simp add*: *pi.inject residual.inject abs-fresh alpha*)
  **}**

**note** *Goal* = *this*
**obtain** *y*::*name* **where** $y \sharp P$ **and** $y \sharp \alpha$ **and** $y \sharp P'$ **and** $y \neq a$
**by**(*generate-fresh name*) (*auto simp add*: *fresh-prod*)
**from** *Input* ‹$y \sharp P$› **have** $a{<}y{>}.([(x, y)] \cdot P) \longmapsto \alpha \prec P'$ **by**(*simp add*: *alphaInput*)
**moreover note** ‹$y \sharp \alpha$› ‹$y \sharp P'$› ‹$y \neq a$›
**moreover from** *A* ‹$y \sharp P$› **have** $\bigwedge u.$ *Prop* $(a{<}u{>})$ $(([(x, y)] \cdot P)[y{::=}u])$
**by**(*simp add*: *renaming name-swap*)
**ultimately show** *?thesis* **by**(*rule Goal*)
**qed**

**lemma** *outputCases*:
**fixes** $P$ :: *pi*
**and** $\alpha$ :: *freeRes*
**and** $P'$ :: *pi*

**assumes** $a\{b\}.P \longmapsto \alpha \prec P'$
**and** *Prop* (*OutputR* $a$ $b$) $P$

**shows** *Prop* $\alpha$ $P'$
**using** *assms*
**by**(*cases rule*: *outputCases'*) (*auto simp add*: *pi.inject residual.inject*)

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

**assumes** $\mathbf{0} \longmapsto e$ *Rs*

**shows** *False*
**using** *assms*
**by** − (*ind-cases* $\mathbf{0} \longmapsto e$ *Rs*)

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

**assumes** $[a{\neq}a]P \longmapsto$ *Rs*

**shows** *False*
**using** *assms*
**by**(*erule-tac mismatchCases'*) *auto*

**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** $F\ a\ b$
**using** *assms*
**by**(*erule-tac matchCases'*, *auto*)

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

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

**shows** $F\ a\ b$
**using** *assms*
**by**(*erule-tac mismatchCases'*) *auto*

**lemma** *sumCases*[*consumes 1*, *case-names Sum1 Sum2*]:
**fixes** $P$ :: *pi*
**and** $Q$ :: *pi*
**and** $Rs$ :: *residual*

**assumes** *Trans*: $P \oplus Q \longmapsto Rs$
**and** *cSum1*: $P \longmapsto Rs \Longrightarrow F$
**and** *cSum2*: $Q \longmapsto Rs \Longrightarrow F$

**shows** $F$
**using** *assms*
**by**(*erule-tac sumCases'*) *auto*

**lemma** *parCasesB*[*consumes 1*, *case-names cPar1 cPar2*]:
**fixes** $P$ :: *pi*
**and** $Q$ :: *pi*
**and** $a$ :: *name*
**and** $x$ :: *name*
**and** $PQ'$ :: *pi*

**assumes** *Trans*: $P \parallel Q \longmapsto a{<}\nu x{>} \prec PQ'$
**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** –
**from** *Trans* **show** *?thesis*

**proof**(*induct rule*: *parcasesB'*, *auto simp add*: *pi.inject residual.inject*)
**fix** $P'$ $y$
**assume** *PTrans*: $P \longmapsto a{<}\nu y{>} \prec P'$
**assume** *yFreshQ*: $y \sharp (Q{::}pi)$
**assume** *absEq*: $[x].PQ' = [y].(P' \parallel Q)$

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

**from** *cFreshP' PTrans* **have** *Trans*: $P \longmapsto a{<}\nu c{>} \prec ([(y, c)] \cdot P')$ **by**(*simp add*: *alphaBoundOutput*)

**from** *cFreshP' cFreshQ* **have** $c \sharp P' \parallel Q$ **by** *simp*
**hence** $[y].(P' \parallel Q) = [c].([(y, c)] \cdot (P' \parallel Q))$
**by**(*auto simp add*: *alpha fresh-left calc-atm*)
**with** *yFreshQ cFreshQ* **have** $[y].(P' \parallel Q) = [c].(([(y, c)] \cdot P') \parallel Q)$
**by**(*simp add*: *name-fresh-fresh*)

**with** *cineqx absEq* **have** *L1*: $PQ' = [(x, c)] \cdot (([(y, c)] \cdot P') \parallel Q)$ **and** *L2*: $x \sharp ([(y, c)] \cdot P') \parallel Q$
**by**(*simp add*: *name-abs-eq*)+

**from** *L2* **have** *xFreshQ*: $x \sharp Q$ **and** *xFreshP'*: $x \sharp [(y, c)] \cdot P'$ **by** *simp*+
**with** *cFreshQ L1* **have** *L3*: $PQ' = ([(x, c)] \cdot [(y, c)] \cdot P') \parallel Q$ **by**(*simp add*: *name-fresh-fresh*)

**from** *Trans xFreshP'* **have** $P \longmapsto a{<}\nu x{>} \prec ([(x, c)] \cdot [(y, c)] \cdot P')$ **by**(*simp add*: *alphaBoundOutput name-swap*)

**thus** *?thesis* **using** *xFreshQ L3*
**by**(*blast intro*: *icPar1B*)
**next**
**fix** $Q'$ $y$
**assume** *QTrans*: $Q \longmapsto a{<}\nu y{>} \prec Q'$
**assume** *yFreshP*: $y \sharp (P{::}pi)$
**assume** *absEq*: $[x].PQ' = [y].(P \parallel Q')$

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

**from** *cFreshQ' QTrans* **have** *Trans*: $Q \longmapsto a{<}\nu c{>} \prec ([(y, c)] \cdot Q')$ **by**(*simp add*: *alphaBoundOutput*)

**from** *cFreshQ' cFreshP* **have** $c \sharp P \parallel Q'$ **by** *simp*
**hence** $[y].(P \parallel Q') = [c].([(y, c)] \cdot (P \parallel Q'))$

**by**(*auto simp add*: *alpha fresh-left calc-atm*)
**with** *yFreshP cFreshP* **have** $[y].(P \parallel Q') = [c].(P \parallel ([(y, c)] \cdot Q'))$
**by**(*simp add*: *name-fresh-fresh*)

**with** *cineqx absEq* **have** *L1*: $PQ' = [(x, c)] \cdot (P \parallel ([(y, c)] \cdot Q'))$ **and** *L2*: $x \sharp P \parallel ([(y, c)] \cdot Q')$
**by**(*simp add*: *name-abs-eq*)+

**from** *L2* **have** *xFreshP*: $x \sharp P$ **and** *xFreshQ'*: $x \sharp [(y, c)] \cdot Q'$ **by** *simp*+
**with** *cFreshP L1* **have** *L3*: $PQ' = P \parallel ([(x, c)] \cdot [(y, c)] \cdot Q')$ **by**(*simp add*: *name-fresh-fresh*)

**from** *Trans xFreshQ'* **have** $Q \longmapsto a{<}\nu x{>} \prec ([(x, c)] \cdot [(y, c)] \cdot Q')$ **by**(*simp add*: *alphaBoundOutput name-swap*)

**thus** *?thesis* **using** *xFreshP L3*
**by**(*blast intro*: *icPar2B*)
**qed**
**qed**

**lemma** *parCasesOutput*[*consumes 1*, *case-names Par1 Par2*]:
**fixes** $P$ :: *pi*
**and** $Q$ :: *pi*
**and** $a$ :: *name*
**and** $b$ :: *name*
**and** $P'$ :: *pi*

**assumes** $P \parallel Q \longmapsto a[b] \prec PQ'$
**and** $\bigwedge P'. [\![P \longmapsto a[b] \prec P']\!] \Longrightarrow F\ (P' \parallel Q)$
**and** $\bigwedge Q'. [\![Q \longmapsto a[b] \prec Q']\!] \Longrightarrow F\ (P \parallel Q')$

**shows** $F\ PQ'$
**using** *assms*
**by**(*erule-tac parCasesF'*, *auto simp add*: *pi.inject residual.inject*)

**lemma** *parCasesInput*[*consumes 1*, *case-names Par1 Par2*]:
**fixes** $P$ :: *pi*
**and** $Q$ :: *pi*
**and** $a$ :: *name*
**and** $b$ :: *name*
**and** $P'$ :: *pi*

**assumes** *Trans*: $P \parallel Q \longmapsto a{<}b{>} \prec PQ'$
**and** *icPar1F*: $\bigwedge P'. [\![P \longmapsto a{<}b{>} \prec P']\!] \Longrightarrow F\ (P' \parallel Q)$
**and** *icPar2F*: $\bigwedge Q'. [\![Q \longmapsto a{<}b{>} \prec Q']\!] \Longrightarrow F\ (P \parallel Q')$

**shows** $F\ PQ'$
**using** *assms*
**by**(*erule-tac parCasesF'*) (*auto simp add*: *pi.inject residual.inject*)

**lemma** *parCasesF*[*consumes 1*, *case-names cPar1 cPar2 cComm1 cComm2 cClose1 cClose2*]:
**fixes** $P$ :: *pi*
**and** $Q$ :: *pi*
**and** $\alpha$ :: *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** −
**from** *Trans* **show** *?thesis*
**proof**(*rule parCasesF'*, *auto*)
**fix** *Pa Pa' Qa* $\alpha'$
**assume** *Trans'*: $Pa \longmapsto \alpha' \prec Pa'$
**assume** *Eq*: $P \parallel Q = Pa \parallel Qa$
**assume** *Eq'*: $\alpha \prec PQ' = \alpha' \prec Pa' \parallel Qa$

**from** *Eq* **have** $P = Pa$ **and** $Q = Qa$
**by**(*simp add*: *pi.inject*)+

**moreover with** *Eq'* **have** $\alpha = \alpha'$ **and** $PQ' = Pa' \parallel Q$
**by**(*simp add*: *residual.inject*)+

**ultimately show** *?thesis* **using** *icPar1F Trans'*
**by** *simp*
**next**
**fix** *Pa Qa Qa'* $\alpha'$
**assume** *Trans'*: $Qa \longmapsto \alpha' \prec Qa'$
**assume** *Eq*: $P \parallel Q = Pa \parallel Qa$
**assume** *Eq'*: $\alpha \prec PQ' = \alpha' \prec Pa \parallel Qa'$

**from** *Eq* **have** $P = Pa$ **and** $Q = Qa$
**by**(*simp add*: *pi.inject*)+

**moreover with** *Eq'* **have** $\alpha = \alpha'$ **and** $PQ' = P \parallel Qa'$
**by**(*simp add*: *residual.inject*)+

**ultimately show** *?thesis* **using** *icPar2F Trans′*
**by** *simp*
**next**
**fix** *Pa Pa′ Qa Qa′ a b*
**assume** *TransP*: $Pa \longmapsto a{<}b{>} \prec Pa'$
**assume** *TransQ*: $Qa \longmapsto a[b] \prec Qa'$
**assume** *Eq*: $P \parallel Q = Pa \parallel Qa$
**assume** *Eq′*: $\alpha \prec PQ' = \tau \prec Pa' \parallel Qa'$

**from** *TransP TransQ Eq Eq′ icComm1* **show** *?thesis*
**by**(*simp add*: *pi.inject residual.inject*)
**next**
**fix** *Pa Pa′ Qa Qa′ a b x*
**assume** *TransP*: $Pa \longmapsto (a::name)[b] \prec Pa'$
**assume** *TransQ*: $Qa \longmapsto a{<}b{>} \prec Qa'$
**assume** *Eq*: $P \parallel Q = Pa \parallel Qa$
**assume** *Eq′*: $\alpha \prec PQ' = \tau \prec Pa' \parallel Qa'$

**from** *TransP TransQ Eq Eq′ icComm2* **show** *?thesis*
**by**(*simp add*: *pi.inject residual.inject*)
**next**
**fix** *Pa Pa′ Qa Qa′ a x*
**assume** *TransP*: $Pa \longmapsto a{<}x{>} \prec Pa'$
**assume** *TransQ*: $Qa \longmapsto a{<}\nu x{>} \prec Qa'$
**assume** *xFreshPa*: $x \sharp Pa$
**assume** *Eq*: $P \parallel Q = Pa \parallel Qa$
**assume** *Eq′*: $\alpha \prec PQ' = \tau \prec {<}\nu x{>}(Pa' \parallel Qa')$

**have** $\exists (c::name).\ c \sharp (Pa, Pa', x, Qa', a, C)$
**by**(*blast intro*: *name-exists-fresh*)
**then obtain** *c*::*name* **where** *cFreshPa*: $c \sharp Pa$ **and** *cFreshPa′*: $c \sharp Pa'$ **and** *cineqy*: $c \neq x$ **and** *cFreshQa′*: $c \sharp Qa'$ **and** *cFreshC*: $c \sharp C$ **and** *cineqa*: $c \neq a$
**by**(*force simp add*: *fresh-prod name-fresh*)

**from** *cFreshQa′* **have** *L1*: $a{<}\nu x{>} \prec Qa' = a{<}\nu c{>} \prec ([(x, c)] \cdot Qa')$
**by**(*simp add*: *alphaBoundOutput*)
**with** *cFreshQa′ cFreshPa′* **have** $c \sharp (Pa' \parallel Qa')$
**by** *simp*
**then have** *L4*: ${<}\nu x{>}(Pa' \parallel Qa') = {<}\nu c{>}(([(x, c)] \cdot Pa') \parallel ([(x, c)] \cdot Qa'))$
**by**(*simp add*: *alphaRes*)

**have** *TransP*: $Pa \longmapsto a{<}c{>} \prec [(x, c)] \cdot Pa'$
**proof** −
**from** *xFreshPa TransP* **have** *xineqa*: $x \neq a$ **by**(*force dest*: *freshAction*)
**from** *TransP* **have** $([(x, c)] \cdot Pa) \longmapsto [(x, c)] \cdot (a{<}x{>} \prec Pa')$
**by**(*rule TransitionsEarly.eqvt*)
**with** *xineqa xFreshPa cFreshPa cineqa* **show** *?thesis*
**by**(*simp add*: *name-fresh-fresh name-calc*)

**qed**

**with** *TransQ L1 L4 icClose1 Eq Eq′ cFreshPa cFreshC* **show** *?thesis*
**by**(*simp add*: *residual.inject*, *simp add*: *pi.inject*)
**next**
**fix** $Pa\ Pa'\ Qa\ Qa'\ a\ x$
**assume** *TransP*: $Pa \longmapsto a{<}\nu x{>} \prec Pa'$
**assume** *TransQ*: $Qa \longmapsto a{<}x{>} \prec Qa'$
**assume** *xFreshQa*: $x \sharp Qa$
**assume** *Eq*: $P \parallel Q = Pa \parallel Qa$
**assume** *Eq′*: $\alpha \prec PQ' = \tau \prec {<}\nu x{>}(Pa' \parallel Qa')$

**have** $\exists (c{::}name).\ c \sharp (Qa, Pa', x, Qa', a, C)$
**by**(*blast intro*: *name-exists-fresh*)
**then obtain** $c{::}name$ **where** *cFreshQa*: $c \sharp Qa$ **and** *cFreshPa′*: $c \sharp Pa'$ **and** *cineqy*: $c \neq x$ **and** *cFreshQa′*: $c \sharp Qa'$ **and** *cFreshC*: $c \sharp C$ **and** *cineqa*: $c \neq a$
**by**(*force simp add*: *fresh-prod name-fresh*)

**from** *cFreshPa′* **have** *L1*: $a{<}\nu x{>} \prec Pa' = a{<}\nu c{>} \prec ([(x, c)] \cdot Pa')$
**by**(*simp add*: *alphaBoundOutput*)
**with** *cFreshQa′ cFreshPa′* **have** $c \sharp (Pa' \parallel Qa')$
**by** *simp*
**then have** *L4*: ${<}\nu x{>}(Pa' \parallel Qa') = {<}\nu c{>}(([(x, c)] \cdot Pa') \parallel ([(x, c)] \cdot Qa'))$
**by**(*simp add*: *alphaRes*)

**have** *TransQ*: $Qa \longmapsto a{<}c{>} \prec [(x, c)] \cdot Qa'$
**proof** −
**from** *xFreshQa TransQ* **have** *xineqa*: $x \neq a$ **by**(*force dest*: *freshAction*)
**from** *TransQ* **have** $([(x, c)] \cdot Qa) \longmapsto [(x, c)] \cdot (a{<}x{>} \prec Qa')$
**by**(*rule TransitionsEarly.eqvt*)
**with** *xineqa xFreshQa cFreshQa cineqa* **show** *?thesis*
**by**(*simp add*: *name-fresh-fresh name-calc*)
**qed**

**with** *TransP L1 L4 icClose2 Eq Eq′ cFreshQa cFreshC* **show** *?thesis*
**by**(*simp add*: *residual.inject*, *simp add*: *pi.inject*)
**qed**
**qed**

**lemma** *resCasesF*[*consumes 2*, *case-names Res*]:
**fixes** $x :: name$
**and** $P :: pi$
**and** $\alpha :: freeRes$
**and** $P' :: pi$

**assumes** *Trans*: ${<}\nu x{>}P \longmapsto \alpha \prec RP'$
**and** *xFreshAlpha*: $x \sharp \alpha$
**and** *rcResF*: $\bigwedge P'.\ P \longmapsto \alpha \prec P' \Longrightarrow F\ ({<}\nu x{>}P')$

**shows** *F RP′*
**proof** −
**from** *Trans* **show** *?thesis*
**proof**(*induct rule*: *resCasesF′*, *auto*)
**fix** *Pa Pa′ β y*
**assume** *PTrans*: $Pa \longmapsto \beta \prec Pa'$
**assume** *yFreshBeta*: $(y::name) \sharp \beta$
**assume** *TermEq*: $<\nu x>P = <\nu y>Pa$
**assume** *ResEq*: $\alpha \prec RP' = \beta \prec <\nu y>Pa'$

**hence** *alphaeqbeta*: $\alpha = \beta$ **and** *L2*: $RP' = <\nu y>Pa'$ **by**(*simp add*: *residual.inject*)+

**have** $\exists (c::name).\ c \sharp (Pa, \alpha, Pa', x, y)$ **by**(*blast intro*: *name-exists-fresh*)
**then obtain** *c*::*name* **where** *cFreshPa*: $c \sharp Pa$ **and** *cFreshAlpha*: $c \sharp \alpha$ **and** *cFreshPa′*: $c \sharp Pa'$ **and** *cineqx*: $x \neq c$ **and** *cineqy*: $c \neq y$
**by**(*force simp add*: *fresh-prod name-fresh*)

**from** *cFreshPa* **have** $<\nu y>Pa = <\nu c>([(y, c)] \cdot Pa)$ **by**(*rule alphaRes*)
**with** *TermEq cineqx* **have** *Peq*: $P = [(x, c)] \cdot [(y, c)] \cdot Pa$ **and** *xeq*: $x \sharp [(y, c)] \cdot Pa$
**by**(*simp add*: *pi.inject name-abs-eq*)+

**from** *PTrans* **have** $([(y, c)] \cdot Pa) \longmapsto [(y, c)] \cdot (\beta \prec Pa')$ **by**(*rule TransitionsEarly.eqvt*)
**with** *yFreshBeta cFreshAlpha alphaeqbeta* **have** *PTrans′*: $([(y, c)] \cdot Pa) \longmapsto \alpha \prec ([(y, c)] \cdot Pa')$
**by**(*simp add*: *name-fresh-fresh*)

**from** *PTrans′* **have** $([(x, c)] \cdot [(y, c)] \cdot Pa) \longmapsto [(x, c)] \cdot (\alpha \prec [(y, c)] \cdot Pa')$
**by**(*rule TransitionsEarly.eqvt*)
**with** *xFreshAlpha cFreshAlpha Peq* **have** *PTrans′′*: $P \longmapsto \alpha \prec [(x, c)] \cdot [(y, c)] \cdot Pa'$
**by**(*simp add*: *name-fresh-fresh*)

**from** *PTrans′ xeq xFreshAlpha* **have** *xeq′*: $x \sharp [(y, c)] \cdot Pa'$
**by**(*nominal-induct* α *rule*: *freeRes.strong-induct*)
(*auto simp add*: *fresh-left calc-atm eqvts dest*: *freshTransition*)

**from** *cFreshPa′* **have** $<\nu y>Pa' = <\nu c>([(y, c)] \cdot Pa')$ **by**(*rule alphaRes*)
**moreover from** *xeq′* **have** $<\nu c>([(y, c)] \cdot Pa') = <\nu x>([(c, x)] \cdot [(y, c)] \cdot Pa')$
**by**(*rule alphaRes*)
**ultimately have** $RP' = <\nu x>([(x, c)] \cdot [(y, c)] \cdot Pa')$ **using** *ResEq*
**by**(*simp add*: *residual.inject name-swap*)

**with** *PTrans′′ xFreshAlpha* **show** *?thesis*
**by**(*blast intro*: *rcResF*)
**qed**

**qed**

**lemma** *resCasesB*[*consumes 2*, *case-names Open Res*]:
 **fixes** $x$ :: *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*: $\bigwedge P'. \llbracket P \longmapsto (OutputR\ a\ y) \prec P';\ a \neq y\rrbracket \Longrightarrow F\ ([(x,\ y)] \cdot P')$
 **and** *rcResB*: $\bigwedge P'. \llbracket P \longmapsto a<\nu x> \prec P';\ y \neq a\rrbracket \Longrightarrow F\ (<\nu y>P')$

 **shows** $F\ RP'$
**proof** −
 **from** *Trans* **show** *?thesis*
 **proof**(*induct rule*: *resCasesB'*, *auto*)
  **fix** $Pa\ Pa'\ aa\ b$
  **assume** *PTrans*: $Pa \longmapsto (aa{::}name)[b] \prec Pa'$
  **assume** *aaineqb*: $aa{\neq}b$
  **assume** *TermEq*: $<\nu y>P = <\nu b>Pa$
  **assume** *ResEq*: $a<\nu x> \prec RP' = aa<\nu b> \prec Pa'$

  **have** $\exists(c{::}name).\ c \sharp (x,\ a,\ aa,\ y,\ Pa,\ Pa',\ b)$ **by**(*blast intro*: *name-exists-fresh*)
  **then obtain** $c$ **where** *cineqx*: $c{\neq}x$ **and** *cFresha*: $c \sharp a$ **and** *cineqy*: $c \neq y$ **and** *cineqaa*: $c \neq aa$ **and** *cFreshPa*: $c \sharp Pa$ **and** *cFreshPa'*: $c \sharp Pa'$ **and** *cineqb*: $c \neq b$
   **by**(*force simp add*: *fresh-prod name-fresh*)

  **from** *cFreshPa* **have** $<\nu b>Pa = <\nu c>([(b,\ c)] \cdot Pa)$ **by**(*rule alphaRes*)
  **with** *cineqy TermEq* **have** *PEq*: $P = [(y,\ c)] \cdot [(b,\ c)] \cdot Pa$ **and** *yFreshPa*: $y \sharp [(b,\ c)] \cdot Pa$
   **by**(*simp add*: *pi.inject name-abs-eq*)+

  **from** *PTrans* **have** $([(b,\ c)] \cdot Pa) \longmapsto ([(b,\ c)] \cdot (aa[b] \prec Pa'))$ **by**(*rule TransitionsEarly.eqvt*)
  **with** *aaineqb cineqaa* **have** *L1*: $([(b,\ c)] \cdot Pa) \longmapsto aa[c] \prec [(b,\ c)] \cdot Pa'$ **by**(*simp add*: *name-calc*)
  **with** *yFreshPa* **have** *yineqaa*: $y \neq aa$ **by**(*force dest*: *freshAction*)
  **from** *L1 yFreshPa cineqy* **have** *yFreshPa'*: $y \sharp [(b,\ c)] \cdot Pa'$ **by**(*force intro*: *freshTransition*)

  **from** *L1* **have** $([(y,\ c)] \cdot [(b,\ c)] \cdot Pa) \longmapsto [(y,\ c)] \cdot (aa[c] \prec [(b,\ c)] \cdot Pa')$
   **by**(*rule TransitionsEarly.eqvt*)
  **with** *cineqaa yineqaa cineqy PEq* **have** *PTrans*: $P \longmapsto aa[y] \prec [(y,\ c)] \cdot [(b,\ c)] \cdot Pa'$
   **by**(*simp add*: *name-calc*)
  **moreover from** *cFreshPa'* **have** $aa<\nu b> \prec Pa' = aa<\nu c> \prec ([(b,\ c)] \cdot Pa')$ **by**(*rule alphaBoundOutput*)

**with** *ResEq cineqx* **have** *ResEq'*: $RP' = [(x, c)] \cdot [(b, c)] \cdot Pa'$ **and** $x \sharp [(b, c)] \cdot Pa'$
**by**(*simp add*: *residual.inject name-abs-eq*)+
**with** *xineqy cineqy cineqx yFreshPa'* **have** $RP' = [(x, y)] \cdot [(y, c)] \cdot [(b, c)] \cdot Pa'$
**by**(*subst pt-perm-compose*[*OF pt-name-inst*, *OF at-name-inst*], *simp add*: *name-calc name-fresh-fresh*)
**moreover from** *ResEq* **have** $a=aa$ **by**(*simp add*: *residual.inject*)
**ultimately show** *?thesis* **using** *yineqaa rcOpen*
**by** *blast*
**next**
**fix** *Pa Pa' aa xa ya*
**assume** *PTrans*: $Pa \longmapsto aa{<}\nu xa{>} \prec Pa'$
**assume** *yaFreshaa*: $(ya::name) \neq aa$
**assume** *yaineqxa*: $ya \neq xa$
**assume** *EqTrans*: $<\nu y>P = <\nu ya>Pa$
**assume** *EqRes*: $a{<}\nu x{>} \prec RP' = aa{<}\nu xa{>} \prec (<\nu ya>Pa')$

**hence** *aeqaa*: $a = aa$ **by**(*simp add*: *residual.inject*)
**with** *yaFreshaa* **have** *yaFresha*: $ya \sharp a$ **by** *simp*

**have** $\exists (c::name).\ c \sharp (Pa', y, xa, ya, x, Pa, aa)$ **by**(*blast intro*: *name-exists-fresh*)
**then obtain** $c$ **where** *cFreshPa'*: $c \sharp Pa'$ **and** *cineqy*: $c \neq y$ **and** *cineqxa*: $c \neq xa$ **and** *cineqya*: $c \neq ya$ **and** *cineqx*: $c \neq x$ **and** *cFreshP*: $c \sharp Pa$ **and** *cFreshaa*: $c \sharp aa$
**by**(*force simp add*: *fresh-prod name-fresh*)

**have** $\exists (d::name).\ d \sharp (Pa, a, x, Pa', c, xa, ya, y)$ **by**(*blast intro*: *name-exists-fresh*)
**then obtain** $d$ **where** *dFreshPa*: $d \sharp Pa$ **and** *dFresha*: $d \sharp a$ **and** *dineqx*: $d \neq x$ **and** *dFreshPa'*: $d \sharp Pa'$ **and** *dineqc*: $d \neq c$ **and** *dineqxa*: $d \neq xa$ **and** *dineqya*: $d \neq ya$ **and** *dineqy*: $d \neq y$
**by**(*force simp add*: *fresh-prod name-fresh*)

**from** *dFreshPa* **have** $<\nu ya>Pa = <\nu d>([(ya, d)] \cdot Pa)$ **by**(*rule alphaRes*)
**with** *EqTrans dineqy* **have** *PEq*: $P = [(y, d)] \cdot [(ya, d)] \cdot Pa$
**and** *yFreshPa*: $y \sharp [(ya, d)] \cdot Pa$
**by**(*simp add*: *pi.inject name-abs-eq*)+

**from** *dFreshPa'* **have** *L1*: $<\nu ya>Pa' = <\nu d>([(ya, d)] \cdot Pa')$ **by**(*rule alphaRes*)
**from** *cFreshPa' dineqc cineqya* **have** $c \sharp <\nu d>([(ya, d)] \cdot Pa')$
**by**(*simp add*: *name-fresh-abs name-calc name-fresh-left*)
**hence** $aa{<}\nu xa{>} \prec (<\nu d>([(ya, d)] \cdot Pa')) = aa{<}\nu c{>} \prec ([(xa, c)] \cdot <\nu d>([(ya, d)] \cdot Pa'))$ (**is** *?LHS* = -)
**by**(*rule alphaBoundOutput*)
**with** *dineqxa dineqc* **have** $?LHS = aa{<}\nu c{>} \prec (<\nu d>([(xa, c)] \cdot [(ya, d)] \cdot Pa'))$
**by**(*simp add*: *name-calc*)
**with** *L1 EqRes cineqx dineqc dineqx* **have**
*RP'Eq*: $RP' = <\nu d>([(x, c)] \cdot [(xa, c)] \cdot [(ya, d)] \cdot Pa')$

**and** *xFreshPa'*: $x \sharp [(xa,\ c)] \cdot [(ya,\ d)] \cdot Pa'$
**by**(*simp add*: *residual.inject name-abs-eq name-fresh-abs name-calc*)+

**from** *PTrans aeqaa* **have** $([(ya,\ d)] \cdot Pa) \longmapsto [(ya,\ d)] \cdot (a{<}\nu xa{>} \prec Pa')$
**by**(*blast intro*: *TransitionsEarly.eqvt*)
**with** *yaineqxa yaFresha dineqxa dFresha* **have** *L1*:
$([(ya,\ d)] \cdot Pa) \longmapsto a{<}\nu xa{>} \prec ([(ya,\ d)] \cdot Pa')$ **by**(*simp add*: *name-calc name-fresh-fresh*)
**with** *yFreshPa* **have** *yineqa*: $y \neq a$ **by**(*force dest*: *freshAction*)
**from** *dineqc cineqya cFreshPa'* **have** $c \sharp [(ya,\ d)] \cdot Pa'$
**by**(*simp add*: *name-fresh-left name-calc*)
**hence** $a{<}\nu xa{>} \prec ([(ya,\ d)] \cdot Pa') = a{<}\nu c{>} \prec ([(xa,\ c)] \cdot [(ya,\ d)] \cdot Pa')$ (**is** *?LHS = -*)
**by**(*rule alphaBoundOutput*)
**with** *xFreshPa'* **have** *L2*: $?LHS = a{<}\nu x{>} \prec ([(c,\ x)] \cdot [(xa,\ c)] \cdot [(ya,\ d)] \cdot Pa')$
**by**(*simp add*: *alphaBoundOutput*)
**with** *L1 PEq* **have** $P \longmapsto [(y,\ d)] \cdot (a{<}\nu x{>} \prec ([(c,\ x)] \cdot [(xa,\ c)] \cdot [(ya,\ d)] \cdot Pa'))$
**by**(*force intro*: *TransitionsEarly.eqvt simp del*: *residual.perm*)
**with** *yineqa dFresha xineqy dineqx* **have** *Trans*: $P \longmapsto a{<}\nu x{>} \prec ([(y,\ d)] \cdot [(c,\ x)] \cdot [(xa,\ c)] \cdot [(ya,\ d)] \cdot Pa')$
**by**(*simp add*: *name-calc name-fresh-fresh*)

**from** *L1 L2 yFreshPa xineqy* **have** $y \sharp [(c,\ x)] \cdot [(xa,\ c)] \cdot [(ya,\ d)] \cdot Pa'$
**by**(*force intro*: *freshTransition*)
**with** *RP'Eq* **have** $RP' = {<}\nu y{>}([(y,\ d)] \cdot [(c,\ x)] \cdot [(xa,\ c)] \cdot [(ya,\ d)] \cdot Pa')$
**by**(*simp add*: *alphaRes name-swap*)

**with** *Trans yineqa* **show** *?thesis*
**by**(*blast intro*: *rcResB*)
**qed**
**qed**

**lemma** *bangInduct*[*consumes 1*, *case-names Par1B Par1F Par2B Par2F Comm1 Comm2 Close1 Close2 Bang*]:
**fixes** $F$ :: *'a::fs-name* $\Rightarrow$ *pi* $\Rightarrow$ *residual* $\Rightarrow$ *bool*
**and** $P$ :: *pi*
**and** *Rs* :: *residual*
**and** $C$ :: *'a::fs-name*

**assumes** *Trans*: $!P \longmapsto Rs$
**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** −
**have** $\bigwedge X\ Y\ C.\ [\![X \longmapsto Y;\ bangPred\ P\ X]\!] \Longrightarrow F\ C\ X\ Y$
**proof** −
**fix** $X\ Y\ C$
**assume** $X \longmapsto Y$ **and** $bangPred\ P\ X$
**thus** $F\ C\ X\ Y$
**proof**(*nominal-induct avoiding*: $C$ *rule*: *TransitionsEarly.strong-induct*)
**case**(*Tau Pa*)
**thus** *?case*
**apply** −
**by**(*ind-cases bangPred* $P$ $(\tau.(Pa))$)
**next**
**case**(*Input x a u Pa C*)
**thus** *?case*
**by** − (*ind-cases bangPred* $P$ $(a{<}x{>}.Pa)$)
**next**
**case**(*Output a b Pa C*)
**thus** *?case*
**by** − (*ind-cases bangPred* $P$ $(a\{b\}.Pa)$)
**next**
**case**(*Match Pa Rs b C*)
**thus** *?case*
**by** − (*ind-cases bangPred* $P$ $([b \frown b]Pa)$)
**next**
**case**(*Mismatch Pa Rs a b C*)
**thus** *?case*
**by** − (*ind-cases bangPred* $P$ $([a \neq b]Pa)$)
**next**
**case**(*Open Pa a b Pa'*)
**thus** *?case*
**by** − (*ind-cases bangPred* $P$ $({<}\nu b{>}Pa)$)
**next**

**case**(*Sum1 Pa Rs Q*)
**thus** *?case*
**by** − (*ind-cases bangPred P* ($Pa \oplus Q$))
**next**
**case**(*Sum2 Q Rs Pa*)
**thus** *?case*
**by** − (*ind-cases bangPred P* ($Pa \oplus Q$))
**next**
**case**(*Par1B Pa a x* $P'$ *Q C*)
**thus** *?case*
**by** − (*ind-cases bangPred P* ($Pa \parallel Q$), *auto simp add*: *pi.inject cPar1B*)
**next**
**case**(*Par1F Pa* $\alpha$ $P'$ *Q C*)
**thus** *?case*
**by** − (*ind-cases bangPred P* ($Pa \parallel Q$), *auto simp add*: *pi.inject cPar1F*)
**next**
**case**(*Par2B Q a x* $Q'$ *Pa*)
**thus** *?case*
**by** − (*ind-cases bangPred P* ($Pa \parallel Q$), *auto simp add*: *pi.inject aux1 cPar2B*)
**next**
**case**(*Par2F Q* $\alpha$ $Q'$ *Pa*)
**thus** *?case*
**by** − (*ind-cases bangPred P* ($Pa \parallel Q$), *auto simp add*: *pi.inject intro*: *cPar2F aux1*)
**next**
**case**(*Comm1 Pa a b* $Pa'$ *Q* $Q'$ *C*)
**thus** *?case*
**by** − (*ind-cases bangPred P* ($Pa \parallel Q$), *auto simp add*: *pi.inject intro*: *cComm1 aux1*)
**next**
**case**(*Comm2 Pa a b* $Pa'$ *Q* $P''$ *C*)
**thus** *?case*
**by** − (*ind-cases bangPred P* ($Pa \parallel Q$), *auto simp add*: *pi.inject intro*: *cComm2 aux1*)
**next**
**case**(*Close1 Pa a x* $Pa'$ *Q* $Q''$ *C*)
**thus** *?case*
**by** − (*ind-cases bangPred P* ($Pa \parallel Q$), *auto simp add*: *pi.inject aux1 cClose1*)
**next**
**case**(*Close2 Pa a x* $Pa'$ *Q* $Q'$ *C*)
**thus** *?case*
**by** − (*ind-cases bangPred P* ($Pa \parallel Q$), *auto simp add*: *pi.inject aux1 cClose2*)
**next**
**case**(*ResB Pa a x* $Pa'$ *y*)
**thus** *?case*
**by** − (*ind-cases bangPred P* ($<\nu y>Pa$))
**next**
**case**(*ResF Pa* $\alpha$ $Pa'$ *y*)
**thus** *?case*

```
      by − (ind-cases bangPred P (<νy>Pa))
    next
      case(Bang Pa Rs)
      thus ?case
        by − (ind-cases bangPred P (!Pa), auto simp add: pi.inject intro: aux2
cBang)
    qed
  qed
  with Trans show ?thesis by(force intro: bangPred.aux1)
qed

end

theory Strong-Early-Sim
  imports Early-Semantics Rel
begin

definition strongSimEarly :: pi ⇒ (pi × pi) set ⇒ pi ⇒ bool (‹- ⇝[-] -› [80, 80,
80] 80) where
  P ⇝[Rel] Q ≡ (∀ a y Q'. Q ⟼a<νy> ≺ Q' ⟶ y ♯ P ⟶ (∃ P'. P ⟼a<νy>
≺ P' ∧ (P', Q') ∈ Rel)) ∧
                (∀ α Q'. Q ⟼α ≺ Q' ⟶ (∃ P'. P ⟼α ≺ P' ∧ (P', Q') ∈ Rel))

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

  assumes P ⇝[A] P'
  and     A ⊆ B

  shows P ⇝[B] P'
using assms
by(fastforce simp add: strongSimEarly-def)

lemma freshUnit[simp]:
  fixes y :: name

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

lemma simCasesCont[consumes 1, case-names Bound Free]:
  fixes P   :: pi
  and   Q   :: pi
  and   Rel :: (pi × pi) set
  and   C   :: 'a::fs-name

  assumes Eqvt:  eqvt Rel
```

**and** *Bound*: $\bigwedge a\ y\ Q'$. $[\![Q \longmapsto a{<}\nu y{>} \prec Q'; y \sharp P; y \sharp Q; y \sharp C]\!] \Longrightarrow \exists P'$. $P \longmapsto a{<}\nu y{>} \prec P' \wedge (P', Q') \in Rel$
**and** *Free*: $\bigwedge \alpha\ Q'$. $Q \longmapsto \alpha \prec Q' \Longrightarrow \exists P'. P \longmapsto \alpha \prec P' \wedge (P', Q') \in Rel$

**shows** $P \rightsquigarrow[Rel]\ Q$
**proof** −
**from** *Free* **show** *?thesis*
**proof**(*auto simp add*: *strongSimEarly-def*)
**fix** $Q'\ a\ y$
**assume** *yFreshP*: $(y::name) \sharp P$
**assume** *Trans*: $Q \longmapsto a{<}\nu y{>} \prec Q'$
**have** $\exists c::name.\ c \sharp (P, Q', y, Q, C)$ **by**(*blast intro*: *name-exists-fresh*)
**then obtain** $c::name$ **where** *cFreshP*: $c \sharp P$ **and** *cFreshQ'*: $c \sharp Q'$ **and** *cFreshC*: $c \sharp C$
**and** *cineqy*: $c \neq y$ **and** $c \sharp Q$
**by**(*force simp add*: *fresh-prod name-fresh*)

**from** *Trans cFreshQ'* **have** $Q \longmapsto a{<}\nu c{>} \prec ([(y, c)] \cdot Q')$ **by**(*simp add*: *alphaBoundOutput*)
**hence** $\exists P'. P \longmapsto a{<}\nu c{>} \prec P' \wedge (P', [(y, c)] \cdot Q') \in Rel$ **using** $\langle c \sharp P\rangle$ $\langle c \sharp Q\rangle$ $\langle c \sharp C\rangle$
**by**(*rule Bound*)
**then obtain** $P'$ **where** *PTrans*: $P \longmapsto a{<}\nu c{>} \prec P'$ **and** *P'RelQ'*: $(P', [(y, c)] \cdot Q') \in Rel$
**by** *blast*

**from** *PTrans yFreshP cineqy* **have** *yFreshP'*: $y \sharp P'$ **by**(*force intro*: *freshTransition*)
**with** *PTrans* **have** $P \longmapsto a{<}\nu y{>} \prec ([(y, c)] \cdot P')$ **by**(*simp add*: *alphaBoundOutput name-swap*)
**moreover have** $([(y, c)] \cdot P', Q') \in Rel$ (**is** *?goal*)
**proof** −
**from** *Eqvt P'RelQ'* **have** $([(y, c)] \cdot P', [(y, c)] \cdot [(y, c)] \cdot Q') \in Rel$
**by**(*rule eqvtRelI*)
**with** *cineqy* **show** *?goal* **by**(*simp add*: *name-calc*)
**qed**
**ultimately show** $\exists P'. P \longmapsto a{<}\nu y{>} \prec P' \wedge (P', Q') \in Rel$ **by** *blast*
**qed**
**qed**

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

**assumes** *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** $P \rightsquigarrow[Rel]\ Q$
**using** *assms*
**by**(*auto simp add*: *strongSimEarly-def*)

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

**assumes** $P \rightsquigarrow[Rel]\ Q$

**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** $Q \longmapsto \alpha \prec Q' \Longrightarrow \exists P'.\ P \longmapsto \alpha \prec P' \wedge (P', Q') \in Rel$
**using** *assms* **by**(*simp add*: *strongSimEarly-def*)+

**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**(*induct rule*: *simCases*)
**case**(*Bound a y Q'*)
**have** *Trans*: $(perm \cdot Q) \longmapsto a{<}\nu y{>} \prec Q'$ **by** *fact*
**have** *yFreshP*: $y \sharp perm \cdot P$ **by** *fact*

**from** *Trans* **have** $(rev\ perm \cdot (perm \cdot Q)) \longmapsto rev\ perm \cdot (a{<}\nu y{>} \prec Q')$
**by**(*rule TransitionsEarly.eqvt*)
**hence** $Q \longmapsto (rev\ perm \cdot a){<}\nu(rev\ perm \cdot y){>} \prec (rev\ perm \cdot Q')$
**by**(*simp add*: *name-rev-per*)
**moreover from** *yFreshP* **have** $(rev\ perm \cdot y) \sharp P$ **by**(*simp add*: *name-fresh-left*)
**ultimately have** $\exists P'.\ P \longmapsto (rev\ perm \cdot a){<}\nu(rev\ perm \cdot y){>} \prec P' \wedge (P', rev$
$perm \cdot Q') \in Rel$ **using** *Sim*
**by**(*force intro*: *elim*)
**then obtain** $P'$ **where** *PTrans*: $P \longmapsto (rev\ perm \cdot a){<}\nu(rev\ perm \cdot y){>} \prec P'$
**and** *P'RelQ'*: $(P', rev\ perm \cdot Q') \in Rel$
**by** *blast*

**from** *PTrans* **have** $(perm \cdot P) \longmapsto perm \cdot ((rev\ perm \cdot a){<}\nu(rev\ perm \cdot y){>} \prec$

$P'$) **by**(*rule TransitionsEarly.eqvt*)
**hence** *L1*: ($perm \cdot P$) $\longmapsto a{<}\nu y{>} \prec$ ($perm \cdot P'$) **by**(*simp add*: *name-per-rev*)
**from** *P′RelQ′ RelRel′* **have** ($P'$, $rev\ perm \cdot Q'$) $\in Rel'$ **by** *blast*
**with** *EqvtRel′* **have** ($perm \cdot P'$, $perm \cdot (rev\ perm \cdot Q')$) $\in Rel'$
**by**(*rule eqvtRelI*)
**hence** ($perm \cdot P'$, $Q'$) $\in Rel'$ **by**(*simp add*: *name-per-rev*)
**with** *L1* **show** *?case* **by** *blast*
**next**
**case**(*Free* $\alpha$ $Q'$)
**have** *Trans*: ($perm \cdot Q$) $\longmapsto \alpha \prec Q'$ **by** *fact*

**from** *Trans* **have** ($rev\ perm \cdot (perm \cdot Q)$) $\longmapsto rev\ perm \cdot (\alpha \prec Q')$
**by**(*rule TransitionsEarly.eqvt*)
**hence** $Q \longmapsto (rev\ perm \cdot \alpha) \prec (rev\ perm \cdot Q')$
**by**(*simp add*: *name-rev-per*)
**with** *Sim* **have** $\exists P'.\ P \longmapsto (rev\ perm \cdot \alpha) \prec P' \wedge (P', (rev\ perm \cdot Q')) \in Rel$
**by**(*force intro*: *elim*)
**then obtain** $P'$ **where** *PTrans*: $P \longmapsto (rev\ perm \cdot \alpha) \prec P'$ **and** *PRel*: ($P'$, ($rev\ perm \cdot Q'$)) $\in Rel$ **by** *blast*

**from** *PTrans* **have** ($perm \cdot P$) $\longmapsto perm \cdot ((rev\ perm \cdot \alpha) \prec P')$ **by**(*rule TransitionsEarly.eqvt*)
**hence** *L1*: ($perm \cdot P$) $\longmapsto \alpha \prec (perm \cdot P')$ **by**(*simp add*: *name-per-rev*)
**from** *PRel EqvtRel′ RelRel′* **have** (($perm \cdot P'$), ($perm \cdot (rev\ perm \cdot Q')$)) $\in Rel'$
**by**(*force intro*: *eqvtRelI*)
**hence** (($perm \cdot P'$), $Q'$) $\in Rel'$ **by**(*simp add*: *name-per-rev*)
**with** *L1* **show** *?case* **by** *blast*
**qed**

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

**assumes** $Id \subseteq Rel$

**shows** $P \leadsto[Rel]\ P$
**using** *assms*
**by**(*auto simp add*: *strongSimEarly-def*)

**lemmas** *fresh-prod*[*simp*]

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

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

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

**shows** $P \leadsto[Rel'']\ R$
**proof** −
**from** *Eqvt'* **show** *?thesis*
**proof**(*induct rule*: *simCasesCont*[**where** *C*=*Q*])
**case**(*Bound a y R'*)
**have** *RTrans*: $R \longmapsto a{<}\nu y{>} \prec R'$ **by** *fact*

**from** *QSimR RTrans* ‹$y \sharp Q$› **have** $\exists Q'.\ Q \longmapsto a{<}\nu y{>} \prec Q' \wedge (Q', R') \in Rel'$
**by**(*rule elim*)
**then obtain** *Q'* **where** *QTrans*: $Q \longmapsto a{<}\nu y{>} \prec Q'$ **and** *Q'Rel'R'*: $(Q', R')$
$\in Rel'$ **by** *blast*
**from** *PSimQ QTrans* ‹$y \sharp P$› **have** $\exists P'.\ P \longmapsto a{<}\nu y{>} \prec P' \wedge (P', Q') \in Rel$
**by**(*rule elim*)
**then obtain** *P'* **where** *PTrans*: $P \longmapsto a{<}\nu y{>} \prec P'$ **and** *P'RelQ'*: $(P', Q') \in$
*Rel* **by** *blast*

**moreover from** *P'RelQ' Q'Rel'R' Trans* **have** $(P', R') \in Rel''$ **by** *blast*

**ultimately show** *?case* **by** *blast*
**next**
**case**(*Free* $\alpha$ *R'*)
**have** *RTrans*: $R \longmapsto \alpha \prec R'$ **by** *fact*
**with** *QSimR* **have** $\exists Q'.\ Q \longmapsto \alpha \prec Q' \wedge (Q', R') \in Rel'$ **by**(*rule elim*)
**then obtain** *Q'* **where** *QTrans*: $Q \longmapsto \alpha \prec Q'$ **and** *Q'RelR'*: $(Q', R') \in Rel'$
**by** *blast*
**from** *PSimQ QTrans* **have** $\exists P'.\ P \longmapsto \alpha \prec P' \wedge (P', Q') \in Rel$ **by**(*rule elim*)
**then obtain** *P'* **where** *PTrans*: $P \longmapsto \alpha \prec P'$ **and** *P'RelQ'*: $(P', Q') \in Rel$
**by** *blast*
**from** *P'RelQ' Q'RelR' Trans* **have** $(P', R') \in Rel''$ **by** *blast*
**with** *PTrans* **show** $\exists P'.\ P \longmapsto \alpha \prec P' \wedge (P', R') \in Rel''$ **by** *blast*
**qed**
**qed**

**end**

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

**lemma** *monoAux*: $A \subseteq B \Longrightarrow P \leadsto[A]\ Q \longrightarrow P \leadsto[B]\ Q$
**by**(*auto intro*: *Strong-Early-Sim.monotonic*)

**coinductive-set** *bisim* :: $(pi \times pi)$ *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** −
**have** *aux*: $X \cup bisim = \{(P,\ Q).\ (P,\ Q) \in X \vee P \sim Q\}$ **by** *blast*

**from** *p* **show** *?thesis*
**by**(*coinduct*, *force dest*: *step simp add*: *aux*)
**qed**

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

**assumes** $(P,\ Q) \in X$
**and** $\bigwedge R\ S.\ (R,\ S) \in X \Longrightarrow R \rightsquigarrow[(X \cup bisim)]\ S$
**and** $\bigwedge R\ S.\ (R,\ S) \in X \Longrightarrow (S,\ R) \in X$

**shows** $P \sim Q$
**using** *assms*
**by**(*coinduct rule*: *bisimCoinductAux*) *auto*

**lemma** *weak-coinduct*[*case-names bisim*, *case-conclusion StrongBisim step*, *consumes 1*]:
**assumes** *p*: $(P,\ Q) \in X$
**and** *step*: $\bigwedge P\ Q.\ (P,\ Q) \in X \Longrightarrow P \rightsquigarrow[X]\ Q \wedge (Q,\ P) \in X$

**shows** $P \sim Q$
**using** *p*
**proof**(*coinduct rule*: *bisimCoinductAux*)
**case** (*bisim P*)
**from** *step*[*OF this*] **show** *?case* **using** *Strong-Early-Sim.monotonic* **by** *blast*
**qed**

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

**assumes** $(P, Q) \in X$
**and** $\bigwedge P\ Q.\ (P, Q) \in X \Longrightarrow P \rightsquigarrow[X]\ Q$
**and** $\bigwedge P\ Q.\ (P, Q) \in X \Longrightarrow (Q, P) \in X$

**shows** $P \sim Q$
**using** *assms*
**by**(*coinduct rule*: *weak-coinduct*) *auto*

**lemma** *monotonic*: *mono*($\lambda p$ *x1 x2*.
$\exists P\ Q.\ x1 = P \wedge$
$x2 = Q \wedge P \rightsquigarrow[\{(xa, x).\ p\ xa\ x\}]\ Q \wedge Q \rightsquigarrow[\{(xa, x).\ p\ xa\ x\}]\ P)$
**apply**(*rule monoI*)
**by**(*auto intro*: *Strong-Early-Sim.monotonic*)

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

**assumes** $P \sim Q$

**shows** $P \rightsquigarrow[bisim]\ Q$
**and** $Q \sim P$
**using** *assms*
**by**(*auto intro*: *bisim.cases*)

**lemma** *bisimClosed*[*eqvt*]:
**fixes** $P$ :: *pi*
**and** $Q$ :: *pi*
**and** $p$ :: *name prm*

**assumes** $P \sim Q$

**shows** $(p \cdot P) \sim (p \cdot Q)$
**proof** −
**let** $?X = \{(p \cdot P, p \cdot Q) \mid (p::name\ prm)\ P\ Q.\ P \sim Q\}$
**from** *assms* **have** $(p \cdot P, p \cdot Q) \in ?X$ **by** *auto*
**thus** *?thesis*
**proof**(*coinduct rule*: *bisimWeakCoinduct*)
**case**(*cSim P Q*)
**moreover** **{**
**fix** *P Q*
**fix** *p*::*name prm*
**assume** $P \rightsquigarrow[bisim]\ Q$

**moreover have** $bisim \subseteq ?X$
**by**(*auto*, *rule-tac x*=[] **in** *exI*) *auto*
**moreover have** *eqvt ?X*
**by**(*auto simp add*: *eqvt-def pt2*[*OF pt-name-inst*, *THEN sym*]) *blast*

```
      ultimately have (p • P) ⇝[?X] (p • Q)
        by(rule Strong-Early-Sim.eqvtI)
    }
    ultimately show ?case by(blast dest: bisimE)
  next
    case(cSym P Q)
    thus ?case by(blast dest: bisimE)
  qed
qed

lemma eqvt[simp]:
  shows eqvt bisim
by(auto simp add: eqvt-def eqvts)

lemma reflexive:
  fixes P :: pi

  shows P ∼ P
proof −
  have (P, P) ∈ Id by simp
  then show ?thesis
    by(coinduct rule: bisimWeakCoinduct) (auto intro: Strong-Early-Sim.reflexive)
qed

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

  assumes PBiSimQ: P ∼ Q
  and     QBiSimR: Q ∼ R

  shows P ∼ R
proof −
  let ?X = bisim O bisim
  from assms have (P, R) ∈ ?X by blast
  thus ?thesis
  proof(coinduct rule: bisimWeakCoinduct)
    case(cSim P Q)
    moreover {
      fix P Q R
      assume P ∼ Q and Q ∼ R
      hence P ⇝[bisim] Q and Q ⇝[bisim] R
        by(metis bisimE)+
      moreover from eqvt have eqvt ?X by(auto simp add: eqvtTrans)
      moreover have bisim O bisim ⊆ ?X by auto

      ultimately have P ⇝[?X] R
        by(rule Strong-Early-Sim.transitive)
```

}
**ultimately show** *?case* **by** *auto*
**next**
**case**(*cSym P Q*)
**thus** *?case* **by**(*auto dest*: *bisimE*)
**qed**
**qed**

**end**

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

**abbreviation** *StrongCongEarlyJudge* (**infixr** ‹$\sim^s$› *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** $P \sim Q$
**using** *assms substClosedSubset* **by** *blast*

**lemma** *eqvt*:
**shows** *eqvt* (*substClosed bisim*)
**by**(*rule eqvtSubstClosed*[*OF Strong-Early-Bisim.eqvt*])

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

**assumes** $P \sim^s Q$

**shows** (*perm* $\cdot$ $P$) $\sim^s$ (*perm* $\cdot$ $Q$)
**using** *assms*
**by**(*rule eqvtRelI*[*OF eqvt*])

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

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

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

**and** $Q :: pi$

**assumes** $P \sim^s Q$

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

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

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

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

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

**assumes** $P \sim^s Q$

**shows** $P[<s>] \sim^s Q[<s>]$
**using** *assms*
**proof**(*auto simp add*: *substClosed-def*)
**fix** $s'$
**assume** $\forall s.\ P[<s>] \sim Q[<s>]$
**hence** $P[<(s@s')>] \sim Q[<(s@s')>]$ **by** *blast*
**moreover have** $P[<(s@s')>] = (P[<s>])[<s'>]$
**by**(*induct* $s'$, *auto*)
**moreover have** $Q[<(s@s')>] = (Q[<s>])[<s'>]$
**by**(*induct* $s'$, *auto*)

**ultimately show** $(P[<s>])[<s'>] \sim (Q[<s>])[<s'>]$
**by** *simp*
**qed**

**end**

**theory** *Strong-Early-Sim-Pres*
**imports** *Strong-Early-Sim*
**begin**

**lemma** *tauPres*:

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

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

**shows** $\tau.(P) \leadsto$[*Rel*] $\tau.(Q)$
**proof**(*induct rule*: *simCases*)
**case**(*Bound a y Q'*)
**have** $\tau.(Q) \longmapsto a{<}\nu y{>} \prec Q'$ **by** *fact*
**hence** *False* **by**(*induct rule*: *tauCases'*, *auto*)
**thus** *?case* **by** *simp*
**next**
**case**(*Free* $\alpha$ *Q'*)
**have** $\tau.(Q) \longmapsto \alpha \prec Q'$ **by** *fact*
**thus** $\exists P'. \tau.(P) \longmapsto \alpha \prec P' \wedge (P', Q') \in$ *Rel*
**proof**(*induct rule*: *tauCases'*, *auto simp add*: *pi.inject residual.inject*)
**have** $\tau.(P) \longmapsto \tau \prec P$ **by**(*rule TransitionsEarly.Tau*)
**with** *PRelQ* **show** $\exists P'. \tau.(P) \longmapsto \tau \prec P' \wedge (P', Q) \in$ *Rel* **by** *blast*
**qed**
**qed**

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

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

**shows** $a{<}x{>}.P \leadsto$[*Rel*] $a{<}x{>}.Q$
**using** *Eqvt*
**proof**(*induct rule*: *simCasesCont*[**where** $C=(x, a, P, Q)$])
**case**(*Bound b y Q'*)
**from** ‹$y \sharp (x, a, P, Q)$› **have** $y \neq x$ $y \neq a$ $y \sharp P$ $y \sharp Q$ **by** *simp+*
**from** ‹$a{<}x{>}.Q \longmapsto b{<}\nu y{>} \prec Q'$› ‹$y \neq a$› ‹$y \neq x$› ‹$y \sharp Q$› **show** *?case*
**by**(*erule-tac inputCases'*) *auto*
**next**
**case**(*Free* $\alpha$ *Q'*)
**from** ‹$a{<}x{>}.Q \longmapsto \alpha \prec Q'$›
**show** *?case*
**proof**(*induct rule*: *inputCases*)
**case**(*cInput u*)
**have** $a{<}x{>}.P \longmapsto a{<}u{>} \prec P[x::=u]$ **by**(*rule Input*)
**moreover from** *PRelQ* **have** $(P[x::=u], Q[x::=u]) \in$ *Rel* **by** *auto*
**ultimately show** *?case* **by** *blast*

  **qed**
**qed**

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

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

  **shows** $a\{b\}.P \rightsquigarrow[Rel]\ a\{b\}.Q$
**proof**(*induct rule*: *simCases*)
  **case**(*Bound c y Q'*)
  **have** $a\{b\}.Q \longmapsto c{<}\nu y{>} \prec Q'$ **by** *fact*
  **hence** *False* **by**(*induct rule*: *outputCases'*, *auto*)
  **thus** $\exists P'.\ a\{b\}.P \longmapsto c{<}\nu y{>} \prec P' \wedge (P', Q') \in Rel$ **by** *simp*
**next**
  **case**(*Free* $\alpha$ *Q'*)
  **have** $a\{b\}.Q \longmapsto \alpha \prec Q'$ **by** *fact*
  **thus** $\exists P'.\ a\{b\}.P \longmapsto \alpha \prec P' \wedge (P', Q') \in Rel$
  **proof**(*induct rule*: *outputCases'*, *auto simp add*: *pi.inject residual.inject*)
    **have** $a\{b\}.P \longmapsto a[b] \prec P$ **by**(*rule TransitionsEarly.Output*)
    **with** *PRelQ* **show** $\exists P'.\ a\{b\}.P \longmapsto a[b] \prec P' \wedge (P', Q) \in Rel$ **by** *blast*
  **qed**
**qed**

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

  **assumes** *PSimQ*: $P \rightsquigarrow[Rel]\ Q$
  **and** *RelRel'*: $Rel \subseteq Rel'$
  **shows** $[a\frown b]P \rightsquigarrow[Rel']\ [a\frown b]Q$
**proof**(*induct rule*: *simCases*)
  **case**(*Bound c y Q'*)
  **have** $(y{::}name) \sharp [a\frown b]P$ **by** *fact*
  **hence** *yFreshP*: $y \sharp P$ **by** *simp*
  **have** $[a\frown b]Q \longmapsto c{<}\nu y{>} \prec Q'$ **by** *fact*
  **thus** *?case*
  **proof**(*induct rule*: *matchCases*)
    **case** *Match*
    **have** $Q \longmapsto c{<}\nu y{>} \prec Q'$ **by** *fact*

```
   with PSimQ yFreshP obtain P' where PTrans: P ⟼c<νy> ≺ P' and
P'RelQ': (P', Q') ∈ Rel
     by(blast dest: elim)

   from PTrans have [a⌢a]P ⟼ c<νy> ≺ P' by(rule Early-Semantics.Match)
   moreover from P'RelQ' RelRel' have (P', Q') ∈ Rel' by blast
   ultimately show ?case by blast
  qed
next
  case(Free α Q')
  assume [a⌢b]Q ⟼ α ≺ Q'
  thus ?case
  proof(induct rule: matchCases)
   case Match
   have Q ⟼ α ≺ Q' by fact
   with PSimQ obtain P' where PTrans: P ⟼ α ≺ P' and P'RelQ': (P', Q')
∈ Rel
     by(blast dest: elim)

   from PTrans have [a⌢a]P ⟼α ≺ P' by(rule TransitionsEarly.Match)
   moreover from P'RelQ' RelRel' have (P', Q') ∈ Rel' by blast
   ultimately show ?case by blast
  qed
qed

lemma mismatchPres:
  fixes P    :: pi
  and   Q    :: pi
  and   a    :: name
  and   b    :: name
  and   Rel  :: (pi × 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(cases a = b)
  assume a = b
  thus ?thesis
   by(auto simp add: strongSimEarly-def)
next
  assume aineqb: a ≠ b
  show ?thesis
  proof(induct rule: simCases)
   case(Bound c x Q')
   have x ♯ [a≠b]P by fact
   hence xFreshP: x ♯ P by simp
   have [a≠b]Q ⟼ c<νx> ≺ Q' by fact
```

**thus** *?case*
**proof**(*induct rule*: *mismatchCases*)
**case** *Mismatch*
**have** $Q \longmapsto c{<}\nu x{>} \prec Q'$ **by** *fact*
**with** *PSimQ xFreshP* **obtain** $P'$ **where** *PTrans*: $P \longmapsto c{<}\nu x{>} \prec P'$
**and** *P′RelQ′*: $(P', Q') \in Rel$
**by**(*blast dest*: *elim*)

**from** *PTrans aineqb* **have** $[a{\neq}b]P \longmapsto c{<}\nu x{>} \prec P'$ **by**(*rule Early-Semantics.Mismatch*)
**moreover from** *P′RelQ′ RelRel′* **have** $(P', Q') \in Rel'$ **by** *blast*
**ultimately show** *?case* **by** *blast*
**qed**
**next**
**case**(*Free* $\alpha$ $Q'$)
**have** $[a{\neq}b]Q \longmapsto \alpha \prec Q'$ **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *mismatchCases*)
**case** *Mismatch*
**have** $Q \longmapsto \alpha \prec Q'$ **by** *fact*
**with** *PSimQ* **obtain** $P'$ **where** *PTrans*: $P \longmapsto \alpha \prec P'$
**and** *PRel*: $(P', Q') \in Rel$
**by**(*blast dest*: *elim*)
**from** *PTrans* ‹$a \neq b$› **have** $[a{\neq}b]P \longmapsto \alpha \prec P'$ **by**(*rule TransitionsEarly.Mismatch*)
**with** *RelRel′ PRel* **show** *?case* **by** *blast*
**qed**
**qed**
**qed**

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

**assumes** $P \leadsto[Rel]\ Q$
**and** *C1*: $Id \subseteq Rel'$
**and** $Rel \subseteq Rel'$

**shows** $P \oplus R \leadsto[Rel']\ Q \oplus R$
**proof**(*induct rule*: *simCases*)
**case**(*Bound a y* $Q'$)
**have** $y \sharp P \oplus R$ **by** *fact*
**hence** $(y{::}name) \sharp P$ **and** $y \sharp R$ **by** *simp+*
**from** ‹$Q \oplus R \longmapsto a{<}\nu y{>} \prec Q'$› **show** *?case*
**proof**(*induct rule*: *sumCases*)
**case** *Sum1*
**from** ‹$P \leadsto[Rel]\ Q$› ‹$Q \longmapsto a{<}\nu y{>} \prec Q'$› ‹$y \sharp P$› **obtain** $P'$ **where** *PTrans*:
$P \longmapsto a{<}\nu y{>} \prec P'$ **and** *P′RelQ′*: $(P', Q') \in Rel$

```
      by(blast dest: elim)

    from PTrans have P ⊕ R ⟼a<νy> ≺ P' by(rule Early-Semantics.Sum1)
    moreover from P'RelQ' ‹Rel ⊆ Rel'› have (P', Q') ∈ Rel' by blast
    ultimately show ?case by blast
  next
    case Sum2
   from ‹R ⟼a<νy> ≺ Q'› have P ⊕ R ⟼a<νy> ≺ Q' by(rule Early-Semantics.Sum2)
    moreover from C1 have (Q', Q') ∈ Rel' by auto
    ultimately show ?case by blast
  qed
next
  case(Free α Q')
  from ‹Q ⊕ R ⟼α ≺ Q'› show ∃P'. P ⊕ R ⟼ α ≺ P' ∧ (P', Q') ∈ Rel'
  proof(induct rule: sumCases)
    case Sum1
    have Q ⟼α ≺ Q' by fact
     with ‹P ⇝[Rel] Q› obtain P' where PTrans: P ⟼α ≺ P' and P'RelQ':
(P', Q') ∈ Rel
      by(blast dest: elim)

    from PTrans have P ⊕ R ⟼α ≺ P' by(rule TransitionsEarly.Sum1)
    moreover from P'RelQ' ‹Rel ⊆ Rel'› have (P', Q') ∈ Rel' by blast
    ultimately show ?case by blast
  next
    case Sum2
    from ‹R ⟼α ≺ Q'› have P ⊕ R ⟼α ≺ Q' by(rule TransitionsEarly.Sum2)
    moreover from C1 have (Q', Q') ∈ Rel' by blast
    ultimately show ?case by blast
  qed
qed

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

  assumes PSimQ:    P ⇝[Rel] Q
  and     RSimT:    R ⇝[Rel'] S
  and     PRelQ:    (P, Q) ∈ Rel
  and     RRel'T:   (R, S) ∈ Rel'
  and     Par:      ⋀P' Q' R' S'. ⟦(P', Q') ∈ Rel; (R', S') ∈ Rel'⟧ ⟹ (P' ∥ R',
Q' ∥ S') ∈ Rel''
  and     Res:      ⋀S T x. (S, T) ∈ Rel'' ⟹ (<νx>S, <νx>T) ∈ Rel''
```

**shows** $P \parallel R \leadsto[Rel''] \; Q \parallel S$
**proof**(*induct rule*: *simCases*)
**case**(*Bound a x Q'*)
**have** $x \sharp P \parallel R$ **by** *fact*
**hence** *xFreshP*: $x \sharp P$ **and** *xFreshR*: $x \sharp R$ **by** *simp+*
**have** $Q \parallel S \longmapsto a{<}\nu x{>} \prec Q'$ **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *parCasesB*)
**case**(*cPar1 Q'*)
**have** $Q \longmapsto a{<}\nu x{>} \prec Q'$ **by** *fact*
**with** *PSimQ xFreshP* **obtain** $P'$ **where** *PTrans*:$P \longmapsto a{<}\nu x{>} \prec P'$ **and** *P'RelQ'*: $(P', Q') \in Rel$
**by**(*blast dest*: *elim*)

**from** *PTrans xFreshR* **have** $P \parallel R \longmapsto a{<}\nu x{>} \prec (P' \parallel R)$ **by**(*rule Early-Semantics.Par1B*)
**moreover from** *P'RelQ' RRel'T* **have** $(P' \parallel R, Q' \parallel S) \in Rel''$ **by**(*rule Par*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*cPar2 S'*)
**have** $S \longmapsto a{<}\nu x{>} \prec S'$ **by** *fact*
**with** *RSimT xFreshR* **obtain** $R'$ **where** *RTrans*:$R \longmapsto a{<}\nu x{>} \prec R'$ **and** *R'Rel'T'*: $(R', S') \in Rel'$
**by**(*blast dest*: *elim*)

**from** *RTrans xFreshP* **have** *ParTrans*: $P \parallel R \longmapsto a{<}\nu x{>} \prec (P \parallel R')$ **by**(*rule Early-Semantics.Par2B*)
**moreover from** *PRelQ R'Rel'T'* **have** $(P \parallel R', Q \parallel S') \in Rel''$ **by**(*rule Par*)
**ultimately show** *?case* **by** *blast*
**qed**
**next**
**case**(*Free* $\alpha$ *QT'*)
**have** $Q \parallel S \longmapsto \alpha \prec QT'$ **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *parCasesF*[*of* - - - - - $(P, R)$])
**case**(*cPar1 Q'*)
**have** $Q \longmapsto \alpha \prec Q'$ **by** *fact*
**with** *PSimQ* **obtain** $P'$ **where** *PTrans*: $P \longmapsto \alpha \prec P'$ **and** *PRel*: $(P', Q') \in Rel$
**by**(*blast dest*: *elim*)

**from** *PTrans* **have** $P \parallel R \longmapsto \alpha \prec P' \parallel R$ **by**(*rule Early-Semantics.Par1F*)
**moreover from** *PRel RRel'T* **have** $(P' \parallel R, Q' \parallel S) \in Rel''$ **by**(*rule Par*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*cPar2 S'*)
**have** $S \longmapsto \alpha \prec S'$ **by** *fact*
**with** *RSimT* **obtain** $R'$ **where** *RTrans*: $R \longmapsto \alpha \prec R'$ **and** *RRel*: $(R', S') \in Rel'$
**by**(*blast dest*: *elim*)

**from** *RTrans* **have** $P \parallel R \longmapsto \alpha \prec P \parallel R'$ **by**(*rule Early-Semantics.Par2F*)
**moreover from** *PRelQ RRel* **have** $(P \parallel R', Q \parallel S') \in Rel''$ **by**(*rule Par*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*cComm1* $Q'$ $S'$ *a b*)
**have** $Q \longmapsto a{<}b{>} \prec Q'$ **by** *fact*
**with** *PSimQ* **obtain** $P'$ **where** *PTrans*: $P \longmapsto a{<}b{>} \prec P'$ **and** $P'RelQ'$: $(P', Q') \in Rel$
**by**(*blast dest*: *elim*)

**have** $S \longmapsto a[b] \prec S'$ **by** *fact*
**with** *RSimT* **obtain** $R'$ **where** *RTrans*: $R \longmapsto a[b] \prec R'$ **and** *RRel*: $(R', S') \in Rel'$
**by**(*blast dest*: *elim*)

**from** *PTrans RTrans* **have** $P \parallel R \longmapsto \tau \prec P' \parallel R'$ **by**(*rule Early-Semantics.Comm1*)
**moreover from** $P'RelQ'$ *RRel* **have** $(P' \parallel R', Q' \parallel S') \in Rel''$ **by**(*rule Par*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*cComm2* $Q'$ $S'$ *a b*)
**have** $Q \longmapsto (OutputR\ a\ b) \prec Q'$ **by** *fact*
**with** *PSimQ* **obtain** $P'$ **where** *PTrans*: $P \longmapsto a[b] \prec P'$ **and** *PRel*: $(P', Q') \in Rel$
**by**(*blast dest*: *elim*)

**have** $S \longmapsto a{<}b{>} \prec S'$ **by** *fact*
**with** *RSimT* **obtain** $R'$ **where** *RTrans*: $R \longmapsto a{<}b{>} \prec R'$ **and** $R'Rel'T'$: $(R', S') \in Rel'$
**by**(*blast dest*: *elim*)

**from** *PTrans RTrans* **have** $P \parallel R \longmapsto \tau \prec P' \parallel R'$ **by**(*rule Early-Semantics.Comm2*)
**moreover from** *PRel* $R'Rel'T'$ **have** $(P' \parallel R', Q' \parallel S') \in Rel''$ **by**(*rule Par*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*cClose1* $Q'$ $S'$ *a x*)
**have** $x \sharp (P, R)$ **by** *fact*
**hence** *xFreshP*: $x \sharp P$ **and** *xFreshR*: $x \sharp R$ **by** *simp*+

**have** $Q \longmapsto a{<}x{>} \prec Q'$ **by** *fact*
**with** *PSimQ* **obtain** $P'$ **where** *PTrans*: $P \longmapsto a{<}x{>} \prec P'$ **and** $P'RelQ'$: $(P', Q') \in Rel$
**by**(*blast dest*: *elim*)

**have** $S \longmapsto a{<}\nu x{>} \prec S'$ **by** *fact*
**with** *RSimT xFreshR* **obtain** $R'$ **where** *RTrans*: $R \longmapsto a{<}\nu x{>} \prec R'$ **and** $R'Rel'T'$: $(R', S') \in Rel'$
**by**(*blast dest*: *elim*)

**from** *PTrans RTrans xFreshP* **have** $P \parallel R \longmapsto \tau \prec {<}\nu x{>}(P' \parallel R')$
**by**(*rule Early-Semantics.Close1*)
**moreover from** $P'RelQ'$ $R'Rel'T'$ **have** $({<}\nu x{>}(P' \parallel R'), {<}\nu x{>}(Q' \parallel S')) \in Rel''$
**by**(*blast intro*: *Par Res*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*cClose2* $Q'$ $S'$ *a x*)
**have** $x \sharp (P, R)$ **by** *fact*
**hence** *xFreshP*: $x \sharp P$ **and** *xFreshR*: $x \sharp R$ **by** *simp+*

**have** $Q \longmapsto a{<}\nu x{>} \prec Q'$ **by** *fact*
**with** *PSimQ xFreshP* **obtain** $P'$ **where** *PTrans*: $P \longmapsto a{<}\nu x{>} \prec P'$ **and** $P'RelQ'$: $(P', Q') \in Rel$
**by**(*blast dest*: *elim*)

**have** $S \longmapsto a{<}x{>} \prec S'$ **by** *fact*
**with** *RSimT* **obtain** $R'$ **where** *RTrans*: $R \longmapsto a{<}x{>} \prec R'$ **and** $R'Rel'T'$: $(R', S') \in Rel'$
**by**(*blast dest*: *elim*)

**from** *PTrans RTrans xFreshR* **have** $P \parallel R \longmapsto \tau \prec {<}\nu x{>}(P' \parallel R')$
**by**(*rule Early-Semantics.Close2*)
**moreover from** $P'RelQ'$ $R'Rel'T'$ **have** $({<}\nu x{>}(P' \parallel R'), {<}\nu x{>}(Q' \parallel S')) \in Rel''$
**by**(*blast intro*: *Par Res*)
**ultimately show** *?case* **by** *blast*
**qed**
**qed**

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

**assumes** *PSimQ*: $P \leadsto[Rel]\ Q$
**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** −
**note** *PSimQ*
**moreover have** *RSimR*: $R \leadsto[Id]\ R$ **by**(*auto intro*: *reflexive*)
**moreover note** *PRelQ* **moreover have** $(R, R) \in Id$ **by** *auto*

**moreover from** *Par* **have** $\bigwedge P\ Q\ R\ T.\ [\![(P, Q) \in Rel; (R, T) \in Id]\!] \Longrightarrow (P \parallel R, Q \parallel T) \in Rel'$
**by** *auto*
**ultimately show** *?thesis* **using** *Res* **by**(*rule parCompose*)
**qed**

**lemma** *resPres*:
**fixes** $P$ :: *pi*
**and** $Q$ :: *pi*
**and** *Rel* :: $(pi \times pi)$ *set*
**and** $x$ :: *name*
**and** *Rel'* :: $(pi \times pi)$ *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** −
**from** *EqvtRel'* **show** *?thesis*
**proof**(*induct rule*: *simCasesCont*[**where** $C = (P, x)$])
**case**(*Bound a y Q'*)
**have** *Trans*: $<\nu x>Q \longmapsto a<\nu y> \prec Q'$ **by** *fact*
**have** $y \sharp (P, x)$ **by** *fact*
**hence** *yineqx*: $y \neq x$ **and** *yFreshP*: $y \sharp (P::pi)$ **by** *simp+*
**from** *Trans yineqx* **show** *?case*
**proof**(*induct rule*: *resCasesB*)
**case**(*Open Q'*)
**have** *QTrans*: $Q \longmapsto (a::name)[x] \prec Q'$ **by** *fact*
**with** *PSimQ* **obtain** $P'$ **where** *PTrans*: $P \longmapsto a[x] \prec P'$ **and** *P'RelQ'*: $(P', Q') \in Rel$
**by**(*blast dest*: *elim*)

**have** $<\nu x>P \longmapsto a<\nu y> \prec ([(y, x)] \cdot P')$
**proof** −
**have** *aineqx*: $a \neq x$ **by** *fact*
**with** *PTrans* **have** $<\nu x>P \longmapsto a<\nu x> \prec P'$ **by**(*rule TransitionsEarly.Open*)
**moreover have** $a<\nu x> \prec P' = a<\nu y> \prec ([(y, x)] \cdot P')$
**proof** −
**from** *PTrans yFreshP* **have** *yFreshP'*: $y \sharp P'$ **by**(*force intro*: *freshTransition*)
**thus** *?thesis* **by**(*simp add*: *alphaBoundOutput name-swap*)
**qed**
**ultimately show** *?thesis* **by** *simp*
**qed**
**moreover from** *EqvtRel P'RelQ' RelRel'* **have** $([(y, x)] \cdot P', [(y, x)] \cdot Q') \in Rel'$

**by**(*blast intro*: *eqvtRelI*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*Res* $Q'$)
**have** *QTrans*: $Q \longmapsto a{<}\nu y{>} \prec Q'$ **by** *fact*

**with** *PSimQ yFreshP* **obtain** $P'$ **where** *PTrans*: $P \longmapsto a{<}\nu y{>} \prec P'$ **and** *P'RelQ'*: $(P', Q') \in Rel$
**by**(*blast dest*: *elim*)

**have** *xineqa*: $x \neq a$ **by** *fact*
**with** *PTrans yineqx* **have** *ResTrans*: $<\nu x>P \longmapsto a{<}\nu y{>} \prec (<\nu x>P')$
**by**(*blast intro*: *ResB*)
**moreover from** *P'RelQ'* **have** $((<\nu x>P'), (<\nu x>Q')) \in Rel'$
**by**(*rule ResSet*)

**ultimately show** $\exists P'. <\nu x>P \longmapsto a{<}\nu y{>} \prec P' \wedge (P', (<\nu x>Q')) \in Rel'$ **by** *blast*
**qed**
**next**
**case**(*Free* $\alpha$ $Q'$)
**have** *Trans*: $<\nu x>Q \longmapsto \alpha \prec Q'$ **by** *fact*
**have** $\exists c$::*name*. $c \sharp (P, Q, Q', \alpha)$ **by**(*blast intro*: *name-exists-fresh*)
**then obtain** *c*::*name* **where** *cFreshQ*: $c \sharp Q$ **and** *cFreshAlpha*: $c \sharp \alpha$ **and** *cFreshQ'*: $c \sharp Q'$ **and** *cFreshP*: $c \sharp P$
**by**(*force simp add*: *fresh-prod*)
**from** *cFreshP* **have** $<\nu x>P = <\nu c>([(x, c)] \cdot P)$ **by**(*simp add*: *alphaRes*)
**moreover have** $\exists P'.<\nu c>([(x, c)] \cdot P) \longmapsto \alpha \prec P' \wedge (P', Q') \in Rel'$
**proof** −
**from** *Trans cFreshQ* **have** $<\nu c>([(x, c)] \cdot Q) \longmapsto \alpha \prec Q'$ **by**(*simp add*: *alphaRes*)
**moreover from** *EqvtRel PSimQ* **have** $([(x, c)] \cdot P) \leadsto[Rel] ([(x, c)] \cdot Q)$
**by**(*blast intro*: *eqvtI*)
**ultimately show** *?thesis* **using** *cFreshAlpha*
**apply** −
**apply**(*erule resCasesF*)
**apply** *auto*
**by**(*blast intro*: *ResF ResSet dest*: *elim*)
**qed**

**ultimately show** $\exists P'.<\nu x>P \longmapsto \alpha \prec P' \wedge (P', Q') \in Rel'$ **by** *auto*
**qed**
**qed**

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

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

**shows** $(resChain\ lst)\ P \rightsquigarrow[Rel]\ (resChain\ lst)\ Q$
**proof** –
**show** *?thesis*
**proof**(*induct lst*)
**from** *PRelQ* **show** $resChain\ []\ P \rightsquigarrow[Rel]\ resChain\ []\ Q$ **by** *simp*
**next**
**fix** *a lst*
**assume** *IH*: $(resChain\ lst\ P) \rightsquigarrow[Rel]\ (resChain\ lst\ Q)$
**moreover from** *Res* **have** $\bigwedge P\ Q\ a.\ (P,\ Q) \in Rel \Longrightarrow (<\nu a>P,\ <\nu a>Q) \in Rel$
**by** *simp*
**moreover have** $Rel \subseteq Rel$ **by** *simp*
**ultimately have** $<\nu a>(resChain\ lst\ P) \rightsquigarrow[Rel]\ <\nu a>(resChain\ lst\ Q)$ **using** *eqvtRel*
**by**(*rule-tac resPres*)
**thus** $resChain\ (a\ \#\ lst)\ P \rightsquigarrow[Rel]\ resChain\ (a\ \#\ lst)\ Q$
**by** *simp*
**qed**
**qed**

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

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

**shows** $!P \rightsquigarrow[bangRel\ Rel]\ !Q$
**proof** –
**let** $?Sim = \lambda P\ Rs.\ (\forall\, a\ x\ Q'.\ Rs = a<\nu x> \prec Q' \longrightarrow x\ \sharp\ P \longrightarrow (\exists\, P'.\ P \longmapsto a<\nu x> \prec P' \wedge (P',\ Q') \in bangRel\ Rel)) \wedge$
$(\forall\, \alpha\ Q'.\ Rs = \alpha \prec Q' \longrightarrow (\exists\, P'.\ P \longmapsto \alpha \prec P' \wedge (P',\ Q') \in bangRel\ Rel))$
**from** *eqvtRel* **have** *EqvtBangRel*: *eqvt*(*bangRel Rel*) **by**(*rule eqvtBangRel*)

**{**
**fix** *Pa Rs*
**assume** $!Q \longmapsto Rs$ **and** $(Pa,\ !Q) \in bangRel\ Rel$
**hence** *?Sim Pa Rs* **using** *PRelQ*
**proof**(*nominal-induct avoiding*: *Pa P rule*: *bangInduct*)
**case**($Par1B\ a\ x\ Q'\ Pa\ P$)
**have** *QTrans*: $Q \longmapsto a<\nu x> \prec Q'$ **by** *fact*

**have** $(Pa,\ Q \parallel !Q) \in$ *bangRel Rel* **and** $x \sharp Pa$ **by** *fact+*
**thus** *?Sim Pa* $(a{<}\nu x{>} \prec (Q' \parallel !Q))$
**proof**(*induct rule*: *BRParCases*)
**case**(*BRPar P R*)
**have** *PRelQ*: $(P,\ Q) \in Rel$ **by** *fact*
**have** *PBRQ*: $(R,\ !Q) \in$ *bangRel Rel* **by** *fact*
**have** $x \sharp P \parallel R$ **by** *fact*
**hence** *xFreshP*: $x \sharp P$ **and** *xFreshR*: $x \sharp R$ **by** *simp+*
**show** *?case*
**proof**(*auto simp add*: *residual.inject alpha′*)
**from** *PRelQ* **have** $P \leadsto[Rel]\ Q$ **by**(*rule Sim*)

**with** *QTrans xFreshP* **obtain** $P'$ **where** *PTrans*: $P \longmapsto a{<}\nu x{>} \prec P'$ **and** *P′RelQ′*: $(P',\ Q') \in Rel$
**by**(*blast dest*: *elim*)

**from** *PTrans xFreshR* **have** $P \parallel R \longmapsto a{<}\nu x{>} \prec (P' \parallel R)$
**by**(*force intro*: *Early-Semantics.Par1B*)
**moreover from** *P′RelQ′ PBRQ* **have** $(P' \parallel R,\ Q' \parallel !Q) \in$ *bangRel Rel* **by**(*rule Rel.BRPar*)
**ultimately show** $\exists P'.\ P \parallel R \longmapsto a{<}\nu x{>} \prec P' \wedge (P',\ Q' \parallel !Q) \in$ *bangRel Rel* **by** *blast*
**next**
**fix** $y$
**assume** $(y{::}name) \sharp Q'$ **and** $y \sharp P$ **and** $y \sharp R$ **and** $y \sharp Q$
**from** *QTrans* ‹$y \sharp Q'$› **have** $Q \longmapsto a{<}\nu y{>} \prec ([(x,\ y)] \cdot Q')$
**by**(*simp add*: *alphaBoundOutput*)
**moreover from** *PRelQ* **have** $P \leadsto[Rel]\ Q$ **by**(*rule Sim*)
**ultimately obtain** $P'$ **where** *PTrans*: $P \longmapsto a{<}\nu y{>} \prec P'$ **and** *P′RelQ′*: $(P',\ [(x,\ y)] \cdot Q') \in Rel$
**using** ‹$y \sharp P$›
**by**(*blast dest*: *elim*)
**from** *PTrans* ‹$y \sharp R$› **have** $P \parallel R \longmapsto a{<}\nu y{>} \prec (P' \parallel R)$ **by**(*force intro*: *Early-Semantics.Par1B*)
**moreover from** *P′RelQ′ PBRQ* **have** $(P' \parallel R,\ ([(x,\ y)] \cdot Q') \parallel !Q) \in$ *bangRel Rel* **by**(*rule Rel.BRPar*)
**with** ‹$x \sharp Q$› ‹$y \sharp Q$› **have** $(P' \parallel R,\ ([(y,\ x)] \cdot Q') \parallel !([(y,\ x)] \cdot Q)) \in$ *bangRel Rel*
**by**(*simp add*: *name-fresh-fresh name-swap*)
**ultimately show** $\exists P'.\ P \parallel R \longmapsto a{<}\nu y{>} \prec P' \wedge (P',\ ([(y,\ x)] \cdot Q') \parallel !([(y,\ x)] \cdot Q)) \in$ *bangRel Rel*
**by** *blast*
**qed**
**qed**
**next**
**case**(*Par1F* $\alpha$ $Q'$ *Pa P*)
**have** *QTrans*: $Q \longmapsto \alpha \prec Q'$ **by** *fact*
**have** $(Pa,\ Q \parallel !Q) \in$ *bangRel Rel* **by** *fact*
**thus** *?case*

**proof**(*induct rule*: *BRParCases*)
**case**(*BRPar P R*)
**have** *PRelQ*: $(P,\ Q) \in$ *Rel* **and** *BR*: $(R,\ !Q) \in$ *bangRel Rel* **by** *fact+*
**show** *?case*
**proof**(*auto simp add*: *residual.inject*)
**from** *PRelQ* **have** $P \leadsto$[*Rel*] $Q$ **by**(*rule Sim*)
**with** *QTrans* **obtain** $P'$ **where** *PTrans*: $P \longmapsto \alpha \prec P'$ **and** *RRel*: $(P',\ Q') \in$ *Rel*
**by**(*blast dest*: *elim*)

**from** *PTrans* **have** $P \parallel R \longmapsto \alpha \prec P' \parallel R$ **by**(*rule TransitionsEarly.Par1F*)
**moreover from** *RRel BR* **have** $(P' \parallel R,\ Q' \parallel !Q) \in$ *bangRel Rel* **by**(*rule Rel.BRPar*)
**ultimately show** $\exists P'.\ P \parallel R \longmapsto \alpha \prec P' \wedge (P',\ Q' \parallel !Q) \in$ *bangRel Rel* **by** *blast*
**qed**
**qed**
**next**
**case**(*Par2B a x* $Q'$ *Pa P*)
**hence** *IH*: $\bigwedge Pa.\ (Pa,\ !Q) \in$ *bangRel Rel* $\Longrightarrow$ *?Sim Pa* $(a{<}\nu x{>} \prec Q')$ **by** *simp*
**have** $(Pa,\ Q \parallel !Q) \in$ *bangRel Rel* **and** $x \sharp Pa$ **by** *fact+*
**thus** *?Sim Pa* $(a{<}\nu x{>} \prec (Q \parallel Q'))$
**proof**(*induct rule*: *BRParCases*)
**case**(*BRPar P R*)
**have** *PRelQ*: $(P,\ Q) \in$ *Rel* **and** *RBRQ*: $(R,\ !Q) \in$ *bangRel Rel* **by** *fact+*
**have** $x \sharp P \parallel R$ **by** *fact*
**hence** *xFreshP*: $x \sharp P$ **and** *xFreshR*: $x \sharp R$ **by** *simp+*

**from** *EqvtBangRel* **show** *?Sim* $(P \parallel R)$ $(a{<}\nu x{>} \prec (Q \parallel Q'))$
**proof**(*auto simp add*: *residual.inject alpha'*)
**from** *RBRQ* **have** *?Sim R* $(a{<}\nu x{>} \prec Q')$ **by**(*rule IH*)
**with** *xFreshR* **obtain** $R'$ **where** *RTrans*: $R \longmapsto a{<}\nu x{>} \prec R'$ **and** $R'BRQ'$: $(R',\ Q') \in$ (*bangRel Rel*)
**by**(*metis elim*)
**from** *RTrans xFreshP* **have** $P \parallel R \longmapsto a{<}\nu x{>} \prec (P \parallel R')$ **by**(*auto intro*: *Early-Semantics.Par2B*)
**moreover from** *PRelQ* $R'BRQ'$ **have** $(P \parallel R',\ Q \parallel Q') \in$ (*bangRel Rel*) **by**(*rule Rel.BRPar*)
**ultimately show** $\exists P'.\ P \parallel R \longmapsto a{<}\nu x{>} \prec P' \wedge (P',\ Q \parallel Q') \in$ *bangRel Rel* **by** *blast*
**next**
**fix** *y*
**assume** $(y{::}name) \sharp Q$ **and** $y \sharp Q'$ **and** $y \sharp P$ **and** $y \sharp R$
**from** *RBRQ* **have** *?Sim R* $(a{<}\nu x{>} \prec Q')$ **by**(*rule IH*)
**with** $\langle y \sharp Q' \rangle$ **have** *?Sim R* $(a{<}\nu y{>} \prec ([(x,\ y)] \cdot Q'))$ **by**(*simp add*: *alphaBoundOutput*)
**with** $\langle y \sharp R \rangle$ **obtain** $R'$ **where** *RTrans*: $R \longmapsto a{<}\nu y{>} \prec R'$ **and** $R'BRQ'$: $(R',\ ([(x,\ y)] \cdot Q')) \in$ (*bangRel Rel*)

**by**(*metis elim*)
**from** *RTrans* $\langle y \sharp P\rangle$ **have** $P \parallel R \longmapsto a{<}\nu y{>} \prec (P \parallel R')$ **by**(*auto intro*: *Early-Semantics.Par2B*)
**moreover from** *PRelQ R'BRQ'* **have** $(P \parallel R',\ Q \parallel ([(x,\ y)] \cdot Q')) \in$ (*bangRel Rel*) **by**(*rule Rel.BRPar*)
**with** $\langle y \sharp Q\rangle$ $\langle x \sharp Q\rangle$ **have** $(P \parallel R',\ ([(y,\ x)] \cdot Q) \parallel ([(y,\ x)] \cdot Q')) \in$ (*bangRel Rel*)
**by**(*simp add*: *name-swap name-fresh-fresh*)
**ultimately show** $\exists P'.\ P \parallel R \longmapsto a{<}\nu y{>} \prec P' \wedge (P',\ ([(y,\ x)] \cdot Q) \parallel ([(y,\ x)] \cdot Q')) \in$ *bangRel Rel* **by** *blast*
**qed**
**qed**
**next**
**case**(*Par2F* $\alpha$ $Q'$ *Pa P*)
**hence** *IH*: $\bigwedge Pa.\ (Pa,\ !Q) \in$ *bangRel Rel* $\Longrightarrow$ *?Sim Pa* $(\alpha \prec Q')$ **by** *simp*
**have** $(Pa,\ Q \parallel !Q) \in$ *bangRel Rel* **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *BRParCases*)
**case**(*BRPar P R*)
**have** *PRelQ*: $(P,\ Q) \in$ *Rel* **and** *RBRQ*: $(R,\ !Q) \in$ *bangRel Rel* **by** *fact+*
**show** *?case*
**proof**(*auto simp add*: *residual.inject*)
**from** *RBRQ IH* **have** $\exists R'.\ R \longmapsto \alpha \prec R' \wedge (R',\ Q') \in$ *bangRel Rel*
**by**(*metis elim*)
**then obtain** $R'$ **where** *RTrans*: $R \longmapsto \alpha \prec R'$ **and** *R'RelQ'*: $(R',\ Q') \in$ *bangRel Rel*
**by** *blast*

**from** *RTrans* **have** $P \parallel R \longmapsto \alpha \prec P \parallel R'$ **by**(*rule TransitionsEarly.Par2F*)
**moreover from** *PRelQ R'RelQ'* **have** $(P \parallel R',\ Q \parallel Q') \in$ *bangRel Rel* **by**(*rule Rel.BRPar*)
**ultimately show** $\exists P'.\ P \parallel R \longmapsto \alpha \prec P' \wedge (P',\ Q \parallel Q') \in$ *bangRel Rel* **by** *blast*
**qed**
**qed**
**next**
**case**(*Comm1 a* $Q'$ *b* $Q''$ *Pa P*)
**hence** *IH*: $\bigwedge Pa.\ (Pa,\ !Q) \in$ *bangRel Rel* $\Longrightarrow$ *?Sim Pa* $(a[b] \prec Q'')$ **by** *simp*
**have** *QTrans*: $Q \longmapsto a{<}b{>} \prec Q'$ **by** *fact*
**have** $(Pa,\ Q \parallel !Q) \in$ *bangRel Rel* **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *BRParCases*)
**case**(*BRPar P R*)
**have** *PRelQ*: $(P,\ Q) \in$ *Rel* **and** *RBRQ*: $(R,\ !Q) \in$ *bangRel Rel* **by** *fact+*
**show** *?case*
**proof**(*auto simp add*: *residual.inject*)
**from** *PRelQ* **have** $P \leadsto[Rel]\ Q$ **by**(*rule Sim*)
**with** *QTrans* **obtain** $P'$ **where** *PTrans*: $P \longmapsto a{<}b{>} \prec P'$ **and** *P'RelQ'*: $(P',\ Q') \in$ *Rel*

**by**(*blast dest*: *elim*)

**from** *IH RBRQ* **have** *RTrans*: $\exists R'.\ R \longmapsto a[b] \prec R' \wedge (R',\ Q'') \in \mathit{bangRel}$ *Rel*
**by**(*metis elim*)
**then obtain** $R'$ **where** *RTrans*: $R \longmapsto a[b] \prec R'$ **and** $R'RelQ''$: $(R',\ Q'') \in$ *bangRel Rel*
**by** *blast*

**from** *PTrans RTrans* **have** $P \parallel R \longmapsto \tau \prec P' \parallel R'$ **by**(*rule Transition-sEarly.Comm1*)
**moreover from** $P'RelQ'$ $R'RelQ''$ **have** $(P' \parallel R',\ Q' \parallel Q'') \in$ *bangRel Rel* **by**(*rule Rel.BRPar*)
**ultimately show** $\exists P'.\ P \parallel R \longmapsto \tau \prec P' \wedge (P',\ Q' \parallel Q'') \in$ *bangRel Rel* **by** *blast*
**qed**
**qed**
**next**
**case**(*Comm2 a b* $Q'$ $Q''$)
**hence** *IH*: $\bigwedge Pa.\ (Pa,\ !Q) \in$ *bangRel Rel* $\Longrightarrow$ *?Sim Pa* $(a{<}b{>} \prec Q'')$ **by** *simp*
**have** *QTrans*: $Q \longmapsto a[b] \prec Q'$ **by** *fact*
**have** $(Pa,\ Q \parallel !Q) \in$ *bangRel Rel* **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *BRParCases*)
**case**(*BRPar P R*)
**have** *PRelQ*: $(P,\ Q) \in$ *Rel* **and** *RBRQ*: $(R, !Q) \in$ *bangRel Rel* **by** *fact+*
**show** *?case*
**proof**(*auto simp add*: *residual.inject*)
**from** *PRelQ* **have** $P \leadsto[Rel]\ Q$ **by**(*rule Sim*)
**with** *QTrans* **obtain** $P'$ **where** *PTrans*: $P \longmapsto a[b] \prec P'$ **and** $P'RelQ'$: $(P',\ Q') \in$ *Rel*
**by**(*blast dest*: *elim*)

**from** *IH RBRQ* **have** *RTrans*: $\exists R'.\ R \longmapsto a{<}b{>} \prec R' \wedge (R',\ Q'') \in$ *bangRel Rel*
**by**(*metis elim*)
**then obtain** $R'$ **where** *RTrans*: $R \longmapsto a{<}b{>} \prec R'$ **and** $R'RelQ''$: $(R',\ Q'') \in$ *bangRel Rel*
**by** *blast*

**from** *PTrans RTrans* **have** $P \parallel R \longmapsto \tau \prec P' \parallel R'$ **by**(*rule Transition-sEarly.Comm2*)
**moreover from** $P'RelQ'$ $R'RelQ''$ **have** $(P' \parallel R',\ Q' \parallel Q'') \in$ *bangRel Rel* **by**(*rule Rel.BRPar*)
**ultimately show** $\exists P'.\ P \parallel R \longmapsto \tau \prec P' \wedge (P',\ Q' \parallel Q'') \in$ *bangRel Rel* **by** *blast*
**qed**
**qed**
**next**

**case**(*Close1 a x Q′ Q′′ Pa P*)
**hence** *IH*: $\bigwedge$*Pa*. (*Pa*, !*Q*) ∈ *bangRel Rel* ⟶ *?Sim Pa* (*a*<*νx*> ≺ *Q′′*) **by** *simp*
**have** *QTrans*: *Q* ⟼ *a*<*x*> ≺ *Q′* **by** *fact*
**have** *xFreshQ*: *x* ♯ *Q* **by** *fact*
**have** (*Pa*, *Q* ∥ !*Q*) ∈ *bangRel Rel* **by** *fact*
**moreover have** *xFreshPa*: *x* ♯ *Pa* **by** *fact*
**ultimately show** *?case*
**proof**(*induct rule*: *BRParCases*)
**case**(*BRPar P R*)
**have** *PRelQ*: (*P*, *Q*) ∈ *Rel* **and** *RBRQ*: (*R*, !*Q*) ∈ *bangRel Rel* **by** *fact+*
**have** *x* ♯ *P* ∥ *R* **by** *fact*
**hence** *xFreshP*: *x* ♯ *P* **and** *xFreshR*: *x* ♯ *R* **by** *simp+*
**show** *?case*
**proof**(*auto simp add*: *residual.inject*)
**from** *PRelQ* **have** *P* ⇝[*Rel*] *Q* **by**(*rule Sim*)
**with** *QTrans xFreshP* **obtain** *P′* **where** *PTrans*: *P* ⟼*a*<*x*> ≺ *P′* **and** *P′RelQ′*: (*P′*, *Q′*) ∈ *Rel*
**by**(*blast dest*: *elim*)

**from** *RBRQ xFreshR IH* **have** ∃ *R′*. *R* ⟼*a*<*νx*> ≺ *R′* ∧ (*R′*, *Q′′*) ∈ *bangRel Rel*
**by**(*metis elim*)
**then obtain** *R′* **where** *RTrans*: *R* ⟼*a*<*νx*> ≺ *R′* **and** *R′RelQ′′*: (*R′*, *Q′′*) ∈ *bangRel Rel*
**by** *blast*

**from** *PTrans RTrans xFreshP* **have** *P* ∥ *R* ⟼*τ* ≺ <*νx*>(*P′* ∥ *R′*)
**by**(*rule Early-Semantics.Close1*)
**moreover from** *P′RelQ′ R′RelQ′′* **have** (<*νx*>(*P′* ∥ *R′*), <*νx*>(*Q′* ∥ *Q′′*)) ∈ *bangRel Rel*
**by**(*force intro*: *Rel.BRPar BRRes*)
**ultimately show** ∃ *P′*. *P* ∥ *R* ⟼ *τ* ≺ *P′* ∧ (*P′*, <*νx*>(*Q′* ∥ *Q′′*)) ∈ *bangRel Rel* **by** *blast*
**qed**
**qed**
**next**
**case**(*Close2 a x Q′ Q′′ Pa P*)
**hence** *IH*: $\bigwedge$*Pa*. (*Pa*, !*Q*) ∈ *bangRel Rel* ⟹ *?Sim Pa* (*a*<*x*> ≺ *Q′′*) **by** *simp*
**have** *QTrans*: *Q* ⟼ *a*<*νx*> ≺ *Q′* **by** *fact*
**have** *xFreshQ*: *x* ♯ *Q* **by** *fact*
**have** (*Pa*, *Q* ∥ !*Q*) ∈ *bangRel Rel* **and** *x* ♯ *Pa* **by** *fact+*
**thus** *?case*
**proof**(*induct rule*: *BRParCases*)
**case**(*BRPar P R*)
**have** *PRelQ*: (*P*, *Q*) ∈ *Rel* **and** *RBRQ*: (*R*, !*Q*) ∈ *bangRel Rel* **by** *fact+*
**have** *x* ♯ *P* ∥ *R* **by** *fact*
**hence** *xFreshP*: *x* ♯ *P* **and** *xFreshR*: *x* ♯ *R* **by** *simp+*
**show** *?case*

**proof**(*auto simp add*: *residual.inject*)
**from** *PRelQ* **have** $P \rightsquigarrow[Rel]\ Q$ **by**(*rule Sim*)
**with** *QTrans xFreshP* **obtain** $P'$ **where** *PTrans*: $P \longmapsto a{<}\nu x{>} \prec P'$ **and** $P'RelQ'$: $(P', Q') \in Rel$
**by**(*blast dest*: *elim*)

**from** *RBRQ IH* **have** $\exists R'.\ R \longmapsto a{<}x{>} \prec R' \wedge (R', Q'') \in bangRel\ Rel$
**by** *auto*
**then obtain** $R'$ **where** *RTrans*: $R \longmapsto a{<}x{>} \prec R'$ **and** $R'RelQ''$: $(R', Q'') \in bangRel\ Rel$
**by** *blast*

**from** *PTrans RTrans xFreshR* **have** $P \parallel R \longmapsto \tau \prec {<}\nu x{>}(P' \parallel R')$
**by**(*rule Early-Semantics.Close2*)
**moreover from** $P'RelQ'$ $R'RelQ''$ **have** $({<}\nu x{>}(P' \parallel R'), {<}\nu x{>}(Q' \parallel Q'')) \in bangRel\ Rel$
**by**(*force intro*: *Rel.BRPar BRRes*)
**ultimately show** $\exists P'.\ P \parallel R \longmapsto \tau \prec P' \wedge (P', {<}\nu x{>}(Q' \parallel Q'')) \in bangRel\ Rel$ **by** *blast*
**qed**
**qed**
**next**
**case**(*Bang Rs Pa P*)
**hence** *IH*: $\bigwedge Pa.\ (Pa, Q \parallel !Q) \in bangRel\ Rel \Longrightarrow$ *?Sim Pa Rs* **by** *simp*
**have** $(Pa, !Q) \in bangRel\ Rel$ **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *BRBangCases*)
**case**(*BRBang P*)
**have** *PRelQ*: $(P, Q) \in Rel$ **by** *fact*
**hence** $(!P, !Q) \in bangRel\ Rel$ **by**(*rule Rel.BRBang*)
**with** *PRelQ* **have** $(P \parallel !P, Q \parallel !Q) \in bangRel\ Rel$ **by**(*rule BRPar*)
**with** *IH* **have** *?Sim* $(P \parallel !P)$ *Rs* **by** *simp*
**thus** *?case* **by**(*force intro*: *TransitionsEarly.Bang*)
**qed**
**qed**
}

**moreover from** *PRelQ* **have** $(!P, !Q) \in bangRel\ Rel$ **by**(*rule BRBang*)
**ultimately show** *?thesis* **by**(*auto simp add*: *strongSimEarly-def*)
**qed**

**end**

**theory** *Strong-Early-Bisim-Pres*
**imports** *Strong-Early-Bisim Strong-Early-Sim-Pres*
**begin**

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

  **assumes** $P \sim Q$

  **shows** $\tau.(P) \sim \tau.(Q)$
**proof** $-$
  **let** $?X = \{(\tau.(P), \tau.(Q)) \mid P\ Q.\ P \sim Q\}$
  **from** $\langle P \sim Q\rangle$ **have** $(\tau.(P), \tau.(Q)) \in ?X$ **by** *auto*
  **thus** *?thesis*
    **by**(*coinduct rule*: *bisimCoinduct*) (*auto intro*: *tauPres dest*: *bisimE*)
**qed**

**lemma** *inputPres*:
  **fixes** $P :: pi$
  **and** $Q :: pi$
  **and** $a :: name$
  **and** $x :: name$

  **assumes** *PSimQ*: $\forall y.\ P[x::=y] \sim Q[x::=y]$

  **shows** $a{<}x{>}.P \sim a{<}x{>}.Q$
**proof** $-$
  **let** $?X = \{(a{<}x{>}.P, a{<}x{>}.Q) \mid a\ x\ P\ Q.\ \forall y.\ P[x::=y] \sim Q[x::=y]\}$
  **{**
    **fix** *axP axQ p*
    **assume** $(axP, axQ) \in ?X$
     **then obtain** $a\ x\ P\ Q$ **where** $A$: $\forall y.\ P[x::=y] \sim Q[x::=y]$ **and** $B$: $axP = a{<}x{>}.P$ **and** $C$: $axQ = a{<}x{>}.Q$
      **by** *auto*
    **have** $\bigwedge y.\ ((p::name\ prm) \cdot P)[(p \cdot x)::=y] \sim (p \cdot Q)[(p \cdot x)::=y]$
    **proof** $-$
      **fix** $y$
      **from** $A$ **have** $P[x::=(rev\ p \cdot y)] \sim Q[x::=(rev\ p \cdot y)]$
        **by** *blast*
      **hence** $(p \cdot (P[x::=(rev\ p \cdot y)])) \sim p \cdot (Q[x::=(rev\ p \cdot y)])$
        **by**(*rule bisimClosed*)
      **thus** $(p \cdot P)[(p \cdot x)::=y] \sim (p \cdot Q)[(p \cdot x)::=y]$
        **by**(*simp add*: *eqvts pt-pi-rev*[*OF pt-name-inst*, *OF at-name-inst*])
    **qed**
    **hence** $((p::name\ prm) \cdot axP, p \cdot axQ) \in ?X$ **using** $B\ C$
      **by** *auto*
  **}**
  **hence** *eqvt ?X* **by**(*simp add*: *eqvt-def*)
  **from** *PSimQ* **have** $(a{<}x{>}.P, a{<}x{>}.Q) \in ?X$ **by** *auto*
  **thus** *?thesis*
  **proof**(*coinduct rule*: *bisimCoinduct*)
    **case**(*cSim P Q*)

**thus** *?case* **using** ‹*eqvt ?X*›
**by**(*force intro*: *inputPres*)
**next**
**case**(*cSym P Q*)
**thus** *?case*
**by**(*blast dest*: *bisimE*)
**qed**
**qed**

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

**assumes** $P \sim Q$

**shows** $a\{b\}.P \sim a\{b\}.Q$
**proof** −
**let** $?X = \{(a\{b\}.P,\ a\{b\}.Q) \mid a\ b\ P\ Q.\ P \sim Q\}$
**from** ‹$P \sim Q$› **have** $(a\{b\}.P,\ a\{b\}.Q) \in\ ?X$ **by** *auto*
**thus** *?thesis*
**by**(*coinduct rule*: *bisimCoinduct*) (*blast intro*: *outputPres dest*: *bisimE*)+
**qed**

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

**assumes** $P \sim Q$

**shows** $[a\frown b]P \sim [a\frown b]Q$
**proof** −
**let** $?X = \{x.\ \exists\, P\ Q\ a\ b.\ P \sim Q \wedge x = ([a\frown b]P,\ [a\frown b]Q)\}$
**from** *assms* **have** $([a\frown b]P,\ [a\frown b]Q) \in\ ?X$ **by** *blast*
**thus** *?thesis*
**by**(*coinduct rule*: *bisimCoinduct*) (*blast intro*: *matchPres dest*: *bisimE*)+
**qed**

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

**assumes** $P \sim Q$

**shows** $[a{\neq}b]P \sim [a{\neq}b]Q$
**proof** $-$
**let** *?X* $= \{x.\ \exists\, P\ Q\ a\ b.\ P \sim Q \wedge x = ([a{\neq}b]P,\ [a{\neq}b]Q)\}$
**from** *assms* **have** $([a{\neq}b]P,\ [a{\neq}b]Q) \in$ *?X* **by** *blast*
**thus** *?thesis*
**by**(*coinduct rule*: *bisimCoinduct*) (*blast intro*: *mismatchPres dest*: *bisimE*)+
**qed**

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

**assumes** $P \sim Q$

**shows** $P \oplus R \sim Q \oplus R$
**proof** $-$
**let** *?X* $= \{(P \oplus R,\ Q \oplus R) \mid P\ Q\ R.\ P \sim Q\}$
**from** *assms* **have** $(P \oplus R,\ Q \oplus R) \in$ *?X* **by** *blast*
**thus** *?thesis*
**by**(*coinduct rule*: *bisimCoinduct*) (*auto dest*: *bisimE intro*: *reflexive sumPres*)
**qed**

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

**assumes** $P \sim Q$

**shows** $<\nu x>P \sim <\nu x>Q$
**proof** $-$
**let** *?X* $= \{x.\ \exists\, P\ Q.\ P \sim Q \wedge (\exists\, a.\ x = (<\nu a>P,\ <\nu a>Q))\}$
**from** *assms* **have** $(<\nu x>P,\ <\nu x>Q) \in$ *?X* **by** *blast*
**thus** *?thesis*
**proof**(*coinduct rule*: *bisimCoinduct*)
**case**(*cSim xP xQ*)
**moreover** **{**
**fix** $P\ Q\ a$
**assume** $P \sim Q$
**hence** $P \leadsto[bisim]\ Q$ **by**(*rule bisimE*)
**moreover have** $\bigwedge P\ Q\ a.\ P \sim Q \Longrightarrow (<\nu a>P,\ <\nu a>Q) \in$ *?X* $\cup$ *bisim* **by** *blast*
**moreover have** *bisim* $\subseteq$ *?X* $\cup$ *bisim* **by** *blast*
**moreover have** *eqvt bisim* **by**(*rule eqvt*)
**moreover have** *eqvt* (*?X* $\cup$ *bisim*) **using** *eqvts*
**by**(*auto simp add*: *eqvt-def*) *blast*
**ultimately have** $<\nu a>P \leadsto[($*?X* $\cup$ *bisim*$)]\ <\nu a>Q$
**by**(*rule Strong-Early-Sim-Pres.resPres*)

```
    }
    ultimately show ?case by auto
  next
    case(cSym xP xQ)
    thus ?case by(auto dest: bisimE)
  qed
qed

lemma parPres:
  fixes P :: pi
  and   Q :: pi
  and   R :: pi
  and   T :: pi

  assumes P ∼ Q

  shows P ∥ R ∼ Q ∥ R
proof −
  let ?X = {(resChain lst (P ∥ R), resChain lst (Q ∥ R)) | lst P Q R. P ∼ Q}
  have BC: ⋀P Q. P ∥ Q = resChain [] (P ∥ Q) by auto
  from assms have (P ∥ R, Q ∥ R) ∈ ?X by(blast intro: BC)
  thus ?thesis
  proof(coinduct rule: bisimWeakCoinduct)
    case(cSim PR QR)
    moreover {
      fix lst P Q R
      assume P ∼ Q
      have eqvt ?X using eqvts by(auto simp add: eqvt-def) blast
      moreover have Res: ⋀P Q x. (P, Q) ∈ ?X ⟹ (<νx>P, <νx>Q) ∈ ?X
        by(auto, rule-tac x=x#lst in exI) auto
      moreover {
        from ‹P ∼ Q› have P ↝[bisim] Q by(rule bisimE)
        moreover note ‹P ∼ Q›
        moreover have ⋀P Q R. P ∼ Q ⟹ (P ∥ R, Q ∥ R) ∈ ?X
          by(blast intro: BC)
        ultimately have P ∥ R ↝[?X] Q ∥ R using Res
          by(rule parPres)
      }

      ultimately have resChain lst (P ∥ R) ↝[?X] resChain lst (Q ∥ R)
        by(rule resChainI)
    }
    ultimately show ?case by auto
  next
    case(cSym P Q)
    thus ?case by(auto dest: bisimE)
  qed
qed
```

**lemma** *bangRelBisimE*:
  **fixes** $P$ :: *pi*
  **and** $Q$ :: *pi*
  **and** *Rel* :: (*pi* × *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** −
  **from** *A* **show** *?thesis*
  **proof**(*induct*)
    **fix** $P\ Q$
    **assume** $(P, Q) \in Rel$
    **hence** $(Q, P) \in Rel$ **by**(*rule Sym*)
    **thus** $(!Q, !P) \in$ *bangRel Rel* **by**(*rule BRBang*)
  **next**
    **fix** $P\ Q\ R\ T$
    **assume** *RRelT*: $(R, T) \in Rel$
    **assume** *IH*: $(Q, P) \in$ *bangRel Rel*
    **from** *RRelT* **have** $(T, R) \in Rel$ **by**(*rule Sym*)
    **thus** $(T \parallel Q, R \parallel P) \in$ *bangRel Rel* **using** *IH* **by**(*rule BRPar*)
  **next**
    **fix** $P\ Q\ a$
    **assume** $(Q, P) \in$ *bangRel Rel*
    **thus** $(<\nu a>Q, <\nu a>P) \in$ *bangRel Rel* **by**(*rule BRRes*)
  **qed**
**qed**

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

  **assumes** *PBiSimQ*: $P \sim Q$

  **shows** $!P \sim !Q$
**proof** −
  **let** *?X* = *bangRel bisim*
    **from** *PBiSimQ* **have** $(!P, !Q) \in$ *?X* **by**(*rule BRBang*)
    **thus** *?thesis*
    **proof**(*coinduct rule*: *bisimWeakCoinduct*)
      **case**(*cSim bP bQ*)
      **{**
        **fix** $P\ Q$
        **assume** $(P, Q) \in$ *?X*
        **hence** $P \leadsto$[*?X*] $Q$
        **proof**(*induct*)
          **fix** $P\ Q$
          **assume** $P \sim Q$

**thus** $!P \rightsquigarrow[?X]\ !Q$ **using** *bisimE*(*1*) *eqvt*
**by**(*rule Strong-Early-Sim-Pres.bangPres*)
**next**
**fix** *P Q R T*
**assume** *RBiSimT*: $R \sim T$
**assume** *PBangRelQ*: $(P, Q) \in ?X$
**assume** *PSimQ*: $P \rightsquigarrow[?X]\ Q$
**from** *RBiSimT* **have** $R \rightsquigarrow[bisim]\ T$ **by**(*blast dest*: *bisimE*)
**thus** $R \parallel P \rightsquigarrow[?X]\ T \parallel Q$ **using** *PSimQ RBiSimT PBangRelQ BRPar BRRes eqvt eqvtBangRel*
**by**(*blast intro*: *Strong-Early-Sim-Pres.parCompose*)
**next**
**fix** *P Q a*
**assume** $P \rightsquigarrow[?X]\ Q$
**moreover from** *eqvtBangRel eqvt* **have** *eqvt ?X* **by** *blast*
**ultimately show** $<\nu a>P \rightsquigarrow[?X]\ <\nu a>Q$ **using** *BRRes* **by**(*blast intro*: *Strong-Early-Sim-Pres.resPres*)
**qed**
**}**
**with** ‹$(bP, bQ) \in ?X$› **show** *?case* **by** *blast*
**next**
**case**(*cSym bP bQ*)
**thus** *?case* **by**(*metis bangRelSymetric bisimE*)
**qed**
**qed**

**end**

**theory** *Strong-Early-Bisim-Subst-Pres*
**imports** *Strong-Early-Bisim-Subst Strong-Early-Bisim-Pres*
**begin**

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

**assumes** $P \sim^s Q$

**shows** $\tau.(P) \sim^s \tau.(Q)$
**using** *assms*
**by**(*force simp add*: *substClosed-def intro*: *Strong-Early-Bisim-Pres.tauPres*)

**lemma** *inputPres*:
**fixes** $P :: pi$
**and** $Q :: pi$
**and** $a :: name$
**and** $x :: name$

**assumes** $P \sim^s Q$

**shows** $a{<}x{>}.P \sim^s a{<}x{>}.Q$
**proof**(*auto simp add*: *substClosed-def*)
**fix** $\sigma$ :: (*name* × *name*) *list*
**{**
**fix** $P\ Q\ a\ x\ \sigma$
**assume** $P \sim^s Q$
**then have** $P[{<}\sigma{>}] \sim^s Q[{<}\sigma{>}]$ **by**(*rule partUnfold*)
**then have** $\forall\, y.\ (P[{<}\sigma{>}])[x{::=}y] \sim (Q[{<}\sigma{>}])[x{::=}y]$
**apply**(*auto simp add*: *substClosed-def*)
**by**(*erule-tac* $x{=}[(x,\ y)]$ **in** *allE*) *auto*
**moreover assume** $x \sharp \sigma$
**ultimately have** $(a{<}x{>}.P)[{<}\sigma{>}] \sim (a{<}x{>}.Q)[{<}\sigma{>}]$
**by**(*force intro*: *Strong-Early-Bisim-Pres.inputPres*)
**}**
**note** *Goal* = *this*

**obtain** $y$::*name* **where** $y \sharp P$ **and** $y \sharp Q$ **and** $y \sharp \sigma$
**by**(*generate-fresh name*) *auto*
**from** ‹$P \sim^s Q$› **have** $([(x,\ y)] \bullet P) \sim^s ([(x,\ y)] \bullet Q)$ **by**(*rule eqvtI*)
**hence** $(a{<}y{>}.([(x,\ y)] \bullet P))[{<}\sigma{>}] \sim (a{<}y{>}.([(x,\ y)] \bullet Q))[{<}\sigma{>}]$ **using** ‹$y \sharp \sigma$›
**by**(*rule Goal*)
**moreover from** ‹$y \sharp P$› ‹$y \sharp Q$› **have** $a{<}x{>}.P = a{<}y{>}.([(x,\ y)] \bullet P)$ **and**
$a{<}x{>}.Q = a{<}y{>}.([(x,\ y)] \bullet Q)$
**by**(*simp add*: *alphaInput*)+

**ultimately show** $(a{<}x{>}.P)[{<}\sigma{>}] \sim (a{<}x{>}.Q)[{<}\sigma{>}]$ **by** *simp*
**qed**

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

**assumes** $P \sim^s Q$

**shows** $a\{b\}.P \sim^s a\{b\}.Q$
**using** *assms*
**by**(*force simp add*: *substClosed-def intro*: *Strong-Early-Bisim-Pres.outputPres*)

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

**assumes** $P \sim^s Q$

**shows** $[a{\frown}b]P \sim^s [a{\frown}b]Q$
**using** *assms*

**by**(*force simp add*: *substClosed-def intro*: *Strong-Early-Bisim-Pres.matchPres*)

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

  **assumes** $P \sim^s Q$

  **shows** $[a \neq b]P \sim^s [a \neq b]Q$
**using** *assms*
**by**(*force simp add*: *substClosed-def intro*: *Strong-Early-Bisim-Pres.mismatchPres*)

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

  **assumes** $P \sim^s Q$

  **shows** $P \oplus R \sim^s Q \oplus R$
**using** *assms*
**by**(*force simp add*: *substClosed-def intro*: *Strong-Early-Bisim-Pres.sumPres*)

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

  **assumes** $P \sim^s Q$

  **shows** $P \parallel R \sim^s Q \parallel R$
**using** *assms*
**by**(*force simp add*: *substClosed-def intro*: *Strong-Early-Bisim-Pres.parPres*)

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

  **assumes** *PeqQ*: $P \sim^s Q$

  **shows** $<\nu x>P \sim^s <\nu x>Q$
**proof**(*auto simp add*: *substClosed-def*)
  **fix** $s$::(*name* × *name*) *list*

  **have** *Res*: $\bigwedge P\ Q\ x\ s.\ [\![P[<s>] \sim Q[<s>];\ x \sharp s]\!] \Longrightarrow (<\nu x>P)[<s>] \sim (<\nu x>Q)[<s>]$
    **by**(*force intro*: *Strong-Early-Bisim-Pres.resPres*)

**have** $\exists$ *c::name. c* $\sharp$ $(P,\ Q,\ s)$ **by**(*blast intro*: *name-exists-fresh*)
**then obtain** *c::name* **where** *cFreshP*: $c \sharp P$ **and** *cFreshQ*: $c \sharp Q$ **and** *cFreshs*: $c \sharp s$
**by**(*force simp add*: *fresh-prod*)

**from** *PeqQ* **have** $P[<([(x,\ c)] \cdot s)>] \sim Q[<([(x,\ c)] \cdot s)>]$ **by**(*simp add*: *substClosed-def*)
**hence** $([(x,\ c)] \cdot P[<([(x,\ c)] \cdot s)>]) \sim ([(x,\ c)] \cdot Q[<([(x,\ c)] \cdot s)>])$ **by**(*rule Strong-Early-Bisim.bisimClosed*)
**hence** $([(x,\ c)] \cdot P)[<s>] \sim ([(x,\ c)] \cdot Q)[<s>]$ **by** *simp*
**hence** $(<\nu c>([(x,\ c)] \cdot P))[<s>] \sim (<\nu c>([(x,\ c)] \cdot Q))[<s>]$ **using** *cFreshs* **by**(*rule Res*)

**moreover from** *cFreshP cFreshQ* **have** $<\nu x>P = <\nu c>([(x,\ c)] \cdot P)$ **and** $<\nu x>Q = <\nu c>([(x,\ c)] \cdot Q)$
**by**(*simp add*: *alphaRes*)+

**ultimately show** $(<\nu x>P)[<s>] \sim (<\nu x>Q)[<s>]$ **by** *simp*
**qed**

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

**assumes** $P \sim^s Q$

**shows** $!P \sim^s !Q$
**using** *assms*
**by**(*force simp add*: *substClosed-def intro*: *Strong-Early-Bisim-Pres.bangPres*)

**end**

**theory** *Early-Tau-Chain*
**imports** *Early-Semantics*
**begin**

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

**lemma** *tauActTauChain*:
**fixes** $P$ :: *pi*
**and** $P'$ :: *pi*

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

**shows** $P \Longrightarrow_\tau P'$
**using** *assms*
**by** *auto*

**lemma** *tauChainAddTau*[*intro*]:
  **fixes** $P$ :: *pi*
  **and** $P'$ :: *pi*
  **and** $P''$ :: *pi*

  **shows** $P \Longrightarrow_\tau P' \Longrightarrow P' \longmapsto \tau \prec P'' \Longrightarrow P \Longrightarrow_\tau P''$
  **and** $P \longmapsto \tau \prec P' \Longrightarrow P' \Longrightarrow_\tau P'' \Longrightarrow P \Longrightarrow_\tau P''$
**by**(*auto dest*: *tauActTauChain*)

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

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

  **shows** $F\ P'$
**using** *assms*
**by**(*drule-tac rtrancl-induct*) *auto*

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

  **assumes** $P \Longrightarrow_\tau P'$

  **shows** $(perm \cdot P) \Longrightarrow_\tau (perm \cdot P')$
**using** *assms*
**proof**(*induct rule*: *tauChainInduct*)
  **case** *id*
  **thus** *?case* **by** *simp*
**next**
  **case**(*ih* $P''$ $P'''$)
  **have** $P \Longrightarrow_\tau P''$ **and** $P'' \longmapsto \tau \prec P'''$ **by** *fact+*
  **hence** $(perm \cdot P'') \longmapsto \tau \prec (perm \cdot P''')$ **by**(*drule-tac TransitionsEarly.eqvt*) *auto*
  **moreover have** $(perm \cdot P) \Longrightarrow_\tau (perm \cdot P'')$ **by** *fact*
  **ultimately show** *?case* **by**(*force dest*: *tauActTauChain*)
**qed**

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

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

**shows** $P \Longrightarrow_\tau P'$
**proof** −
**have** *rev perm* $\cdot$ (*perm* $\cdot$ $P$) = $P$ **by**(*simp add*: *pt-rev-pi*[*OF pt-name-inst*, *OF at-name-inst*])
**moreover have** *rev perm* $\cdot$ (*perm* $\cdot$ $P'$) = $P'$ **by**(*simp add*: *pt-rev-pi*[*OF pt-name-inst*, *OF at-name-inst*])
**ultimately show** *?thesis* **using** *assms*
**by**(*drule-tac perm=rev perm* **in** *eqvtChainI*, *simp*)
**qed**

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

**shows** $P \Longrightarrow_\tau P' = (perm \cdot P) \Longrightarrow_\tau (perm \cdot P')$
**by**(*blast intro*: *eqvtChainE eqvtChainI*)

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

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

**shows** $x \sharp P'$
**using** *assms*
**proof**(*induct rule*: *tauChainInduct*)
**case** *id*
**thus** *?case* **by** *simp*
**next**
**case**(*ih* $P'$ $P''$)
**have** $x \sharp P$ **and** $x \sharp P \Longrightarrow x \sharp P'$ **by** *fact+*
**hence** $x \sharp P'$ **by** *simp*
**moreover have** $P' \longmapsto \tau \prec P''$ **by** *fact*
**ultimately show** *?case* **by**(*force intro*: *freshTransition*)
**qed**

**lemma** *matchChain*:
**fixes** $b$ :: *name*
**and** $P$ :: *pi*
**and** $P'$ :: *pi*

**assumes** $P \Longrightarrow_\tau P'$
**and** $P \neq P'$

**shows** $[b{\frown}b]P \Longrightarrow_\tau P'$

**using** *assms*
**proof**(*induct rule*: *tauChainInduct*)
  **case** *id*
  **thus** *?case* **by** *simp*
**next**
  **case**(*ih* $P''$ $P'''$)
  **have** $P''TransP'''$: $P'' \longmapsto\tau \prec P'''$ **by** *fact*
  **show** $[b\frown b]P \Longrightarrow_\tau P'''$
  **proof**(*cases* $P = P''$)
    **assume** $P=P''$
    **moreover with** $P''TransP'''$ **have** $[b\frown b]P \longmapsto\tau \prec P'''$ **by**(*force intro*: *Match*)
    **thus** $[b\frown b]P \Longrightarrow_\tau P'''$ **by**(*rule tauActTauChain*)
  **next**
    **assume** $P \neq P''$
    **moreover have** $P \neq P'' \Longrightarrow [b\frown b]P \Longrightarrow_\tau P''$ **by** *fact*
    **ultimately show** $[b\frown b]P \Longrightarrow_\tau P'''$ **using** $P''TransP'''$ **by**(*blast*)
  **qed**
**qed**

**lemma** *mismatchChain*:
  **fixes** $a$ :: *name*
  **and** $b$ :: *name*
  **and** $P$ :: *pi*
  **and** $P'$ :: *pi*

  **assumes** *PChain*: $P \Longrightarrow_\tau P'$
  **and** *aineqb*: $a \neq b$
  **and** *PineqP'*: $P \neq P'$

  **shows** $[a\neq b]P \Longrightarrow_\tau P'$
**proof** −
  **from** *PChain PineqP'* **show** *?thesis*
  **proof**(*induct rule*: *tauChainInduct*)
    **case** *id*
    **thus** *?case* **by** *simp*
  **next**
    **case**(*ih* $P''$ $P'''$)
    **have** $P''TransP'''$: $P'' \longmapsto\tau \prec P'''$ **by** *fact*
    **show** $[a\neq b]P \Longrightarrow_\tau P'''$
    **proof**(*cases* $P = P''$)
      **assume** $P=P''$
      **moreover with** *aineqb* $P''TransP'''$ **have** $[a\neq b]P \longmapsto\tau \prec P'''$ **by**(*force intro*:
*Mismatch*)
      **thus** $[a\neq b]P \Longrightarrow_\tau P'''$ **by**(*rule tauActTauChain*)
    **next**
      **assume** $P \neq P''$
      **moreover have** $P \neq P'' \Longrightarrow [a\neq b]P \Longrightarrow_\tau P''$ **by** *fact*
      **ultimately show** $[a\neq b]P \Longrightarrow_\tau P'''$ **using** $P''TransP'''$ **by**(*blast*)
    **qed**

**qed**
**qed**

**lemma** *sum1Chain*:
**fixes** $P$ :: *pi*
**and** $P'$ :: *pi*
**and** $Q$ :: *pi*

**assumes** $P \Longrightarrow_\tau P'$
**and** $P \neq P'$

**shows** $P \oplus Q \Longrightarrow_\tau P'$
**using** *assms*
**proof**(*induct rule*: *tauChainInduct*)
**case** *id*
**thus** *?case* **by** *simp*
**next**
**case**(*ih* $P''$ $P'''$)
**have** $P''$*TransP*$'''$: $P'' \longmapsto\tau \prec P'''$ **by** *fact*
**show** $P \oplus Q \Longrightarrow_\tau P'''$
**proof**(*cases* $P = P''$)
**assume** $P=P''$
**moreover with** $P''$*TransP*$'''$ **have** $P \oplus Q \longmapsto\tau \prec P'''$ **by**(*force intro*: *Sum1*)
**thus** $P \oplus Q \Longrightarrow_\tau P'''$ **by**(*force intro*: *tauActTauChain*)
**next**
**assume** $P \neq P''$
**moreover have** $P \neq P'' \Longrightarrow P \oplus Q \Longrightarrow_\tau P''$ **by** *fact*
**ultimately show** $P \oplus Q \Longrightarrow_\tau P'''$ **using** $P''$*TransP*$'''$ **by**(*force dest*: *tauAct-TauChain*)
**qed**
**qed**

**lemma** *sum2Chain*:
**fixes** $P$ :: *pi*
**and** $Q$ :: *pi*
**and** $Q'$ :: *pi*

**assumes** $Q \Longrightarrow_\tau Q'$
**and** $Q \neq Q'$

**shows** $P \oplus Q \Longrightarrow_\tau Q'$
**using** *assms*
**proof**(*induct rule*: *tauChainInduct*)
**case** *id*
**thus** *?case* **by** *simp*
**next**
**case**(*ih* $Q''$ $Q'''$)
**have** $Q''$*TransQ*$'''$: $Q'' \longmapsto\tau \prec Q'''$ **by** *fact*
**show** $P \oplus Q \Longrightarrow_\tau Q'''$

**proof**(*cases* $Q = Q''$)
  **assume** $Q{=}Q''$
  **moreover with** $Q''TransQ'''$ **have** $P \oplus Q \longmapsto\tau \prec Q'''$ **by**(*force intro*: *Sum2*)
  **thus** $P \oplus Q \Longrightarrow_\tau Q'''$ **by**(*force intro*: *tauActTauChain*)
**next**
  **assume** $Q \neq Q''$
  **moreover have** $Q \neq Q'' \Longrightarrow P \oplus Q \Longrightarrow_\tau Q''$ **by** *fact*
  **ultimately show** $P \oplus Q \Longrightarrow_\tau Q'''$ **using** $Q''TransQ'''$ **by** *blast*
**qed**
**qed**

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

  **assumes** $P \Longrightarrow_\tau P'$

  **shows** $P \parallel Q \Longrightarrow_\tau P' \parallel Q$
**using** *assms*
**proof**(*induct rule*: *tauChainInduct*)
  **case** *id*
  **thus** *?case* **by** *simp*
**next**
  **case**(*ih* $P''$ $P'$)
  **have** $P''TransP'$: $P'' \longmapsto\tau \prec P'$ **by** *fact*
  **have** *IH*: $P \parallel Q \Longrightarrow_\tau P'' \parallel Q$ **by** *fact*

  **have** $P'' \parallel Q \longmapsto\tau \prec P' \parallel Q$ **using** $P''TransP'$ **by**(*force intro*: *Par1F*)
  **thus** $P \parallel Q \Longrightarrow_\tau P' \parallel Q$ **using** *IH* **by**(*force dest*: *tauActTauChain*)
**qed**

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

  **assumes** $Q \Longrightarrow_\tau Q'$

  **shows** $P \parallel Q \Longrightarrow_\tau P \parallel Q'$
**using** *assms*
**proof**(*induct rule*: *tauChainInduct*)
  **case** *id*
  **thus** *?case* **by** *simp*
**next**
  **case**(*ih* $Q''$ $Q'$)
  **have** $Q''TransQ'$: $Q'' \longmapsto\tau \prec Q'$ **by** *fact*
  **have** *IH*: $P \parallel Q \Longrightarrow_\tau P \parallel Q''$ **by** *fact*

**have** $P \parallel Q'' \longmapsto\tau \prec P \parallel Q'$ **using** *Q''TransQ'* **by**(*force intro*: *Par2F*)
**thus** $P \parallel Q \Longrightarrow_\tau P \parallel Q'$ **using** *IH* **by**(*force dest*: *tauActTauChain*)
**qed**

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

**assumes** $P \Longrightarrow_\tau P'$
**and** $Q \Longrightarrow_\tau Q'$

**shows** $P \parallel Q \Longrightarrow_\tau P' \parallel Q'$
**proof** −
**from** ‹$P \Longrightarrow_\tau P'$› **have** $P \parallel Q \Longrightarrow_\tau P' \parallel Q$ **by**(*rule Par1Chain*)
**moreover from** ‹$Q \Longrightarrow_\tau Q'$› **have** $P' \parallel Q \Longrightarrow_\tau P' \parallel Q'$ **by**(*rule Par2Chain*)
**ultimately show** *?thesis* **by** *auto*
**qed**

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

**assumes** $P \Longrightarrow_\tau P'$

**shows** $<\nu a>P \Longrightarrow_\tau <\nu a>P'$
**using** *assms*
**proof**(*induct rule*: *tauChainInduct*)
**case** *id*
**thus** *?case* **by** *simp*
**next**
**case**(*ih* $P''$ $P'''$)
**have** $P'' \longmapsto\tau \prec P'''$ **by** *fact*
**hence** $<\nu a>P'' \longmapsto\tau \prec <\nu a>P'''$ **by**(*force intro*: *ResF*)
**moreover have** $<\nu a>P \Longrightarrow_\tau <\nu a>P''$ **by** *fact*
**ultimately show** *?case* **by**(*force dest*: *tauActTauChain*)
**qed**

**lemma** *substChain*:
**fixes** $P$ :: *pi*
**and** $x$ :: *name*
**and** $b$ :: *name*
**and** $P'$ :: *pi*

**assumes** *PTrans*: $P[x::=b] \Longrightarrow_\tau P'$

**shows** $P[x::=b] \Longrightarrow_\tau P'[x::=b]$

**proof**(*cases* $x$=$b$)
  **assume** $x = b$
  **with** *PTrans* **show** *?thesis* **by** *simp*
**next**
  **assume** $x \neq b$
  **hence** $x \sharp P[x::=b]$ **by**(*simp add*: *fresh-fact2*)
  **with** *PTrans* **have** $x \sharp P'$ **by**(*force intro*: *freshChain*)
  **hence** $P' = P'[x::=b]$ **by**(*simp add*: *forget*)
  **with** *PTrans* **show** *?thesis* **by** *simp*
**qed**

**lemma** *bangChain*:
  **fixes** $P$ :: *pi*
  **and** $P'$ :: *pi*

  **assumes** *PTrans*: $P \parallel !P \Longrightarrow_\tau P'$
  **and** $P'$*ineq*: $P' \neq P \parallel !P$

  **shows** $!P \Longrightarrow_\tau P'$
**using** *assms*
**proof**(*induct rule*: *tauChainInduct*)
  **case** *id*
  **thus** *?case* **by** *simp*
**next**
  **case**(*ih* $P'$ $P''$)
  **show** *?case*
  **proof**(*cases* $P' = P \parallel !P$)
    **case** *True*
    **from** ‹$P' \longmapsto\tau \prec P''$› ‹$P' = P \parallel !P$› **have** $!P \longmapsto\tau \prec P''$ **by**(*blast intro*: *Bang*)
    **thus** *?thesis* **by** *auto*
  **next**
    **case** *False*
    **from** ‹$P' \neq P \parallel !P$› **have** $!P \Longrightarrow_\tau P'$ **by**(*rule ih*)
    **with** ‹$P' \longmapsto\tau \prec P''$› **show** *?thesis* **by**(*auto dest*: *tauActTauChain*)
  **qed**
**qed**

**end**

**theory** *Weak-Early-Step-Semantics*
  **imports** *Early-Tau-Chain*
**begin**

**lemma** *inputSupportDerivative*:
  **assumes** $P \longmapsto a{<}x{>} \prec P'$

  **shows** $(supp\ P') - \{x\} \subseteq supp\ P$
**using** *assms*
**apply**(*nominal-induct rule*: *inputInduct*)

**apply**(*auto simp add*: *pi.supp abs-supp supp-atm*)
**apply**(*rule ccontr*)
**apply**(*simp add*: *fresh-def*[*symmetric*])
**apply**(*drule-tac fresh-fact1*)
**apply**(*rotate-tac 4*)
**apply** *assumption*
**apply**(*simp add*: *fresh-def*)
**apply** *force*
**apply**(*case-tac* $x \sharp P$)
**apply**(*drule-tac fresh-fact1*)
**apply**(*rotate-tac 2*)
**apply** *assumption*
**apply**(*simp add*: *fresh-def*)
**apply** *force*
**apply**(*rotate-tac 2*)
**apply**(*drule-tac fresh-fact2*)
**apply**(*simp add*: *fresh-def*)
**by** *force*

**lemma** *outputSupportDerivative*:
  **fixes** $P$ :: *pi*
  **and** $a$ :: *name*
  **and** $b$ :: *name*
  **and** $P'$ :: *pi*

  **assumes** $P \longmapsto a[b] \prec P'$

  **shows** $(supp\ P') \subseteq ((supp\ P)::name\ set)$
**using** *assms*
**by**(*nominal-induct rule*: *outputInduct*) (*auto simp add*: *pi.supp abs-supp*)

**lemma** *boundOutputSupportDerivative*:
  **assumes** $P \longmapsto a<\nu x> \prec P'$
  **and** $x \sharp P$

  **shows** $(supp\ P') - \{x\} \subseteq supp\ P$
**using** *assms*
**by**(*nominal-induct rule*: *boundOutputInduct*) (*auto simp add*: *pi.supp abs-supp supp-atm dest*: *outputSupportDerivative*)

**lemma** *tauSupportDerivative*:

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

  **shows** $((supp\ P')::name\ set) \subseteq supp\ P$
**using** *assms*
**proof**(*nominal-induct rule*: *tauInduct*)
  **case**(*Tau P*)
  **thus** *?case* **by**(*force simp add*: *pi.supp*)

**next**
**case**(*Match P*)
**thus** *?case* **by**(*force simp add*: *pi.supp*)
**next**
**case**(*Mismatch P*)
**thus** *?case* **by**(*force simp add*: *pi.supp*)
**next**
**case**(*Sum1 P*)
**thus** *?case* **by**(*force simp add*: *pi.supp*)
**next**
**case**(*Sum2 P*)
**thus** *?case* **by**(*force simp add*: *pi.supp*)
**next**
**case**(*Par1 P*)
**thus** *?case* **by**(*force simp add*: *pi.supp*)
**next**
**case**(*Par2 P*)
**thus** *?case* **by**(*force simp add*: *pi.supp*)
**next**
**case**(*Comm1 P a b P′ Q Q′*)
**from** $\langle P \longmapsto a{<}b{>} \prec P'\rangle$ **have** $(supp\ P') - \{b\} \subseteq supp\ P$ **by**(*rule inputSupportDerivative*)
**moreover from** $\langle Q \longmapsto a[b] \prec Q'\rangle$ **have** $((supp\ Q')::name\ set) \subseteq supp\ Q$ **by**(*rule outputSupportDerivative*)
**moreover from** $\langle Q \longmapsto a[b] \prec Q'\rangle$ **have** $b \in supp\ Q$
**by**(*nominal-induct rule*: *outputInduct*) (*auto simp add*: *pi.supp abs-supp supp-atm*)
**ultimately show** *?case* **by**(*auto simp add*: *pi.supp*)
**next**
**case**(*Comm2 P a b P′ Q Q′*)
**from** $\langle P \longmapsto a[b] \prec P'\rangle$ **have** $((supp\ P')::name\ set) \subseteq supp\ P$ **by**(*rule outputSupportDerivative*)
**moreover from** $\langle Q \longmapsto a{<}b{>} \prec Q'\rangle$ **have** $(supp\ Q') - \{b\} \subseteq supp\ Q$ **by**(*rule inputSupportDerivative*)
**moreover from** $\langle P \longmapsto a[b] \prec P'\rangle$ **have** $b \in supp\ P$
**by**(*nominal-induct rule*: *outputInduct*) (*auto simp add*: *pi.supp abs-supp supp-atm*)
**ultimately show** *?case* **by**(*auto simp add*: *pi.supp*)
**next**
**case**(*Close1 P a x P′ Q Q′*)
**thus** *?case* **by**(*auto dest*: *inputSupportDerivative boundOutputSupportDerivative simp add*: *abs-supp pi.supp*)
**next**
**case**(*Close2 P a x P′ Q Q′*)
**thus** *?case* **by**(*auto dest*: *inputSupportDerivative boundOutputSupportDerivative simp add*: *abs-supp pi.supp*)
**next**
**case**(*Res P P′ x*)
**thus** *?case* **by**(*force simp add*: *pi.supp abs-supp*)
**next**
**case**(*Bang P P′*)

**thus** *?case* **by**(*force simp add*: *pi.supp*)
**qed**

**lemma** *tauChainSupportDerivative*:
**fixes** $P$ :: *pi*
**and** $P'$ :: *pi*

**assumes** $P \Longrightarrow_\tau P'$

**shows** $((supp\ P')$::*name set*$) \subseteq (supp\ P)$
**using** *assms*
**by**(*induct rule*: *tauChainInduct*) (*auto dest*: *tauSupportDerivative*)

**definition** *outputTransition* :: *pi* ⇒ *name* ⇒ *name* ⇒ *pi* ⇒ *bool* (‹- ⟹-<ν-> ≺ -› [*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* ⇒ *freeRes*⇒ *pi* ⇒ *bool* (‹- ⟹- ≺ -› [*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** $\alpha$ :: *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** $[\![P \Longrightarrow_\tau P'''; P''' \longmapsto a{<}\nu x{>} \prec P''; P'' \Longrightarrow_\tau P']\!] \Longrightarrow P \Longrightarrow a{<}\nu x{>} \prec P'$
**by**(*auto simp add*: *outputTransition-def freeTransition-def*)

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

**shows** $P \Longrightarrow \alpha \prec P' \Longrightarrow (\exists P''\ P'''.\ P \Longrightarrow_\tau P'' \wedge P'' \longmapsto \alpha \prec P''' \wedge P''' \Longrightarrow_\tau P')$
**and** $P \Longrightarrow a{<}\nu x{>} \prec P' \Longrightarrow \exists P''\ P'''.\ P \Longrightarrow_\tau P''' \wedge P''' \longmapsto a{<}\nu x{>} \prec P'' \wedge P'' \Longrightarrow_\tau P'$
**by**(*auto simp add*: *outputTransition-def freeTransition-def*)

**lemma** *weakTransitionAlpha*:

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

**assumes** *PTrans*: $P \Longrightarrow a{<}\nu x{>} \prec P'$
**and** $y \sharp P$

**shows** $P \Longrightarrow a{<}\nu y{>} \prec ([(x, y)] \cdot P')$
**proof**(*cases* $y=x$)
**case** *True*
**with** *PTrans* **show** *?thesis* **by** *simp*
**next**
**case** *False*
**from** *PTrans* **obtain** $P''$ $P'''$ **where** *PChain*: $P \Longrightarrow_\tau P'''$
**and** $P'''$*Trans*: $P''' \longmapsto a{<}\nu x{>} \prec P''$
**and** $P''$*Chain*: $P'' \Longrightarrow_\tau P'$
**by**(*force dest*: *transitionE*)
**note** *PChain*
**moreover from** *PChain* ‹$y \sharp P$› **have** $y \sharp P'''$ **by**(*rule freshChain*)
**with** $P'''$*Trans* **have** $y \sharp P''$ **using** ‹$y \neq x$› **by**(*rule freshTransition*)
**with** $P'''$*Trans* **have** $P''' \longmapsto a{<}\nu y{>} \prec ([(x, y)] \cdot P'')$ **by**(*simp add*: *alphaBoundOutput name-swap*)
**moreover from** $P''$*Chain* **have** $([(x, y)] \cdot P'') \Longrightarrow_\tau ([(x, y)] \cdot P')$
**by**(*rule eqvtChainI*)
**ultimately show** *?thesis* **by**(*rule transitionI*)
**qed**

**lemma** *singleActionChain*:
**fixes** $P$ :: *pi*
**and** *Rs* :: *residual*

**shows** $P \longmapsto a{<}\nu x{>} \prec P' \Longrightarrow P \Longrightarrow a{<}\nu x{>} \prec P'$
**and** $P \longmapsto \alpha \prec P' \Longrightarrow P \Longrightarrow \alpha \prec P'$
**proof** −
**have** $P \Longrightarrow_\tau P$ **by** *simp*
**moreover assume** $P \longmapsto a{<}\nu x{>} \prec P'$
**moreover have** $P' \Longrightarrow_\tau P'$ **by** *simp*
**ultimately show** $P \Longrightarrow a{<}\nu x{>} \prec P'$
**by**(*rule transitionI*)
**next**
**have** $P \Longrightarrow_\tau P$ **by** *simp*
**moreover assume** $P \longmapsto \alpha \prec P'$
**moreover have** $P' \Longrightarrow_\tau P'$ **by** *simp*
**ultimately show** $P \Longrightarrow \alpha \prec P'$
**by**(*rule transitionI*)
**qed**

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

**shows** $\tau.(P) \Longrightarrow \tau \prec \; P$
**proof** −
**have** $\tau.(P) \Longrightarrow_\tau \tau.(P)$ **by** *simp*
**moreover have** $\tau.(P) \longmapsto \tau \prec P$ **by**(*rule Early-Semantics.Tau*)
**moreover have** $P \Longrightarrow_\tau P$ **by** *simp*
**ultimately show** *?thesis* **by**(*rule transitionI*)
**qed**

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

**shows** $a{<}x{>}.P \Longrightarrow a{<}u{>} \prec P[x{::=}u]$
**proof** −
**have** $a{<}x{>}.P \Longrightarrow_\tau a{<}x{>}.P$ **by** *simp*
**moreover have** $a{<}x{>}.P \longmapsto a{<}u{>} \prec P[x{::=}u]$ **by**(*rule Early-Semantics.Input*)
**moreover have** $P[x{::=}u] \Longrightarrow_\tau P[x{::=}u]$ **by** *simp*
**ultimately show** *?thesis* **by**(*rule transitionI*)
**qed**

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

**shows** $a\{b\}.P \Longrightarrow a[b] \prec P$
**proof** −
**have** $a\{b\}.P \Longrightarrow_\tau a\{b\}.P$ **by** *simp*
**moreover have** $a\{b\}.P \longmapsto a[b] \prec P$ **by**(*rule Early-Semantics.Output*)
**moreover have** $P \Longrightarrow_\tau P$ **by** *simp*
**ultimately show** *?thesis* **by**(*rule transitionI*)
**qed**

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

**shows** $P \Longrightarrow b{<}\nu x{>} \prec P' \Longrightarrow [a \frown a]P \Longrightarrow b{<}\nu x{>} \prec P'$
**and** $P \Longrightarrow \alpha \prec P' \Longrightarrow [a \frown a]P \Longrightarrow \alpha \prec P'$
**proof** −

**assume** $P \Longrightarrow b{<}\nu x{>} \prec P'$
**then obtain** $P''$ $P'''$ **where** *PChain*: $P \Longrightarrow_\tau P'''$
**and** $P'''$*Trans*: $P''' \longmapsto b{<}\nu x{>} \prec P''$
**and** $P''$*Chain*: $P'' \Longrightarrow_\tau P'$
**by**(*force dest*: *transitionE*)
**show** $[a \frown a]P \Longrightarrow b{<}\nu x{>} \prec P'$
**proof**(*cases* $P = P'''$)
**case** *True*
**have** $[a \frown a]P \Longrightarrow_\tau [a \frown a]P$ **by** *simp*
**moreover from** ‹$P = P'''$› $P'''$*Trans* **have** $[a \frown a]P \longmapsto b{<}\nu x{>} \prec P''$
**by**(*rule-tac Early-Semantics.Match*) *auto*
**ultimately show** *?thesis* **using** $P''$*Chain* **by**(*rule transitionI*)
**next**
**case** *False*
**from** *PChain* ‹$P \neq P'''$› **have** $[a \frown a]P \Longrightarrow_\tau P'''$ **by**(*rule matchChain*)
**thus** *?thesis* **using** $P'''$*Trans* $P''$*Chain* **by**(*rule transitionI*)
**qed**
**next**
**assume** $P \Longrightarrow \alpha \prec P'$
**then obtain** $P''$ $P'''$ **where** *PChain*: $P \Longrightarrow_\tau P'''$
**and** $P'''$*Trans*: $P''' \longmapsto \alpha \prec P''$
**and** $P''$*Chain*: $P'' \Longrightarrow_\tau P'$
**by**(*force dest*: *transitionE*)
**show** $[a \frown a]P \Longrightarrow \alpha \prec P'$
**proof**(*cases* $P = P'''$)
**case** *True*
**have** $[a \frown a]P \Longrightarrow_\tau [a \frown a]P$ **by** *simp*
**moreover from** ‹$P = P'''$› $P'''$*Trans* **have** $[a \frown a]P \longmapsto \alpha \prec P''$
**by**(*rule-tac Early-Semantics.Match*) *auto*
**ultimately show** *?thesis* **using** $P''$*Chain* **by**(*rule transitionI*)
**next**
**case** *False*
**from** *PChain* ‹$P \neq P'''$› **have** $[a \frown a]P \Longrightarrow_\tau P'''$ **by**(*rule matchChain*)
**thus** *?thesis* **using** $P'''$*Trans* $P''$*Chain* **by**(*rule transitionI*)
**qed**
**qed**

**lemma** *Mismatch*:
**fixes** $P$ :: *pi*
**and** $c$ :: *name*
**and** $x$ :: *name*
**and** $a$ :: *name*
**and** $b$ :: *name*
**and** $P'$ :: *pi*
**and** $\alpha$ :: *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** −

**assume** $P \Longrightarrow c{<}\nu x{>} \prec P'$
**then obtain** $P''$ $P'''$ **where** *PChain*: $P \Longrightarrow_\tau P'''$
**and** $P'''$*Trans*: $P''' \longmapsto c{<}\nu x{>} \prec P''$
**and** $P''$*Chain*: $P'' \Longrightarrow_\tau P'$
**by**(*force dest*: *transitionE*)
**assume** $a \neq b$
**show** $[a{\neq}b]P \Longrightarrow c{<}\nu x{>} \prec P'$
**proof**(*cases* $P = P'''$)
**case** *True*
**have** $[a{\neq}b]P \Longrightarrow_\tau [a{\neq}b]P$ **by** *simp*
**moreover from** ‹$P = P'''$› ‹$a \neq b$› $P'''$*Trans* **have** $[a{\neq}b]P \longmapsto c{<}\nu x{>} \prec P''$
**by**(*rule-tac Early-Semantics.Mismatch*) *auto*
**ultimately show** *?thesis* **using** $P''$*Chain* **by**(*rule transitionI*)
**next**
**case** *False*
**from** *PChain* ‹$a \neq b$› ‹$P \neq P'''$› **have** $[a{\neq}b]P \Longrightarrow_\tau P'''$ **by**(*rule mismatchChain*)
**thus** *?thesis* **using** $P'''$*Trans* $P''$*Chain* **by**(*rule transitionI*)
**qed**
**next**
**assume** $P \Longrightarrow \alpha \prec P'$
**then obtain** $P''$ $P'''$ **where** *PChain*: $P \Longrightarrow_\tau P'''$
**and** $P'''$*Trans*: $P''' \longmapsto \alpha \prec P''$
**and** $P''$*Chain*: $P'' \Longrightarrow_\tau P'$
**by**(*force dest*: *transitionE*)
**assume** $a \neq b$
**show** $[a{\neq}b]P \Longrightarrow \alpha \prec P'$
**proof**(*cases* $P = P'''$)
**case** *True*
**have** $[a{\neq}b]P \Longrightarrow_\tau [a{\neq}b]P$ **by** *simp*
**moreover from** ‹$P = P'''$› ‹$a \neq b$› $P'''$*Trans* **have** $[a{\neq}b]P \longmapsto \alpha \prec P''$
**by**(*rule-tac Early-Semantics.Mismatch*) *auto*
**ultimately show** *?thesis* **using** $P''$*Chain* **by**(*rule transitionI*)
**next**
**case** *False*
**from** *PChain* ‹$a \neq b$› ‹$P \neq P'''$› **have** $[a{\neq}b]P \Longrightarrow_\tau P'''$ **by**(*rule mismatchChain*)
**thus** *?thesis* **using** $P'''$*Trans* $P''$*Chain* **by**(*rule transitionI*)
**qed**
**qed**

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

**assumes** *PTrans*: $P \Longrightarrow a[b] \prec P'$
**and** $a \neq b$

**shows** ${<}\nu b{>}P \Longrightarrow a{<}\nu b{>} \prec P'$

**proof** −
  **from** *PTrans* **obtain** $P''$ $P'''$ **where** *PChain*: $P \Longrightarrow_\tau P'''$
    **and** $P'''$*Trans*: $P''' \longmapsto a[b] \prec P''$
    **and** $P''$*Chain*: $P'' \Longrightarrow_\tau P'$
    **by**(*force dest*: *transitionE*)
  **from** *PChain* **have** $<\nu b>P \Longrightarrow_\tau <\nu b>P'''$ **by**(*rule ResChain*)
  **moreover from** $P'''$*Trans* $\langle a \neq b\rangle$ **have** $<\nu b>P''' \longmapsto a<\nu b> \prec P''$ **by**(*rule Open*)
  **ultimately show** *?thesis* **using** $P''$*Chain* **by**(*rule transitionI*)
**qed**

**lemma** *Sum1*:
  **fixes** $P$ :: *pi*
  **and** $a$ :: *name*
  **and** $x$ :: *name*
  **and** $P'$ :: *pi*
  **and** $Q$ :: *pi*
  **and** $\alpha$ :: *freeRes*

  **shows** $P \Longrightarrow a<\nu x> \prec P' \Longrightarrow P \oplus Q \Longrightarrow a<\nu x> \prec P'$
  **and** $P \Longrightarrow \alpha \prec P' \Longrightarrow P \oplus Q \Longrightarrow \alpha \prec P'$
**proof** −
  **assume** $P \Longrightarrow a<\nu x> \prec P'$
  **then obtain** $P''$ $P'''$ **where** *PChain*: $P \Longrightarrow_\tau P'''$
    **and** $P'''$*Trans*: $P''' \longmapsto a<\nu x> \prec P''$
    **and** $P''$*Chain*: $P'' \Longrightarrow_\tau P'$
    **by**(*force dest*: *transitionE*)
  **show** $P \oplus Q \Longrightarrow a<\nu x> \prec P'$
  **proof**(*cases* $P = P'''$)
    **case** *True*
    **have** $P \oplus Q \Longrightarrow_\tau P \oplus Q$ **by** *simp*
    **moreover from** $P'''$*Trans* $\langle P = P'''\rangle$ **have** $P \oplus Q \longmapsto a<\nu x> \prec P''$ **by**(*blast intro*: *Sum1*)
    **ultimately show** *?thesis* **using** $P''$*Chain* **by**(*rule transitionI*)
  **next**
    **case** *False*
    **from** *PChain* $\langle P \neq P'''\rangle$ **have** $P \oplus Q \Longrightarrow_\tau P'''$ **by**(*rule sum1Chain*)
    **thus** *?thesis* **using** $P'''$*Trans* $P''$*Chain* **by**(*rule transitionI*)
  **qed**
**next**
  **assume** $P \Longrightarrow \alpha \prec P'$
  **then obtain** $P''$ $P'''$ **where** *PChain*: $P \Longrightarrow_\tau P'''$
    **and** $P'''$*Trans*: $P''' \longmapsto \alpha \prec P''$
    **and** $P''$*Chain*: $P'' \Longrightarrow_\tau P'$
    **by**(*force dest*: *transitionE*)
  **show** $P \oplus Q \Longrightarrow \alpha \prec P'$
  **proof**(*cases* $P = P'''$)
    **case** *True*
    **have** $P \oplus Q \Longrightarrow_\tau P \oplus Q$ **by** *simp*

**moreover from** $P'''Trans$ ‹$P = P'''$› **have** $P \oplus Q \longmapsto\alpha \prec P''$ **by**(*blast intro*: *Sum1*)
**ultimately show** *?thesis* **using** $P''Chain$ **by**(*rule transitionI*)
**next**
**case** *False*
**from** *PChain* ‹$P \neq P'''$› **have** $P \oplus Q \Longrightarrow_\tau P'''$ **by**(*rule sum1Chain*)
**thus** *?thesis* **using** $P'''Trans$ $P''Chain$ **by**(*rule transitionI*)
**qed**
**qed**

**lemma** *Sum2*:
**fixes** $Q$ :: *pi*
**and** $a$ :: *name*
**and** $x$ :: *name*
**and** $Q'$ :: *pi*
**and** $P$ :: *pi*
**and** $\alpha$ :: *freeRes*

**shows** $Q \Longrightarrow a{<}\nu x{>} \prec Q' \Longrightarrow P \oplus Q \Longrightarrow a{<}\nu x{>} \prec Q'$
**and** $Q \Longrightarrow\alpha \prec Q' \Longrightarrow P \oplus Q \Longrightarrow\alpha \prec Q'$
**proof** −
**assume** $Q \Longrightarrow a{<}\nu x{>} \prec Q'$
**then obtain** $Q''$ $Q'''$ **where** *QChain*: $Q \Longrightarrow_\tau Q'''$
**and** $Q'''Trans$: $Q''' \longmapsto a{<}\nu x{>} \prec Q''$
**and** $Q''Chain$: $Q'' \Longrightarrow_\tau Q'$
**by**(*force dest*: *transitionE*)
**show** $P \oplus Q \Longrightarrow a{<}\nu x{>} \prec Q'$
**proof**(*cases* $Q = Q'''$)
**case** *True*
**have** $P \oplus Q \Longrightarrow_\tau P \oplus Q$ **by** *simp*
**moreover from** $Q'''Trans$ ‹$Q = Q'''$› **have** $P \oplus Q \longmapsto a{<}\nu x{>} \prec Q''$ **by**(*blast intro*: *Sum2*)
**ultimately show** *?thesis* **using** $Q''Chain$ **by**(*rule transitionI*)
**next**
**case** *False*
**from** *QChain* ‹$Q \neq Q'''$› **have** $P \oplus Q \Longrightarrow_\tau Q'''$ **by**(*rule sum2Chain*)
**thus** *?thesis* **using** $Q'''Trans$ $Q''Chain$ **by**(*rule transitionI*)
**qed**
**next**
**assume** $Q \Longrightarrow\alpha \prec Q'$
**then obtain** $Q''$ $Q'''$ **where** *QChain*: $Q \Longrightarrow_\tau Q'''$
**and** $Q'''Trans$: $Q''' \longmapsto\alpha \prec Q''$
**and** $Q''Chain$: $Q'' \Longrightarrow_\tau Q'$
**by**(*force dest*: *transitionE*)
**show** $P \oplus Q \Longrightarrow\alpha \prec Q'$
**proof**(*cases* $Q = Q'''$)
**case** *True*
**have** $P \oplus Q \Longrightarrow_\tau P \oplus Q$ **by** *simp*
**moreover from** $Q'''Trans$ ‹$Q = Q'''$› **have** $P \oplus Q \longmapsto\alpha \prec Q''$ **by**(*blast intro*:

*Sum2*)
  **ultimately show** *?thesis* **using** $Q''$*Chain* **by**(*rule transitionI*)
 **next**
  **case** *False*
  **from** *QChain* ‹$Q \neq Q'''$› **have** $P \oplus Q \Longrightarrow_\tau Q'''$ **by**(*rule sum2Chain*)
  **thus** *?thesis* **using** $Q'''$*Trans* $Q''$*Chain* **by**(*rule transitionI*)
 **qed**
**qed**

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

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

 **shows** $P \parallel Q \Longrightarrow a{<}\nu x{>} \prec (P' \parallel Q)$
**proof** −
 **from** *PTrans* **obtain** $P''$ $P'''$ **where** *PChain*: $P \Longrightarrow_\tau P'''$
   **and** $P'''$*Trans*: $P''' \longmapsto a{<}\nu x{>} \prec P''$
   **and** $P''$*Chain*: $P'' \Longrightarrow_\tau P'$
  **by**(*blast dest*: *transitionE*)
 **from** *PChain* **have** $P \parallel Q \Longrightarrow_\tau P''' \parallel Q$ **by**(*rule Par1Chain*)
 **moreover from** $P'''$*Trans* ‹$x \sharp Q$› **have** $P''' \parallel Q \longmapsto a{<}\nu x{>} \prec (P'' \parallel Q)$ **by**(*rule Early-Semantics.Par1B*)
 **moreover from** $P''$*Chain* **have** $P'' \parallel Q \Longrightarrow_\tau P' \parallel Q$ **by**(*rule Par1Chain*)
 **ultimately show** $P \parallel Q \Longrightarrow a{<}\nu x{>} \prec (P' \parallel Q)$ **by**(*rule transitionI*)
**qed**

**lemma** *Par1F*:
 **fixes** $P$ :: *pi*
 **and** $\alpha$ :: *freeRes*
 **and** $P'$ :: *pi*
 **and** $Q$ :: *pi*

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

 **shows** $P \parallel Q \Longrightarrow \alpha \prec (P' \parallel Q)$
**proof** −
 **from** *PTrans* **obtain** $P''$ $P'''$ **where** *PChain*: $P \Longrightarrow_\tau P'''$
   **and** $P'''$*Trans*: $P''' \longmapsto \alpha \prec P''$
   **and** $P''$*Chain*: $P'' \Longrightarrow_\tau P'$
  **by**(*blast dest*: *transitionE*)
 **from** *PChain* **have** $P \parallel Q \Longrightarrow_\tau P''' \parallel Q$ **by**(*rule Par1Chain*)
 **moreover from** $P'''$*Trans* **have** $P''' \parallel Q \longmapsto \alpha \prec (P'' \parallel Q)$ **by**(*rule Early-Semantics.Par1F*)
 **moreover from** $P''$*Chain* **have** $P'' \parallel Q \Longrightarrow_\tau P' \parallel Q$ **by**(*rule Par1Chain*)

**ultimately show** *?thesis* **by**(*rule transitionI*)
**qed**

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

**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** −
**from** *QTrans* **obtain** $Q''$ $Q'''$ **where** *QChain*: $Q \Longrightarrow_\tau Q'''$
**and** $Q'''$*Trans*: $Q''' \longmapsto a{<}\nu x{>} \prec Q''$
**and** $Q''$*Chain*: $Q'' \Longrightarrow_\tau Q'$
**by**(*blast dest*: *transitionE*)
**from** *QChain* **have** $P \parallel Q \Longrightarrow_\tau P \parallel Q'''$ **by**(*rule Par2Chain*)
**moreover from** $Q'''$*Trans* ‹$x \sharp P$› **have** $P \parallel Q''' \longmapsto a{<}\nu x{>} \prec (P \parallel Q'')$ **by**(*rule Early-Semantics.Par2B*)
**moreover from** $Q''$*Chain* **have** $P \parallel Q'' \Longrightarrow_\tau P \parallel Q'$ **by**(*rule Par2Chain*)
**ultimately show** $P \parallel Q \Longrightarrow a{<}\nu x{>} \prec (P \parallel Q')$ **by**(*rule transitionI*)
**qed**

**lemma** *Par2F*:
**fixes** $Q$ :: *pi*
**and** $\alpha$ :: *freeRes*
**and** $Q'$ :: *pi*
**and** $P$ :: *pi*

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

**shows** $P \parallel Q \Longrightarrow \alpha \prec (P \parallel Q')$
**proof** −
**from** *QTrans* **obtain** $Q''$ $Q'''$ **where** *QChain*: $Q \Longrightarrow_\tau Q'''$
**and** $Q'''$*Trans*: $Q''' \longmapsto \alpha \prec Q''$
**and** $Q''$*Chain*: $Q'' \Longrightarrow_\tau Q'$
**by**(*blast dest*: *transitionE*)
**from** *QChain* **have** $P \parallel Q \Longrightarrow_\tau P \parallel Q'''$ **by**(*rule Par2Chain*)
**moreover from** $Q'''$*Trans* **have** $P \parallel Q''' \longmapsto \alpha \prec (P \parallel Q'')$ **by**(*rule Early-Semantics.Par2F*)
**moreover from** $Q''$*Chain* **have** $P \parallel Q'' \Longrightarrow_\tau P \parallel Q'$ **by**(*rule Par2Chain*)
**ultimately show** *?thesis* **by**(*rule transitionI*)
**qed**

**lemma** *Comm1*:
**fixes** $P$ :: *pi*
**and** $a$ :: *name*

**and** $b$ :: *name*
**and** $P'$ :: *pi*
**and** $Q$ :: *pi*
**and** $Q'$ :: *pi*

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

**shows** $P \parallel Q \Longrightarrow \tau \prec P' \parallel Q'$
**proof** $-$
**from** *PTrans* **obtain** $P''$ $P'''$ **where** *PChain*: $P \Longrightarrow_\tau P'''$
**and** $P'''$*Trans*: $P''' \longmapsto a{<}b{>} \prec P''$
**and** $P''$*Chain*: $P'' \Longrightarrow_\tau P'$
**by**(*blast dest*: *transitionE*)
**from** *QTrans* **obtain** $Q''$ $Q'''$ **where** *QChain*: $Q \Longrightarrow_\tau Q'''$
**and** $Q'''$*Trans*: $Q''' \longmapsto a[b] \prec Q''$
**and** $Q''$*Chain*: $Q'' \Longrightarrow_\tau Q'$
**by**(*blast dest*: *transitionE*)

**from** *PChain QChain* **have** $P \parallel Q \Longrightarrow_\tau P''' \parallel Q'''$ **by**(*rule chainPar*)
**moreover from** $P'''$*Trans* $Q'''$*Trans* **have** $P''' \parallel Q''' \longmapsto \tau \prec P'' \parallel Q''$
**by**(*rule Early-Semantics.Comm1*)
**moreover from** $P''$*Chain* $Q''$*Chain* **have** $P'' \parallel Q'' \Longrightarrow_\tau P' \parallel Q'$ **by**(*rule chain-Par*)
**ultimately show** *?thesis* **by**(*rule transitionI*)
**qed**

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

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

**shows** $P \parallel Q \Longrightarrow \tau \prec P' \parallel Q'$
**proof** $-$
**from** *PTrans* **obtain** $P''$ $P'''$ **where** *PChain*: $P \Longrightarrow_\tau P'''$
**and** $P'''$*Trans*: $P''' \longmapsto a[b] \prec P''$
**and** $P''$*Chain*: $P'' \Longrightarrow_\tau P'$
**by**(*blast dest*: *transitionE*)
**from** *QTrans* **obtain** $Q''$ $Q'''$ **where** *QChain*: $Q \Longrightarrow_\tau Q'''$
**and** $Q'''$*Trans*: $Q''' \longmapsto a{<}b{>} \prec Q''$
**and** $Q''$*Chain*: $Q'' \Longrightarrow_\tau Q'$
**by**(*blast dest*: *transitionE*)

**from** *PChain QChain* **have** $P \parallel Q \Longrightarrow_\tau P''' \parallel Q'''$ **by**(*rule chainPar*)
**moreover from** *P'''Trans Q'''Trans* **have** $P''' \parallel Q''' \longmapsto \tau \prec P'' \parallel Q''$
**by**(*rule Early-Semantics.Comm2*)
**moreover from** *P''Chain Q''Chain* **have** $P'' \parallel Q'' \Longrightarrow_\tau P' \parallel Q'$ **by**(*rule chainPar*)
**ultimately show** *?thesis* **by**(*rule transitionI*)
**qed**

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

**assumes** *PTrans*: $P \Longrightarrow a{<}x{>} \prec P'$
**and** *QTrans*: $Q \Longrightarrow a{<}\nu x{>} \prec Q'$
**and** $x \sharp P$

**shows** $P \parallel Q \Longrightarrow \tau \prec {<}\nu x{>}(P' \parallel Q')$
**proof** −
**from** *PTrans* **obtain** $P'''$ $P''$ **where** *PChain*: $P \Longrightarrow_\tau P'''$
**and** *P'''Trans*: $P''' \longmapsto a{<}x{>} \prec P''$
**and** *P''Chain*: $P'' \Longrightarrow_\tau P'$
**by**(*blast dest*: *transitionE*)
**from** *QTrans* **obtain** $Q''$ $Q'''$ **where** *QChain*: $Q \Longrightarrow_\tau Q'''$
**and** *Q'''Trans*: $Q''' \longmapsto a{<}\nu x{>} \prec Q''$
**and** *Q''Chain*: $Q'' \Longrightarrow_\tau Q'$
**by**(*blast dest*: *transitionE*)

**from** *PChain QChain* **have** $P \parallel Q \Longrightarrow_\tau P''' \parallel Q'''$ **by**(*rule chainPar*)
**moreover from** *PChain* ‹$x \sharp P$› **have** $x \sharp P'''$ **by**(*rule freshChain*)
**with** *P'''Trans Q'''Trans* **have** $P''' \parallel Q''' \longmapsto \tau \prec {<}\nu x{>}(P'' \parallel Q'')$
**by**(*rule Early-Semantics.Close1*)
**moreover from** *P''Chain Q''Chain* **have** $P'' \parallel Q'' \Longrightarrow_\tau P' \parallel Q'$ **by**(*rule chainPar*)
**hence** ${<}\nu x{>}(P'' \parallel Q'') \Longrightarrow_\tau {<}\nu x{>}(P' \parallel Q')$ **by**(*rule ResChain*)
**ultimately show** *?thesis* **by**(*rule transitionI*)
**qed**

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

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

**shows** $P \parallel Q \Longrightarrow \tau \prec {<}\nu x{>}(P' \parallel Q')$
**proof** −
**from** *PTrans* **obtain** $P''$ $P'''$ **where** *PChain*: $P \Longrightarrow_\tau P'''$
**and** $P'''$*Trans*: $P''' \longmapsto a{<}\nu x{>} \prec P''$
**and** $P''$*Chain*: $P'' \Longrightarrow_\tau P'$
**by**(*blast dest*: *transitionE*)
**from** *QTrans* **obtain** $Q''$ $Q'''$ **where** *QChain*: $Q \Longrightarrow_\tau Q'''$
**and** $Q'''$*Trans*: $Q''' \longmapsto a{<}x{>} \prec Q''$
**and** $Q''$*Chain*: $Q'' \Longrightarrow_\tau Q'$
**by**(*blast dest*: *transitionE*)

**from** *PChain QChain* **have** $P \parallel Q \Longrightarrow_\tau P''' \parallel Q'''$ **by**(*rule chainPar*)
**moreover from** *QChain* ‹$x \sharp Q$› **have** $x \sharp Q'''$ **by**(*rule freshChain*)

**with** $P'''$*Trans* $Q'''$*Trans* **have** $P''' \parallel Q''' \longmapsto \tau \prec {<}\nu x{>}(P'' \parallel Q'')$
**by**(*rule Early-Semantics.Close2*)
**moreover from** $P''$*Chain* $Q''$*Chain* **have** $P'' \parallel Q'' \Longrightarrow_\tau P' \parallel Q'$ **by**(*rule chain-Par*)
**hence** ${<}\nu x{>}(P'' \parallel Q'') \Longrightarrow_\tau {<}\nu x{>}(P' \parallel Q')$ **by**(*rule ResChain*)
**ultimately show** *?thesis* **by**(*rule transitionI*)
**qed**

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

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

**shows** ${<}\nu x{>}P \Longrightarrow \alpha \prec {<}\nu x{>}P'$
**proof** −
**from** *PTrans* **obtain** $P''$ $P'''$ **where** *PChain*: $P \Longrightarrow_\tau P'''$
**and** $P'''$*Trans*: $P''' \longmapsto \alpha \prec P''$
**and** $P''$*Chain*: $P'' \Longrightarrow_\tau P'$
**by**(*blast dest*: *transitionE*)

**from** *PChain* **have** ${<}\nu x{>}P \Longrightarrow_\tau {<}\nu x{>}P'''$ **by**(*rule ResChain*)
**moreover from** $P'''$*Trans* ‹$x \sharp \alpha$› **have** ${<}\nu x{>}P''' \longmapsto \alpha \prec {<}\nu x{>}P''$
**by**(*rule Early-Semantics.ResF*)
**moreover from** $P''$*Chain* **have** ${<}\nu x{>}P'' \Longrightarrow_\tau {<}\nu x{>}P'$ **by**(*rule ResChain*)
**ultimately show** *?thesis* **by**(*rule transitionI*)
**qed**

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

**assumes** *PTrans*: $P \Longrightarrow a{<}\nu x{>} \prec P'$
**and** $y \neq a$
**and** $y \neq x$

**shows** $<\nu y>P \Longrightarrow a{<}\nu x{>} \prec (<\nu y>P')$
**proof** −
**from** *PTrans* **obtain** $P''$ $P'''$ **where** *PChain*: $P \Longrightarrow_\tau P'''$
**and** $P'''$*Trans*: $P''' \longmapsto a{<}\nu x{>} \prec P''$
**and** $P''$*Chain*: $P'' \Longrightarrow_\tau P'$
**by**(*blast dest*: *transitionE*)

**from** *PChain* **have** $<\nu y>P \Longrightarrow_\tau <\nu y>P'''$ **by**(*rule ResChain*)
**moreover from** $P'''$*Trans* ‹$y \neq a$› ‹$y \neq x$› **have** $<\nu y>P''' \longmapsto a{<}\nu x{>} \prec (<\nu y>P'')$
**by**(*rule Early-Semantics.ResB*)
**moreover from** $P''$*Chain* **have** $<\nu y>P'' \Longrightarrow_\tau <\nu y>P'$ **by**(*rule ResChain*)
**ultimately show** *?thesis* **by**(*rule transitionI*)
**qed**

**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** −
**assume** *PTrans*: $P \parallel !P \Longrightarrow a{<}\nu x{>} \prec P'$

**from** *PTrans* **obtain** $P''$ $P'''$ **where** *PChain*: $P \parallel !P \Longrightarrow_\tau P'''$
**and** $P'''$*Trans*: $P''' \longmapsto a{<}\nu x{>} \prec P''$
**and** $P''$*Chain*: $P'' \Longrightarrow_\tau P'$
**by**(*force dest*: *transitionE*)

**show** $!P \Longrightarrow a{<}\nu x{>} \prec P'$
**proof**(*cases* $P''' = P \parallel !P$)
**case** *True*
**have** $!P \Longrightarrow_\tau !P$ **by** *simp*
**moreover from** $P'''$*Trans* ‹$P''' = P \parallel !P$› **have** $!P \longmapsto a{<}\nu x{>} \prec P''$ **by**(*blast intro*: *Early-Semantics.Bang*)
**ultimately show** *?thesis* **using** $P''$*Chain* **by**(*rule transitionI*)

  **next**
    **case** *False*
    **from** *PChain* ‹$P''' \neq P \parallel !P$› **have** $!P \Longrightarrow_\tau P'''$ **by**(*rule bangChain*)
    **with** *P'''Trans P''Chain* **show** *?thesis* **by**(*blast intro*: *transitionI*)
  **qed**
**next**
  **fix** $\alpha$ $P'$ $P$
  **assume** $P \parallel !P \Longrightarrow\alpha \prec P'$

  **then obtain** $P''$ $P'''$ **where** *PChain*: $P \parallel !P \Longrightarrow_\tau P''$
                         **and** *P''Trans*: $P'' \longmapsto\alpha \prec P'''$
                         **and** *P'''Chain*: $P''' \Longrightarrow_\tau P'$
    **by**(*force dest*: *transitionE*)

  **show** $!P \Longrightarrow\alpha \prec P'$
  **proof**(*cases* $P'' = P \parallel !P$)
    **assume** $P'' = P \parallel !P$
    **moreover with** *P''Trans* **have** $!P \longmapsto\alpha \prec P'''$ **by**(*blast intro*: *Bang*)
     **ultimately show** *?thesis* **using** *PChain P'''Chain* **by**(*rule-tac transitionI*, *auto*)
  **next**
    **assume** $P'' \neq P \parallel !P$
    **with** *PChain* **have** $!P \Longrightarrow_\tau P''$ **by**(*rule bangChain*)
    **with** *P''Trans P'''Chain* **show** *?thesis* **by**(*blast intro*: *transitionI*)
  **qed**
**qed**

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

  **assumes** $P \Longrightarrow\tau \prec P'$

  **shows** $P \Longrightarrow_\tau P'$
**using** *assms*
**by**(*force dest*: *transitionE tauActTauChain*)

**lemma** *chainTransitionAppend*:
  **fixes** $P$ :: *pi*
  **and**   $P'$ :: *pi*
  **and**   $Rs$ :: *residual*
  **and**   $a$ :: *name*
  **and**   $x$ :: *name*
  **and**   $P''$ :: *pi*
  **and**   $\alpha$ :: *freeRes*

  **shows** $P \Longrightarrow a{<}\nu x{>} \prec P'' \Longrightarrow P'' \Longrightarrow_\tau P' \Longrightarrow P \Longrightarrow a{<}\nu x{>} \prec P'$
  **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** −
  **assume** *PTrans*: $P \Longrightarrow a{<}\nu x{>} \prec P''$
  **assume** $P''$*Chain*: $P'' \Longrightarrow_\tau P'$

  **from** *PTrans* **obtain** $P'''$ $P''''$ **where** *PChain*: $P \Longrightarrow_\tau P''''$
  **and** $P''''$*Trans*: $P'''' \longmapsto a{<}\nu x{>} \prec P'''$
  **and** $P'''$*Chain*: $P''' \Longrightarrow_\tau P''$
    **by**(*blast dest*: *transitionE*)

  **from** $P'''$*Chain* $P''$*Chain* **have** $P''' \Longrightarrow_\tau P'$ **by** *auto*
  **with** *PChain* $P''''$*Trans* **show** $P \Longrightarrow a{<}\nu x{>} \prec P'$ **by**(*rule transitionI*)
**next**
  **assume** *PTrans*: $P \Longrightarrow\alpha \prec P''$
  **assume** $P''$*Chain*: $P'' \Longrightarrow_\tau P'$

  **from** *PTrans* **obtain** $P'''$ $P''''$ **where** *PChain*: $P \Longrightarrow_\tau P''''$
  **and** $P''''$*Trans*: $P'''' \longmapsto\alpha \prec P'''$
  **and** $P'''$*Chain*: $P''' \Longrightarrow_\tau P''$
    **by**(*blast dest*: *transitionE*)

  **from** $P'''$*Chain* $P''$*Chain* **have** $P''' \Longrightarrow_\tau P'$ **by** *auto*
  **with** *PChain* $P''''$*Trans* **show** $P \Longrightarrow\alpha \prec P'$ **by**(*rule transitionI*)
**next**
  **assume** *PChain*: $P \Longrightarrow_\tau P''$
  **assume** $P''$*Trans*: $P'' \Longrightarrow a{<}\nu x{>} \prec P'$

  **from** $P''$*Trans* **obtain** $P'''$ $P''''$ **where** $P''$*Chain*: $P'' \Longrightarrow_\tau P''''$
  **and** $P''''$*Trans*: $P'''' \longmapsto a{<}\nu x{>} \prec P'''$
  **and** $P'''$*Chain*: $P''' \Longrightarrow_\tau P'$
    **by**(*blast dest*: *transitionE*)

  **from** *PChain* $P''$*Chain* **have** $P \Longrightarrow_\tau P''''$ **by** *auto*
  **thus** $P \Longrightarrow a{<}\nu x{>} \prec P'$ **using** $P''''$*Trans* $P'''$*Chain* **by**(*rule transitionI*)
**next**
  **assume** *PChain*: $P \Longrightarrow_\tau P''$
  **assume** $P''$*Trans*: $P'' \Longrightarrow\alpha \prec P'$

  **from** $P''$*Trans* **obtain** $P'''$ $P''''$ **where** $P''$*Chain*: $P'' \Longrightarrow_\tau P''''$
  **and** $P''''$*Trans*: $P'''' \longmapsto\alpha \prec P'''$
  **and** $P'''$*Chain*: $P''' \Longrightarrow_\tau P'$
    **by**(*blast dest*: *transitionE*)

  **from** *PChain* $P''$*Chain* **have** $P \Longrightarrow_\tau P''''$ **by** *auto*
  **thus** $P \Longrightarrow\alpha \prec P'$ **using** $P''''$*Trans* $P'''$*Chain* **by**(*rule transitionI*)
**qed**

**lemma** *freshBoundOutputTransition*:
  **fixes** $P$ :: *pi*

**and** $a$ :: *name*
**and** $x$ :: *name*
**and** $P'$ :: *pi*
**and** $c$ :: *name*

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

**shows** $c \sharp P'$
**proof** −
**from** *PTrans* **obtain** $P''$ $P'''$ **where** *PChain*: $P \Longrightarrow_\tau P'''$
**and** $P'''$*Trans*: $P''' \longmapsto a{<}\nu x{>} \prec P''$
**and** $P''$*Chain*: $P'' \Longrightarrow_\tau P'$
**by**(*blast dest*: *transitionE*)

**from** *PChain* ‹$c \sharp P$› **have** $c \sharp P'''$ **by**(*rule freshChain*)
**with** $P'''$*Trans* **have** $c \sharp P''$ **using** ‹$c \neq x$› **by**(*rule Early-Semantics.freshTransition*)
**with** $P''$*Chain* **show** $c \sharp P'$ **by**(*rule freshChain*)
**qed**

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

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

**shows** $c \sharp P'$
**proof** −
**from** *PTrans* **have** $P \Longrightarrow_\tau P'$ **by**(*rule tauTransitionChain*)
**thus** *?thesis* **using** ‹$c \sharp P$› **by**(*rule freshChain*)
**qed**

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

**assumes** *PTrans*: $P \Longrightarrow a[b] \prec P'$
**and** $c \sharp P$

**shows** $c \sharp P'$
**proof** −
**from** *PTrans* **obtain** $P''$ $P'''$ **where** *PChain*: $P \Longrightarrow_\tau P'''$
**and** $P'''$*Trans*: $P''' \longmapsto a[b] \prec P''$
**and** $P''$*Chain*: $P'' \Longrightarrow_\tau P'$

**by**(*blast dest*: *transitionE*)

**from** *PChain* ‹$c \sharp P$› **have** $c \sharp P'''$ **by**(*rule freshChain*)
**with** *P'''Trans* **have** $c \sharp P''$ **by**(*rule Early-Semantics.freshTransition*)
**with** *P''Chain* **show** *?thesis* **by**(*rule freshChain*)
**qed**

**lemma** *eqvtI*[*eqvt*]:
**fixes** $P$ :: *pi*
**and** $a$ :: *name*
**and** $x$ :: *name*
**and** $P'$ :: *pi*
**and** $p$ :: *name prm*
**and** $\alpha$ :: *freeRes*

**shows** $P \Longrightarrow a{<}\nu x{>} \prec P' \Longrightarrow (p \cdot P) \Longrightarrow (p \cdot a){<}\nu(p \cdot x){>} \prec (p \cdot P')$
**and** $P \Longrightarrow \alpha \prec P' \Longrightarrow (p \cdot P) \Longrightarrow (p \cdot \alpha) \prec (p \cdot P')$
**proof** −
**assume** $P \Longrightarrow a{<}\nu x{>} \prec P'$
**then obtain** $P''$ $P'''$ **where** *PChain*: $P \Longrightarrow_\tau P'''$
**and** *P'''Trans*: $P''' \longmapsto a{<}\nu x{>} \prec P''$
**and** *P''Chain*: $P'' \Longrightarrow_\tau P'$
**by**(*blast dest*: *transitionE*)

**from** *PChain* **have** $(p \cdot P) \Longrightarrow_\tau (p \cdot P''')$ **by**(*rule eqvtChainI*)
**moreover from** *P'''Trans* **have** $(p \cdot P''') \longmapsto (p \cdot (a{<}\nu x{>} \prec P''))$
**by**(*rule TransitionsEarly.eqvt*)
**hence** $(p \cdot P''') \longmapsto (p \cdot a){<}\nu(p \cdot x){>} \prec (p \cdot P'')$
**by**(*simp add*: *eqvts*)
**moreover from** *P''Chain* **have** $(p \cdot P'') \Longrightarrow_\tau (p \cdot P')$ **by**(*rule eqvtChainI*)
**ultimately show** $(p \cdot P) \Longrightarrow (p \cdot a){<}\nu(p \cdot x){>} \prec (p \cdot P')$
**by**(*rule transitionI*)
**next**
**assume** $P \Longrightarrow \alpha \prec P'$
**then obtain** $P''$ $P'''$ **where** *PChain*: $P \Longrightarrow_\tau P'''$
**and** *P'''Trans*: $P''' \longmapsto \alpha \prec P''$
**and** *P''Chain*: $P'' \Longrightarrow_\tau P'$
**by**(*blast dest*: *transitionE*)

**from** *PChain* **have** $(p \cdot P) \Longrightarrow_\tau (p \cdot P''')$ **by**(*rule eqvtChainI*)
**moreover from** *P'''Trans* **have** $(p \cdot P''') \longmapsto (p \cdot (\alpha \prec P''))$
**by**(*rule TransitionsEarly.eqvt*)
**hence** $(p \cdot P''') \longmapsto (p \cdot \alpha) \prec (p \cdot P'')$
**by**(*simp add*: *eqvts*)
**moreover from** *P''Chain* **have** $(p \cdot P'') \Longrightarrow_\tau (p \cdot P')$ **by**(*rule eqvtChainI*)
**ultimately show** $(p \cdot P) \Longrightarrow (p \cdot \alpha) \prec (p \cdot P')$
**by**(*rule transitionI*)
**qed**

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

  **assumes** *PTrans*: $P \Longrightarrow a<b> \prec P'$
  **and** $c \sharp P$
  **and** $c \neq b$

  **shows** $c \sharp P'$
**proof** −
  **from** *PTrans* **obtain** $P''$ $P'''$ **where** *PChain*: $P \Longrightarrow_\tau P'''$
    **and** $P'''$*Trans*: $P''' \longmapsto a<b> \prec P''$
    **and** $P''$*Chain*: $P'' \Longrightarrow_\tau P'$
    **by**(*blast dest*: *transitionE*)

  **from** *PChain* ‹$c \sharp P$› **have** $c \sharp P'''$ **by**(*rule freshChain*)
  **with** $P'''$*Trans* **have** $c \sharp P''$ **using** ‹$c \neq b$› **by**(*rule Early-Semantics.freshInputTransition*)
  **with** $P''$*Chain* **show** *?thesis* **by**(*rule freshChain*)
**qed**

**lemmas** *freshTransition* = *freshBoundOutputTransition freshOutputTransition*
  *freshInputTransition freshTauTransition*

**end**

**theory** *Weak-Early-Semantics*
  **imports** *Weak-Early-Step-Semantics*
**begin**

**definition** *weakFreeTransition* :: *pi* ⇒ *freeRes* ⇒ *pi* ⇒ *bool* (‹- $\Longrightarrow\hat{}$ - $\prec$ -› [*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** $\alpha$ :: *freeRes*
  **and** $P'$ :: *pi*

  **shows** $P \Longrightarrow\alpha \prec P' \Longrightarrow P \Longrightarrow\hat{}\alpha \prec P'$
  **and** $P \Longrightarrow\hat{}\tau \prec P$
**by**(*auto simp add*: *weakFreeTransition-def*)

**lemma** *transitionCases*[*consumes 1*, *case-names Step Stay*]:
  **fixes** $P$ :: *pi*
  **and** $\alpha$ :: *freeRes*
  **and** $P'$ :: *pi*

**assumes** $P \Longrightarrow\hat{}\ \alpha \prec P'$
**and** $P \Longrightarrow\alpha \prec P' \Longrightarrow F\ \alpha\ P'$
**and** $F\ (\tau)\ P$

**shows** $F\ \alpha\ P'$
**using** *assms*
**by**(*auto simp add*: *weakFreeTransition-def*)

**lemma** *singleActionChain*:
**fixes** $P$ :: *pi*
**and** $\alpha$ :: *freeRes*
**and** $P'$ :: *pi*

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

**shows** $P \Longrightarrow\hat{}\ \alpha \prec P'$
**using** *assms*
**by**(*auto dest*: *singleActionChain intro*: *weakTransitionI*)

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

**shows** $\tau.(P) \Longrightarrow\hat{}\ \tau \prec\ P$
**by**(*auto intro*: *Weak-Early-Step-Semantics.Tau*
*simp add*: *weakFreeTransition-def*)

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

**shows** $a{<}x{>}.P \Longrightarrow\hat{}\ a{<}u{>} \prec P[x{::=}u]$
**by**(*auto intro*: *Weak-Early-Step-Semantics.Input*
*simp add*: *weakFreeTransition-def*)

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

**shows** $a\{b\}.P \Longrightarrow\hat{}\ a[b] \prec P$
**by**(*auto intro*: *Weak-Early-Step-Semantics.Output*
*simp add*: *weakFreeTransition-def*)

**lemma** *Par1F*:
**fixes** $P$ :: *pi*
**and** $\alpha$ :: *freeRes*

**and** $P'$ :: *pi*
**and** $Q$ :: *pi*

**assumes** $P \Longrightarrow\hat{}\ \alpha \prec P'$

**shows** $P \parallel Q \Longrightarrow\hat{}\ \alpha \prec (P' \parallel Q)$
**using** *assms*
**by**(*auto intro*: *Weak-Early-Step-Semantics.Par1F*
*simp add*: *weakFreeTransition-def residual.inject*)

**lemma** *Par2F*:
**fixes** $Q$ :: *pi*
**and** $\alpha$ :: *freeRes*
**and** $Q'$ :: *pi*
**and** $P$ :: *pi*

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

**shows** $P \parallel Q \Longrightarrow\hat{}\ \alpha \prec (P \parallel Q')$
**using** *assms*
**by**(*auto intro*: *Weak-Early-Step-Semantics.Par2F*
*simp add*: *weakFreeTransition-def residual.inject*)

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

**assumes** $P \Longrightarrow\hat{}\ \alpha \prec P'$
**and** $x \sharp \alpha$

**shows** $<\nu x>P \Longrightarrow\hat{}\ \alpha \prec <\nu x>P'$
**using** *assms*
**by**(*auto intro*: *Weak-Early-Step-Semantics.ResF*
*simp add*: *weakFreeTransition-def residual.inject*)

**lemma** *Bang*:
**fixes** $P$ :: *pi*
**and** $Rs$ :: *residual*

**assumes** $P \parallel !P \Longrightarrow\hat{}\ \alpha \prec P'$
**and** $P' \neq P \parallel !P$

**shows** $!P \Longrightarrow\hat{}\ \alpha \prec P'$
**using** *assms*
**by**(*auto intro*: *Weak-Early-Step-Semantics.Bang*
*simp add*: *weakFreeTransition-def residual.inject*)

**lemma** *tauTransitionChain*[*simp*]:
**fixes** $P$ :: *pi*
**and** $P'$ :: *pi*

**shows** $P \Longrightarrow^{\hat{}} \tau \prec P' = P \Longrightarrow_\tau P'$
**apply**(*auto dest*: *Weak-Early-Step-Semantics.tauTransitionChain*
*simp add*: *weakFreeTransition-def*)
**by**(*erule rtrancl.cases*) (*auto intro*: *transitionI*)

**lemma** *tauStepTransitionChain*[*simp*]:
**fixes** $P$ :: *pi*
**and** $P'$ :: *pi*

**assumes** $P \neq P'$

**shows** $P \Longrightarrow \tau \prec P' = P \Longrightarrow_\tau P'$
**using** *assms*
**apply**(*auto dest*: *Weak-Early-Step-Semantics.tauTransitionChain*
*simp add*: *weakFreeTransition-def*)
**by**(*erule rtrancl.cases*) (*auto intro*: *transitionI*)

**lemma** *chainTransitionAppend*:
**fixes** $P$ :: *pi*
**and** $P'$ :: *pi*
**and** $Rs$ :: *residual*
**and** $a$ :: *name*
**and** $x$ :: *name*
**and** $P''$ :: *pi*
**and** $\alpha$ :: *freeRes*

**shows** $P \Longrightarrow_\tau P'' \Longrightarrow P'' \Longrightarrow^{\hat{}} \alpha \prec P' \Longrightarrow P \Longrightarrow^{\hat{}} \alpha \prec P'$
**and** $P \Longrightarrow^{\hat{}} \alpha \prec P'' \Longrightarrow P'' \Longrightarrow_\tau P' \Longrightarrow P \Longrightarrow^{\hat{}} \alpha \prec P'$
**by**(*auto intro*: *chainTransitionAppend simp add*: *weakFreeTransition-def dest*: *Weak-Early-Step-Semantics.tauT*

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

**assumes** $P \Longrightarrow^{\hat{}} \tau \prec P'$
**and** $c \sharp P$

**shows** $c \sharp P'$
**using** *assms*
**by**(*auto intro*: *Weak-Early-Step-Semantics.freshTauTransition*
*simp add*: *weakFreeTransition-def residual.inject*)

**lemma** *freshOutputTransition*:
**fixes** $P$ :: *pi*

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

**assumes** $P \Longrightarrow\hat{} a[b] \prec P'$
**and** $c \sharp P$

**shows** $c \sharp P'$
**using** *assms*
**by**(*auto intro*: *Weak-Early-Step-Semantics.freshOutputTransition*
*simp add*: *weakFreeTransition-def residual.inject*)

**lemma** *eqvtI*:
**fixes** $P$ :: *pi*
**and** $\alpha$ :: *freeRes*
**and** $P'$ :: *pi*
**and** $p$ :: *name prm*

**assumes** $P \Longrightarrow\hat{} \alpha \prec P'$

**shows** $(p \cdot P) \Longrightarrow\hat{} (p \cdot \alpha) \prec (p \cdot P')$
**using** *assms*
**by**(*auto intro*: *Weak-Early-Step-Semantics.eqvtI*
*simp add*: *weakFreeTransition-def residual.inject*)

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

**assumes** $P \Longrightarrow\hat{} a{<}b{>} \prec P'$
**and** $c \sharp P$
**and** $c \neq b$

**shows** $c \sharp P'$
**using** *assms*
**by**(*auto intro*: *Weak-Early-Step-Semantics.freshInputTransition*
*simp add*: *weakFreeTransition-def residual.inject*)

**lemmas** *freshTransition* = *freshBoundOutputTransition freshOutputTransition*
*freshInputTransition freshTauTransition*

**end**

**theory** *Weak-Early-Sim*
**imports** *Weak-Early-Semantics Strong-Early-Sim-Pres*

**begin**

**definition** *weakSimulation* :: $pi \Rightarrow (pi \times pi)\ set \Rightarrow pi \Rightarrow bool$ ($\langle$- $\leadsto$<-> -$\rangle$ [*80*, *80*, *80*] *80*)
  **where** $P \leadsto<Rel>\ Q \equiv (\forall\, a\ x\ Q'.\ Q \longmapsto a<\nu x> \prec Q' \wedge x \sharp P \longrightarrow (\exists P'.\ P \Longrightarrow a<\nu x> \prec P' \wedge (P',\ Q') \in Rel)) \wedge$
  $(\forall\, \alpha\ Q'.\ Q \longmapsto \alpha \prec Q' \longrightarrow (\exists P'.\ P \Longrightarrow\hat{}\ \alpha \prec P' \wedge (P',\ Q') \in Rel))$

**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** $P \leadsto<B>\ P'$
**using** *assms*
**by**(*simp add*: *weakSimulation-def*) *blast*

**lemma** *simCasesCont*[*consumes 1*, *case-names Bound Free*]:
  **fixes** $P$ :: $pi$
  **and** $Q$ :: $pi$
  **and** $Rel$ :: $(pi \times pi)\ set$
  **and** $C$ :: $'a$::*fs-name*

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

  **shows** $P \leadsto<Rel>\ Q$
**proof**(*auto simp add*: *weakSimulation-def*)
  **fix** $a\ x\ Q'$
  **assume** *QTrans*: $Q \longmapsto a<\nu x> \prec Q'$ **and** $x \sharp P$
  **obtain** $c$::*name* **where** $c \sharp P$ **and** $c \sharp Q$ **and** $c \neq a$ **and** $c \sharp Q'$ **and** $c \sharp C$ **and** $c \neq x$
    **by**(*generate-fresh name*) *auto*

  **from** *QTrans* $\langle c \sharp Q'\rangle$ **have** $Q \longmapsto a<\nu c> \prec ([(x,\ c)] \cdot Q')$ **by**(*simp add*: *alphaBoundOutput*)
  **then obtain** $P'$ **where** *PTrans*: $P \Longrightarrow a<\nu c> \prec P'$ **and** *P'RelQ'*: $(P',\ [(x,\ c)] \cdot Q') \in Rel$
    **using** $\langle c \sharp P\rangle$ $\langle c \sharp Q\rangle$ $\langle c \neq a\rangle$ $\langle c \sharp C\rangle$
    **by**(*drule-tac Bound*) *auto*

  **from** *PTrans* $\langle x \sharp P\rangle$ $\langle c \neq x\rangle$ **have** $P \Longrightarrow a<\nu x> \prec ([(x,\ c)] \cdot P')$

**by**(*force intro*: *weakTransitionAlpha simp add*: *name-swap*)
**moreover from** *Eqvt* $P'RelQ'$ **have** $([(x, c)] \cdot P', [(x, c)] \cdot [(x, c)] \cdot Q') \in Rel$
**by**(*rule eqvtRelI*)
**hence** $([(x, c)] \cdot P', Q') \in Rel$ **by** *simp*
**ultimately show** $\exists P'. P \Longrightarrow a<\nu x> \prec P' \wedge (P', Q') \in Rel$
**by** *blast*
**next**
**fix** $\alpha$ $Q'$
**assume** $Q \longmapsto \alpha \prec Q'$
**thus** $\exists P'. P \Longrightarrow^{\hat{}} \alpha \prec P' \wedge (P', Q') \in Rel$
**by**(*rule Free*)
**qed**

**lemma** *simCases*[*case-names Bound Free*]:
**fixes** $P$ :: *pi*
**and** $Q$ :: *pi*
**and** *Rel* :: (*pi* × *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** $\bigwedge Q'\ \alpha.\ Q \longmapsto \alpha \prec Q' \Longrightarrow \exists P'. P \Longrightarrow^{\hat{}} \alpha \prec P' \wedge (P', Q') \in Rel$

**shows** $P \leadsto<Rel> Q$
**using** *assms*
**by**(*auto simp add*: *weakSimulation-def*)

**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** $Q \longmapsto \alpha \prec Q' \Longrightarrow \exists P'. P \Longrightarrow^{\hat{}} \alpha \prec P' \wedge (P', Q') \in Rel$
**using** *assms* **by**(*simp add*: *weakSimulation-def*)+

**lemma** *weakSimTauChain*:
**fixes** $P$ :: *pi*
**and** *Rel* :: (*pi* × *pi*) *set*
**and** *Rel'* :: (*pi* × *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** −
**from** *QChain* **show** *?thesis*
**proof**(*induct rule*: *tauChainInduct*)
**case** *id*
**moreover have** $P \Longrightarrow_\tau P$ **by** *simp*
**ultimately show** *?case* **using** *PSimQ PRelQ* **by** *blast*
**next**
**case**(*ih* $Q'$ $Q''$)
**have** $\exists P'.\ P \Longrightarrow_\tau P' \wedge (P', Q') \in Rel$ **by** *fact*
**then obtain** $P'$ **where** *PChain*: $P \Longrightarrow_\tau P'$ **and** $P'Rel'Q'$: $(P', Q') \in Rel$ **by** *blast*
**from** $P'Rel'Q'$ **have** $P' \leadsto$<*Rel*> $Q'$ **by**(*rule PSimQ*)
**moreover have** $Q'Trans$: $Q' \longmapsto \tau \prec Q''$ **by** *fact*
**ultimately obtain** $P''$ **where** $P'Trans$: $P' \Longrightarrow\hat{}\ \tau \prec P''$ **and** $P''RelQ''$: $(P'', Q'') \in Rel$
**by**(*blast dest*: *simE*)
**from** $P'Trans$ **have** $P' \Longrightarrow_\tau P''$ **by** *simp*
**with** *PChain* **have** $P \Longrightarrow_\tau P''$ **by** *auto*
**with** $P''RelQ''$ **show** *?case* **by** *blast*
**qed**
**qed**

**lemma** *simE2*:
**fixes** $P$ :: *pi*
**and** *Rel* :: (*pi* × *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** −
**assume** *QTrans*: $Q \Longrightarrow a$<$\nu x$> $\prec Q'$ **and** $x \sharp P$
**from** *QTrans* **obtain** $Q''$ $Q'''$ **where** *QChain*: $Q \Longrightarrow_\tau Q'''$
**and** $Q'''Trans$: $Q''' \longmapsto a$<$\nu x$> $\prec Q''$
**and** $Q''Chain$: $Q'' \Longrightarrow_\tau Q'$
**by**(*blast dest*: *transitionE*)

**from** *QChain PRelQ Sim* **obtain** $P'''$ **where** *PChain*: $P \Longrightarrow_\tau P'''$ **and** $P'''RelQ'''$: $(P''', Q''') \in Rel$
**by**(*blast dest*: *weakSimTauChain*)

**from** *PChain* $\langle x \sharp P\rangle$ **have** $x \sharp P'''$ **by**(*rule freshChain*)

**from** $P'''RelQ'''$ **have** $P''' \rightsquigarrow$<*Rel*> $Q'''$ **by**(*rule Sim*)
**with** $Q'''Trans$ $\langle x \sharp P'''\rangle$ **obtain** $P''$ **where** $P'''Trans$: $P''' \Longrightarrow a$<$\nu x$> $\prec P''$
**and** $P''RelQ''$: $(P'', Q'') \in Rel$
**by**(*blast dest*: *simE*)

**from** $Q''Chain$ $P''RelQ''$ *Sim* **obtain** $P'$ **where** $P''Chain$: $P'' \Longrightarrow_\tau P'$ **and** $P'RelQ'$: $(P', Q') \in Rel$
**by**(*blast dest*: *weakSimTauChain*)
**from** *PChain* $P'''Trans$ $P''Chain$ **have** $P \Longrightarrow a$<$\nu x$> $\prec P'$
**by**(*blast dest*: *Weak-Early-Step-Semantics.chainTransitionAppend*)
**with** $P'RelQ'$ **show** $\exists P'. P \Longrightarrow a$<$\nu x$> $\prec P' \wedge (P', Q') \in Rel$ **by** *blast*
**next**
**assume** $Q \Longrightarrow^\wedge \alpha \prec Q'$
**thus** $\exists P'. P \Longrightarrow^\wedge \alpha \prec P' \wedge (P', Q') \in Rel$
**proof**(*induct rule*: *transitionCases*)
**case** *Step*
**have** $Q \Longrightarrow \alpha \prec Q'$ **by** *fact*
**then obtain** $Q''$ $Q'''$ **where** *QChain*: $Q \Longrightarrow_\tau Q''$
**and** $Q''Trans$: $Q'' \longmapsto \alpha \prec Q'''$
**and** $Q'''Chain$: $Q''' \Longrightarrow_\tau Q'$
**by**(*blast dest*: *transitionE*)

**from** *QChain PRelQ Sim* **have** $\exists P''. P \Longrightarrow_\tau P'' \wedge (P'', Q'') \in Rel$
**by**(*rule weakSimTauChain*)
**then obtain** $P''$ **where** *PChain*: $P \Longrightarrow_\tau P''$ **and** $P''RelQ''$: $(P'', Q'') \in Rel$ **by** *blast*
**from** $P''RelQ''$ **have** $P'' \rightsquigarrow$<*Rel*> $Q''$ **by**(*rule Sim*)
**with** $Q''Trans$ **obtain** $P'''$ **where** $P''Trans$: $P'' \Longrightarrow^\wedge \alpha \prec P'''$
**and** $P'''RelQ'''$: $(P''', Q''') \in Rel$
**by**(*blast dest*: *simE*)

**have** $\exists P'. P''' \Longrightarrow_\tau P' \wedge (P', Q') \in Rel$ **using** $Q'''Chain$ $P'''RelQ'''$ *Sim*
**by**(*rule weakSimTauChain*)
**then obtain** $P'$ **where** $P'''Chain$: $P''' \Longrightarrow_\tau P'$ **and** $P'RelQ'$: $(P', Q') \in Rel$ **by** *blast*

**from** *PChain* $P''Trans$ $P'''Chain$ **have** $P \Longrightarrow^\wedge \alpha \prec P'$
**by**(*blast dest*: *chainTransitionAppend*)
**with** $P'RelQ'$ **show** *?case* **by** *blast*
**next**
**case** *Stay*
**have** $P \Longrightarrow^\wedge \tau \prec P$ **by** *simp*
**thus** *?case* **using** *PRelQ* **by** *blast*

qed
qed

**lemma** *eqvtI*:
**fixes** $P$ :: *pi*
**and** $Q$ :: *pi*
**and** *Rel* :: (*pi* × *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**(*induct rule*: *simCases*)
**case**(*Bound Q' a x*)
**have** *xFreshP*: $x \sharp perm \cdot P$ **by** *fact*
**have** *QTrans*: $(perm \cdot Q) \longmapsto a$<$\nu x$> $\prec Q'$ **by** *fact*

**hence** $(rev\ perm \cdot (perm \cdot Q)) \longmapsto rev\ perm \cdot (a$<$\nu x$> $\prec Q')$ **by**(*rule eqvts*)
**hence** $Q \longmapsto (rev\ perm \cdot a)$<$\nu(rev\ perm \cdot x)$> $\prec (rev\ perm \cdot Q')$
**by**(*simp add*: *name-rev-per*)
**moreover from** *xFreshP* **have** $(rev\ perm \cdot x) \sharp P$ **by**(*simp add*: *name-fresh-left*)
**ultimately obtain** $P'$ **where** *PTrans*: $P \Longrightarrow(rev\ perm \cdot a)$<$\nu(rev\ perm \cdot x)$> $\prec P'$
**and** *P'RelQ'*: $(P', rev\ perm \cdot Q') \in Rel$ **using** *PSimQ*
**by**(*blast dest*: *simE*)

**from** *PTrans* **have** $(perm \cdot P) \Longrightarrow(perm \cdot rev\ perm \cdot a)$<$\nu(perm \cdot rev\ perm \cdot x)$> $\prec perm \cdot P'$
**by**(*rule eqvts*)
**hence** $(perm \cdot P) \Longrightarrow a$<$\nu x$> $\prec (perm \cdot P')$ **by**(*simp add*: *name-per-rev*)
**moreover from** *P'RelQ' RelRel'* **have** $(P', rev\ perm \cdot Q') \in Rel'$ **by** *blast*
**with** *EqvtRel'* **have** $(perm \cdot P', perm \cdot (rev\ perm \cdot Q')) \in Rel'$
**by**(*rule eqvtRelI*)
**hence** $(perm \cdot P', Q') \in Rel'$ **by**(*simp add*: *name-per-rev*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*Free Q' α*)
**have** *QTrans*: $(perm \cdot Q) \longmapsto \alpha \prec Q'$ **by** *fact*

**hence** $(rev\ perm \cdot (perm \cdot Q)) \longmapsto rev\ perm \cdot (\alpha \prec Q')$ **by**(*rule eqvts*)
**hence** $Q \longmapsto (rev\ perm \cdot \alpha) \prec (rev\ perm \cdot Q')$ **by**(*simp add*: *name-rev-per*)
**with** *PSimQ* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow^{\hat{}} (rev\ perm \cdot \alpha) \prec P'$
**and** *PRel*: $(P', (rev\ perm \cdot Q')) \in Rel$
**by**(*blast dest*: *simE*)

**from** *PTrans* **have** $(perm \cdot P) \Longrightarrow^{\hat{}} (perm \cdot rev\ perm \cdot \alpha) \prec perm \cdot P'$
**by**(*rule Weak-Early-Semantics.eqvtI*)

**hence** *L1*: $(perm \cdot P) \Longrightarrow\hat{}\ \alpha \prec (perm \cdot P')$ **by**(*simp add*: *name-per-rev*)
**from** *PRel EqvtRel′ RelRel′* **have** $((perm \cdot P'), (perm \cdot (rev\ perm \cdot Q'))) \in Rel'$
**by**(*force intro*: *eqvtRelI*)
**hence** $((perm \cdot P'), Q') \in Rel'$ **by**(*simp add*: *name-per-rev*)
**with** *L1* **show** *?case* **by** *blast*
**qed**

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

**assumes** $Id \subseteq Rel$

**shows** $P \rightsquigarrow<Rel> P$
**using** *assms*
**by**(*auto intro*: *Weak-Early-Step-Semantics.singleActionChain*
*simp add*: *weakSimulation-def weakFreeTransition-def*)

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

**assumes** *QSimR*: $Q \rightsquigarrow<Rel'> R$
**and** *Eqvt*: *eqvt Rel*
**and** *Eqvt″*: *eqvt* $Rel''$
**and** *Trans*: $Rel\ O\ Rel' \subseteq Rel''$
**and** *Sim*: $\bigwedge S\ T.\ (S, T) \in Rel \Longrightarrow S \rightsquigarrow<Rel> T$
**and** *PRelQ*: $(P, Q) \in Rel$

**shows** $P \rightsquigarrow<Rel''> R$
**proof** −
**from** *Eqvt″* **show** *?thesis*
**proof**(*induct rule*: *simCasesCont*[**where** $C=Q$])
**case**(*Bound a x R′*)
**have** *RTrans*: $R \longmapsto a<\nu x> \prec R'$ **by** *fact*
**from** $\langle x \sharp Q \rangle$ *QSimR RTrans* **obtain** $Q'$ **where** *QTrans*: $Q \Longrightarrow a<\nu x> \prec Q'$
**and** *Q′Rel′R′*: $(Q', R') \in Rel'$
**by**(*blast dest*: *simE*)

**from** *Sim Eqvt PRelQ QTrans* $\langle x \sharp P \rangle$
**obtain** $P'$ **where** *PTrans*: $P \Longrightarrow a<\nu x> \prec P'$ **and** *P′RelQ′*: $(P', Q') \in Rel$
**by**(*drule-tac simE2*) *auto*

**moreover from** $P'RelQ'$ $Q'Rel'R'$ *Trans* **have** $(P', R') \in Rel''$ **by** *blast*
**ultimately show** *?case* **by** *blast*
**next**
**case**(*Free* $\alpha$ $R'$)
**have** *RTrans*: $R \longmapsto \alpha \prec R'$ **by** *fact*
**with** *QSimR* **obtain** $Q'$ **where** *QTrans*: $Q \Longrightarrow\hat{}\ \alpha \prec Q'$ **and** $Q'RelR'$: $(Q', R') \in Rel'$
**by**(*blast dest*: *simE*)
**from** *Sim Eqvt PRelQ QTrans* **have** $\exists P'.\ P \Longrightarrow\hat{}\ \alpha \prec P' \wedge (P', Q') \in Rel$
**by**(*blast intro*: *simE2*)
**then obtain** $P'$ **where** *PTrans*: $P \Longrightarrow\hat{}\ \alpha \prec P'$ **and** $P'RelQ'$: $(P', Q') \in Rel$ **by** *blast*
**from** $P'RelQ'$ $Q'RelR'$ *Trans* **have** $(P', R') \in Rel''$ **by** *blast*
**with** *PTrans* **show** *?case* **by** *blast*
**qed**
**qed**

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

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

**shows** $P \leadsto<Rel''>\ R$
**proof** −
**from** $Eqvt''$ **show** *?thesis*
**proof**(*induct rule*: *simCasesCont*[**where** $C=Q$])
**case**(*Bound a x* $R'$)
**have** *RTrans*: $R \longmapsto a<\nu x> \prec R'$ **by** *fact*
**from** *QSimR RTrans* ‹$x \sharp Q$› **obtain** $Q'$ **where** *QTrans*: $Q \longmapsto a<\nu x> \prec Q'$
**and** $Q'Rel'R'$: $(Q', R') \in Rel'$
**by**(*blast dest*: *Strong-Early-Sim.elim*)

**with** *PSimQ QTrans* ‹$x \sharp P$› **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow a<\nu x> \prec P'$ **and** $P'RelQ'$: $(P', Q') \in Rel$
**by**(*blast dest*: *simE*)
**moreover from** $P'RelQ'$ $Q'Rel'R'$ *Trans* **have** $(P', R') \in Rel''$ **by** *blast*
**ultimately show** *?case* **by** *blast*
**next**
**case**(*Free* $\alpha$ $R'$)
**have** *RTrans*: $R \longmapsto \alpha \prec R'$ **by** *fact*
**with** *QSimR* **obtain** $Q'$ **where** *QTrans*: $Q \longmapsto \alpha \prec Q'$ **and** $Q'RelR'$: $(Q', R')$

$\in$ *Rel'*
      **by**(*blast dest*: *Strong-Early-Sim.elim*)
    **from** *PSimQ QTrans* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow\hat{}\ \alpha \prec P'$ **and** *P'RelQ'*: $(P', Q') \in$ *Rel*
      **by**(*blast dest*: *simE*)
    **from** *P'RelQ' Q'RelR' Trans* **have** $(P', R') \in$ *Rel''* **by** *blast*
    **with** *PTrans* **show** *?case* **by** *blast*
  **qed**
**qed**

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

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

  **shows** $P \leadsto$<*Rel*> $Q$
**proof**(*induct rule*: *simCases*)
  **case**(*Bound* $Q'$ *a x*)
  **have** $Q \longmapsto a<\nu x> \prec Q'$ **by** *fact*
  **with** *PSimQ* ‹$x \sharp P$› **obtain** $P'$ **where** *PTrans*: $P \longmapsto a<\nu x> \prec P'$ **and** *P'RelQ'*: $(P', Q') \in$ *Rel*
    **by**(*blast dest*: *Strong-Early-Sim.elim*)
  **from** *PTrans* **have** $P \Longrightarrow a<\nu x> \prec P'$
    **by**(*force intro*: *Weak-Early-Step-Semantics.singleActionChain simp add*: *weak-FreeTransition-def*)
  **with** *P'RelQ'* **show** *?case* **by** *blast*
**next**
  **case**(*Free* $Q'$ $\alpha$)
  **have** $Q \longmapsto \alpha \prec Q'$ **by** *fact*
  **with** *PSimQ* **obtain** $P'$ **where** *PTrans*: $P \longmapsto \alpha \prec P'$ **and** *P'RelQ'*: $(P', Q') \in$ *Rel*
    **by**(*blast dest*: *Strong-Early-Sim.elim*)
  **from** *PTrans* **have** $P \Longrightarrow\hat{}\ \alpha \prec P'$ **by**(*rule Weak-Early-Semantics.singleActionChain*)
  **with** *P'RelQ'* **show** *?case* **by** *blast*
**qed**

**end**

**theory** *Weak-Early-Bisim*
  **imports** *Weak-Early-Sim Strong-Early-Bisim*
**begin**

**lemma** *monoAux*: $A \subseteq B \Longrightarrow P \leadsto$<*A*> $Q \longrightarrow P \leadsto$<*B*> $Q$
**by**(*auto intro*: *Weak-Early-Sim.monotonic*)

**coinductive-set** *weakBisim* :: (*pi* $\times$ *pi*) *set*
**where**

*step*: $[\![P \rightsquigarrow<weakBisim> Q; (Q, P) \in weakBisim]\!] \Longrightarrow (P, Q) \in weakBisim$
**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*: $\bigwedge P\ Q.\ (P, Q) \in X \Longrightarrow P \rightsquigarrow<(X \cup weakBisim)> Q \wedge (Q, P) \in X \cup weakBisim$

**shows** $P \approx Q$
**proof** −
**have** *aux*: $X \cup weakBisim = \{(P, Q).\ (P, Q) \in X \vee P \approx Q\}$ **by** *blast*

**from** *p* **show** *?thesis*
**by**(*coinduct*, *force dest*: *step simp add*: *aux*)
**qed**

**lemma** *weakBisimWeakCoinductAux*[*case-names weakBisim*, *case-conclusion weakBisim step*, *consumes 1*]:
**assumes** *p*: $(P, Q) \in X$
**and** *step*: $\bigwedge P\ Q.\ (P, Q) \in X \Longrightarrow P \rightsquigarrow<X> Q \wedge (Q, P) \in X$

**shows** $P \approx Q$
**using** *p*
**proof**(*coinduct rule*: *weakBisimCoinductAux*)
**case** (*weakBisim P*)
**from** *step*[*OF this*] **show** *?case* **using** *Weak-Early-Sim.monotonic* **by** *blast*
**qed**

**lemma** *weakBisimCoinduct*[*consumes 1*, *case-names cSim cSym*]:
**fixes** *P* :: *pi*
**and** *Q* :: *pi*
**and** *X* :: (*pi* × *pi*) *set*

**assumes** $(P, Q) \in X$
**and** $\bigwedge R\ S.\ (R, S) \in X \Longrightarrow R \rightsquigarrow<(X \cup weakBisim)> S$
**and** $\bigwedge R\ S.\ (R, S) \in X \Longrightarrow (S, R) \in X$

**shows** $P \approx Q$
**using** *assms*
**by**(*coinduct rule*: *weakBisimCoinductAux*) *auto*

**lemma** *weakBisimWeakCoinduct*[*consumes 1*, *case-names cSim cSym*]:
**fixes** *P* :: *pi*
**and** *Q* :: *pi*
**and** *X* :: (*pi* × *pi*) *set*

**assumes** $(P, Q) \in X$
**and** $\bigwedge P\ Q.\ (P, Q) \in X \Longrightarrow P \leadsto$<$X$> $Q$
**and** $\bigwedge P\ Q.\ (P, Q) \in X \Longrightarrow (Q, P) \in X$

**shows** $P \approx Q$
**using** *assms*
**by**(*coinduct rule*: *weakBisimWeakCoinductAux*) *auto*

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

**assumes** $P \approx Q$

**shows** $P \leadsto$<*weakBisim*> $Q$
**and** $Q \approx P$
**using** *assms*
**by**(*auto dest*: *weakBisim.cases*)

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

**assumes** $P \leadsto$<*weakBisim*> $Q$
**and** $Q \approx P$

**shows** $P \approx Q$
**using** *assms*
**by**(*auto intro*: *weakBisim.intros*)

**lemma** *eqvt*[*simp*]:
**shows** *eqvt weakBisim*
**proof**(*auto simp add*: *eqvt-def*)
**let** *?X* = $\{x.\ \exists P\ Q$ (*perm*::*name prm*). $P \approx Q \wedge x = (perm \cdot P, perm \cdot Q)\}$
**fix** $P$ $Q$
**fix** *perm*::*name prm*
**assume** *PBiSimQ*: $P \approx Q$

**hence** $(perm \cdot P, perm \cdot Q) \in$ *?X* **by** *blast*
**moreover have** $\bigwedge P\ Q$ *perm*::*name prm*. $[\![P \leadsto$<*weakBisim*> $Q]\!] \Longrightarrow (perm \cdot P)$ $\leadsto$<*?X*> $(perm \cdot Q)$
**proof** −
**fix** $P$ $Q$
**fix** *perm*::*name prm*
**assume** $P \leadsto$<*weakBisim*> $Q$

**moreover have** *weakBisim* $\subseteq$ *?X*
**proof**(*auto*)

**fix** $P$ $Q$
**assume** $P \approx Q$
**moreover have** $P = ([]::name\ prm) \cdot P$ **and** $Q = ([]::name\ prm) \cdot Q$ **by** *auto*
**ultimately show** $\exists P'\ Q'.\ P' \approx Q' \wedge (\exists (perm::name\ prm).\ P = perm \cdot P' \wedge Q = perm \cdot Q')$
**by** *blast*
**qed**

**moreover have** *eqvt ?X*
**proof**(*auto simp add*: *eqvt-def*)
**fix** $P$ $Q$
**fix** *perm1*::*name prm*
**fix** *perm2*::*name prm*

**assume** $P \approx Q$
**moreover have** $perm1 \cdot perm2 \cdot P = (perm1\ @\ perm2) \cdot P$ **by**(*simp add*: *pt2*[*OF pt-name-inst*])
**moreover have** $perm1 \cdot perm2 \cdot Q = (perm1\ @\ perm2) \cdot Q$ **by**(*simp add*: *pt2*[*OF pt-name-inst*])

**ultimately show** $\exists P'\ Q'.\ P' \approx Q' \wedge (\exists (perm::name\ prm).\ perm1 \cdot perm2 \cdot P = perm \cdot P' \wedge$
$perm1 \cdot perm2 \cdot Q = perm \cdot Q')$
**by** *blast*
**qed**

**ultimately show** $(perm \cdot P) \rightsquigarrow$<*?X*> $(perm \cdot Q)$
**by**(*rule Weak-Early-Sim.eqvtI*)
**qed**

**ultimately show** $(perm \cdot P) \approx (perm \cdot Q)$ **by**(*coinduct rule*: *weakBisimWeakCoinductAux*, *blast dest*: *weakBisimE*)
**qed**

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

**assumes** $P \approx Q$

**shows** $(perm \cdot P) \approx (perm \cdot Q)$
**using** *assms*
**by**(*rule eqvtRelI*[*OF eqvt*])

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

```
  assumes P ∼ Q

  shows P ≈ Q
proof −
  from ‹P ∼ Q› show ?thesis
  proof(coinduct rule: weakBisimWeakCoinduct)
    case(cSim P Q)
    from ‹P ∼ Q› have P ⇝[bisim] Q by(rule bisimE)
    thus P ⇝<bisim> Q by(rule strongSimWeakSim)
  next
    case(cSym P Q)
    thus ?case by(rule bisimE)
  qed
qed

lemma reflexive:
  fixes P :: pi

  shows P ≈ P
proof −
  have (P, P) ∈ Id by simp
  thus ?thesis
    by(coinduct rule: weakBisimCoinduct) (auto intro: Weak-Early-Sim.reflexive)
qed

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

  assumes P ≈ Q

  shows Q ≈ P
using assms
by(auto dest: weakBisimE)

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

  assumes P ≈ Q
  and     Q ≈ R

  shows P ≈ R
proof −
  let ?X = weakBisim O weakBisim
  from assms have (P, R) ∈ ?X by blast
  thus ?thesis
```

**proof**(*coinduct rule*: *weakBisimCoinduct*)
**case**(*cSim P R*)
**from** ‹$(P, R) \in ?X$› **obtain** $Q$ **where** $P \approx Q$ **and** $Q \approx R$ **by** *auto*
**from** ‹$Q \approx R$› **have** $Q \leadsto$<*weakBisim*> $R$ **by**(*rule weakBisimE*)
**moreover have** *eqvt ?X* **by** *auto*
**moreover have** $?X \subseteq ?X$ **by** *simp*
**ultimately show** $P \leadsto$<($?X \cup$ *weakBisim*)> $R$ **using** *weakBisimE*(*1*) ‹$P \approx Q$›
**by**(*rule-tac Weak-Early-Sim.transitive*) *auto*
**next**
**case**(*cSym P R*)
**thus** *?case* **by**(*auto dest*: *symetric*)
**qed**
**qed**

**lemma** *weakBisimWeakUpto*[*case-names cSim cSym*, *consumes 1*]:
**assumes** *p*: $(P, Q) \in X$
**and** *Eqvt*: *eqvt X*
**and** *rSim*: $\bigwedge P\ Q.\ (P, Q) \in X \Longrightarrow P \leadsto$<(*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** −
**let** *?X* = *weakBisim O X O weakBisim*
**let** *?Y* = *weakBisim O X O bisim*
**from** *Eqvt eqvt* **have** *eqvt ?X* **by** *blast*
**from** *Strong-Early-Bisim.eqvt Eqvt eqvt* **have** *eqvt ?Y* **by** *blast*

**from** ‹$(P, Q) \in X$› **have** $(P, Q) \in ?X$ **by**(*blast intro*: *Strong-Early-Bisim.reflexive reflexive*)
**thus** *?thesis*
**proof**(*coinduct rule*: *weakBisimWeakCoinduct*)
**case**(*cSim P Q*)
**{**
**fix** $P\ P'\ Q'\ Q$
**assume** $P \approx P'$ **and** $(P', Q') \in X$ **and** $Q' \approx Q$
**from** ‹$Q' \approx Q$› **have** $Q' \leadsto$<*weakBisim*> $Q$ **by**(*rule weakBisimE*)
**moreover note** ‹*eqvt ?Y*› ‹*eqvt ?X*›
**moreover have** *?Y O weakBisim* $\subseteq$ *?X* **by**(*blast dest*: *strongBisimWeakBisim transitive*)
**moreover {**
**fix** $P\ Q$
**assume** $(P, Q) \in ?Y$
**then obtain** $P'\ Q'$ **where** $P \approx P'$ **and** $(P', Q') \in X$ **and** $Q' \sim Q$ **by** *auto*
**from** ‹$(P', Q') \in X$› **have** $P' \leadsto$<*?Y*> $Q'$ **by**(*rule rSim*)
**moreover from** ‹$Q' \sim Q$› **have** $Q' \leadsto$[*bisim*] $Q$ **by**(*rule bisimE*)
**moreover note** ‹*eqvt ?Y*›
**moreover have** *?Y O bisim* $\subseteq$ *?Y* **by**(*auto dest*: *Strong-Early-Bisim.transitive*)
**ultimately have** $P' \leadsto$<*?Y*> $Q$ **by**(*rule strongAppend*)

**moreover note** ‹$P \approx P'$›
**moreover have** *weakBisim O ?Y* $\subseteq$ *?Y* **by**(*blast dest*: *transitive*)
**ultimately have** *P* ⇝<*?Y*> *Q* **using** *weakBisimE*(*1*) *eqvt* ‹*eqvt ?Y*›
**by**(*rule-tac Weak-Early-Sim.transitive*)
**}**
**moreover from** ‹($P'$, $Q'$) ∈ *X*› **have** ($P'$, $Q'$) ∈ *?Y* **by**(*blast intro*: *reflexive Strong-Early-Bisim.reflexive*)
**ultimately have** $P'$ ⇝<*?X*> *Q* **by**(*rule Weak-Early-Sim.transitive*)
**moreover note** ‹$P \approx P'$›
**moreover have** *weakBisim O ?X* $\subseteq$ *?X* **by**(*blast dest*: *transitive*)
**ultimately have** *P* ⇝<*?X*> *Q* **using** *weakBisimE*(*1*) *eqvt* ‹*eqvt ?X*›
**by**(*rule-tac Weak-Early-Sim.transitive*)
**}**
**with** ‹(*P*, *Q*) ∈ *?X*› **show** *?case* **by** *auto*
**next**
**case**(*cSym P Q*)
**thus** *?case*
**apply** *auto*
**by**(*blast dest*: *bisimE rSym weakBisimE*)
**qed**
**qed**

**lemma** *weakBisimUpto*[*case-names cSim cSym*, *consumes 1*]:
**assumes** *p*: (*P*, *Q*) ∈ *X*
**and** *Eqvt*: *eqvt X*
**and** *rSim*: $\bigwedge$*R S*. (*R*, *S*) ∈ *X* ⟹ *R* ⇝<(*weakBisim O* (*X* ∪ *weakBisim*) *O bisim*)> *S*
**and** *rSym*: $\bigwedge$*R S*. (*R*, *S*) ∈ *X* ⟹ (*S*, *R*) ∈ *X*

**shows** $P \approx Q$
**proof** −
**from** *p* **have** (*P*, *Q*) ∈ *X* ∪ *weakBisim* **by** *simp*
**thus** *?thesis* **using** *Eqvt*
**apply**(*coinduct rule*: *weakBisimWeakUpto*)
**apply**(*auto dest*: *rSim rSym weakBisimE*)
**apply**(*rule Weak-Early-Sim.monotonic*)
**apply**(*blast dest*: *weakBisimE*)
**apply**(*auto simp add*: *relcomp-unfold*)
**by**(*metis reflexive Strong-Early-Bisim.reflexive transitive*)
**qed**

**lemma** *transitive-coinduct-weak*[*case-names cSim cSym*, *consumes 2*]:
**assumes** *p*: (*P*, *Q*) ∈ *X*
**and** *Eqvt*: *eqvt X*
**and** *rSim*: $\bigwedge$*P Q*. (*P*, *Q*) ∈ *X* ⟹ *P* ⇝<(*bisim O X O bisim*)> *Q*
**and** *rSym*: $\bigwedge$ *P Q*. (*P*, *Q*) ∈ *X* ⟹ (*Q*, *P*) ∈ *bisim O X O bisim*

**shows** $P \approx Q$

**proof** −
  **let** *?X* = *bisim O X O bisim*
  **from** ‹(*P*, *Q*) ∈ *X*› **have** (*P*, *Q*) ∈ *?X* **by**(*blast intro*: *Strong-Early-Bisim.reflexive*)
  **thus** *?thesis*
  **proof**(*coinduct rule*: *weakBisimWeakCoinduct*)
    **case**(*cSim P Q*)
    **{**
      **fix** $P$ $P'$ $Q'$ $Q$
      **assume** *PBisimP'*: $P \sim P'$
      **assume** *P'SimQ'*: $P' \leadsto$<*?X*> $Q'$
      **assume** *Q'SimQ*: $Q' \leadsto$[*bisim*] $Q$

      **have** $P \leadsto$<*?X*> $Q$
      **proof** −
        **have** $P' \leadsto$<*?X*> $Q$
        **proof** −
          **have** *?X O bisim* ⊆ *?X* **by**(*blast intro*: *Strong-Early-Bisim.transitive*)
          **moreover from** *Strong-Early-Bisim.eqvt Eqvt* **have** *eqvt ?X* **by** *blast*
          **ultimately show** *?thesis* **using** *P'SimQ' Q'SimQ*
            **by**(*rule-tac strongAppend*)
        **qed**
        **moreover have** *eqvt bisim* **by**(*rule Strong-Early-Bisim.eqvt*)
        **moreover from** *Strong-Early-Bisim.eqvt Eqvt* **have** *eqvt ?X* **by** *blast*
      **moreover have** *bisim O ?X* ⊆ *?X* **by**(*blast intro*: *Strong-Early-Bisim.transitive*)
          **moreover have** $\bigwedge P\ Q.\ P \sim Q \Longrightarrow P \leadsto$<*bisim*> $Q$ **by**(*blast dest*: *Strong-Early-Bisim.bisimE strongSimWeakSim*)
      **ultimately show** *?thesis* **using** *PBisimP'* **by**(*rule Weak-Early-Sim.transitive*)
      **qed**
    **}**
    **thus** *?case* **using** ‹(*P*, *Q*) ∈ *?X*› *rSim* **by** (*blast dest*: *Strong-Early-Bisim.bisimE*)
  **next**
    **case**(*cSym P Q*)
    **{**
      **fix** $P$ $P'$ $Q'$ $Q$
      **assume** $P \sim P'$ **and** $(P', Q') \in X$ **and** $Q' \sim Q$
      **from** ‹$(P', Q') \in X$› **have** $(Q', P') \in$ *?X* **by**(*rule rSym*)
      **with** ‹$P \sim P'$› ‹$Q' \sim Q$› **have** $(Q, P) \in$ *?X*
        **apply** *auto*
        **apply**(*drule-tac Strong-Early-Bisim.bisimE*(*2*))
        **apply**(*drule Strong-Early-Bisim.transitive*[**where** *Q*=*P'*])
        **apply** *assumption*
        **apply**(*drule-tac Strong-Early-Bisim.bisimE*(*2*))
        **apply**(*drule Strong-Early-Bisim.transitive*[**where** *Q*=*Q'*])
        **apply** *assumption*
        **by** *auto*
    **}**
    **thus** *?case* **using** ‹(*P*, *Q*) ∈ *?X*› **by** *auto*
  **qed**
**qed**

**end**

**theory** *Weak-Early-Step-Sim*
  **imports** *Weak-Early-Sim Strong-Early-Sim*
**begin**

**definition** *weakStepSimulation* :: *pi* ⇒ (*pi* × *pi*) *set* ⇒ *pi* ⇒ *bool* (‹- ⇝«-» -› [*80*, *80*, *80*] *80*) **where**
  *P* ⇝«*Rel*» *Q* ≡ (∀ *Q′ a x*. *Q* ⟼*a*<ν*x*> ≺ *Q′* ⟶ *x* ♯ *P* ⟶ (∃ *P′* . *P* ⟹*a*<ν*x*> ≺ *P′* ∧ (*P′*, *Q′*) ∈ *Rel*)) ∧
    (∀ *Q′ α*. *Q* ⟼*α* ≺ *Q′* ⟶ (∃ *P′*. *P* ⟹*α* ≺ *P′* ∧ (*P′*, *Q′*) ∈ *Rel*))

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

  **assumes** *P* ⇝«*A*» *P′*
  **and** *A* ⊆ *B*

  **shows** *P* ⇝«*B*» *P′*
**using** *assms*
**by**(*simp add*: *weakStepSimulation-def*) *blast*

**lemma** *simCasesCont*[*consumes 1*, *case-names Bound Free*]:
  **fixes** *P* :: *pi*
  **and** *Q* :: *pi*
  **and** *Rel* :: (*pi* × *pi*) *set*
  **and** *C* :: *′a*::*fs-name*

  **assumes** *Eqvt*: *eqvt Rel*
  **and** *Bound*: ⋀*a x Q′*. ⟦*x* ♯ *C*; *Q* ⟼ *a*<ν*x*> ≺ *Q′*⟧ ⟹ ∃ *P′*. *P* ⟹ *a*<ν*x*> ≺ *P′* ∧ (*P′*, *Q′*) ∈ *Rel*
  **and** *Free*: ⋀*α Q′*. *Q* ⟼ *α* ≺ *Q′* ⟹ ∃ *P′*. *P* ⟹ *α* ≺ *P′* ∧ (*P′*, *Q′*) ∈ *Rel*

  **shows** *P* ⇝«*Rel*» *Q*
**proof** −
  **from** *Free* **show** *?thesis*
  **proof**(*auto simp add*: *weakStepSimulation-def*)
    **fix** *Q′ a x*
    **assume** *xFreshP*: (*x*::*name*) ♯ *P*
    **assume** *Trans*: *Q* ⟼ *a*<ν*x*> ≺ *Q′*
    **have** ∃ *c*::*name*. *c* ♯ (*P*, *Q′*, *x*, *C*) **by**(*blast intro*: *name-exists-fresh*)
    **then obtain** *c*::*name* **where** *cFreshP*: *c* ♯ *P* **and** *cFreshQ′*: *c* ♯ *Q′* **and** *cFreshC*: *c* ♯ *C*
      **and** *cineqx*: *c* ≠ *x*

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

**from** *Trans cFreshQ'* **have** $Q \longmapsto a{<}\nu c{>} \prec ([(x, c)] \cdot Q')$ **by**(*simp add*: *alphaBoundOutput*)
**with** *cFreshC* **have** $\exists P'. P \Longrightarrow a{<}\nu c{>} \prec P' \wedge (P', [(x, c)] \cdot Q') \in Rel$
**by**(*rule Bound*)
**then obtain** $P'$ **where** *PTrans*: $P \Longrightarrow a{<}\nu c{>} \prec P'$ **and** *P'RelQ'*: $(P', [(x, c)] \cdot Q') \in Rel$
**by** *blast*

**from** *PTrans* ‹$x \sharp P$› ‹$c \neq x$› **have** $P \Longrightarrow a{<}\nu x{>} \prec ([(x, c)] \cdot P')$
**by**(*simp add*: *weakTransitionAlpha name-swap*)
**moreover from** *Eqvt P'RelQ'* **have** $([(x, c)] \cdot P', [(x, c)] \cdot [(x, c)] \cdot Q') \in Rel$
**by**(*rule eqvtRelI*)
**with** ‹$c \neq x$› **have** $([(x, c)] \cdot P', Q') \in Rel$
**by** *simp*
**ultimately show** $\exists P'. P \Longrightarrow a{<}\nu x{>} \prec P' \wedge (P', Q') \in Rel$ **by** *blast*
**qed**
**qed**

**lemma** *simCases*[*consumes 0*, *case-names Bound Free*]:
**fixes** $P$ :: *pi*
**and** $Q$ :: *pi*
**and** *Rel* :: (*pi* × *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*» $Q$
**using** *assms*
**by**(*auto simp add*: *weakStepSimulation-def*)

**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** $Q \longmapsto \alpha \prec Q' \Longrightarrow \exists P'. P \Longrightarrow \alpha \prec P' \wedge (P', Q') \in Rel$
**using** *assms* **by**(*simp add*: *weakStepSimulation-def*)+

**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** −
  **assume** *QTrans*: $Q \Longrightarrow a<\nu x> \prec Q'$
  **assume** $x \sharp P$

  **from** *QTrans* **obtain** $Q''\ Q'''$ **where** *QChain*: $Q \Longrightarrow_\tau Q''$
    **and** $Q''$*Trans*: $Q'' \longmapsto a<\nu x> \prec Q'''$
    **and** $Q'''$*Chain*: $Q''' \Longrightarrow_\tau Q'$
    **by**(*blast dest*: *transitionE*)

  **from** *QChain PRelQ Sim* **have** $\exists P''.\ P \Longrightarrow_\tau P'' \wedge (P'',\ Q'') \in Rel$
    **by**(*rule weakSimTauChain*)
  **then obtain** $P''$ **where** *PChain*: $P \Longrightarrow_\tau P''$ **and** $P''$*RelQ*$''$: $(P'',\ Q'') \in Rel$ **by** *blast*
  **from** *PChain* ‹$x \sharp P$› **have** *xFreshP*$''$: $x \sharp P''$ **by**(*rule freshChain*)

  **from** $P''$*RelQ*$''$ **have** $P'' \leadsto<Rel>\ Q''$ **by**(*rule Sim*)
  **with** $Q''$*Trans xFreshP*$''$ **obtain** $P'''$ **where** $P''$*Trans*: $P'' \Longrightarrow a<\nu x> \prec P'''$
    **and** $P'''$*RelQ*$'''$: $(P''',\ Q''') \in Rel$
    **by**(*blast dest*: *Weak-Early-Sim.simE*)

  **have** $\exists P'.\ P''' \Longrightarrow_\tau P' \wedge (P',\ Q') \in Rel$ **using** $Q'''$*Chain* $P'''$*RelQ*$'''$ *Sim*
    **by**(*rule weakSimTauChain*)
  **then obtain** $P'$ **where** $P'''$*Chain*: $P''' \Longrightarrow_\tau P'$ **and** $P'$*RelQ*$'$: $(P',\ Q') \in Rel$ **by** *blast*

  **from** *PChain* $P''$*Trans* $P'''$*Chain* **have** $P \Longrightarrow a<\nu x> \prec P'$
    **by**(*blast dest*: *Weak-Early-Step-Semantics.chainTransitionAppend*)
  **with** $P'$*RelQ*$'$ **show** $\exists P'.\ P \Longrightarrow a<\nu x> \prec P' \wedge (P',\ Q') \in Rel$
    **by** *blast*
**next**
  **assume** $Q \Longrightarrow \alpha \prec Q'$

  **then obtain** $Q''\ Q'''$ **where** *QChain*: $Q \Longrightarrow_\tau Q''$

and $Q''Trans$: $Q'' \longmapsto \alpha \prec Q'''$
and $Q'''Chain$: $Q''' \Longrightarrow_\tau Q'$
**by**(*blast dest*: *transitionE*)
**from** *QChain Q''Trans Q'''Chain* **show** $\exists P'. P \Longrightarrow\alpha \prec P' \wedge (P', Q') \in Rel$
**proof**(*induct arbitrary*: $\alpha$ $Q'''$ $Q'$ *rule*: *tauChainInduct*)
**case** *id*
**from** *PSimQ* $\langle Q \longmapsto\alpha \prec Q'''\rangle$ **have** $\exists P'. P \Longrightarrow\alpha \prec P' \wedge (P', Q''') \in Rel$
**by**(*blast dest*: *simE*)
**then obtain** $P'''$ **where** *PTrans*: $P \Longrightarrow\alpha \prec P'''$ **and** $P'RelQ'''$: $(P''', Q''') \in Rel$
**by** *blast*

**have** $\exists P'. P''' \Longrightarrow_\tau P' \wedge (P', Q') \in Rel$ **using** $\langle Q''' \Longrightarrow_\tau Q'\rangle$ $P'RelQ'''$ *Sim*
**by**(*rule Weak-Early-Sim.weakSimTauChain*)
**then obtain** $P'$ **where** $P'''Chain$: $P''' \Longrightarrow_\tau P'$ **and** $P'RelQ'$: $(P', Q') \in Rel$ **by** *blast*

**from** *P'''Chain PTrans* **have** $P \Longrightarrow\alpha \prec P'$
**by**(*blast dest*: *Weak-Early-Step-Semantics.chainTransitionAppend*)

**with** $P'RelQ'$ **show** *?case* **by** *blast*
**next**
**case**(*ih* $Q''''$ $Q''$ $\alpha$ $Q'''$ $Q'$)
**have** $Q'' \Longrightarrow_\tau Q''$ **by** *simp*
**with** $\langle Q'''' \longmapsto\tau \prec Q''\rangle$ **obtain** $P''$ **where** *PChain*: $P \Longrightarrow\tau \prec P''$ **and** $P''RelQ''$: $(P'', Q'') \in Rel$
**by**(*drule-tac ih*) *auto*

**from** $P''RelQ''$ **have** $P'' \rightsquigarrow$<*Rel*> $Q''$ **by**(*rule Sim*)
**hence** $\exists P'''. P'' \Longrightarrow\hat{}\alpha \prec P''' \wedge (P''', Q''') \in Rel$ **using** $\langle Q'' \longmapsto\alpha \prec Q'''\rangle$
**by**(*rule Weak-Early-Sim.simE*)
**then obtain** $P'''$ **where** $P''Trans$: $P'' \Longrightarrow\hat{}\alpha \prec P'''$
**and** $P'''RelQ'''$: $(P''', Q''') \in Rel$
**by** *blast*
**from** $\langle Q''' \Longrightarrow_\tau Q'\rangle$ $P'''RelQ'''$ *Sim* **have** $\exists P'. P''' \Longrightarrow_\tau P' \wedge (P', Q') \in Rel$
**by**(*rule Weak-Early-Sim.weakSimTauChain*)
**then obtain** $P'$ **where** $P'''Chain$: $P''' \Longrightarrow_\tau P'$
**and** $P'RelQ'$: $(P', Q') \in Rel$
**by** *blast*
**from** *PChain P''Trans* **have** $P \Longrightarrow\alpha \prec P'''$
**apply**(*auto simp add*: *freeTransition-def weakFreeTransition-def*)
**apply**(*drule tauActTauChain*, *auto*)
**by**(*rule-tac x=P'''aa* **in** *exI*) *auto*
**hence** $P \Longrightarrow\alpha \prec P'$ **using** *P'''Chain*
**by**(*rule Weak-Early-Step-Semantics.chainTransitionAppend*)
**with** $P'RelQ'$ **show** *?case* **by** *blast*
**qed**
**qed**

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

 **assumes** *PSimQ*: $P \leadsto$«*Rel*» $Q$
 **and** *RelRel'*: $Rel \subseteq Rel'$
 **and** *EqvtRel'*: *eqvt Rel'*

 **shows** $(perm \cdot P) \leadsto$«*Rel'*» $(perm \cdot Q)$
**proof**(*induct rule*: *simCases*)
 **case**(*Bound a x Q'*)
 **have** *xFreshP*: $x \sharp perm \cdot P$ **by** *fact*
 **have** *QTrans*: $(perm \cdot Q) \longmapsto a{<}\nu x{>} \prec Q'$ **by** *fact*

 **hence** $(rev\ perm \cdot (perm \cdot Q)) \longmapsto rev\ perm \cdot (a{<}\nu x{>} \prec Q')$ **by**(*rule eqvt*)
 **hence** $Q \longmapsto (rev\ perm \cdot a){<}\nu(rev\ perm \cdot x){>} \prec (rev\ perm \cdot Q')$
 **by**(*simp add*: *name-rev-per*)
 **moreover from** *xFreshP* **have** $(rev\ perm \cdot x) \sharp P$ **by**(*simp add*: *name-fresh-left*)
 **ultimately obtain** $P'$ **where** *PTrans*: $P \Longrightarrow (rev\ perm \cdot a){<}\nu(rev\ perm \cdot x){>}$
$\prec P'$
 **and** *P'RelQ'*: $(P', rev\ perm \cdot Q') \in Rel$ **using** *PSimQ*
 **by**(*blast dest*: *simE*)

 **from** *PTrans* **have** $(perm \cdot P) \Longrightarrow (perm \cdot rev\ perm \cdot a){<}\nu(perm \cdot rev\ perm \cdot$
$x){>} \prec perm \cdot P'$
 **by**(*rule Weak-Early-Step-Semantics.eqvtI*)
 **hence** *L1*: $(perm \cdot P) \Longrightarrow a{<}\nu x{>} \prec (perm \cdot P')$ **by**(*simp add*: *name-per-rev*)
 **from** *P'RelQ' RelRel'* **have** $(P', rev\ perm \cdot Q') \in Rel'$ **by** *blast*
 **with** *EqvtRel'* **have** $(perm \cdot P', perm \cdot (rev\ perm \cdot Q')) \in Rel'$
 **by**(*rule eqvtRelI*)
 **hence** $(perm \cdot P', Q') \in Rel'$ **by**(*simp add*: *name-per-rev*)
 **with** *L1* **show** *?case* **by** *blast*
**next**
 **case**(*Free* $\alpha$ *Q'*)
 **have** *QTrans*: $(perm \cdot Q) \longmapsto \alpha \prec Q'$ **by** *fact*

 **hence** $(rev\ perm \cdot (perm \cdot Q)) \longmapsto rev\ perm \cdot (\alpha \prec Q')$ **by**(*rule eqvts*)
 **hence** $Q \longmapsto (rev\ perm \cdot \alpha) \prec (rev\ perm \cdot Q')$ **by**(*simp add*: *name-rev-per*)
 **with** *PSimQ* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow (rev\ perm \cdot \alpha) \prec P'$
 **and** *PRel*: $(P', (rev\ perm \cdot Q')) \in Rel$
 **by**(*blast dest*: *simE*)

 **from** *PTrans* **have** $(perm \cdot P) \Longrightarrow (perm \cdot rev\ perm \cdot \alpha) \prec perm \cdot P'$
 **by**(*rule Weak-Early-Step-Semantics.eqvtI*)
 **hence** *L1*: $(perm \cdot P) \Longrightarrow \alpha \prec (perm \cdot P')$ **by**(*simp add*: *name-per-rev*)
 **from** *PRel EqvtRel' RelRel'* **have** $((perm \cdot P'), (perm \cdot (rev\ perm \cdot Q'))) \in Rel'$
 **by**(*force intro*: *eqvtRelI*)

```
  hence ((perm • P′), Q′) ∈ Rel′ by(simp add: name-per-rev)
  with L1 show ?case by blast
qed



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

  assumes Id ⊆ Rel

  shows P ⇝«Rel» P
using assms
by(auto intro: Weak-Early-Step-Semantics.singleActionChain
   simp add: weakStepSimulation-def weakFreeTransition-def)

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: ⋀S T. (S, T) ∈ Rel ⟹ S ⇝<Rel> T
  and     PRelQ: (P, Q) ∈ Rel

  shows P ⇝«Rel″» R
proof -
  from Eqvt″ show ?thesis
  proof(induct rule: simCasesCont[of - (P, Q)])
    case(Bound a x R′)
    have x ♯ (P, Q) by fact
    hence xFreshP: x ♯ P and xFreshQ: x ♯ Q by(simp add: fresh-prod)+
    have RTrans: R ⟼a<νx> ≺ R′ by fact
    from xFreshQ QSimR RTrans obtain Q′ where QTrans: Q ⟹ a<νx> ≺ Q′
                                      and Q′Rel′R′: (Q′, R′) ∈ Rel′
      by(blast dest: simE)

    with PSimQ Sim Eqvt PRelQ QTrans xFreshP have ∃P′. P ⟹ a<νx> ≺ P′
∧ (P′, Q′) ∈ Rel
      by(blast intro: simE2)
    then obtain P′ where PTrans: P ⟹ a<νx> ≺ P′ and P′RelQ′: (P′, Q′) ∈
```

*Rel* **by** *blast*
    **moreover from** *P′RelQ′ Q′Rel′R′ Trans* **have** $(P', R') \in$ *Rel″* **by** *blast*
    **ultimately show** *?case* **by** *blast*
  **next**
    **case**(*Free* $\alpha$ *R′*)
    **have** *RTrans*: $R \longmapsto \alpha \prec R'$ **by** *fact*
    **with** *QSimR* **obtain** *Q′* **where** *QTrans*: $Q \Longrightarrow \alpha \prec Q'$ **and** *Q′RelR′*: $(Q', R') \in$ *Rel′*
      **by**(*blast dest*: *simE*)
    **from** *PSimQ Sim Eqvt PRelQ QTrans* **have** $\exists P'. P \Longrightarrow \alpha \prec P' \land (P', Q') \in$ *Rel*
      **by**(*blast intro*: *simE2*)
    **then obtain** *P′* **where** *PTrans*: $P \Longrightarrow \alpha \prec P'$ **and** *P′RelQ′*: $(P', Q') \in$ *Rel* **by** *blast*
    **from** *P′RelQ′ Q′RelR′ Trans* **have** $(P', R') \in$ *Rel″* **by** *blast*
    **with** *PTrans* **show** *?case* **by** *blast*
  **qed**
**qed**

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

  **assumes** *PSimQ*: *P* ⇝[*Rel*] *Q*

  **shows** *P* ⇝«*Rel*» *Q*
**proof**(*induct rule*: *simCases*)
  **case**(*Bound a x Q′*)
  **have** $Q \longmapsto a<\nu x> \prec Q'$ **and** $x \sharp P$ **by** *fact+*
  **with** *PSimQ* **obtain** *P′* **where** *PTrans*: $P \longmapsto a<\nu x> \prec P'$ **and** *P′RelQ′*: $(P', Q') \in$ *Rel*
    **by**(*blast dest*: *Strong-Early-Sim.elim*)
  **from** *PTrans* **have** $P \Longrightarrow a<\nu x> \prec P'$
    **by**(*force intro*: *Weak-Early-Step-Semantics.singleActionChain simp add*: *weak-FreeTransition-def*)
  **with** *P′RelQ′* **show** *?case* **by** *blast*
**next**
  **case**(*Free* $\alpha$ *Q′*)
  **have** $Q \longmapsto \alpha \prec Q'$ **by** *fact*
  **with** *PSimQ* **obtain** *P′* **where** *PTrans*: $P \longmapsto \alpha \prec P'$ **and** *P′RelQ′*: $(P', Q') \in$ *Rel*
    **by**(*blast dest*: *Strong-Early-Sim.elim*)
  **from** *PTrans* **have** $P \Longrightarrow \alpha \prec P'$ **by**(*rule Weak-Early-Step-Semantics.singleActionChain*)
  **with** *P′RelQ′* **show** *?case* **by** *blast*
**qed**

**lemma** *weakSimWeakEqSim*:
  **fixes** *P* :: *pi*

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

**assumes** *P* ⇝«*Rel*» *Q*

**shows** *P* ⇝<*Rel*> *Q*
**using** *assms*
**by**(*force simp add*: *weakStepSimulation-def Weak-Early-Sim.weakSimulation-def weak-FreeTransition-def*)

**end**

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

**definition** *weakCongruence* :: *pi* ⇒ *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*
**using** *assms*
**by**(*auto simp add*: *weakCongruence-def*)

**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*
**using** *assms*
**by**(*auto simp add*: *weakCongruence-def*)

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

**assumes** *P* ≃ *Q*

**shows** $P \rightsquigarrow\llangle weakBisim\rrangle\ Q$
**and** $Q \rightsquigarrow\llangle weakBisim\rrangle\ P$
**using** *assms*
**by**(*auto simp add*: *weakCongruence-def*)

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

**assumes** $P \rightsquigarrow\llangle weakBisim\rrangle\ Q$
**and** $Q \rightsquigarrow\llangle weakBisim\rrangle\ P$

**shows** $P \simeq Q$
**using** *assms*
**by**(*auto simp add*: *weakCongruence-def*)

**lemma** *eqvtI*[*eqvt*]:
**fixes** $P :: pi$
**and** $Q :: pi$
**and** $p :: name\ prm$

**assumes** $P \simeq Q$

**shows** $(p \bullet P) \simeq (p \bullet Q)$
**using** *assms*
**by**(*auto simp add*: *weakCongruence-def intro*: *eqvtI*)

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

**assumes** $P \sim Q$

**shows** $P \simeq Q$
**proof** −
**have** $\bigwedge P\ Q.\ P \rightsquigarrow[bisim]\ Q \Longrightarrow P \rightsquigarrow\llangle weakBisim\rrangle\ Q$
**proof** −
**fix** $P\ Q$
**assume** $P \rightsquigarrow[bisim]\ Q$
**hence** $P \rightsquigarrow\llangle bisim\rrangle\ Q$ **by**(*rule Weak-Early-Step-Sim.strongSimWeakSim*)
**moreover have** $bisim \subseteq weakBisim$
**by**(*auto intro*: *strongBisimWeakBisim*)
**ultimately show** $P \rightsquigarrow\llangle weakBisim\rrangle\ Q$ **by**(*rule Weak-Early-Step-Sim.monotonic*)
**qed**
**with** *assms* **show** *?thesis*
**by**(*blast intro*: *weakCongI dest*: *Strong-Early-Bisim.bisimE*)
**qed**

**lemma** *congruenceWeakBisim*:

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

**assumes** $P \simeq Q$

**shows** $P \approx Q$
**using** *assms*
**proof** $-$
**let** $?X = \{(P, Q) \mid P\ Q.\ P \simeq Q\}$
**from** *assms* **have** $(P, Q) \in ?X$ **by** *simp*
**thus** *?thesis*
**proof**(*induct rule*: *weakBisimCoinduct*)
**case**(*cSim P Q*)
**{**
**fix** $P\ Q$
**assume** $P \simeq Q$
**hence** $P \leadsto$«*weakBisim*» $Q$ **by**(*simp add*: *weakCongruence-def*)
**hence** $P \leadsto$«($?X \cup$ *weakBisim*)» $Q$ **by**(*rule-tac Weak-Early-Step-Sim.monotonic*)
*auto*
**hence** $P \leadsto$<($?X \cup$ *weakBisim*)> $Q$ **by**(*rule weakSimWeakEqSim*)
**}**
**with** ‹$(P, Q) \in ?X$› **show** *?case* **by** *auto*
**next**
**case**(*cSym P Q*)
**thus** *?case* **by**(*auto simp add*: *weakCongruence-def*)
**qed**
**qed**

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

**shows** $P \simeq P$
**proof** $-$
**from** *Weak-Early-Bisim.reflexive* **have** $\bigwedge P.\ P \leadsto$«*weakBisim*» $P$
**by**(*blast intro*: *Weak-Early-Step-Sim.reflexive*)
**thus** *?thesis*
**by**(*force simp add*: *substClosed-def weakCongruence-def*)
**qed**

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

**assumes** $P \simeq Q$

**shows** $Q \simeq P$
**using** *assms*
**by**(*force simp add*: *substClosed-def weakCongruence-def*)

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

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

**shows** $P \simeq R$
**proof** −
**have** *Goal*: $\bigwedge P\ Q\ R.\ [\![P \leadsto$«*weakBisim*» $Q$; $Q \leadsto$«*weakBisim*» $R$; $P \approx Q]\!] \Longrightarrow$
$P \leadsto$«*weakBisim*» $R$
**using** *Weak-Early-Bisim.eqvt Weak-Early-Bisim.weakBisimE Weak-Early-Bisim.transitive*
**by**(*blast intro*: *Weak-Early-Step-Sim.transitive*)
**from** *assms* **show** *?thesis*
**apply**(*simp add*: *weakCongruence-def*) **using** *assms*
**by**(*blast intro*: *Goal dest*: *congruenceWeakBisim symetric*)
**qed**

**end**

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

**consts** *weakBisimSubst* :: (*pi* × *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** $P \approx Q$
**using** *assms substClosedSubset*
**by** *blast*

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

**assumes** $P \sim^s Q$

**shows** $P \approx^s Q$
**using** *assms*
**by**(*auto simp add*: *substClosed-def intro*: *strongBisimWeakBisim*)

**lemma** *eqvt*:
  **shows** *eqvt* (*substClosed weakBisim*)
**by**(*rule eqvtSubstClosed*[*OF Weak-Early-Bisim.eqvt*])

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

  **assumes** $P \approx^s Q$

  **shows** $(perm \cdot P) \approx^s (perm \cdot Q)$
**using** *assms*
**by**(*rule eqvtRelI*[*OF eqvt*])

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

  **shows** $P \approx^s P$
**by**(*force simp add*: *substClosed-def intro*: *Weak-Early-Bisim.reflexive*)

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

  **assumes** $P \approx^s Q$

  **shows** $Q \approx^s P$
**using** *assms*
**by**(*force simp add*: *substClosed-def intro*: *Weak-Early-Bisim.symetric*)

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

  **assumes** $P \approx^s Q$
  **and** $Q \approx^s R$

  **shows** $P \approx^s R$
**using** *assms*
**by**(*force simp add*: *substClosed-def intro*: *Weak-Early-Bisim.transitive*)

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

  **assumes** $P \approx^s Q$

**shows** $P[<s>] \approx^s Q[<s>]$
**using** *assms*
**proof**(*auto simp add*: *substClosed-def*)
**fix** $s'$
**assume** $\forall s.\ P[<s>] \approx Q[<s>]$
**hence** $P[<(s@s')>] \approx Q[<(s@s')>]$ **by** *blast*
**moreover have** $P[<(s@s')>] = (P[<s>])[<s'>]$
**by**(*induct* $s'$, *auto*)
**moreover have** $Q[<(s@s')>] = (Q[<s>])[<s'>]$
**by**(*induct* $s'$, *auto*)

**ultimately show** $(P[<s>])[<s'>] \approx (Q[<s>])[<s'>]$
**by** *simp*
**qed**

**end**

**theory** *Weak-Early-Cong-Subst*
**imports** *Weak-Early-Cong Weak-Early-Bisim-Subst Strong-Early-Bisim-Subst*
**begin**

**consts** *congruenceSubst* :: (*pi* × *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* × *name*) *list*

**assumes** $P \simeq^s Q$

**shows** $P[<s>] \leadsto$«*weakBisim*» $Q[<s>]$
**and** $Q[<s>] \leadsto$«*weakBisim*» $P[<s>]$
**proof** −
**from** *assms* **show** $P[<s>] \leadsto$«*weakBisim*» $Q[<s>]$ **by**(*simp add*: *weakCongruenceSubst-def weakCongruence-def*)
**next**
**from** *assms* **show** $Q[<s>] \leadsto$«*weakBisim*» $P[<s>]$ **by**(*simp add*: *weakCongruenceSubst-def weakCongruence-def*)
**qed**

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

**assumes** $\bigwedge s.\ P[<s>] \leadsto$«*weakBisim*» $Q[<s>]$

**and** $\bigwedge s.\ Q[<s>] \rightsquigarrow«weakBisim»\ P[<s>]$

**shows** $P \simeq^s Q$
**using** *assms*
**by**(*simp add*: *weakCongruenceSubst-def weakCongruence-def*)

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

**assumes** $P \simeq^s Q$

**shows** $P \simeq Q$
**using** *assms*
**apply**(*simp add*: *weakCongruenceSubst-def weakCongruence-def*)
**apply**(*erule-tac x*=[] **in** *allE*)
**by** *auto*

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

**assumes** $P \simeq^s Q$

**shows** $(p \cdot P) \simeq^s (p \cdot Q)$
**proof**(*simp add*: *weakCongruenceSubst-def*, *rule allI*)
**fix** $s$
**from** *assms* **have** $P[<(rev\ p\ \cdot\ s)>] \simeq Q[<(rev\ p\ \cdot\ s)>]$ **by**(*auto simp add*: *weakCongruenceSubst-def*)
**thus** $(p \cdot P)[<s>] \simeq (p \cdot Q)[<s>]$ **by**(*drule-tac p*=*p* **in** *Weak-Early-Cong.eqvtI*) (*simp add*: *eqvts name-per-rev*)
**qed**

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

**assumes** $P \sim^s Q$

**shows** $P \simeq^s Q$
**using** *assms*
**by**(*auto intro*: *strongBisimWeakCong simp add*: *substClosed-def weakCongruence-Subst-def*)

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

**assumes** $P \simeq^s Q$

**shows** $P \approx^s Q$
**using** *assms*
**by**(*auto intro*: *congruenceWeakBisim simp add*: *substClosed-def weakCongruenceSubst-def*)

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

**shows** $P \simeq^s P$
**proof** −
**from** *Weak-Early-Bisim.reflexive* **have** $\bigwedge P.\ P \leadsto\langle\langle weakBisim\rangle\rangle\ P$
**by**(*blast intro*: *Weak-Early-Step-Sim.reflexive*)
**thus** *?thesis*
**by**(*force simp add*: *weakCongruenceSubst-def weakCongruence-def*)
**qed**

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

**assumes** $P \simeq^s Q$

**shows** $Q \simeq^s P$
**using** *assms* **by**(*auto simp add*: *weakCongruenceSubst-def weakCongruence-def*)

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

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

**shows** $P \simeq^s R$
**using** *assms* **by**(*auto simp add*: *weakCongruenceSubst-def intro*: *Weak-Early-Cong.transitive*)

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

**assumes** $P \simeq^s Q$

**shows** $P[<s>] \simeq^s Q[<s>]$
**using** *assms*
**proof**(*auto simp add*: *weakCongruenceSubst-def*)
**fix** $s'$

**assume** $\forall s.\ P[<s>] \simeq Q[<s>]$
**hence** $P[<(s@s')>] \simeq Q[<(s@s')>]$ **by** *blast*
**moreover have** $P[<(s@s')>] = (P[<s>])[<s'>]$
**by**(*induct* $s'$, *auto*)
**moreover have** $Q[<(s@s')>] = (Q[<s>])[<s'>]$
**by**(*induct* $s'$, *auto*)

**ultimately show** $(P[<s>])[<s'>] \simeq (Q[<s>])[<s'>]$
**by** *simp*
**qed**

**end**

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

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

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

**shows** $\tau.(P) \rightsquigarrow«Rel» \tau.(Q)$
**proof**(*induct rule*: *simCases*)
**case**(*Bound a x Q'*)
**have** $\tau.(Q) \longmapsto a<\nu x> \prec Q'$ **by** *fact*
**hence** *False* **by**(*induct rule*: *tauCases'*, *auto*)
**thus** *?case* **by** *simp*
**next**
**case**(*Free* $\alpha$ *Q'*)
**have** $\tau.(Q) \longmapsto(\alpha \prec Q')$ **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *tauCases'*, *auto simp add*: *pi.inject residual.inject*)
**have** $\tau.(P) \Longrightarrow \tau \prec P$ **by**(*rule Weak-Early-Step-Semantics.Tau*)
**with** *PRelQ* **show** $\exists P'.\ \tau.(P) \Longrightarrow \tau \prec P' \wedge (P', Q) \in Rel$ **by** *blast*
**qed**
**qed**

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

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

**and** *Eqvt*: *eqvt Rel*

**shows** $a{<}x{>}.P \leadsto «Rel» \ a{<}x{>}.Q$
**using** *Eqvt*
**proof**(*induct rule*: *simCasesCont*[**where** $C$=($x$, $a$, $P$, $Q$)])
**case**(*Bound b y Q'*)
**from** ‹$y \sharp (x, a, P, Q)$› **have** $y \neq x$ $y \neq a$ $y \sharp P$ $y \sharp Q$ **by** *simp+*
**from** ‹$a{<}x{>}.Q \longmapsto b{<}\nu y{>} \prec Q'$› ‹$y \neq a$› ‹$y \neq x$› ‹$y \sharp Q$› **show** *?case*
**by**(*erule-tac inputCases'*) *auto*
**next**
**case**(*Free* $\alpha$ *Q'*)
**from** ‹$a{<}x{>}.Q \longmapsto \alpha \prec Q'$›
**show** *?case*
**proof**(*induct rule*: *inputCases*)
**case**(*cInput u*)
**have** $a{<}x{>}.P \Longrightarrow (a{<}u{>}) \prec (P[x{::=}u])$
**by**(*rule Weak-Early-Step-Semantics.Input*)
**moreover from** *PRelQ* **have** $(P[x{::=}u], Q[x{::=}u]) \in Rel$ **by** *auto*
**ultimately show** *?case* **by** *blast*
**qed**
**qed**

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

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

**shows** $a\{b\}.P \leadsto «Rel» \ a\{b\}.Q$
**proof**(*induct rule*: *simCases*)
**case**(*Bound c x Q'*)
**have** $a\{b\}.Q \longmapsto c{<}\nu x{>} \prec Q'$ **by** *fact*
**hence** *False* **by**(*induct rule*: *outputCases'*, *auto*)
**thus** *?case* **by** *simp*
**next**
**case**(*Free* $\alpha$ *Q'*)
**have** $a\{b\}.Q \longmapsto \alpha \prec Q'$ **by** *fact*
**thus** $\exists P'.\ a\{b\}.P \Longrightarrow \alpha \prec P' \wedge (P', Q') \in Rel$
**proof**(*induct rule*: *outputCases'*, *auto simp add*: *pi.inject residual.inject*)
**have** $a\{b\}.P \Longrightarrow a[b] \prec P$ **by**(*rule Weak-Early-Step-Semantics.Output*)
**with** *PRelQ* **show** $\exists P'.\ a\{b\}.P \Longrightarrow a[b] \prec P' \wedge (P', Q) \in Rel$ **by** *blast*
**qed**
**qed**

**lemma** *matchPres*:

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

**assumes** $PSimQ$: $P \rightsquigarrow\text{«}Rel\text{»}\ Q$
**and** $RelRel'$: $Rel \subseteq Rel'$

**shows** $[a\frown b]P \rightsquigarrow\text{«}Rel'\text{»}\ [a\frown b]Q$
**proof**(*induct rule*: *simCases*)
**case**($Bound\ c\ x\ Q'$)
**have** $x \sharp [a\frown b]P$ **by** *fact*
**hence** $xFreshP$: $(x::name) \sharp P$ **by** *simp*
**have** $[a\frown b]Q \longmapsto c<\nu x> \prec Q'$ **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *matchCases*)
**case** *Match*
**have** $Q \longmapsto c<\nu x> \prec Q'$ **by** *fact*
**with** *PSimQ xFreshP* **obtain** $P'$ **where** $PTrans$: $P \Longrightarrow c<\nu x> \prec P'$
**and** $P'RelQ'$: $(P', Q') \in Rel$
**by**(*blast dest*: *simE*)
**from** *PTrans* **have** $[a\frown a]P \Longrightarrow c<\nu x> \prec P'$ **by**(*rule Weak-Early-Step-Semantics.Match*)
**moreover from** $P'RelQ'$ $RelRel'$ **have** $(P', Q') \in Rel'$ **by** *blast*
**ultimately show** *?case* **by** *blast*
**qed**
**next**
**case**($Free\ \alpha\ Q'$)
**have** $[a\frown b]Q \longmapsto\alpha \prec Q'$ **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *matchCases*)
**case** *Match*
**have** $Q \longmapsto \alpha \prec Q'$ **by** *fact*
**with** *PSimQ* **obtain** $P'$ **where** $PTrans$: $P \Longrightarrow\alpha \prec P'$ **and** $PRel$: $(P', Q') \in$
$Rel$
**by**(*blast dest*: *simE*)
**from** *PTrans* **have** $[a\frown a]P \Longrightarrow\alpha \prec P'$ **by**(*rule Weak-Early-Step-Semantics.Match*)
**with** $RelRel'$ *PRel* **show** *?case* **by** *blast*
**qed**
**qed**

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

**assumes** *PSimQ*: $P \rightsquigarrow$«*Rel*» $Q$
**and** *RelRel′*: $Rel \subseteq Rel'$

**shows** $[a{\neq}b]P \rightsquigarrow$«*Rel′*» $[a{\neq}b]Q$
**proof**(*induct rule*: *simCases*)
**case**(*Bound c x Q′*)
**have** $x \sharp [a{\neq}b]P$ **by** *fact*
**hence** *xFreshP*: $(x{::}name) \sharp P$ **by** *simp*
**have** $[a{\neq}b]Q \longmapsto c{<}\nu x{>} \prec Q'$ **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *mismatchCases*)
**case** *Mismatch*
**have** *aineqb*: $a \neq b$ **by** *fact*
**have** $Q \longmapsto c{<}\nu x{>} \prec Q'$ **by** *fact*
**with** *PSimQ xFreshP* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow c{<}\nu x{>} \prec P'$
**and** *P′RelQ′*: $(P', Q') \in Rel$
**by**(*blast dest*: *simE*)
**from** *PTrans aineqb* **have** $[a{\neq}b]P \Longrightarrow c{<}\nu x{>} \prec P'$ **by**(*rule Weak-Early-Step-Semantics.Mismatch*)
**moreover from** *P′RelQ′ RelRel′* **have** $(P', Q') \in Rel'$ **by** *blast*
**ultimately show** *?case* **by** *blast*
**qed**
**next**
**case**(*Free* $\alpha$ *Q′*)
**have** $[a{\neq}b]Q \longmapsto \alpha \prec Q'$ **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *mismatchCases*)
**case** *Mismatch*
**have** $Q \longmapsto \alpha \prec Q'$ **by** *fact*
**with** *PSimQ* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow \alpha \prec P'$ **and** *PRel*: $(P', Q') \in$
*Rel*
**by**(*blast dest*: *simE*)
**from** *PTrans* ‹$a \neq b$› **have** $[a{\neq}b]P \Longrightarrow \alpha \prec P'$ **by**(*rule Weak-Early-Step-Semantics.Mismatch*)
**with** *RelRel′ PRel* **show** *?case* **by** *blast*
**qed**
**qed**

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

**assumes** *PSimQ*: $P \rightsquigarrow$«*Rel*» $Q$
**and** *RelRel′*: $Rel \subseteq Rel'$
**and** *C*: $Id \subseteq Rel'$

**shows** $P \oplus R \rightsquigarrow$«*Rel′*» $Q \oplus R$
**proof**(*induct rule*: *simCases*)
**case**(*Bound a x Q′*)

**have** $x \sharp P \oplus R$ **by** *fact*
**hence** *xFreshP*: $(x::name) \sharp P$ **and** *xFreshR*: $x \sharp R$ **by** *simp+*
**have** $Q \oplus R \longmapsto a<\nu x> \prec Q'$ **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *sumCases*)
**case** *Sum1*
**have** $Q \longmapsto a<\nu x> \prec Q'$ **by** *fact*
**with** *xFreshP PSimQ* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow a<\nu x> \prec P'$ **and** *P'RelQ'*: $(P', Q') \in Rel$
**by**(*blast dest*: *simE*)
**from** *PTrans* **have** $P \oplus R \Longrightarrow a<\nu x> \prec P'$ **by**(*rule Weak-Early-Step-Semantics.Sum1*)
**moreover from** *P'RelQ' RelRel'* **have** $(P', Q') \in Rel'$ **by** *blast*
**ultimately show** *?case* **by** *blast*
**next**
**case** *Sum2*
**from** ‹$R \longmapsto a<\nu x> \prec Q'$› **have** $P \oplus R \longmapsto a<\nu x> \prec Q'$ **by**(*rule Early-Semantics.Sum2*)
**hence** $P \oplus R \Longrightarrow a<\nu x> \prec Q'$ **by**(*rule Weak-Early-Step-Semantics.singleActionChain*)
**moreover from** *C* **have** $(Q', Q') \in Rel'$ **by** *blast*
**ultimately show** *?case* **by** *blast*
**qed**
**next**
**case**(*Free* $\alpha$ $Q'$)
**have** $Q \oplus R \longmapsto \alpha \prec Q'$ **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *sumCases*)
**case** *Sum1*
**have** $Q \longmapsto \alpha \prec Q'$ **by** *fact*
**with** *PSimQ* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow \alpha \prec P'$ **and** *PRel*: $(P', Q') \in Rel$
**by**(*blast dest*: *simE*)
**from** *PTrans* **have** $P \oplus R \Longrightarrow \alpha \prec P'$ **by**(*rule Weak-Early-Step-Semantics.Sum1*)
**with** *RelRel' PRel* **show** *?case* **by** *blast*
**next**
**case** *Sum2*
**from** ‹$R \longmapsto \alpha \prec Q'$› **have** $P \oplus R \longmapsto \alpha \prec Q'$ **by**(*rule Early-Semantics.Sum2*)
**hence** $P \oplus R \Longrightarrow \alpha \prec Q'$ **by**(*rule Weak-Early-Step-Semantics.singleActionChain*)
**moreover from** *C* **have** $(Q', Q') \in Rel'$ **by** *blast*
**ultimately show** *?case* **by** *blast*
**qed**
**qed**

**lemma** *parPres*:
**fixes** $P$ :: *pi*
**and** $Q$ :: *pi*
**and** $R$ :: *pi*
**and** $T$ :: *pi*
**and** *Rel* :: $(pi \times pi)$ *set*
**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*: $\bigwedge S\ T\ U.\ (S, T) \in Rel \Longrightarrow (S \parallel U, T \parallel U) \in Rel'$
**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** −
**show** *?thesis*
**proof**(*induct rule*: *simCases*)
**case**(*Bound a x Q′*)
**have** $x \sharp P \parallel R$ **by** *fact*
**hence** *xFreshP*: $x \sharp P$ **and** *xFreshR*: $x \sharp R$ **by** *simp+*
**have** $Q \parallel R \longmapsto a<\nu x> \prec Q'$ **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *parCasesB*)
**case**(*cPar1 Q′*)
**have** *QTrans*: $Q \longmapsto a<\nu x> \prec Q'$ **by** *fact*
**from** *xFreshP PSimQ QTrans* **obtain** $P'$ **where** *PTrans*:$P \Longrightarrow a<\nu x> \prec P'$
**and** *P′RelQ′*: $(P', Q') \in Rel$
**by**(*blast dest*: *simE*)
**from** *PTrans xFreshR* **have** $P \parallel R \Longrightarrow a<\nu x> \prec (P' \parallel R)$ **by**(*rule Weak-Early-Step-Semantics.Par1B*)
**moreover from** *P′RelQ′* **have** $(P' \parallel R, Q' \parallel R) \in Rel'$ **by**(*rule Par*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*cPar2 R′*)
**from** ‹$R \longmapsto a<\nu x> \prec R'$› ‹$x \sharp P$› **have** $P \parallel R \longmapsto a<\nu x> \prec (P \parallel R')$
**by**(*rule Early-Semantics.Par2B*)
**hence** $P \parallel R \Longrightarrow a<\nu x> \prec (P \parallel R')$ **by**(*rule Weak-Early-Step-Semantics.singleActionChain*)
**moreover from** *PRelQ* **have** $(P \parallel R', Q \parallel R') \in Rel'$ **by**(*rule Par*)
**ultimately show** *?case* **by** *blast*
**qed**
**next**
**case**(*Free α QR′*)
**have** $Q \parallel R \longmapsto \alpha \prec QR'$ **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *parCasesF*[*of* - - - - - $(P, R)$])
**case**(*cPar1 Q′*)
**have** $Q \longmapsto \alpha \prec Q'$ **by** *fact*
**with** *PSimQ* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow \alpha \prec P'$ **and** *PRel*: $(P', Q')$
$\in Rel$
**by**(*blast dest*: *simE*)
**from** *PTrans* **have** *Trans*: $P \parallel R \Longrightarrow \alpha \prec P' \parallel R$ **by**(*rule Weak-Early-Step-Semantics.Par1F*)
**moreover from** *PRel* **have** $(P' \parallel R, Q' \parallel R) \in Rel'$ **by**(*blast intro*: *Par*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*cPar2 R′*)
**from** ‹$R \longmapsto \alpha \prec R'$› **have** $P \parallel R \longmapsto \alpha \prec (P \parallel R')$
**by**(*rule Early-Semantics.Par2F*)

**hence** $P \parallel R \Longrightarrow\alpha \prec (P \parallel R')$ **by**(*rule Weak-Early-Step-Semantics.singleActionChain*)
  **moreover from** *PRelQ* **have** $(P \parallel R',\ Q \parallel\ R') \in Rel'$ **by**(*rule Par*)
  **ultimately show** *?case* **by** *blast*
**next**
  **case**(*cComm1* $Q'$ $R'$ *a b*)
  **have** *QTrans*: $Q \longmapsto a{<}b{>} \prec Q'$ **and** *RTrans*: $R \longmapsto a[b] \prec R'$ **by** *fact+*

  **from** *PSimQ QTrans* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow a{<}b{>} \prec P'$
                              **and** $P'RelQ'$: $(P',\ Q') \in Rel$
    **by**(*blast dest*: *simE*)
**from** *RTrans* **have** $R \Longrightarrow a[b] \prec R'$ **by**(*rule Weak-Early-Step-Semantics.singleActionChain*)
**with** *PTrans* **have** $P \parallel R \Longrightarrow \tau \prec P' \parallel R'$ **by**(*rule Weak-Early-Step-Semantics.Comm1*)
  **moreover from** $P'RelQ'$ **have** $(P' \parallel R',\ Q' \parallel R') \in Rel'$ **by**(*rule Par*)
  **ultimately show** *?case* **by** *blast*
**next**
  **case**(*cComm2* $Q'$ $R'$ *a b*)
  **have** *QTrans*: $Q \longmapsto a[b] \prec Q'$ **and** *RTrans*: $R \longmapsto a{<}b{>} \prec R'$ **by** *fact+*

  **from** *PSimQ QTrans* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow a[b] \prec P'$
                              **and** $P'RelQ'$: $(P',\ Q') \in Rel$
    **by**(*blast dest*: *simE*)

**from** *RTrans* **have** $R \Longrightarrow a{<}b{>} \prec R'$ **by**(*rule Weak-Early-Step-Semantics.singleActionChain*)
**with** *PTrans* **have** $P \parallel R \Longrightarrow \tau \prec P' \parallel R'$ **by**(*rule Weak-Early-Step-Semantics.Comm2*)
  **moreover from** $P'RelQ'$ **have** $(P' \parallel R',\ Q' \parallel R') \in Rel'$ **by**(*rule Par*)
  **ultimately show** *?case* **by** *blast*
**next**
  **case**(*cClose1* $Q'$ $R'$ *a x*)
  **have** *QTrans*: $Q \longmapsto a{<}x{>} \prec Q'$ **and** *RTrans*: $R \longmapsto a{<}\nu x{>} \prec R'$ **by** *fact+*
  **have** $x \sharp (P,\ R)$ **by** *fact*
  **hence** *xFreshP*: $x \sharp P$ **and** *xFreshR*: $x \sharp R$ **by**(*simp add*: *fresh-prod*)+

  **from** *PSimQ QTrans* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow a{<}x{>} \prec P'$
                              **and** $P'RelQ'$: $(P',\ Q') \in Rel$
    **by**(*blast dest*: *simE*)

**from** *RTrans* **have** $R \Longrightarrow a{<}\nu x{>} \prec R'$ **by**(*rule Weak-Early-Step-Semantics.singleActionChain*)
  **with** *PTrans* **have** *Trans*: $P \parallel R \Longrightarrow \tau \prec {<}\nu x{>}(P' \parallel R')$ **using** ‹$x \sharp P$›
    **by**(*rule Weak-Early-Step-Semantics.Close1*)
  **moreover from** $P'RelQ'$ **have** $({<}\nu x{>}(P' \parallel R'),\ {<}\nu x{>}(Q' \parallel R')) \in Rel'$
    **by**(*blast intro*: *Par Res*)
  **ultimately show** *?case* **by** *blast*
**next**
  **case**(*cClose2* $Q'$ $R'$ *a x*)
  **have** *QTrans*: $Q \longmapsto a{<}\nu x{>} \prec Q'$ **and** *RTrans*: $R \longmapsto a{<}x{>} \prec R'$ **by** *fact+*
  **have** $x \sharp (P,\ R)$ **by** *fact*
  **hence** *xFreshR*: $x \sharp R$ **and** *xFreshP*: $x \sharp P$ **by**(*simp add*: *fresh-prod*)+

  **from** *PSimQ QTrans xFreshP* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow a{<}\nu x{>} \prec P'$

**and** $P'RelQ'$: $(P', Q') \in Rel$
**by**(*blast dest*: *simE*)

**from** *RTrans* **have** $R \Longrightarrow a{<}x{>} \prec R'$ **by**(*rule Weak-Early-Step-Semantics.singleActionChain*)
**with** *PTrans* **have** *Trans*: $P \parallel R \Longrightarrow \tau \prec {<}\nu x{>}(P' \parallel R')$ **using** $\langle x \sharp R\rangle$
**by**(*rule Weak-Early-Step-Semantics.Close2*)
**moreover from** $P'RelQ'$ **have** $({<}\nu x{>}(P' \parallel R'), {<}\nu x{>}(Q' \parallel R')) \in Rel'$
**by**(*blast intro*: *Par Res*)
**ultimately show** *?case* **by** *blast*
**qed**
**qed**
**qed**

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

**assumes** *PSimQ*: $P \leadsto«Rel» Q$
**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** −
**from** $EqvtRel'$ **show** *?thesis*
**proof**(*induct rule*: *simCasesCont*[*of* - $(P, x)$])
**case**(*Bound a y* $Q'$)
**have** *Trans*: ${<}\nu x{>}Q \longmapsto a{<}\nu y{>} \prec Q'$ **by** *fact*
**have** $y \sharp (P, x)$ **by** *fact*
**hence** *yineqx*: $y \neq x$ **and** *yFreshP*: $y \sharp P$ **by**(*simp add*: *fresh-prod*)+
**from** *Trans yineqx* **show** *?case*
**proof**(*induct rule*: *resCasesB*)
**case**(*Open* $Q'$)
**have** *QTrans*: $Q \longmapsto a[x] \prec Q'$ **and** *aineqx*: $a \neq x$ **by** *fact*+

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

**from** *PTrans aineqx* **have** ${<}\nu x{>}P \Longrightarrow a{<}\nu x{>} \prec P'$ **by**(*rule Weak-Early-Step-Semantics.Open*)
**hence** ${<}\nu x{>}P \Longrightarrow a{<}\nu y{>} \prec ([(y, x)] \cdot P')$ **using** $\langle y \sharp P\rangle$ $\langle y \neq x\rangle$
**by**(*force simp add*: *weakTransitionAlpha abs-fresh name-swap*)

**moreover from** *EqvtRel* $P'RelQ'$ $RelRel'$ **have** $([(y, x)] \cdot P', [(y, x)] \cdot Q') \in$
$Rel'$

**by**(*blast intro*: *eqvtRelI*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*Res* $Q'$)
**have** *QTrans*: $Q \longmapsto a<\nu y> \prec Q'$ **and** *xineqa*: $x \neq a$ **by** *fact+*

**from** *PSimQ yFreshP QTrans* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow a<\nu y> \prec P'$
**and** $P'RelQ'$: $(P', Q') \in Rel$
**by**(*blast dest*: *simE*)
**from** *PTrans xineqa yineqx yFreshP* **have** *ResTrans*: $<\nu x>P \Longrightarrow a<\nu y> \prec (<\nu x>P')$
**by**(*blast intro*: *Weak-Early-Step-Semantics.ResB*)
**moreover from** $P'RelQ'$ **have** $((<\nu x>P'), (<\nu x>Q')) \in Rel'$
**by**(*rule C1*)
**ultimately show** *?case* **by** *blast*
**qed**
**next**
**case**(*Free* $\alpha$ $Q'$)
**have** *QTrans*: $<\nu x>Q \longmapsto \alpha \prec Q'$ **by** *fact*
**have** $\exists\, c$::*name*. $c \sharp (P, Q, Q', \alpha)$ **by**(*blast intro*: *name-exists-fresh*)
**then obtain** *c*::*name* **where** *cFreshQ*: $c \sharp Q$ **and** *cFreshAlpha*: $c \sharp \alpha$ **and** $cFreshQ'$: $c \sharp Q'$ **and** *cFreshP*: $c \sharp P$
**by**(*force simp add*: *fresh-prod*)
**from** *cFreshP* **have** $<\nu x>P = <\nu c>([(x, c)] \cdot P)$ **by**(*simp add*: *alphaRes*)
**moreover have** $\exists P'.<\nu c>([(x, c)] \cdot P) \Longrightarrow \alpha \prec P' \wedge (P', Q') \in Rel'$
**proof** −
**from** *QTrans cFreshQ* **have** $<\nu c>([(x, c)] \cdot Q) \longmapsto \alpha \prec Q'$ **by**(*simp add*: *alphaRes*)
**moreover have** $c \sharp \alpha$ **by**(*rule cFreshAlpha*)
**moreover from** *PSimQ EqvtRel* **have** $([(x, c)] \cdot P) \leadsto«Rel» ([(x, c)] \cdot Q)$
**by**(*blast intro*: *eqvtI*)
**ultimately show** *?thesis*
**apply**(*induct rule*: *resCasesF*, *auto simp add*: *residual.inject pi.inject name-abs-eq*)
**by**(*blast intro*: *Weak-Early-Step-Semantics.ResF C1 dest*: *simE*)
**qed**

**ultimately show** *?case* **by** *force*
**qed**
**qed**

**lemma** *resChainI*:
**fixes** $P$ :: *pi*
**and** $Q$ :: *pi*
**and** *Rel* :: (*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*: $P \rightsquigarrow$«*Rel*» $Q$

**shows** (*resChain lst*) $P \rightsquigarrow$«*Rel*» (*resChain lst*) $Q$
**proof** −
**show** *?thesis*
**proof**(*induct lst*)
**from** *PRelQ* **show** *resChain* [] $P \rightsquigarrow$«*Rel*» *resChain* [] $Q$ **by** *simp*
**next**
**fix** *a lst*
**assume** *IH*: (*resChain lst P*) $\rightsquigarrow$«*Rel*» (*resChain lst Q*)
**moreover from** *Res* **have** $\bigwedge P\ Q\ a.\ (P, Q) \in Rel \Longrightarrow (<\nu a>P, <\nu a>Q) \in Rel$
**by** *simp*
**moreover have** $Rel \subseteq Rel$ **by** *simp*
**ultimately have** $<\nu a>$(*resChain lst P*) $\rightsquigarrow$«*Rel*» $<\nu a>$(*resChain lst Q*) **using** *eqvtRel*
**by**(*rule-tac resPres*)

**thus** *resChain* $(a \# lst)$ $P \rightsquigarrow$«*Rel*» *resChain* $(a \# lst)$ $Q$
**by** *simp*
**qed**
**qed**

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

**assumes** *PRelQ*: $(P, Q) \in Rel$
**and** *Sim*: $\bigwedge R\ S.\ (R, S) \in Rel \Longrightarrow R \rightsquigarrow$«$Rel'$» $S$
**and** *C1*: $Rel \subseteq Rel'$
**and** *eqvtRel*: *eqvt* $Rel'$

**shows** $!P \rightsquigarrow$«*bangRel* $Rel'$» $!Q$
**proof** −
**let** $?Sim = \lambda P\ Rs.\ (\forall a\ x\ Q'.\ Rs = a<\nu x> \prec Q' \longrightarrow x \sharp P \longrightarrow (\exists P'.\ P \Longrightarrow a<\nu x> \prec P' \land (P', Q') \in bangRel\ Rel')) \land$
$(\forall \alpha\ Q'.\ Rs = \alpha \prec Q' \longrightarrow (\exists P'.\ P \Longrightarrow \alpha \prec P' \land (P', Q') \in bangRel\ Rel'))$
**from** *eqvtRel* **have** *EqvtBangRel*: *eqvt*(*bangRel* $Rel'$) **by**(*rule eqvtBangRel*)
**from** *C1* **have** $BRelRel'$: $\bigwedge P\ Q.\ (P, Q) \in bangRel\ Rel \Longrightarrow (P, Q) \in bangRel\ Rel'$
**by**(*auto intro*: *bangRelSubset*)

{
**fix** *Pa Rs*
**assume** $!Q \longmapsto Rs$ **and** $(Pa, !Q) \in bangRel\ Rel$
**hence** *?Sim Pa Rs* **using** *PRelQ*
**proof**(*nominal-induct avoiding*: *Pa P rule*: *bangInduct*)

**case**(*Par1B a x Q′ Pa P*)
**have** *QTrans*: $Q \longmapsto a{<}\nu x{>} \prec Q'$ **by** *fact*
**have** $(Pa,\ Q \parallel !Q) \in$ *bangRel Rel* **and** $x \sharp Pa$ **by** *fact+*
**thus** *?Sim Pa* $(a{<}\nu x{>} \prec (Q' \parallel !Q))$
**proof**(*induct rule*: *BRParCases*)
**case**(*BRPar P R*)
**have** *PRelQ*: $(P,\ Q) \in Rel$ **by** *fact*
**have** *PBRQ*: $(R,\ !Q) \in$ *bangRel Rel* **by** *fact*
**have** $x \sharp P \parallel R$ **by** *fact*
**hence** *xFreshP*: $x \sharp P$ **and** *xFreshR*: $x \sharp R$ **by** *simp+*
**show** *?case*
**proof**(*auto simp add*: *residual.inject alpha′*)
**from** *PRelQ* **have** $P \rightsquigarrow$«*Rel′*» $Q$ **by**(*rule Sim*)

**with** *QTrans xFreshP* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow a{<}\nu x{>} \prec P'$ **and** *P′RelQ′*: $(P',\ Q') \in Rel'$
**by**(*blast dest*: *simE*)

**from** *PTrans xFreshR* **have** $P \parallel R \Longrightarrow a{<}\nu x{>} \prec (P' \parallel R)$
**by**(*force intro*: *Weak-Early-Step-Semantics.Par1B*)
**moreover from** *P′RelQ′ PBRQ BRelRel′* **have** $(P' \parallel R,\ Q' \parallel !Q) \in$ *bangRel Rel′* **by**(*blast intro*: *Rel.BRPar*)
**ultimately show** $\exists P'.\ P \parallel R \Longrightarrow a{<}\nu x{>} \prec P' \wedge (P',\ Q' \parallel !Q) \in$ *bangRel Rel′* **by** *blast*
**next**
**fix** $y$
**assume** $(y{::}name) \sharp Q'$ **and** $y \sharp P$ **and** $y \sharp R$ **and** $y \sharp Q$
**from** *QTrans* ‹$y \sharp Q'$› **have** $Q \longmapsto a{<}\nu y{>} \prec ([(x,\ y)] \cdot Q')$
**by**(*simp add*: *alphaBoundOutput*)
**moreover from** *PRelQ* **have** $P \rightsquigarrow$«*Rel′*» $Q$ **by**(*rule Sim*)
**ultimately obtain** $P'$ **where** *PTrans*: $P \Longrightarrow a{<}\nu y{>} \prec P'$ **and** *P′RelQ′*: $(P',\ [(x,\ y)] \cdot Q') \in Rel'$
**using** ‹$y \sharp P$›
**by**(*blast dest*: *simE*)
**from** *PTrans* ‹$y \sharp R$› **have** $P \parallel R \Longrightarrow a{<}\nu y{>} \prec (P' \parallel R)$ **by**(*force intro*: *Weak-Early-Step-Semantics.Par1B*)
**moreover from** *P′RelQ′ PBRQ BRelRel′* **have** $(P' \parallel R,\ ([(x,\ y)] \cdot Q') \parallel !Q) \in$ *bangRel Rel′* **by**(*metis Rel.BRPar*)
**with** ‹$x \sharp Q$› ‹$y \sharp Q$› **have** $(P' \parallel R,\ ([(y,\ x)] \cdot Q') \parallel !([(y,\ x)] \cdot Q)) \in$ *bangRel Rel′*
**by**(*simp add*: *name-fresh-fresh name-swap*)
**ultimately show** $\exists P'.\ P \parallel R \Longrightarrow a{<}\nu y{>} \prec P' \wedge (P',\ ([(y,\ x)] \cdot Q') \parallel !([(y,\ x)] \cdot Q)) \in$ *bangRel Rel′*
**by** *blast*
**qed**
**qed**
**next**
**case**(*Par1F* $\alpha$ *Q′ Pa P*)
**have** *QTrans*: $Q \longmapsto \alpha \prec Q'$ **by** *fact*

**have** $(Pa, Q \parallel !Q) \in$ *bangRel Rel* **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *BRParCases*)
**case**(*BRPar P R*)
**have** *PRelQ*: $(P, Q) \in$ *Rel* **and** *BR*: $(R, !Q) \in$ *bangRel Rel* **by** *fact+*
**show** *?case*
**proof**(*auto simp add*: *residual.inject*)
**from** *PRelQ* **have** $P \rightsquigarrow$«*Rel′*» $Q$ **by**(*rule Sim*)
**with** *QTrans* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow\alpha \prec P'$ **and** *RRel*: $(P',$
$Q') \in Rel'$
**by**(*blast dest*: *simE*)

**from** *PTrans* **have** $P \parallel R \Longrightarrow\alpha \prec P' \parallel R$ **by**(*rule Weak-Early-Step-Semantics.Par1F*)
**moreover from** *RRel BR BRelRel′* **have** $(P' \parallel R, Q' \parallel !Q) \in$ *bangRel*
*Rel′* **by**(*metis Rel.BRPar*)
**ultimately show** $\exists P'. P \parallel R \Longrightarrow\alpha \prec P' \wedge (P', Q' \parallel !Q) \in$ *bangRel Rel′*
**by** *blast*
**qed**
**qed**
**next**
**case**(*Par2B a x Q′ Pa P*)
**hence** *IH*: $\bigwedge Pa. (Pa, !Q) \in$ *bangRel Rel* $\Longrightarrow$ *?Sim Pa* $(a<\nu x> \prec Q')$ **by**
*simp*
**have** $(Pa, Q \parallel !Q) \in$ *bangRel Rel* **and** $x \sharp Pa$ **by** *fact+*
**thus** *?Sim Pa* $(a<\nu x> \prec (Q \parallel Q'))$
**proof**(*induct rule*: *BRParCases*)
**case**(*BRPar P R*)
**have** *PRelQ*: $(P, Q) \in$ *Rel* **and** *RBRQ*: $(R, !Q) \in$ *bangRel Rel* **by** *fact+*
**have** $x \sharp P \parallel R$ **by** *fact*
**hence** *xFreshP*: $x \sharp P$ **and** *xFreshR*: $x \sharp R$ **by** *simp+*

**from** *EqvtBangRel* **show** *?Sim* $(P \parallel R)$ $(a<\nu x> \prec (Q \parallel Q'))$
**proof**(*auto simp add*: *residual.inject alpha′*)
**from** *RBRQ* **have** *?Sim R* $(a<\nu x> \prec Q')$ **by**(*rule IH*)
**with** *xFreshR* **obtain** $R'$ **where** *RTrans*: $R \Longrightarrow a<\nu x> \prec R'$ **and** *R′BRQ′*:
$(R', Q') \in$ (*bangRel Rel′*)
**by**(*metis simE*)
**from** *RTrans xFreshP* **have** $P \parallel R \Longrightarrow a<\nu x> \prec (P \parallel R')$ **by**(*auto intro*:
*Weak-Early-Step-Semantics.Par2B*)
**moreover from** *PRelQ R′BRQ′ C1* **have** $(P \parallel R', Q \parallel Q') \in$ (*bangRel*
*Rel′*) **by**(*blast dest*: *Rel.BRPar*)
**ultimately show** $\exists P'. P \parallel R \Longrightarrow a<\nu x> \prec P' \wedge (P', Q \parallel Q') \in$ *bangRel*
*Rel′* **by** *blast*
**next**
**fix** $y$
**assume** $(y::name) \sharp Q$ **and** $y \sharp Q'$ **and** $y \sharp P$ **and** $y \sharp R$
**from** *RBRQ* **have** *?Sim R* $(a<\nu x> \prec Q')$ **by**(*rule IH*)
**with** ‹$y \sharp Q'$› **have** *?Sim R* $(a<\nu y> \prec ([(x, y)] \cdot Q'))$ **by**(*simp add*:
*alphaBoundOutput*)

**with** ‹$y \sharp R$› **obtain** $R'$ **where** *RTrans*: $R \Longrightarrow a{<}\nu y{>} \prec R'$ **and** *R'BRQ'*: $(R', ([(x, y)] \cdot Q')) \in (bangRel\ Rel')$
**by**(*metis simE*)
**from** *RTrans* ‹$y \sharp P$› **have** $P \parallel R \Longrightarrow a{<}\nu y{>} \prec (P \parallel R')$ **by**(*auto intro*: *Weak-Early-Step-Semantics.Par2B*)
**moreover from** *PRelQ R'BRQ' C1* **have** $(P \parallel R', Q \parallel ([(x, y)] \cdot Q')) \in (bangRel\ Rel')$ **by**(*blast dest*: *Rel.BRPar*)
**with** ‹$y \sharp Q$› ‹$x \sharp Q$› **have** $(P \parallel R', ([(y, x)] \cdot Q) \parallel ([(y, x)] \cdot Q')) \in (bangRel\ Rel')$
**by**(*simp add*: *name-swap name-fresh-fresh*)
**ultimately show** $\exists P'. P \parallel R \Longrightarrow a{<}\nu y{>} \prec P' \wedge (P', ([(y, x)] \cdot Q) \parallel ([(y, x)] \cdot Q')) \in bangRel\ Rel'$ **by** *blast*
**qed**
**qed**
**next**
**case**(*Par2F* $\alpha$ $Q'$ *Pa P*)
**hence** *IH*: $\bigwedge Pa. (Pa, !Q) \in bangRel\ Rel \Longrightarrow ?Sim\ Pa\ (\alpha \prec Q')$ **by** *simp*
**have** $(Pa, Q \parallel !Q) \in bangRel\ Rel$ **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *BRParCases*)
**case**(*BRPar P R*)
**have** *PRelQ*: $(P, Q) \in Rel$ **and** *RBRQ*: $(R, !Q) \in bangRel\ Rel$ **by** *fact+*
**show** *?case*
**proof**(*auto simp add*: *residual.inject*)
**from** *RBRQ IH* **have** $\exists R'. R \Longrightarrow \alpha \prec R' \wedge (R', Q') \in bangRel\ Rel'$
**by**(*metis simE*)
**then obtain** $R'$ **where** *RTrans*: $R \Longrightarrow \alpha \prec R'$ **and** *R'RelQ'*: $(R', Q') \in bangRel\ Rel'$
**by** *blast*

**from** *RTrans* **have** $P \parallel R \Longrightarrow \alpha \prec P \parallel R'$ **by**(*rule Weak-Early-Step-Semantics.Par2F*)
**moreover from** *PRelQ R'RelQ' C1* **have** $(P \parallel R', Q \parallel Q') \in bangRel\ Rel'$ **by**(*blast dest*: *Rel.BRPar*)
**ultimately show** $\exists P'. P \parallel R \Longrightarrow \alpha \prec P' \wedge (P', Q \parallel Q') \in bangRel\ Rel'$ **by** *blast*
**qed**
**qed**
**next**
**case**(*Comm1 a Q' b Q'' Pa P*)
**hence** *IH*: $\bigwedge Pa. (Pa, !Q) \in bangRel\ Rel \Longrightarrow ?Sim\ Pa\ (a[b] \prec Q'')$ **by** *simp*
**have** *QTrans*: $Q \longmapsto a{<}b{>} \prec Q'$ **by** *fact*
**have** $(Pa, Q \parallel !Q) \in bangRel\ Rel$ **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *BRParCases*)
**case**(*BRPar P R*)
**have** *PRelQ*: $(P, Q) \in Rel$ **and** *RBRQ*: $(R, !Q) \in bangRel\ Rel$ **by** *fact+*
**show** *?case*
**proof**(*auto simp add*: *residual.inject*)
**from** *PRelQ* **have** $P \leadsto«Rel'» Q$ **by**(*rule Sim*)

```
        with QTrans obtain P' where PTrans: P ⟹a<b> ≺ P' and P'RelQ':
(P', Q') ∈ Rel'
          by(blast dest: simE)

        from IH RBRQ have RTrans: ∃R'. R ⟹a[b] ≺ R' ∧ (R', Q'') ∈ bangRel
Rel'
          by(metis simE)
        then obtain R' where RTrans: R ⟹a[b] ≺ R' and R'RelQ'': (R', Q'')
∈ bangRel Rel'
          by blast

      from PTrans RTrans have P ∥ R ⟹τ ≺ P' ∥ R' by(rule Weak-Early-Step-Semantics.Comm1)
        moreover from P'RelQ' R'RelQ'' have (P' ∥ R', Q' ∥ Q'') ∈ bangRel
Rel' by(rule Rel.BRPar)
        ultimately show ∃P'. P ∥ R ⟹τ ≺ P' ∧ (P', Q' ∥ Q'') ∈ bangRel Rel'
by blast
      qed
    qed
  next
    case(Comm2 a b Q' Q'')
    hence IH: ⋀Pa. (Pa, !Q) ∈ bangRel Rel ⟹ ?Sim Pa (a<b> ≺ Q'') by simp
    have QTrans: Q ⟼ a[b] ≺ Q' by fact
    have (Pa, Q ∥ !Q) ∈ bangRel Rel by fact
    thus ?case
    proof(induct rule: BRParCases)
      case(BRPar P R)
      have PRelQ: (P, Q) ∈ Rel and RBRQ: (R, !Q) ∈ bangRel Rel by fact+
      show ?case
      proof(auto simp add: residual.inject)
        from PRelQ have P ⇝«Rel'» Q by(rule Sim)
        with QTrans obtain P' where PTrans: P ⟹a[b] ≺ P' and P'RelQ':
(P', Q') ∈ Rel'
          by(blast dest: simE)

          from IH RBRQ have RTrans: ∃R'. R ⟹a<b> ≺ R' ∧ (R', Q'') ∈
bangRel Rel'
          by(metis simE)
          then obtain R' where RTrans: R ⟹a<b> ≺ R' and R'RelQ'': (R',
Q'') ∈ bangRel Rel'
          by blast

      from PTrans RTrans have P ∥ R ⟹τ ≺ P' ∥ R' by(rule Weak-Early-Step-Semantics.Comm2)
        moreover from P'RelQ' R'RelQ'' have (P' ∥ R', Q' ∥ Q'') ∈ bangRel
Rel' by(rule Rel.BRPar)
        ultimately show ∃P'. P ∥ R ⟹τ ≺ P' ∧ (P', Q' ∥ Q'') ∈ bangRel Rel'
by blast
      qed
    qed
  next
```

**case**(*Close1 a x Q′ Q′′ Pa P*)
**hence** *IH*: $\bigwedge Pa.\ (Pa,\ !Q) \in bangRel\ Rel \longrightarrow\ ?Sim\ Pa\ (a{<}\nu x{>} \prec Q'')$ **by** *simp*
**have** *QTrans*: $Q \longmapsto a{<}x{>} \prec Q'$ **by** *fact*
**have** *xFreshQ*: $x \sharp Q$ **by** *fact*
**have** $(Pa,\ Q \parallel !Q) \in bangRel\ Rel$ **by** *fact*
**moreover have** *xFreshPa*: $x \sharp Pa$ **by** *fact*
**ultimately show** *?case*
**proof**(*induct rule*: *BRParCases*)
**case**(*BRPar P R*)
**have** *PRelQ*: $(P,\ Q) \in Rel$ **and** *RBRQ*: $(R,\ !Q) \in bangRel\ Rel$ **by** *fact+*
**have** $x \sharp P \parallel R$ **by** *fact*
**hence** *xFreshP*: $x \sharp P$ **and** *xFreshR*: $x \sharp R$ **by** *simp+*
**show** *?case*
**proof**(*auto simp add*: *residual.inject*)
**from** *PRelQ* **have** $P \rightsquigarrow$«$Rel'$» $Q$ **by**(*rule Sim*)
**with** *QTrans xFreshP* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow a{<}x{>} \prec P'$ **and** *P′RelQ′*: $(P',\ Q') \in Rel'$
**by**(*blast dest*: *simE*)

**from** *RBRQ xFreshR IH* **have** $\exists R'.\ R \Longrightarrow a{<}\nu x{>} \prec R' \wedge (R',\ Q'') \in bangRel\ Rel'$
**by**(*metis simE*)
**then obtain** $R'$ **where** *RTrans*: $R \Longrightarrow a{<}\nu x{>} \prec R'$ **and** *R′RelQ′′*: $(R',\ Q'') \in bangRel\ Rel'$
**by** *blast*

**from** *PTrans RTrans xFreshP* **have** $P \parallel R \Longrightarrow \tau \prec {<}\nu x{>}(P' \parallel R')$
**by**(*rule Weak-Early-Step-Semantics.Close1*)
**moreover from** *P′RelQ′ R′RelQ′′* **have** $({<}\nu x{>}(P' \parallel R'),\ {<}\nu x{>}(Q' \parallel Q'')) \in bangRel\ Rel'$
**by**(*force intro*: *Rel.BRPar BRRes*)
**ultimately show** $\exists P'.\ P \parallel R \Longrightarrow \tau \prec P' \wedge (P',\ {<}\nu x{>}(Q' \parallel Q'')) \in bangRel\ Rel'$ **by** *blast*
**qed**
**qed**
**next**
**case**(*Close2 a x Q′ Q′′ Pa P*)
**hence** *IH*: $\bigwedge Pa.\ (Pa,\ !Q) \in bangRel\ Rel \Longrightarrow\ ?Sim\ Pa\ (a{<}x{>} \prec Q'')$ **by** *simp*
**have** *QTrans*: $Q \longmapsto a{<}\nu x{>} \prec Q'$ **by** *fact*
**have** *xFreshQ*: $x \sharp Q$ **by** *fact*
**have** $(Pa,\ Q \parallel !Q) \in bangRel\ Rel$ **and** $x \sharp Pa$ **by** *fact+*
**thus** *?case*
**proof**(*induct rule*: *BRParCases*)
**case**(*BRPar P R*)
**have** *PRelQ*: $(P,\ Q) \in Rel$ **and** *RBRQ*: $(R,\ !Q) \in bangRel\ Rel$ **by** *fact+*
**have** $x \sharp P \parallel R$ **by** *fact*
**hence** *xFreshP*: $x \sharp P$ **and** *xFreshR*: $x \sharp R$ **by** *simp+*
**show** *?case*

**proof**(*auto simp add*: *residual.inject*)
**from** *PRelQ* **have** $P \rightsquigarrow$«*Rel'*» *Q* **by**(*rule Sim*)
**with** *QTrans xFreshP* **obtain** *P'* **where** *PTrans*: $P \Longrightarrow a$<$\nu x$> $\prec P'$ **and** *P'RelQ'*: $(P', Q') \in Rel'$
**by**(*blast dest*: *simE*)

**from** *RBRQ IH* **have** $\exists R'.\ R \Longrightarrow a$<$x$> $\prec R' \wedge (R', Q'') \in$ *bangRel Rel'*
**by** *auto*
**then obtain** *R'* **where** *RTrans*: $R \Longrightarrow a$<$x$> $\prec R'$ **and** *R'RelQ''*: $(R', Q'') \in$ *bangRel Rel'*
**by** *blast*

**from** *PTrans RTrans xFreshR* **have** $P \parallel R \Longrightarrow\tau \prec$ <$\nu x$>$(P' \parallel R')$
**by**(*rule Weak-Early-Step-Semantics.Close2*)
**moreover from** *P'RelQ' R'RelQ''* **have** (<$\nu x$>$(P' \parallel R')$, <$\nu x$>$(Q' \parallel Q''))$ $\in$ *bangRel Rel'*
**by**(*force intro*: *Rel.BRPar BRRes*)
**ultimately show** $\exists P'.\ P \parallel R \Longrightarrow\tau \prec P' \wedge (P',$ <$\nu x$>$(Q' \parallel Q'')) \in$ *bangRel Rel'* **by** *blast*
**qed**
**qed**
**next**
**case**(*Bang Rs Pa P*)
**hence** *IH*: $\bigwedge Pa.\ (Pa, Q \parallel !Q) \in$ *bangRel Rel* $\Longrightarrow$ *?Sim Pa Rs* **by** *simp*
**have** $(Pa, !Q) \in$ *bangRel Rel* **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *BRBangCases*)
**case**(*BRBang P*)
**have** *PRelQ*: $(P, Q) \in Rel$ **by** *fact*
**hence** $(!P, !Q) \in$ *bangRel Rel* **by**(*rule Rel.BRBang*)
**with** *PRelQ* **have** $(P \parallel !P, Q \parallel !Q) \in$ *bangRel Rel* **by**(*rule BRPar*)
**with** *IH* **have** *?Sim* $(P \parallel !P)$ *Rs* **by** *simp*
**thus** *?case* **by**(*force intro*: *Weak-Early-Step-Semantics.Bang*)
**qed**
**qed**
}

**moreover from** *PRelQ* **have** $(!P, !Q) \in$ *bangRel Rel* **by**(*rule BRBang*)
**ultimately show** *?thesis* **by**(*auto simp add*: *weakStepSimulation-def*)
**qed**

**end**

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

**lemma** *tauPres*:
**fixes** *P* :: *pi*

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

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

**shows** $\tau.(P) \leadsto<Rel> \tau.(Q)$
**proof**(*induct rule*: *simCases*)
**case**(*Bound* $Q'$ $a$ $x$)
**have** $\tau.(Q) \longmapsto a<\nu x> \prec Q'$ **by** *fact*
**hence** *False* **by**(*induct rule*: *tauCases'*, *auto*)
**thus** *?case* **by** *simp*
**next**
**case**(*Free* $Q'$ $\alpha$)
**have** $\tau.(Q) \longmapsto(\alpha \prec Q')$ **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *tauCases'*, *auto simp only*: *pi.inject residual.inject*)
**have** $\tau.(P) \Longrightarrow\hat{}\ \tau \prec P$ **by**(*rule Tau*)
**with** *PRelQ* **show** $\exists P'.\ \tau.(P) \Longrightarrow\hat{}\tau \prec P' \wedge (P', Q) \in Rel$ **by** *blast*
**qed**
**qed**

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

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

**shows** $a<x>.P \leadsto<Rel> a<x>.Q$
**using** *Eqvt*
**proof**(*induct rule*: *simCasesCont*[**where** $C=(x, a, P, Q)$])
**case**(*Bound* $b$ $y$ $Q'$)
**from** ‹$y \sharp (x, a, P, Q)$› **have** $y \neq x$ $y \neq a$ $y \sharp P$ $y \sharp Q$ **by** *simp+*
**from** ‹$a<x>.Q \longmapsto b<\nu y> \prec Q'$› ‹$y \neq a$› ‹$y \neq x$› ‹$y \sharp Q$› **show** *?case*
**by**(*erule-tac inputCases'*) *auto*
**next**
**case**(*Free* $\alpha$ $Q'$)
**from** ‹$a<x>.Q \longmapsto \alpha \prec Q'$›
**show** *?case*
**proof**(*induct rule*: *inputCases*)
**case**(*cInput* $u$)
**have** $a<x>.P \Longrightarrow\hat{}(a<u>) \prec P[x::=u]$
**by**(*rule Input*)
**moreover from** *PRelQ* **have** $(P[x::=u], Q[x::=u]) \in Rel$ **by** *auto*
**ultimately show** *?case* **by** *blast*

**qed**
**qed**

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

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

**shows** $a\{b\}.P \rightsquigarrow$<*Rel*> $a\{b\}.Q$
**proof**(*induct rule*: *simCases*)
**case**(*Bound* $Q'$ $c$ $x$)
**have** $a\{b\}.Q \longmapsto c{<}\nu x{>} \prec Q'$ **by** *fact*
**hence** *False* **by**(*induct rule*: *outputCases'*, *auto*)
**thus** *?case* **by** *simp*
**next**
**case**(*Free* $Q'$ $\alpha$)
**have** $a\{b\}.Q \longmapsto \alpha \prec Q'$ **by** *fact*
**thus** $\exists P'.\ a\{b\}.P \Longrightarrow^{\wedge} \alpha \prec P' \wedge (P', Q') \in Rel$
**proof**(*induct rule*: *outputCases'*, *auto simp add*: *pi.inject residual.inject*)
**have** $a\{b\}.P \Longrightarrow^{\wedge} a[b] \prec P$ **by**(*rule Output*)
**with** *PRelQ* **show** $\exists P'.\ a\{b\}.P \Longrightarrow^{\wedge} a[b] \prec P' \wedge (P', Q) \in Rel$ **by** *blast*
**qed**
**qed**

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

**assumes** *PSimQ*: $P \rightsquigarrow$<*Rel*> $Q$
**and** *RelRel'*: $Rel \subseteq Rel'$
**and** *RelStay*: $\bigwedge R\ S\ c.\ (R, S) \in Rel \Longrightarrow ([c\frown c]R, S) \in Rel$

**shows** $[a\frown b]P \rightsquigarrow$<*Rel'*> $[a\frown b]Q$
**proof**(*induct rule*: *simCases*)
**case**(*Bound* $Q'$ $c$ $x$)
**have** $x \sharp [a\frown b]P$ **by** *fact*
**hence** *xFreshP*: $(x::name) \sharp P$ **by** *simp*
**have** $[a\frown b]Q \longmapsto c{<}\nu x{>} \prec Q'$ **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *matchCases*)
**case** *Match*

have $Q \longmapsto c<\nu x> \prec Q'$ **by** *fact*
with *PSimQ xFreshP* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow c<\nu x> \prec P'$
**and** $P'RelQ'$: $(P', Q') \in Rel$
**by**(*blast dest*: *simE*)
**from** *PTrans* **have** $[a\frown a]P \Longrightarrow c<\nu x> \prec P'$ **by**(*rule Weak-Early-Step-Semantics.Match*)
**moreover from** $P'RelQ'$ $RelRel'$ **have** $(P', Q') \in Rel'$ **by** *blast*
**ultimately show** *?case* **by** *blast*
**qed**
**next**
**case**(*Free* $Q'$ $\alpha$)
**have** $[a\frown b]Q \longmapsto \alpha \prec Q'$ **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *matchCases*)
**case** *Match*
**have** $Q \longmapsto \alpha \prec Q'$ **by** *fact*
**with** *PSimQ* **obtain** $P'$ **where** $P \Longrightarrow^{\wedge} \alpha \prec P'$ **and** $(P', Q') \in Rel$
**by**(*blast dest*: *simE*)
**thus** *?case*
**proof**(*induct rule*: *transitionCases*)
**case** *Step*
**have** $P \Longrightarrow \alpha \prec P'$ **by** *fact*
**hence** $[a\frown a]P \Longrightarrow \alpha \prec P'$ **by**(*rule Weak-Early-Step-Semantics.Match*)
**with** $RelRel'$ ‹$(P', Q') \in Rel$› **show** *?case* **by**(*force simp add*: *weakFreeTransition-def*)
**next**
**case** *Stay*
**have** $[a\frown a]P \Longrightarrow^{\wedge} \tau \prec [a\frown a]P$ **by**(*simp add*: *weakFreeTransition-def*)
**moreover from** ‹$(P, Q') \in Rel$› **have** $([a\frown a]P, Q') \in Rel$ **by**(*blast intro*: *RelStay*)
**ultimately show** *?case* **using** $RelRel'$ **by** *blast*
**qed**
**qed**
**qed**

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

**assumes** *PSimQ*: $P \rightsquigarrow<Rel> Q$
**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** $[a\neq b]P \rightsquigarrow<Rel'> [a\neq b]Q$
**proof**(*induct rule*: *simCases*)
**case**(*Bound* $Q'$ $c$ $x$)

**have** $x \sharp [a{\neq}b]P$ **by** *fact*
**hence** *xFreshP*: $(x{::}name) \sharp P$ **by** *simp*
**have** $[a{\neq}b]Q \longmapsto c{<}\nu x{>} \prec Q'$ **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *mismatchCases*)
**case** *Mismatch*
**have** *aineqb*: $a \neq b$ **by** *fact*
**have** $Q \longmapsto c{<}\nu x{>} \prec Q'$ **by** *fact*
**with** *PSimQ xFreshP* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow c{<}\nu x{>} \prec P'$
**and** $P'RelQ'$: $(P', Q') \in Rel$
**by**(*blast dest*: *simE*)
**from** *PTrans aineqb* **have** $[a{\neq}b]P \Longrightarrow c{<}\nu x{>} \prec P'$ **by**(*rule Weak-Early-Step-Semantics.Mismatch*)
**moreover from** $P'RelQ'$ *RelRel'* **have** $(P', Q') \in Rel'$ **by** *blast*
**ultimately show** *?case* **by** *blast*
**qed**
**next**
**case**(*Free* $Q'$ $\alpha$)
**have** $[a{\neq}b]Q \longmapsto \alpha \prec Q'$ **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *mismatchCases*)
**case** *Mismatch*
**have** *aineqb*: $a \neq b$ **by** *fact*
**have** $Q \longmapsto \alpha \prec Q'$ **by** *fact*
**with** *PSimQ* **obtain** $P'$ **where** $P \Longrightarrow\hat{}\ \alpha \prec P'$ **and** $(P', Q') \in Rel$
**by**(*blast dest*: *simE*)
**thus** *?case*
**proof**(*induct rule*: *transitionCases*)
**case** *Step*
**have** $P \Longrightarrow \alpha \prec P'$ **by** *fact*
**hence** $[a{\neq}b]P \Longrightarrow \alpha \prec P'$ **using** *aineqb* **by**(*rule Weak-Early-Step-Semantics.Mismatch*)
**with** *RelRel'* ‹$(P', Q') \in Rel$› **show** *?case* **by**(*force simp add*: *weakFreeTransition-def*)
**next**
**case** *Stay*
**have** $[a{\neq}b]P \Longrightarrow\hat{}\ \tau \prec [a{\neq}b]P$ **by**(*simp add*: *weakFreeTransition-def*)
**moreover from** ‹$(P, Q') \in Rel$› *aineqb* **have** $([a{\neq}b]P, Q') \in Rel$ **by**(*blast intro*: *RelStay*)
**ultimately show** *?case* **using** *RelRel'* **by** *blast*
**qed**
**qed**
**qed**

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

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

**assumes** $PSimQ$: $P \leadsto<Rel> Q$
**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** $-$
  **show** *?thesis*
  **proof**(*induct rule*: *simCases*)
    **case**(*Bound* $Q'$ $a$ $x$)
    **have** $x \sharp P \parallel R$ **by** *fact*
    **hence** *xFreshP*: $x \sharp P$ **and** *xFreshR*: $x \sharp R$ **by** *simp+*
    **have** $Q \parallel S \longmapsto a<\nu x> \prec Q'$ **by** *fact*
    **thus** *?case*
    **proof**(*induct rule*: *parCasesB*)
      **case**(*cPar1* $Q'$)
      **have** *QTrans*: $Q \longmapsto a<\nu x> \prec Q'$ **and** *xFreshT*: $x \sharp S$ **by** *fact+*
      **from** *xFreshP PSimQ QTrans* **obtain** $P'$ **where** *PTrans*:$P \Longrightarrow a<\nu x> \prec P'$
                                  **and** $P'RelQ'$: $(P', Q') \in Rel$
        **by**(*blast dest*: *simE*)
      **from** *PTrans xFreshR* **have** $P \parallel R \Longrightarrow a<\nu x> \prec (P' \parallel R)$ **by**(*rule Weak-Early-Step-Semantics.Par1B*)
      **moreover from** $P'RelQ'$ $RRel'T$ **have** $(P' \parallel R, Q' \parallel S) \in Rel''$ **by**(*rule Par*)
      **ultimately show** *?case* **by** *blast*
    **next**
      **case**(*cPar2* $S'$)
      **have** *STrans*: $S \longmapsto a<\nu x> \prec S'$ **and** *xFreshQ*: $x \sharp Q$ **by** *fact+*
      **from** *xFreshR RSimT STrans* **obtain** $R'$ **where** *RTrans*:$R \Longrightarrow a<\nu x> \prec R'$
                                  **and** $R'Rel'T'$: $(R', S') \in Rel'$
        **by**(*blast dest*: *simE*)
      **from** *RTrans xFreshP xFreshR* **have** *ParTrans*: $P \parallel R \Longrightarrow a<\nu x> \prec (P \parallel R')$
        **by**(*blast intro*: *Weak-Early-Step-Semantics.Par2B*)
      **moreover from** *PRelQ* $R'Rel'T'$ **have** $(P \parallel R', Q \parallel S') \in Rel''$ **by**(*rule Par*)
      **ultimately show** *?case* **by** *blast*
    **qed**
  **next**
    **case**(*Free* $QT'$ $\alpha$)
    **have** $Q \parallel S \longmapsto \alpha \prec QT'$ **by** *fact*
    **thus** *?case*
    **proof**(*induct rule*: *parCasesF*[*of* - - - - - $(P, R)$])
      **case**(*cPar1* $Q'$)
      **have** $Q \longmapsto \alpha \prec Q'$ **by** *fact*
      **with** *PSimQ* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow^{\wedge} \alpha \prec P'$ **and** *PRel*: $(P', Q') \in Rel$

**by**(*blast dest*: *simE*)
**from** *PTrans* **have** *Trans*: $P \parallel R \Longrightarrow^{\hat{}} \alpha \prec P' \parallel R$ **by**(*rule Weak-Early-Semantics.Par1F*)
**moreover from** *PRel RRel'T* **have** $(P' \parallel R,\ Q' \parallel S) \in Rel''$ **by**(*blast intro*: *Par*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*cPar2* $S'$)
**have** $S \longmapsto \alpha \prec S'$ **by** *fact*
**with** *RSimT* **obtain** $R'$ **where** *RTrans*: $R \Longrightarrow^{\hat{}} \alpha \prec R'$ **and** *RRel*: $(R',\ S') \in Rel'$
**by**(*blast dest*: *simE*)
**from** *RTrans* **have** *Trans*: $P \parallel R \Longrightarrow^{\hat{}} \alpha \prec P \parallel R'$ **by**(*rule Weak-Early-Semantics.Par2F*)
**moreover from** *PRelQ RRel* **have** $(P \parallel R',\ Q \parallel S') \in Rel''$ **by**(*blast intro*: *Par*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*cComm1* $Q'$ $S'$ *a b*)
**have** *QTrans*: $Q \longmapsto a{<}b{>} \prec Q'$ **and** *STrans*: $S \longmapsto a[b] \prec S'$ **by** *fact+*

**from** *PSimQ QTrans* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow a{<}b{>} \prec P'$
**and** $P'RelQ'$: $(P',\ Q') \in Rel$
**by**(*fastforce dest*: *simE simp add*: *weakFreeTransition-def*)

**from** *RSimT STrans* **obtain** $R'$ **where** *RTrans*: $R \Longrightarrow a[b] \prec R'$
**and** *RRel*: $(R',\ S') \in Rel'$
**by**(*fastforce dest*: *simE simp add*: *weakFreeTransition-def*)

**from** *PTrans RTrans* **have** $P \parallel R \Longrightarrow \tau \prec P' \parallel R'$ **by**(*rule Weak-Early-Step-Semantics.Comm1*)
**hence** $P \parallel R \Longrightarrow^{\hat{}} \tau \prec P' \parallel R'$
**by**(*auto simp add*: *trancl-into-rtrancl dest*: *Weak-Early-Step-Semantics.tauTransitionChain*)

**moreover from** $P'RelQ'$ *RRel* **have** $(P' \parallel R',\ Q' \parallel S') \in Rel''$ **by**(*rule Par*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*cComm2* $Q'$ $S'$ *a b*)
**have** *QTrans*: $Q \longmapsto a[b] \prec Q'$ **and** *STrans*: $S \longmapsto a{<}b{>} \prec S'$ **by** *fact+*

**from** *PSimQ QTrans* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow a[b] \prec P'$
**and** *PRel*: $(P',\ Q') \in Rel$
**by**(*fastforce dest*: *simE simp add*: *weakFreeTransition-def*)

**from** *RSimT STrans* **obtain** $R'$ **where** *RTrans*: $R \Longrightarrow a{<}b{>} \prec R'$
**and** $R'Rel'T'$: $(R',\ S') \in Rel'$
**by**(*fastforce dest*: *simE simp add*: *weakFreeTransition-def*)

**from** *PTrans RTrans* **have** $P \parallel R \Longrightarrow \tau \prec P' \parallel R'$ **by**(*rule Weak-Early-Step-Semantics.Comm2*)
**hence** $P \parallel R \Longrightarrow^{\hat{}} \tau \prec P' \parallel R'$
**by**(*auto simp add*: *trancl-into-rtrancl dest*: *Weak-Early-Step-Semantics.tauTransitionChain*)
**moreover from** *PRel* $R'Rel'T'$ **have** $(P' \parallel R',\ Q' \parallel S') \in Rel''$ **by**(*rule Par*)

**ultimately show** *?case* **by** *blast*
**next**
**case**(*cClose1* $Q'$ $S'$ *a x*)
**have** *QTrans*: $Q \longmapsto a{<}x{>} \prec Q'$ **and** *STrans*: $S \longmapsto a{<}\nu x{>} \prec S'$ **by** *fact*+
**have** $x \sharp (P, R)$ **by** *fact*
**hence** *xFreshP*: $x \sharp P$ **and** *xFreshR*: $x \sharp R$ **by**(*simp add*: *fresh-prod*)+

**from** *PSimQ QTrans* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow a{<}x{>} \prec P'$
**and** $P'RelQ'$: $(P', Q') \in Rel$
**by**(*fastforce dest*: *simE simp add*: *weakFreeTransition-def*)

**from** *RSimT STrans xFreshR* **obtain** $R'$ **where** *RTrans*: $R \Longrightarrow a{<}\nu x{>} \prec R'$
**and** $R'Rel'T'$: $(R', S') \in Rel'$
**by**(*blast dest*: *simE*)

**from** *PTrans RTrans xFreshP* **have** *Trans*: $P \parallel R \Longrightarrow \tau \prec {<}\nu x{>}(P' \parallel R')$
**by**(*rule Weak-Early-Step-Semantics.Close1*)
**hence** $P \parallel R \Longrightarrow\hat{}\ \tau \prec {<}\nu x{>}(P' \parallel R')$
**by**(*auto simp add*: *trancl-into-rtrancl dest*: *Weak-Early-Step-Semantics.tauTransitionChain*)
**moreover from** $P'RelQ'$ $R'Rel'T'$ **have** $({<}\nu x{>}(P' \parallel R'), {<}\nu x{>}(Q' \parallel S'))$
$\in Rel''$
**by**(*blast intro*: *Par Res*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*cClose2* $Q'$ $S'$ *a x*)
**have** *QTrans*: $Q \longmapsto a{<}\nu x{>} \prec Q'$ **and** *STrans*: $S \longmapsto a{<}x{>} \prec S'$ **by** *fact*+
**have** $x \sharp (P, R)$ **by** *fact*
**hence** *xFreshR*: $x \sharp R$ **and** *xFreshP*: $x \sharp P$ **by**(*simp add*: *fresh-prod*)+

**from** *PSimQ QTrans xFreshP* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow a{<}\nu x{>} \prec P'$
**and** $P'RelQ'$: $(P', Q') \in Rel$
**by**(*blast dest*: *simE*)

**from** *RSimT STrans* **obtain** $R'$ **where** *RTrans*: $R \Longrightarrow a{<}x{>} \prec R'$
**and** $R'Rel'T'$: $(R', S') \in Rel'$
**by**(*fastforce dest*: *simE simp add*: *weakFreeTransition-def*)
**from** *PTrans RTrans xFreshR* **have** *Trans*: $P \parallel R \Longrightarrow \tau \prec {<}\nu x{>}(P' \parallel R')$
**by**(*rule Weak-Early-Step-Semantics.Close2*)
**hence** $P \parallel R \Longrightarrow\hat{}\ \tau \prec {<}\nu x{>}(P' \parallel R')$
**by**(*auto simp add*: *trancl-into-rtrancl dest*: *Weak-Early-Step-Semantics.tauTransitionChain*)
**moreover from** $P'RelQ'$ $R'Rel'T'$ **have** $({<}\nu x{>}(P' \parallel R'), {<}\nu x{>}(Q' \parallel S'))$
$\in Rel''$
**by**(*blast intro*: *Par Res*)
**ultimately show** *?case* **by** *blast*
**qed**
**qed**
**qed**

**lemma** *parPres*:

**fixes** $P$ :: *pi*
**and** $Q$ :: *pi*
**and** $R$ :: *pi*
**and** $a$ :: *name*
**and** *Rel* :: (*pi* × *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** −
**note** *PSimQ*
**moreover have** *RSimR*: $R \leadsto<Id> R$ **by**(*auto intro*: *reflexive*)
**moreover note** *PRelQ* **moreover have** $(R, R) \in Id$ **by** *auto*
**moreover from** *Par* **have** $\bigwedge P\ Q\ R\ T.\ [\![(P, Q) \in Rel; (R, T) \in Id]\!] \Longrightarrow (P \parallel R, Q \parallel T) \in Rel'$
**by** *auto*
**ultimately show** *?thesis* **using** *Res* **by**(*rule parCompose*)
**qed**

**lemma** *resPres*:
**fixes** $P$ :: *pi*
**and** $Q$ :: *pi*
**and** *Rel* :: (*pi* × *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** −
**from** *EqvtRel'* **show** *?thesis*
**proof**(*induct rule*: *simCasesCont*[**where** $C=(P, x)$])
**case**(*Bound a y Q'*)
**have** *Trans*: $<\nu x>Q \longmapsto a<\nu y> \prec Q'$ **by** *fact*
**have** $y \sharp (P, x)$ **by** *fact*
**hence** *yineqx*: $y \neq x$ **and** *yFreshP*: $y \sharp P$ **by**(*simp add*: *fresh-prod*)+
**from** *Trans yineqx* **show** *?case*
**proof**(*induct rule*: *resCasesB*)
**case**(*Open Q'*)
**have** *QTrans*: $Q \longmapsto a[x] \prec Q'$ **and** *aineqx*: $a \neq x$ **by** *fact*+

**from** *PSimQ QTrans* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow\hat{} a[x] \prec P'$
**and** $P'RelQ'$: $(P', Q') \in Rel$
**by**(*blast dest*: *simE*)

**from** *PTrans aineqx* **have** $<\nu x>P \Longrightarrow a<\nu x> \prec P'$
**by**(*force intro*: *Weak-Early-Step-Semantics.Open simp add*: *weakFreeTransition-def*)
**with** ‹$y \sharp P$› ‹$y \neq x$› **have** $<\nu x>P \Longrightarrow a<\nu y> \prec ([(y, x)] \cdot P')$
**by**(*force intro*: *weakTransitionAlpha simp add*: *abs-fresh name-swap*)
**moreover from** *EqvtRel* $P'RelQ'$ *RelRel'* **have** $([(y, x)] \cdot P', [(y, x)] \cdot Q') \in Rel'$
**by**(*blast intro*: *eqvtRelI*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*Res* $Q'$)
**have** *QTrans*: $Q \longmapsto a<\nu y> \prec Q'$ **and** *xineqa*: $x \neq a$ **by** *fact+*

**from** *PSimQ yFreshP QTrans* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow a<\nu y> \prec P'$
**and** $P'RelQ'$: $(P', Q') \in Rel$
**by**(*blast dest*: *simE*)
**from** *PTrans xineqa yineqx yFreshP* **have** *ResTrans*: $<\nu x>P \Longrightarrow a<\nu y> \prec (<\nu x>P')$
**by**(*blast intro*: *Weak-Early-Step-Semantics.ResB*)
**moreover from** $P'RelQ'$ **have** $((<\nu x>P'), (<\nu x>Q')) \in Rel'$
**by**(*rule ResRel*)
**ultimately show** *?case* **by** *blast*
**qed**
**next**
**case**(*Free* $\alpha$ $Q'$)
**have** *QTrans*: $<\nu x>Q \longmapsto \alpha \prec Q'$ **by** *fact*
**have** $\exists$ *c*::*name*. $c \sharp (P, Q, Q', \alpha)$ **by**(*blast intro*: *name-exists-fresh*)
**then obtain** *c*::*name* **where** *cFreshQ*: $c \sharp Q$ **and** *cFreshAlpha*: $c \sharp \alpha$ **and** *cFreshQ'*: $c \sharp Q'$ **and** *cFreshP*: $c \sharp P$
**by**(*force simp add*: *fresh-prod*)
**from** *cFreshP* **have** $<\nu x>P = <\nu c>([(x, c)] \cdot P)$ **by**(*simp add*: *alphaRes*)
**moreover have** $\exists P'.<\nu c>([(x, c)] \cdot P) \Longrightarrow\hat{}\ \alpha \prec P' \wedge (P', Q') \in Rel'$
**proof** −
**from** *QTrans cFreshQ* **have** $<\nu c>([(x, c)] \cdot Q) \longmapsto \alpha \prec Q'$ **by**(*simp add*: *alphaRes*)
**moreover have** $c \sharp \alpha$ **by**(*rule cFreshAlpha*)
**moreover from** *PSimQ EqvtRel* **have** $([(x, c)] \cdot P) \leadsto<Rel> ([(x, c)] \cdot Q)$
**by**(*blast intro*: *eqvtI*)
**ultimately show** *?thesis*
**apply**(*induct rule*: *resCasesF*, *auto simp add*: *residual.inject pi.inject name-abs-eq*)
**by**(*blast intro*: *ResF ResRel dest*: *simE*)
**qed**

**ultimately show** *?case* **by** *force*

```
  qed
qed

lemma resChainI:
  fixes P   :: pi
  and   Q   :: pi
  and   Rel :: (pi × pi) set
  and   lst :: name list

  assumes eqvtRel: eqvt Rel
  and     Res:     ⋀R S y. (R, S) ∈ Rel ⟹ (<νy>R, <νy>S) ∈ Rel
  and     PRelQ:   P ↝<Rel> Q

  shows (resChain lst) P ↝<Rel> (resChain lst) Q
proof -
  show ?thesis
  proof(induct lst)
    from PRelQ show resChain [] P ↝<Rel> resChain [] Q by simp
  next
    fix a lst
    assume IH: (resChain lst P) ↝<Rel> (resChain lst Q)
    moreover from Res have ⋀P Q a. (P, Q) ∈ Rel ⟹ (<νa>P, <νa>Q) ∈
Rel
      by simp
    moreover have Rel ⊆ Rel by simp
    ultimately have <νa>(resChain lst P) ↝<Rel> <νa>(resChain lst Q) using
eqvtRel
      by(rule-tac resPres)
    thus resChain (a # lst) P ↝<Rel> resChain (a # lst) Q
      by simp
  qed
qed

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' ⟹ (<νx>R, <νx>S) ∈ Rel'

  and     RelStay:    ⋀R S. (R ∥ !R, S) ∈ Rel' ⟹ (!R, S) ∈ Rel'
  and     BangRelRel': (bangRel Rel) ⊆ Rel'
  and     eqvtRel':   eqvt Rel'
```

**shows** $!P \leadsto$<$Rel'$> $!Q$
**proof** $-$
**let** *?Sim* $= \lambda P\ Rs.\ (\forall\, a\ x\ Q'.\ Rs = a$<$\nu x$> $\prec Q' \longrightarrow x \sharp P \longrightarrow (\exists\, P'.\ P \Longrightarrow a$<$\nu x$> $\prec P' \wedge (P', Q') \in Rel')) \wedge$
$(\forall\, \alpha\ Q'.\ Rs = \alpha \prec Q' \longrightarrow (\exists\, P'.\ P \Longrightarrow\hat{}\ \alpha \prec P' \wedge (P', Q') \in Rel'))$
**{**
**fix** *Rs P*
**assume** $!Q \longmapsto Rs$ **and** $(P, !Q) \in$ *bangRel Rel*
**hence** *?Sim P Rs* **using** *PRelQ*
**proof**(*nominal-induct avoiding*: *P rule*: *bangInduct*)
**case**(*Par1B a x Q'*)
**have** *QTrans*: $Q \longmapsto a$<$\nu x$> $\prec Q'$ **and** *xFreshQ*: $x \sharp Q$ **by** *fact+*
**have** $(P, Q \parallel !Q) \in$ *bangRel Rel* **and** $x \sharp P$ **by** *fact+*
**thus** *?case*
**proof**(*induct rule*: *BRParCases*)
**case**(*BRPar P R*)
**have** *PRelQ*: $(P, Q) \in Rel$ **and** *RBangRelT*: $(R, !Q) \in$ *bangRel Rel* **by** *fact+*
**have** $x \sharp P \parallel R$ **by** *fact*
**hence** *xFreshP*: $x \sharp P$ **and** *xFreshR*: $x \sharp R$ **by** *simp+*
**from** *PRelQ* **have** *PSimQ*: $P \leadsto$<$Rel$> $Q$ **by**(*rule Sim*)
**from** ‹$x \sharp P$› ‹$x \sharp Q$› **show** *?case*
**proof**(*auto simp add*: *residual.inject alpha' name-fresh-fresh*)
**from** *PSimQ QTrans xFreshP* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow a$<$\nu x$> $\prec P'$
**and** *P'RelQ'*: $(P', Q') \in Rel$
**by**(*blast dest*: *simE*)
**from** *PTrans xFreshR* **have** $P \parallel R \Longrightarrow a$<$\nu x$>$\prec (P' \parallel R)$
**by**(*rule Weak-Early-Step-Semantics.Par1B*)
**moreover from** *P'RelQ' RBangRelT BangRelRel'* **have** $(P' \parallel R, Q' \parallel !Q) \in Rel'$
**by**(*blast intro*: *Rel.BRPar*)
**ultimately show** $\exists\, P'.\ P \parallel R \Longrightarrow a$<$\nu x$> $\prec P' \wedge (P', Q' \parallel !Q) \in Rel'$ **by** *blast*
**next**
**fix** *y*
**assume** $(y{::}name) \sharp Q'$ **and** $y \sharp P$ **and** $y \sharp R$
**from** *QTrans* ‹$y \sharp Q'$› **have** $Q \longmapsto a$<$\nu y$> $\prec ([(x, y)] \cdot Q')$ **by**(*simp add*: *alphaBoundOutput*)
**with** *PSimQ* ‹$y \sharp P$› **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow a$<$\nu y$> $\prec P'$
**and** *P'RelQ'*: $(P', [(x, y)] \cdot Q') \in Rel$
**by**(*blast dest*: *simE*)
**from** *PTrans* ‹$y \sharp R$› **have** $P \parallel R \Longrightarrow a$<$\nu y$>$\prec (P' \parallel R)$ **by**(*rule Weak-Early-Step-Semantics.Par1B*)
**moreover from** *P'RelQ' RBangRelT BangRelRel'* **have** $(P' \parallel R, ([(y, x)] \cdot Q') \parallel !Q) \in Rel'$
**by**(*fastforce intro*: *Rel.BRPar simp add*: *name-swap*)
**ultimately show** $\exists\, P'.\ P \parallel R \Longrightarrow a$<$\nu y$> $\prec P' \wedge (P', ([(y, x)] \cdot Q') \parallel$

$!Q) \in Rel'$ **by** *blast*
**qed**
**qed**
**next**
**case**(*Par1F* $\alpha$ $Q'$ $P$)
**have** *QTrans*: $Q \longmapsto \alpha \prec Q'$ **by** *fact*
**have** $(P, Q \parallel !Q) \in$ *bangRel Rel* **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *BRParCases*)
**case**(*BRPar P R*)
**have** *PRelQ*: $(P, Q) \in Rel$ **and** *RBangRelQ*: $(R, !Q) \in$ *bangRel Rel* **by** *fact+*
**show** *?case*
**proof**(*auto simp add*: *residual.inject*)
**from** *PRelQ* **have** $P \rightsquigarrow$<*Rel*> $Q$ **by**(*rule Sim*)
**with** *QTrans* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow\hat{} \alpha \prec P'$ **and** *P'RelQ'*: $(P', Q') \in Rel$
**by**(*blast dest*: *simE*)

**from** *PTrans* **have** $P \parallel R \Longrightarrow\hat{} \alpha \prec P' \parallel R$ **by**(*rule Weak-Early-Semantics.Par1F*)
**moreover from** *P'RelQ'* *RBangRelQ* **have** $(P' \parallel R, Q' \parallel !Q) \in$ *bangRel Rel*
**by**(*rule Rel.BRPar*)
**ultimately show** $\exists P'. P \parallel R \Longrightarrow\hat{} \alpha \prec P' \wedge (P', Q' \parallel !Q) \in Rel'$ **using** *BangRelRel'* **by** *blast*
**qed**
**qed**
**next**
**case**(*Par2B a x* $Q'$ $P$)
**hence** *IH*: $\bigwedge P. (P, !Q) \in$ *bangRel Rel* $\Longrightarrow$ *?Sim P* (*a*<$\nu x$> $\prec Q'$) **by** *simp*
**have** *xFreshQ*: $x \sharp Q$ **by** *fact*
**have** $(P, Q \parallel !Q) \in$ *bangRel Rel* **and** $x \sharp P$ **by** *fact+*
**thus** *?case*
**proof**(*induct rule*: *BRParCases*)
**case**(*BRPar P R*)
**have** *PRelQ*: $(P, Q) \in Rel$ **and** *RBangRelQ*: $(R, !Q) \in$ *bangRel Rel* **by** *fact+*
**have** $x \sharp P \parallel R$ **by** *fact*
**hence** *xFreshP*: $x \sharp P$ **and** *xFreshR*: $x \sharp R$ **by** *simp+*
**show** *?case* **using** ‹$x \sharp Q$›
**proof**(*auto simp add*: *residual.inject alpha' name-fresh-fresh*)
**from** *IH RBangRelQ* **have** *?Sim R* (*a*<$\nu x$> $\prec Q'$) **by** *blast*
**with** *xFreshR* **obtain** $R'$ **where** *RTrans*: $R \Longrightarrow a$<$\nu x$> $\prec R'$ **and** *R'BangRelQ'*: $(R', Q') \in Rel'$
**by**(*blast dest*: *simE*)
**from** *RTrans xFreshP* **have** $P \parallel R \Longrightarrow a$<$\nu x$> $\prec (P \parallel R')$
**by**(*auto intro*: *Weak-Early-Step-Semantics.Par2B*)
**moreover from** *PRelQ* *R'BangRelQ'* **have** $(P \parallel R', Q \parallel Q') \in Rel'$
**by**(*rule ParComp*)

**ultimately show** $\exists P'. \ P \parallel R \Longrightarrow a{<}\nu x{>} \prec P' \wedge (P', Q \parallel Q') \in Rel'$ **by** *blast*
**next**
**fix** $y$
**assume** $(y{::}name) \sharp Q'$ **and** $y \sharp R$ **and** $y \sharp P$
**from** *IH RBangRelQ* **have** $?Sim\ R\ (a{<}\nu x{>} \prec Q')$ **by** *blast*
**with** ‹$y \sharp Q'$› **have** $?Sim\ R\ (a{<}\nu y{>} \prec ([(x, y)] \cdot Q'))$ **by**(*simp add*: *alphaBoundOutput*)
**with** ‹$y \sharp R$›**obtain** $R'$ **where** *RTrans*: $R \Longrightarrow a{<}\nu y{>} \prec R'$ **and** $R'BangRelQ'$: $(R', [(x, y)] \cdot Q') \in Rel'$
**by**(*blast dest*: *simE*)
**from** *RTrans* ‹$y \sharp P$› **have** $P \parallel R \Longrightarrow a{<}\nu y{>} \prec (P \parallel R')$
**by**(*auto intro*: *Weak-Early-Step-Semantics.Par2B*)
**moreover from** *PRelQ* $R'BangRelQ'$ **have** $(P \parallel R', Q \parallel ([(y, x)] \cdot Q')) \in Rel'$
**by**(*fastforce intro*: *ParComp simp add*: *name-swap*)
**ultimately show** $\exists P'. \ P \parallel R \Longrightarrow a{<}\nu y{>} \prec P' \wedge (P', Q \parallel ([(y, x)] \cdot Q')) \in Rel'$ **by** *blast*
**qed**
**qed**
**next**
**case**(*Par2F* $\alpha$ $Q'$ $P$)
**hence** *IH*: $\bigwedge P. \ (P, !Q) \in bangRel\ Rel \Longrightarrow ?Sim\ P\ (\alpha \prec Q')$ **by** *simp*
**have** $(P, Q \parallel !Q) \in bangRel\ Rel$ **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *BRParCases*)
**case**(*BRPar P R*)
**have** *PRelQ*: $(P, Q) \in Rel$ **and** *RBangRelQ*: $(R, !Q) \in bangRel\ Rel$ **by** *fact+*
**show** *?case*
**proof**(*auto simp add*: *residual.inject*)
**from** *RBangRelQ* **have** $?Sim\ R\ (\alpha \prec Q')$ **by**(*rule IH*)
**then obtain** $R'$ **where** *RTrans*: $R \Longrightarrow\hat{}\ \alpha \prec R'$ **and** $R'RelQ'$: $(R', Q') \in Rel'$
**by**(*blast dest*: *simE*)
**from** *RTrans* **have** $P \parallel R \Longrightarrow\hat{}\ \alpha \prec P \parallel R'$ **by**(*rule Weak-Early-Semantics.Par2F*)
**moreover from** *PRelQ* $R'RelQ'$ **have** $(P \parallel R', Q \parallel Q') \in Rel'$ **by**(*rule ParComp*)
**ultimately show** $\exists P'. \ P \parallel R \Longrightarrow\hat{}\ \alpha \prec P' \wedge (P', Q \parallel Q') \in Rel'$ **by** *blast*
**qed**
**qed**
**next**
**case**(*Comm1 a* $Q'$ *b* $Q''$ *P*)
**hence** *IH*: $\bigwedge P. \ (P, !Q) \in bangRel\ Rel \Longrightarrow ?Sim\ P\ (a[b] \prec Q'')$ **by** *simp*
**have** *QTrans*: $Q \longmapsto a{<}b{>} \prec Q'$ **by** *fact*
**have** $(P, Q \parallel !Q) \in bangRel\ Rel$ **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *BRParCases*)
**case**(*BRPar P R*)

**have** *PRelQ*: $(P, Q) \in Rel$ **and** *RBangRelQ*: $(R, !Q) \in bangRel\ Rel$ **by** *fact+*
**show** *?case*
**proof**(*auto simp add*: *residual.inject*)
**from** *PRelQ* **have** $P \rightsquigarrow$<*Rel*> $Q$ **by**(*rule Sim*)
**with** *QTrans* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow a$<b> $\prec P'$ **and** $P'RelQ'$: $(P', Q') \in Rel$
**by**(*fastforce dest*: *simE simp add*: *weakFreeTransition-def*)

**from** *RBangRelQ* **have** *?Sim R* $(a[b] \prec Q'')$ **by**(*rule IH*)
**then obtain** $R'$ **where** *RTrans*: $R \Longrightarrow a[b] \prec R'$
**and** $R'RelQ''$: $(R', Q'') \in Rel'$
**by**(*fastforce dest*: *simE simp add*: *weakFreeTransition-def*)
**from** *PTrans RTrans* **have** $P \parallel R \Longrightarrow \tau \prec (P' \parallel R')$
**by**(*rule Weak-Early-Step-Semantics.Comm1*)
**hence** $P \parallel R \Longrightarrow_\tau P' \parallel R'$
**by**(*auto simp add*: *trancl-into-rtrancl dest*: *Weak-Early-Step-Semantics.tauTransitionChain*)
**moreover from** $P'RelQ'$ $R'RelQ''$ **have** $(P' \parallel R', Q' \parallel Q'') \in Rel'$
**by**(*rule ParComp*)
**ultimately show** $\exists P'. (P \parallel R, P') \in \{(P, P'). P \longmapsto \tau \prec P'\}^* \wedge (P', Q' \parallel Q'') \in Rel'$
**by** *auto*
**qed**
**qed**
**next**
**case**(*Comm2 a b* $Q'$ $Q''$ *P*)
**hence** *IH*: $\bigwedge P. (P, !Q) \in bangRel\ Rel \Longrightarrow$ *?Sim P* $(a$<b> $\prec Q'')$ **by** *simp*
**have** *QTrans*: $Q \longmapsto a[b] \prec Q'$ **by** *fact*
**have** $(P, Q \parallel !Q) \in bangRel\ Rel$ **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *BRParCases*)
**case**(*BRPar P R*)
**have** *PRelQ*: $(P, Q) \in Rel$ **and** *RBangRelQ*: $(R, !Q) \in bangRel\ Rel$ **by** *fact+*
**show** *?case*
**proof**(*auto simp add*: *residual.inject*)
**from** *PRelQ* **have** $P \rightsquigarrow$<*Rel*> $Q$ **by**(*rule Sim*)
**with** *QTrans* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow a[b] \prec P'$ **and** $P'RelQ'$: $(P', Q') \in Rel$
**by**(*fastforce dest*: *simE simp add*: *weakFreeTransition-def*)

**from** *RBangRelQ* **have** *?Sim R* $(a$<b> $\prec Q'')$ **by**(*rule IH*)
**then obtain** $R'$ **where** *RTrans*: $R \Longrightarrow a$<b> $\prec R'$ **and** $R'BangRelQ''$: $(R', Q'') \in Rel'$
**by**(*fastforce dest*: *simE simp add*: *weakFreeTransition-def*)

**from** *PTrans RTrans* **have** $P \parallel R \Longrightarrow \tau \prec (P' \parallel R')$
**by**(*rule Weak-Early-Step-Semantics.Comm2*)
**hence** $P \parallel R \Longrightarrow_\tau P' \parallel R'$

**by**(*auto simp add*: *trancl-into-rtrancl dest*: *Weak-Early-Step-Semantics.tauTransitionChain*)
**moreover from** $P'RelQ'$ $R'BangRelQ''$ **have** $(P' \parallel R', Q' \parallel Q'') \in Rel'$
**by**(*rule ParComp*)
**ultimately show** $\exists P'. (P \parallel R, P') \in \{(P, P'). P \longmapsto \tau \prec P'\}^* \wedge (P', Q' \parallel Q'') \in Rel'$ **by** *auto*
**qed**
**qed**
**next**
**case**(*Close1 a x Q' Q'' P*)
**hence** *IH*: $\bigwedge P. (P, !Q) \in bangRel\ Rel \Longrightarrow ?Sim\ P\ (a<\nu x> \prec Q'')$ **by** *simp*
**have** *QTrans*: $Q \longmapsto a<x> \prec Q'$ **by** *fact*
**have** $(P, Q \parallel !Q) \in bangRel\ Rel$ **and** $x \sharp P$ **by** *fact+*
**thus** *?case*
**proof**(*induct rule*: *BRParCases*)
**case**(*BRPar P R*)
**have** *PRelQ*: $(P, Q) \in Rel$ **and** *RBangRelQ*: $(R, !Q) \in bangRel\ Rel$ **by** *fact+*
**have** $x \sharp P \parallel R$ **by** *fact*
**hence** *xFreshR*: $x \sharp R$ **and** *xFreshP*: $x \sharp P$ **by** *simp+*
**show** *?case*
**proof**(*auto simp add*: *residual.inject*)
**from** *PRelQ* **have** $P \leadsto<Rel> Q$ **by**(*rule Sim*)
**with** *QTrans* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow a<x> \prec P'$ **and** $P'RelQ'$: $(P', Q') \in Rel$
**by**(*fastforce dest*: *simE simp add*: *weakFreeTransition-def*)

**from** *RBangRelQ* **have** $?Sim\ R\ (a<\nu x> \prec Q'')$ **by**(*rule IH*)
**with** *xFreshR* **obtain** $R'$ **where** *RTrans*: $R \Longrightarrow a<\nu x> \prec R'$
**and** $R'RelQ''$: $(R', Q'') \in Rel'$
**by**(*blast dest*: *simE*)

**from** *PTrans RTrans xFreshP* **have** $P \parallel R \Longrightarrow \tau \prec <\nu x>(P' \parallel R')$
**by**(*rule Weak-Early-Step-Semantics.Close1*)
**moreover from** $P'RelQ'$ $R'RelQ''$ **have** $(<\nu x>(P' \parallel R'), <\nu x>(Q' \parallel Q'')) \in Rel'$
**by**(*force intro*: *ParComp Res*)
**ultimately show** $\exists P'. (P \parallel R, P') \in \{(P, P'). P \longmapsto \tau \prec P'\}^* \wedge (P', <\nu x>(Q' \parallel Q'')) \in Rel'$ **by** *auto*
**qed**
**qed**
**next**
**case**(*Close2 a x Q' Q'' P*)
**hence** *IH*: $\bigwedge P. (P, !Q) \in bangRel\ Rel \Longrightarrow ?Sim\ P\ (a<x> \prec Q'')$ **by** *simp*
**have** *QTrans*: $Q \longmapsto a<\nu x> \prec Q'$ **by** *fact*
**have** $(P, Q \parallel !Q) \in bangRel\ Rel$ **and** $x \sharp P$ **by** *fact+*
**thus** *?case*
**proof**(*induct rule*: *BRParCases*)
**case**(*BRPar P R*)
**have** *PRelQ*: $(P, Q) \in Rel$ **and** *RBangRelQ*: $(R, !Q) \in bangRel\ Rel$ **by**

*fact+*
**have** $x \sharp P \parallel R$ **by** *fact*
**hence** *xFreshP*: $x \sharp P$ **and** *xFreshR*: $x \sharp R$ **by** *simp+*
**show** *?case*
**proof**(*auto simp add*: *residual.inject*)
**from** *PRelQ* **have** $P \leadsto<Rel>\ Q$ **by**(*rule Sim*)
**with** *QTrans xFreshP* **obtain** $P'$ **where** *PTrans*: $P \Longrightarrow a<\nu x> \prec P'$
**and** $P'RelQ'$: $(P', Q') \in Rel$
**by**(*blast dest*: *simE*)

**from** *RBangRelQ* **have** *?Sim R* $(a<x> \prec Q'')$ **by**(*rule IH*)
**with** *xFreshR* **obtain** $R'$ **where** *RTrans*: $R \Longrightarrow a<x> \prec R'$
**and** $R'RelQ''$: $(R', Q'') \in Rel'$
**by**(*fastforce simp add*: *weakFreeTransition-def*)
**from** *PTrans RTrans xFreshR* **have** $P \parallel R \Longrightarrow \tau \prec <\nu x>(P' \parallel R')$
**by**(*rule Weak-Early-Step-Semantics.Close2*)
**moreover from** $P'RelQ'$ $R'RelQ''$ **have** $(<\nu x>(P' \parallel R'), <\nu x>(Q' \parallel Q'')) \in Rel'$
**by**(*force intro*: *ParComp Res*)
**ultimately show** $\exists P'. (P \parallel R, P') \in \{(P, P'). P \longmapsto \tau \prec P'\}^* \wedge (P', <\nu x>(Q' \parallel Q'')) \in Rel'$ **by** *auto*
**qed**
**qed**
**next**
**case**(*Bang Rs*)
**hence** *IH*: $\bigwedge P. (P, Q \parallel !Q) \in bangRel\ Rel \Longrightarrow ?Sim\ P\ Rs$ **by** *simp*
**have** $(P, !Q) \in bangRel\ Rel$ **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *BRBangCases*)
**case**(*BRBang P*)
**have** *PRelQ*: $(P, Q) \in Rel$ **by** *fact*
**hence** $(!P, !Q) \in bangRel\ Rel$ **by**(*rule Rel.BRBang*)
**with** *PRelQ* **have** $(P \parallel !P, Q \parallel !Q) \in bangRel\ Rel$ **by**(*rule Rel.BRPar*)
**hence** *IH*: *?Sim* $(P \parallel !P)$ *Rs* **by**(*rule IH*)
**show** *?case*
**proof**(*intro conjI allI impI*)
**fix** $Q'\ a\ x$
**assume** $Rs = a<\nu x> \prec Q'$ **and** $x \sharp !P$
**then obtain** $P'$ **where** *PTrans*: $(P \parallel !P) \Longrightarrow a<\nu x> \prec P'$
**and** $P'RelQ'$: $(P', Q') \in Rel'$ **using** *IH*
**by**(*auto simp add*: *residual.inject*)
**from** *PTrans* **have** $!P \Longrightarrow a<\nu x> \prec P'$
**by**(*force intro*: *Weak-Early-Step-Semantics.Bang simp add*: *weakFreeTransition-def*)
**with** $P'RelQ'$ **show** $\exists P'. !P \Longrightarrow a<\nu x> \prec P' \wedge (P', Q') \in Rel'$ **by** *blast*
**next**
**fix** $Q'\ \alpha$
**assume** $Rs = \alpha \prec Q'$
**then obtain** $P'$ **where** *PTrans*: $(P \parallel !P) \Longrightarrow\hat{}\ \alpha \prec P'$

**and** $P'RelQ'$: $(P', Q') \in Rel'$ **using** *IH*
**by** *auto*
**from** *PTrans* **show** $\exists P'.\ !P \Longrightarrow\hat{}\alpha \prec P' \wedge (P', Q') \in Rel'$ **using** $P'RelQ'$
**proof**(*induct rule*: *transitionCases*)
**case** *Step*
**have** $P \parallel !P \Longrightarrow\alpha \prec P'$ **by** *fact*
**hence** $!P \Longrightarrow\alpha \prec P'$ **by**(*rule Weak-Early-Step-Semantics.Bang*)
**with** $P'RelQ'$ **show** *?case* **by**(*force simp add*: *weakFreeTransition-def*)
**next**
**case** *Stay*
**have** $!P \Longrightarrow\hat{}\tau \prec !P$ **by**(*simp add*: *weakFreeTransition-def*)
**moreover assume** $(P \parallel !P, Q') \in Rel'$
**hence** $(!P, Q') \in Rel'$ **by**(*blast intro*: *RelStay*)
**ultimately show** *?case* **by** *blast*
**qed**
**qed**
**qed**
**qed**
}
**moreover from** *PRelQ* **have** $(!P, !Q) \in bangRel\ Rel$ **by**(*rule Rel.BRBang*)
**ultimately show** *?thesis* **by**(*auto simp add*: *weakSimulation-def*)
**qed**

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

**assumes** *PBangRelQ*: $(P, Q) \in bangRel\ Rel$
**and** *Sim*: $\bigwedge R\ S.\ (R, S) \in Rel \Longrightarrow R \rightsquigarrow<Rel> S$

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

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

**shows** $P \rightsquigarrow<Rel'> Q$
**proof** −
**from** *PBangRelQ* **show** *?thesis*
**proof**(*induct rule*: *bangRel.induct*)
**case**(*BRBang P Q*)
**have** *PRelQ*: $(P, Q) \in Rel$ **by** *fact*
**thus** *?case* **using** *ParComp Res BangRelRel' eqvtRel' Eqvt RelStay Sim*
**by**(*rule-tac bangPres*)

**next**
**case**(*BRPar P Q R T*)
**have** (*P*, *Q*) ∈ *Rel* **by** *fact*
**moreover hence** *P* ⇝<*Rel*> *Q* **by**(*rule Sim*)
**moreover have** *R* ⇝<*Rel′*> *T* **by** *fact*
**moreover have** (*R*, *T*) ∈ *bangRel Rel* **by** *fact*
**ultimately show** *?case* **using** *ParComp eqvtRel′ Res Eqvt BangRelRel′*
**by**(*blast intro*: *parCompose*)
**next**
**case**(*BRRes P Q x*)
**have** *P* ⇝<*Rel′*> *Q* **by** *fact*
**thus** *?case* **using** *BangRelRel′ eqvtRel′ Res* **by**(*blast intro*: *resPres*)
**qed**
**qed**

**end**

**theory** *Strong-Early-Late-Comp*
**imports** *Strong-Late-Bisim-Subst-SC Strong-Early-Bisim-Subst*
**begin**

**abbreviation** *TransitionsLate-judge* (‹- $\longmapsto_l$ -› [*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′* ≡ (*Late-Semantics.BoundR* (*Late-Semantics.InputS a*) *x P′*)
**abbreviation** *Transitions-OutputjudgeLate* (‹-[-] $\prec_l$ -› [*80*, *80*] *80*) **where** *a*[*b*] $\prec_l$ *P′* ≡ (*Late-Semantics.FreeR* (*Late-Semantics.OutputR a b*) *P′*)
**abbreviation** *Transitions-BoundOutputjudgeLate* (‹-<ν-> $\prec_l$ -› [*80*, *80*] *80*) **where** *a*<ν*x*> $\prec_l$ *P′* ≡ (*Late-Semantics.BoundR* (*Late-Semantics.BoundOutputS a*) *x P′*)
**abbreviation** *Transitions-TaujudgeLate* (‹τ $\prec_l$ -› *80*) **where** τ $\prec_l$ *P′* ≡ (*Late-Semantics.FreeR Late-Semantics.TauR P′*)

**abbreviation** *Transitions-InputjudgeEarly* (‹-<-> $\prec_e$ -› [*80*, *80*] *80*) **where** *a*<*x*> $\prec_e$ *P′* ≡ (*Early-Semantics.FreeR* (*Early-Semantics.InputR a x*) *P′*)
**abbreviation** *Transitions-OutputjudgeEarly* (‹-[-] $\prec_e$ -› [*80*, *80*] *80*) **where** *a*[*b*] $\prec_e$ *P′* ≡ (*Early-Semantics.FreeR* (*Early-Semantics.OutputR a b*) *P′*)
**abbreviation** *Transitions-BoundOutputjudgeEarly* (‹-<ν-> $\prec_e$ -› [*80*, *80*] *80*) **where** *a*<ν*x*> $\prec_e$ *P′* ≡(*Early-Semantics.BoundOutputR a x P′*)
**abbreviation** *Transitions-TaujudgeEarly* (‹τ $\prec_e$ -› *80*) **where** τ $\prec_e$ *P′* ≡ (*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** $P \longmapsto_l a[b] \prec_l P'$
**using** *assms*
**proof**(*nominal-induct rule*: *Early-Semantics.outputInduct*)
**case**(*Output a b P*)
**show** *?case* **by**(*rule Late-Semantics.Output*)
**next**
**case**(*Match P a b P' c*)
**have** $P \longmapsto_l a[b] \prec_l P'$ **by** *fact*
**thus** *?case* **by**(*rule Late-Semantics.Match*)
**next**
**case**(*Mismatch P a b P' c d*)
**from** ‹$P \longmapsto_l a[b] \prec_l P'$› ‹$c \neq d$›
**show** *?case* **by**(*rule Late-Semantics.Mismatch*)
**next**
**case**(*Sum1 P a b P' Q*)
**have** $P \longmapsto_l a[b] \prec_l P'$ **by** *fact*
**thus** *?case* **by**(*rule Late-Semantics.Sum1*)
**next**
**case**(*Sum2 Q a b Q' P*)
**have** $Q \longmapsto_l a[b] \prec_l Q'$ **by** *fact*
**thus** *?case* **by**(*rule Late-Semantics.Sum2*)
**next**
**case**(*Par1 P a b P' Q*)
**have** $P \longmapsto_l a[b] \prec_l P'$ **by** *fact*
**thus** *?case* **by**(*rule Late-Semantics.Par1F*)
**next**
**case**(*Par2 Q a b Q' P*)
**have** $Q \longmapsto_l a[b] \prec_l Q'$ **by** *fact*
**thus** *?case* **by**(*rule Late-Semantics.Par2F*)
**next**
**case**(*Res P a b P' x*)
**have** $P \longmapsto_l a[b] \prec_l P'$ **and** $x \neq a$ **and** $x \neq b$ **by** *fact+*
**thus** *?case* **by**(*force intro*: *Late-Semantics.ResF*)
**next**
**case**(*Bang P a b P'*)
**have** $P \parallel !P \longmapsto_l a[b] \prec_l P'$ **by** *fact*
**thus** *?case* **by**(*rule Late-Semantics.Bang*)
**qed**

**lemma** *lateEarlyOutput*:
**fixes** $P$ :: *pi*
**and** $a$ :: *name*
**and** $b$ :: *name*
**and** $P'$ :: *pi*

**assumes** $P \longmapsto_l a[b] \prec_l P'$

**shows** $P \longmapsto_e a[b] \prec_e P'$
**using** *assms*
**proof**(*nominal-induct rule*: *Late-Semantics.outputInduct*)
**case**(*Output a b P*)
**thus** *?case* **by**(*rule Early-Semantics.Output*)
**next**
**case**(*Match P a b P' c*)
**have** $P \longmapsto_e a[b] \prec_e P'$ **by** *fact*
**thus** *?case* **by**(*rule Early-Semantics.Match*)
**next**
**case**(*Mismatch P a b P' c d*)
**have** $P \longmapsto_e a[b] \prec_e P'$ **and** $c \neq d$ **by** *fact+*
**thus** *?case* **by**(*rule Early-Semantics.Mismatch*)
**next**
**case**(*Sum1 P a b P' Q*)
**have** $P \longmapsto_e a[b] \prec_e P'$ **by** *fact*
**thus** *?case* **by**(*rule Early-Semantics.Sum1*)
**next**
**case**(*Sum2 Q a b Q' P*)
**have** $Q \longmapsto_e a[b] \prec_e Q'$ **by** *fact*
**thus** *?case* **by**(*rule Early-Semantics.Sum2*)
**next**
**case**(*Par1 P a b P' Q*)
**have** $P \longmapsto_e a[b] \prec_e P'$ **by** *fact*
**thus** *?case* **by**(*rule Early-Semantics.Par1F*)
**next**
**case**(*Par2 Q a b Q' P*)
**have** $Q \longmapsto_e a[b] \prec_e Q'$ **by** *fact*
**thus** *?case* **by**(*rule Early-Semantics.Par2F*)
**next**
**case**(*Res P a b P' x*)
**have** $P \longmapsto_e a[b] \prec_e P'$ **and** $x \neq a$ **and** $x \neq b$ **by** *fact+*
**thus** *?case* **by**(*force intro*: *Early-Semantics.ResF*)
**next**
**case**(*Bang P a b P'*)
**have** $P \parallel !P \longmapsto_e a[b] \prec_e P'$ **by** *fact*
**thus** *?case* **by**(*rule Early-Semantics.Bang*)
**qed**

**lemma** *outputEq*:
**fixes** $P$ :: *pi*
**and** $a$ :: *name*
**and** $b$ :: *name*
**and** $P'$ :: *pi*

**shows** $P \longmapsto_e a[b] \prec_e P' = P \longmapsto_l a[b] \prec_l P'$
**by**(*auto intro*: *lateEarlyOutput earlyLateOutput*)

**lemma** *lateEarlyBoundOutput*:
  **fixes** $P$ :: *pi*
  **and** $a$ :: *name*
  **and** $x$ :: *name*
  **and** $P'$ :: *pi*

  **assumes** $P \longmapsto_l a{<}\nu x{>} \prec_l P'$

  **shows** $P \longmapsto_e a{<}\nu x{>} \prec_e P'$
**proof** −
  **have** *Goal*: $\bigwedge P\ a\ x\ P'.\ [\![P \longmapsto_l a{<}\nu x{>} \prec_l P';\ x \sharp P]\!] \Longrightarrow P \longmapsto_e a{<}\nu x{>} \prec_e P'$
  **proof** −
    **fix** $P\ a\ x\ P'$
    **assume** $P \longmapsto_l a{<}\nu x{>} \prec_l P'$ **and** $x \sharp P$
    **thus** $P \longmapsto_e a{<}\nu x{>} \prec_e P'$
    **proof**(*nominal-induct rule*: *Late-Semantics.boundOutputInduct*)
      **case**(*Match P a x P' b*)
      **have** $P \longmapsto_e a{<}\nu x{>} \prec_e P'$ **by** *fact*
      **thus** *?case* **by**(*rule Early-Semantics.Match*)
    **next**
      **case**(*Mismatch P a x P' b c*)
      **have** $P \longmapsto_e a{<}\nu x{>} \prec_e P'$ **and** $b \neq c$ **by** *fact+*
      **thus** *?case* **by**(*rule Early-Semantics.Mismatch*)
    **next**
      **case**(*Open P a x P'*)
      **have** $P \longmapsto_l a[x] \prec_l P'$ **by** *fact*
      **hence** $P \longmapsto_e a[x] \prec_e P'$ **by**(*rule lateEarlyOutput*)
      **moreover have** $a \neq x$ **by** *fact*
      **ultimately show** *?case* **by**(*rule Early-Semantics.Open*)
    **next**
      **case**(*Sum1 P Q a x P'*)
      **have** $P \longmapsto_e a{<}\nu x{>} \prec_e P'$ **by** *fact*
      **thus** *?case* **by**(*rule Early-Semantics.Sum1*)
    **next**
      **case**(*Sum2 P Q a x Q'*)
      **have** $Q \longmapsto_e a{<}\nu x{>} \prec_e Q'$ **by** *fact*
      **thus** *?case* **by**(*rule Early-Semantics.Sum2*)
    **next**
      **case**(*Par1 P P' Q a x*)
      **have** $P \longmapsto_e a{<}\nu x{>} \prec_e P'$ **and** $x \sharp Q$ **by** *fact+*
      **thus** *?case* **by**(*rule Early-Semantics.Par1B*)
    **next**
      **case**(*Par2 P Q Q' a x*)
      **have** $Q \longmapsto_e a{<}\nu x{>} \prec_e Q'$ **and** $x \sharp P$ **by** *fact+*
      **thus** *?case* **by**(*rule Early-Semantics.Par2B*)
    **next**
      **case**(*Res P P' a x y*)
      **have** $P \longmapsto_e a{<}\nu x{>} \prec_e P'$ **and** $y \neq a$ **and** $y \neq x$ **by** *fact+*
      **thus** *?case* **by**(*force intro*: *Early-Semantics.ResB*)

**next**
**case**(*Bang P a x P'*)
**have** $P \parallel !P \longmapsto_e a{<}\nu x{>} \prec_e P'$ **by** *fact*
**thus** *?case* **by**(*rule Early-Semantics.Bang*)
**qed**
**qed**

**have** $\exists$ *c::name. c* $\sharp$ $(P, P', x)$ **by**(*blast intro*: *name-exists-fresh*)
**then obtain** *c::name* **where** *cFreshP*: $c \sharp P$ **and** *cFreshP'*: $c \sharp P'$ **and** $c \neq x$
**by**(*force simp add*: *fresh-prod*)
**from** *assms cFreshP'* **have** $P \longmapsto_l a{<}\nu c{>} \prec_l ([(x, c)] \cdot P')$
**by**(*simp add*: *Late-Semantics.alphaBoundResidual*)
**hence** $P \longmapsto_e a{<}\nu c{>} \prec_e ([(x, c)] \cdot P')$ **using** *cFreshP*
**by**(*rule Goal*)
**moreover from** *cFreshP'* ‹$c \neq x$› **have** $x \sharp [(x, c)] \cdot P'$ **by**(*simp add*: *name-fresh-left name-calc*)
**ultimately show** *?thesis* **by**(*simp add*: *Early-Semantics.alphaBoundOutput name-swap*)
**qed**

**lemma** *earlyLateBoundOutput*:
**fixes** $P$ :: *pi*
**and** $a$ :: *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** −
**have** *Goal*: $\bigwedge P\ a\ x\ P'.\ [\![P \longmapsto_e a{<}\nu x{>} \prec_e P';\ x \sharp P]\!] \Longrightarrow P \longmapsto_l a{<}\nu x{>} \prec_l P'$
**proof** −
**fix** $P\ a\ x\ P'$
**assume** $P \longmapsto_e a{<}\nu x{>} \prec P'$ **and** $x \sharp P$
**thus** $P \longmapsto_l a{<}\nu x{>} \prec_l P'$
**proof**(*nominal-induct rule*: *Early-Semantics.boundOutputInduct*)
**case**(*Match P a x P' b*)
**have** $P \longmapsto_l a{<}\nu x{>} \prec P'$ **by** *fact*
**thus** *?case* **by**(*rule Late-Semantics.Match*)
**next**
**case**(*Mismatch P a x P' b c*)
**have** $P \longmapsto_l a{<}\nu x{>} \prec P'$ **and** $b \neq c$ **by** *fact+*
**thus** *?case* **by**(*rule Late-Semantics.Mismatch*)
**next**
**case**(*Open P a x P'*)
**have** $P \longmapsto_e a[x] \prec_e P'$ **by** *fact*
**hence** $P \longmapsto_l a[x] \prec_l P'$ **by**(*rule earlyLateOutput*)
**moreover have** $a \neq x$ **by** *fact*
**ultimately show** *?case* **by**(*rule Late-Semantics.Open*)
**next**

**case**(*Sum1 P Q a x P'*)
**have** $P \longmapsto_l a{<}\nu x{>} \prec_l P'$ **by** *fact*
**thus** *?case* **by**(*rule Late-Semantics.Sum1*)
**next**
**case**(*Sum2 P Q a x Q'*)
**have** $Q \longmapsto_l a{<}\nu x{>} \prec_l Q'$ **by** *fact*
**thus** *?case* **by**(*rule Late-Semantics.Sum2*)
**next**
**case**(*Par1 P P' Q a x*)
**have** $P \longmapsto_l a{<}\nu x{>} \prec_l P'$ **and** $x \sharp Q$ **by** *fact+*
**thus** *?case* **by**(*rule Late-Semantics.Par1B*)
**next**
**case**(*Par2 P Q Q' a x*)
**have** $Q \longmapsto_l a{<}\nu x{>} \prec_l Q'$ **and** $x \sharp P$ **by** *fact+*
**thus** *?case* **by**(*rule Late-Semantics.Par2B*)
**next**
**case**(*Res P P' a x y*)
**have** $P \longmapsto_l a{<}\nu x{>} \prec_l P'$ **and** $y \neq a$ **and** $y \neq x$ **by** *fact+*
**thus** *?case* **by**(*force intro*: *Late-Semantics.ResB*)
**next**
**case**(*Bang P a x P'*)
**have** $P \parallel !P \longmapsto_l a{<}\nu x{>} \prec P'$ **by** *fact*
**thus** *?case* **by**(*rule Late-Semantics.Bang*)
**qed**
**qed**

**have** $\exists$ *c*::*name*. $c \sharp (P, P', x)$ **by**(*blast intro*: *name-exists-fresh*)
**then obtain** *c*::*name* **where** *cFreshP*: $c \sharp P$ **and** *cFreshP'*: $c \sharp P'$ **and** $c \neq x$
**by**(*force simp add*: *fresh-prod*)
**from** *assms cFreshP'* **have** $P \longmapsto_e a{<}\nu c{>} \prec_e ([(x, c)] \cdot P')$
**by**(*simp add*: *Early-Semantics.alphaBoundOutput*)
**hence** $P \longmapsto_l a{<}\nu c{>} \prec_l ([(x, c)] \cdot P')$ **using** *cFreshP*
**by**(*rule Goal*)
**moreover from** *cFreshP'* ‹$c \neq x$› **have** $x \sharp [(x, c)] \cdot P'$ **by**(*simp add*: *name-fresh-left name-calc*)
**ultimately show** *?thesis* **by**(*simp add*: *Late-Semantics.alphaBoundResidual name-swap*)
**qed**

**lemma** *BoundOutputEq*:
**fixes** $P$ :: *pi*
**and** $a$ :: *name*
**and** $x$ :: *name*
**and** $P'$ :: *pi*

**shows** $P \longmapsto_e a{<}\nu x{>} \prec_e P' = P \longmapsto_l a{<}\nu x{>} \prec_l P'$
**by**(*auto intro*: *earlyLateBoundOutput lateEarlyBoundOutput*)

**lemma** *lateEarlyInput*:
**fixes** $P$ :: *pi*

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

**assumes** *PTrans*: $P \longmapsto_l a{<}x{>} \prec_l P'$

**shows** $P \longmapsto_e a{<}u{>} \prec_e (P'[x{::=}u])$
**proof** −
**have** *Goal*: $\bigwedge P\ a\ x\ P'\ u.\ [\![P \longmapsto_l a{<}x{>} \prec_l P';\ x \sharp P]\!] \Longrightarrow P \longmapsto_e a{<}u{>} \prec_e (P'[x{::=}u])$
**proof** −
**fix** $P\ a\ x\ P'\ u$
**assume** $P \longmapsto_l a{<}x{>} \prec_l P'$ **and** $x \sharp P$
**thus** $P \longmapsto_e a{<}u{>} \prec_e (P'[x{::=}u])$
**proof**(*nominal-induct avoiding*: *u rule*: *Late-Semantics.inputInduct*)
**case**(*Input a x P*)
**thus** *?case* **by**(*rule Early-Semantics.Input*)
**next**
**case**(*Match P a x P' b u*)
**have** $P \longmapsto_e a{<}u{>} \prec_e (P'[x{::=}u])$ **by** *fact*
**thus** *?case* **by**(*rule Early-Semantics.Match*)
**next**
**case**(*Mismatch P a x P' b c u*)
**have** $P \longmapsto_e a{<}u{>} \prec_e (P'[x{::=}u])$ **by** *fact*
**moreover have** $b \neq c$ **by** *fact*
**ultimately show** *?case* **by**(*rule Early-Semantics.Mismatch*)
**next**
**case**(*Sum1 P Q a x P'*)
**have** $P \longmapsto_e a{<}u{>} \prec_e (P'[x{::=}u])$ **by** *fact*
**thus** *?case* **by**(*rule Early-Semantics.Sum1*)
**next**
**case**(*Sum2 P Q a x Q'*)
**have** $Q \longmapsto_e a{<}u{>} \prec_e (Q'[x{::=}u])$ **by** *fact*
**thus** *?case* **by**(*rule Early-Semantics.Sum2*)
**next**
**case**(*Par1 P P' Q a x*)
**have** $P \longmapsto_e a{<}u{>} \prec_e (P'[x{::=}u])$ **by** *fact*
**hence** $P \parallel Q \longmapsto_e a{<}u{>} \prec_e (P'[x{::=}u] \parallel Q)$ **by**(*rule Early-Semantics.Par1F*)
**moreover have** $x \sharp Q$ **by** *fact*
**ultimately show** *?case* **by**(*simp add*: *forget*)
**next**
**case**(*Par2 P Q Q' a x*)
**have** $Q \longmapsto_e a{<}u{>} \prec_e (Q'[x{::=}u])$ **by** *fact*
**hence** $P \parallel Q \longmapsto_e a{<}u{>} \prec_e (P \parallel Q'[x{::=}u])$ **by**(*rule Early-Semantics.Par2F*)
**moreover have** $x \sharp P$ **by** *fact*
**ultimately show** *?case* **by**(*simp add*: *forget*)
**next**
**case**(*Res P P' a x y u*)

**have** $P \longmapsto_e a{<}u{>} \prec_e (P'[x{::=}u])$ **and** $y \neq a$ **and** *yinequ*: $y \sharp u$ **by** *fact+*
**hence** $<\nu y>P \longmapsto_e a{<}u{>} \prec_e <\nu y>(P'[x{::=}u])$ **by**(*force intro*: *Early-Semantics.ResF*)
**moreover have** $y \neq x$ **by** *fact*
**ultimately show** *?case* **using** *yinequ* **by** *simp*
**next**
**case**(*Bang P a x P' u*)
**have** $P \parallel !P \longmapsto_e a{<}u{>} \prec_e (P'[x{::=}u])$ **by** *fact*
**thus** *?case* **by**(*rule Early-Semantics.Bang*)
**qed**
**qed**

**have** $\exists\, c{::}name.\ c \sharp (P, P')$ **by**(*blast intro*: *name-exists-fresh*)
**then obtain** *c::name* **where** *cFreshP*: $c \sharp P$ **and** *cFreshP'*: $c \sharp P'$
**by**(*force simp add*: *fresh-prod*)
**from** *assms cFreshP'* **have** $P \longmapsto_l a{<}c{>} \prec_l ([(x, c)] \cdot P')$
**by**(*simp add*: *Late-Semantics.alphaBoundResidual*)
**hence** $P \longmapsto_e a{<}u{>} \prec_e ([(x, c)] \cdot P')[c{::=}u]$ **using** *cFreshP*
**by**(*rule Goal*)
**with** *cFreshP'* **show** *?thesis* **by**(*simp add*: *renaming name-swap*)
**qed**

**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** −
**{**
**fix** $P\ a\ u\ P'$
**assume** $P \longmapsto_e a{<}u{>} \prec_e P'$
**hence** $\exists P''\ x.\ P \longmapsto_l a{<}x{>} \prec_l P'' \wedge P' = P''[x{::=}u]$
**proof**(*nominal-induct rule*: *Early-Semantics.inputInduct*)
**case**(*cInput a x P u*)
**have** $a{<}x{>}.P \longmapsto_l a{<}x{>} \prec P$ **by**(*rule Late-Semantics.Input*)
**thus** *?case* **by** *blast*
**next**
**case**(*cMatch P a u P' b*)
**have** $\exists P''\ x.\ P \longmapsto_l a{<}x{>} \prec P'' \wedge P' = P''[x{::=}u]$ **by** *fact*
**then obtain** $P''\ x$ **where** *PTrans*: $P \longmapsto_l a{<}x{>} \prec P''$ **and** *P'eqP''*: $P' = P''[x{::=}u]$ **by** *blast*
**from** *PTrans* **have** $[b \frown b]P \longmapsto_l a{<}x{>} \prec P''$ **by**(*rule Late-Semantics.Match*)
**with** *P'eqP''* **show** *?case* **by** *blast*

**next**
**case**(*cMismatch* $P\ a\ u\ P'\ b\ c$)
**have** $\exists P''\ x.\ P \longmapsto_l a{<}x{>} \prec P'' \wedge P' = P''[x::=u]$ **by** *fact*
**then obtain** $P''\ x$ **where** *PTrans*: $P \longmapsto_l a{<}x{>} \prec P''$ **and** $P'eqP''$: $P' = P''[x::=u]$ **by** *blast*
**have** $b \neq c$ **by** *fact*
**with** *PTrans* **have** $[b{\neq}c]P \longmapsto_l a{<}x{>} \prec P''$ **by**(*rule Late-Semantics.Mismatch*)
**with** $P'eqP''$ **show** *?case* **by** *blast*
**next**
**case**(*cSum1* $P\ a\ u\ P'\ Q$)
**have** $\exists P''\ x.\ P \longmapsto_l a{<}x{>} \prec P'' \wedge P' = P''[x::=u]$ **by** *fact*
**then obtain** $P''\ x$ **where** *PTrans*: $P \longmapsto_l a{<}x{>} \prec P''$ **and** $P'eqP''$: $P' = P''[x::=u]$ **by** *blast*
**from** *PTrans* **have** $P \oplus Q \longmapsto_l a{<}x{>} \prec P''$ **by**(*rule Late-Semantics.Sum1*)
**with** $P'eqP''$ **show** *?case* **by** *blast*
**next**
**case**(*cSum2* $Q\ a\ u\ Q'\ P$)
**have** $\exists Q''\ x.\ Q \longmapsto_l a{<}x{>} \prec Q'' \wedge Q' = Q''[x::=u]$ **by** *fact*
**then obtain** $Q''\ x$ **where** *QTrans*: $Q \longmapsto_l a{<}x{>} \prec Q''$ **and** $Q'eqQ''$: $Q' = Q''[x::=u]$ **by** *blast*
**from** *QTrans* **have** $P \oplus Q \longmapsto_l a{<}x{>} \prec Q''$ **by**(*rule Late-Semantics.Sum2*)
**with** $Q'eqQ''$ **show** *?case* **by** *blast*
**next**
**case**(*cPar1* $P\ a\ u\ P'\ Q$)
**have** $\exists P''\ x.\ P \longmapsto_l a{<}x{>} \prec P'' \wedge P' = P''[x::=u]$ **by** *fact*
**then obtain** $P''\ x$ **where** *PTrans*: $P \longmapsto_l a{<}x{>} \prec P''$ **and** $P'eqP''$: $P' = P''[x::=u]$ **by** *blast*
**have** $\exists c{::}name.\ c \sharp (Q,\ P'')$ **by**(*blast intro*: *name-exists-fresh*)
**then obtain** $c{::}name$ **where** *cFreshQ*: $c \sharp Q$ **and** $cFreshP''$: $c \sharp P''$ **by**(*force simp add*: *fresh-prod*)
**from** *PTrans* $cFreshP''$ **have** $P \longmapsto_l a{<}c{>} \prec [(x,\ c)] \cdot P''$ **by**(*simp add*: *Late-Semantics.alphaBoundResidual*)
**hence** $P \parallel Q \longmapsto_l a{<}c{>} \prec ([(x,\ c)] \cdot P'') \parallel Q$ **using** $\langle c \sharp Q\rangle$ **by**(*rule Late-Semantics.Par1B*)
**moreover from** *cFreshQ* $cFreshP''$ $P'eqP''$ **have** $P' \parallel Q = ((([(x,\ c)] \cdot P'') \parallel Q)[c::=u]$
**by**(*simp add*: *forget renaming name-swap*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*cPar2* $Q\ a\ u\ Q'\ P$)
**have** $\exists Q''\ x.\ Q \longmapsto_l a{<}x{>} \prec Q'' \wedge Q' = Q''[x::=u]$ **by** *fact*
**then obtain** $Q''\ x$ **where** *QTrans*: $Q \longmapsto_l a{<}x{>} \prec Q''$ **and** $Q'eqQ''$: $Q' = Q''[x::=u]$ **by** *blast*
**have** $\exists c{::}name.\ c \sharp (P,\ Q'')$ **by**(*blast intro*: *name-exists-fresh*)
**then obtain** $c{::}name$ **where** *cFreshP*: $c \sharp P$ **and** $cFreshQ''$: $c \sharp Q''$ **by**(*force simp add*: *fresh-prod*)
**from** *QTrans* $cFreshQ''$ **have** $Q \longmapsto_l a{<}c{>} \prec [(x,\ c)] \cdot Q''$ **by**(*simp add*: *Late-Semantics.alphaBoundResidual*)
**hence** $P \parallel Q \longmapsto_l a{<}c{>} \prec P \parallel ([(x,\ c)] \cdot Q'')$ **using** $\langle c \sharp P\rangle$ **by**(*rule*

*Late-Semantics.Par2B*)
**moreover from** *cFreshP cFreshQ''* $Q'eqQ''$ **have** $P \parallel Q' = (P \parallel ([(x, c)] \cdot Q''))[c{::=}u]$
**by**(*simp add*: *forget renaming name-swap*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*cRes P a u P' y*)
**have** $\exists P'' x.\ P \longmapsto_l a<x> \prec P'' \wedge P' = P''[x{::=}u]$ **by** *fact*
**then obtain** $P''$ $x$ **where** *PTrans*: $P \longmapsto_l a<x> \prec P''$ **and** $P'eqP''$: $P' = P''[x{::=}u]$ **by** *blast*
**have** *yinequ*: $y \neq u$ **by** *fact*
**have** $\exists c{::}name.\ c \sharp (y, P'')$ **by**(*blast intro*: *name-exists-fresh*)
**then obtain** $c{::}name$ **where** *cineqy*: $c \neq y$ **and** $cFreshP''$: $c \sharp P''$ **by**(*force simp add*: *fresh-prod*)
**from** *PTrans* $cFreshP''$ **have** $P \longmapsto_l a<c> \prec [(x, c)] \cdot P''$ **by**(*simp add*: *Late-Semantics.alphaBoundResidual*)
**moreover have** $y \neq a$ **by** *fact*
**ultimately have** $<\nu y>P \longmapsto_l a<c> \prec <\nu y>(([(x, c)] \cdot P''))$ **using** *cineqy*
**by**(*force intro*: *Late-Semantics.ResB*)
**moreover from** *cineqy* $cFreshP''$ $P'eqP''$ *yinequ* **have** $<\nu y>P' = (<\nu y>([(x, c)] \cdot P''))[c{::=}u]$
**by**(*simp add*: *renaming name-swap*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*cBang P a u P'*)
**have** $\exists P'' x.\ P \parallel !P \longmapsto_l a<x> \prec P'' \wedge P' = P''[x{::=}u]$ **by** *fact*
**then obtain** $P''$ $x$ **where** *PTrans*: $P \parallel !P \longmapsto_l a<x> \prec P''$ **and** $P'eqP''$: $P' = P''[x{::=}u]$ **by** *blast*
**from** *PTrans* **have** $!P \longmapsto_l a<x> \prec P''$ **by**(*rule Late-Semantics.Bang*)
**with** $P'eqP''$ **show** *?case* **by** *blast*
**qed**
**}**
**with** *assms* **obtain** $P''$ $y$ **where** *PTrans*: $P \longmapsto_l a<y> \prec P''$ **and** $P'eqP''$: $P' = P''[y{::=}u]$ **by** *blast*
**show** *?thesis*
**proof**(*cases x=y*)
**case** *True*
**from** *PTrans* $P'eqP''$ $\langle x = y \rangle$ **show** *?thesis* **by** *blast*
**next**
**case** *False*
**from** *PTrans* $\langle x \neq y \rangle$ $\langle x \sharp P \rangle$ **have** $x \sharp P''$ **by**(*fastforce dest*: *freshBoundDerivative simp add*: *residual.inject*)
**with** *PTrans* **have** $P \longmapsto_l a<x> \prec_l ([(x, y)] \cdot P'')$
**by**(*simp add*: *Late-Semantics.alphaBoundResidual*)
**moreover from** $\langle x \sharp P'' \rangle$ **have** $P''[y{::=}u] = ([(x, y)] \cdot P'')[x{::=}u]$ **by**(*simp add*: *renaming name-swap*)
**ultimately show** *?thesis* **using** $P'eqP''$ **by** *blast*
**qed**
**qed**

**lemma** *lateEarlyTau*:
**fixes** $P$ :: *pi*
**and** $P'$ :: *pi*

**assumes** $P \longmapsto_l \tau \prec_l P'$

**shows** $P \longmapsto_e \tau \prec_e P'$
**using** *assms*
**proof**(*nominal-induct rule*: *Late-Semantics.tauInduct*)
**case**(*Tau P*)
**thus** *?case* **by**(*rule Early-Semantics.Tau*)
**next**
**case**(*Match P P' a*)
**have** $P \longmapsto_e \tau \prec_e P'$ **by** *fact*
**thus** $[a \frown a]P \longmapsto_e \tau \prec_e P'$ **by**(*rule Early-Semantics.Match*)
**next**
**case**(*Mismatch P P' a b*)
**have** $P \longmapsto_e \tau \prec_e P'$ **by** *fact*
**moreover have** $a \neq b$ **by** *fact*
**ultimately show** $[a \neq b]P \longmapsto_e \tau \prec_e P'$ **by**(*rule Early-Semantics.Mismatch*)
**next**
**case**(*Sum1 P P' Q*)
**have** $P \longmapsto_e \tau \prec_e P'$ **by** *fact*
**thus** $P \oplus Q \longmapsto_e \tau \prec_e P'$ **by**(*rule Early-Semantics.Sum1*)
**next**
**case**(*Sum2 Q Q' P*)
**have** $Q \longmapsto_e \tau \prec_e Q'$ **by** *fact*
**thus** $P \oplus Q \longmapsto_e \tau \prec_e Q'$ **by**(*rule Early-Semantics.Sum2*)
**next**
**case**(*Par1 P P' Q*)
**have** $P \longmapsto_e \tau \prec_e P'$ **by** *fact*
**thus** $P \parallel Q \longmapsto_e \tau \prec_e P' \parallel Q$ **by**(*rule Early-Semantics.Par1F*)
**next**
**case**(*Par2 Q Q' P*)
**have** $Q \longmapsto_e \tau \prec_e Q'$ **by** *fact*
**thus** $P \parallel Q \longmapsto_e \tau \prec_e P \parallel Q'$ **by**(*rule Early-Semantics.Par2F*)
**next**
**case**(*Comm1 P a x P' Q b Q'*)
**have** $P \longmapsto_e a{<}b{>} \prec_e P'[x{::=}b]$
**proof** $-$
**have** $P \longmapsto_l a{<}x{>} \prec P'$ **by** *fact*
**thus** *?thesis* **by**(*rule lateEarlyInput*)
**qed**
**moreover have** $Q \longmapsto_e a[b] \prec_e Q'$
**proof** $-$
**have** $Q \longmapsto_l a[b] \prec_l Q'$ **by** *fact*
**thus** *?thesis* **by**(*rule lateEarlyOutput*)
**qed**

**ultimately show** *?case* **by**(*rule Early-Semantics.Comm1*)
**next**
**case**(*Comm2 P a b P′ Q x Q′*)
**have** $P \longmapsto_e a[b] \prec_e P'$
**proof** −
**have** $P \longmapsto_l a[b] \prec_l P'$ **by** *fact*
**thus** *?thesis* **by**(*rule lateEarlyOutput*)
**qed**
**moreover have** $Q \longmapsto_e a<b> \prec_e Q'[x::=b]$
**proof** −
**have** $Q \longmapsto_l a<x> \prec_l Q'$ **by** *fact*
**thus** *?thesis* **by**(*rule lateEarlyInput*)
**qed**
**ultimately show** *?case* **by**(*rule Early-Semantics.Comm2*)
**next**
**case**(*Close1 P a x P′ Q y Q′*)
**have** $P \longmapsto_e a<y> \prec_e P'[x::=y]$
**proof** −
**have** $P \longmapsto_l a<x> \prec P'$ **by** *fact*
**thus** *?thesis* **by**(*rule lateEarlyInput*)
**qed**
**moreover have** $Q \longmapsto_e a<\nu y> \prec Q'$
**proof** −
**have** $Q \longmapsto_l a<\nu y> \prec_l Q'$ **by** *fact*
**thus** *?thesis* **by**(*rule lateEarlyBoundOutput*)
**qed**
**moreover have** $y \sharp P$ **by** *fact*
**ultimately show** *?case* **by**(*rule Early-Semantics.Close1*)
**next**
**case**(*Close2 P a y P′ Q x Q′*)
**have** $P \longmapsto_e a<\nu y> \prec P'$
**proof** −
**have** $P \longmapsto_l a<\nu y> \prec_l P'$ **by** *fact*
**thus** *?thesis* **by**(*rule lateEarlyBoundOutput*)
**qed**
**moreover have** $Q \longmapsto_e a<y> \prec_e Q'[x::=y]$
**proof** −
**have** $Q \longmapsto_l a<x> \prec_l Q'$ **by** *fact*
**thus** *?thesis* **by**(*rule lateEarlyInput*)
**qed**
**moreover have** $y \sharp Q$ **by** *fact*
**ultimately show** *?case* **by**(*rule Early-Semantics.Close2*)
**next**
**case**(*Res P P′ x*)
**have** $P \longmapsto_e \tau \prec_e P'$ **by** *fact*
**thus** *?case* **by**(*force intro*: *Early-Semantics.ResF*)
**next**
**case**(*Bang P P′*)
**have** $P \parallel !P \longmapsto_e \tau \prec_e P'$ **by** *fact*

**thus** *?case* **by**(*rule Early-Semantics.Bang*)
**qed**

**lemma** *earlyLateTau*:
**fixes** $P$ :: *pi*
**and** $P'$ :: *pi*

**assumes** $P \longmapsto_e \tau \prec_e P'$

**shows** $P \longmapsto_l \tau \prec_l P'$
**using** *assms*
**proof**(*nominal-induct rule*: *Early-Semantics.tauInduct*)
**case**(*Tau P*)
**thus** *?case* **by**(*rule Late-Semantics.Tau*)
**next**
**case**(*Match P P' a*)
**have** $P \longmapsto_l \tau \prec_l P'$ **by** *fact*
**thus** *?case* **by**(*rule Late-Semantics.Match*)
**next**
**case**(*Mismatch P P' a b*)
**have** $P \longmapsto_l \tau \prec_l P'$ **by** *fact*
**moreover have** $a \neq b$ **by** *fact*
**ultimately show** *?case* **by**(*rule Late-Semantics.Mismatch*)
**next**
**case**(*Sum1 P P' Q*)
**have** $P \longmapsto_l \tau \prec_l P'$ **by** *fact*
**thus** *?case* **by**(*rule Late-Semantics.Sum1*)
**next**
**case**(*Sum2 Q Q' P*)
**have** $Q \longmapsto_l \tau \prec_l Q'$ **by** *fact*
**thus** *?case* **by**(*rule Late-Semantics.Sum2*)
**next**
**case**(*Par1 P P' Q*)
**have** $P \longmapsto_l \tau \prec_l P'$ **by** *fact*
**thus** *?case* **by**(*rule Late-Semantics.Par1F*)
**next**
**case**(*Par2 Q Q' P*)
**have** $Q \longmapsto_l \tau \prec_l Q'$ **by** *fact*
**thus** *?case* **by**(*rule Late-Semantics.Par2F*)
**next**
**case**(*Comm1 P a b P' Q Q'*)
**have** $P \longmapsto_e a{<}b{>} \prec_e P'$ **by** *fact*
**moreover obtain** $x$::*name* **where** $x \sharp P$ **by**(*generate-fresh name*) *auto*
**ultimately obtain** $P''$ **where** *PTrans*: $P \longmapsto_l a{<}x{>} \prec P''$ **and** $P'eqP''$: $P' = P''[x{::=}b]$
**by**(*blast dest*: *earlyLateInput*)
**have** $Q \longmapsto_e a[b] \prec_e Q'$ **by** *fact*
**hence** $Q \longmapsto_l a[b] \prec_l Q'$ **by**(*rule earlyLateOutput*)
**with** *PTrans* $P'eqP''$ **show** *?case*

**by**(*blast intro*: *Late-Semantics.Comm1*)
**next**
**case**(*Comm2 P a b P' Q Q'*)
**have** $P \longmapsto_e a[b] \prec_e P'$ **by** *fact*
**hence** *QTrans*: $P \longmapsto_l a[b] \prec_l P'$ **by**(*rule earlyLateOutput*)
**have** $Q \longmapsto_e a{<}b{>} \prec_e Q'$ **by** *fact*
**moreover obtain** $x$::*name* **where** $x \sharp Q$ **by**(*generate-fresh name*) *auto*
**ultimately obtain** $Q''$ $x$ **where** $Q \longmapsto_l a{<}x{>} \prec Q''$ **and** $Q' = Q''[x{::=}b]$
**by**(*blast dest*: *earlyLateInput*)
**with** *QTrans* **show** *?case*
**by**(*blast intro*: *Late-Semantics.Comm2*)
**next**
**case**(*Close1 P a x P' Q Q'*)
**have** $P \longmapsto_e a{<}x{>} \prec_e P'$ **and** $x \sharp P$ **by** *fact+*
**then obtain** $P''$ **where** $P \longmapsto_l a{<}x{>} \prec P''$ **and** $P' = P''[x{::=}x]$
**by**(*blast dest*: *earlyLateInput*)

**moreover have** $Q \longmapsto_e a{<}\nu x{>} \prec_e Q'$ **by** *fact*
**hence** $Q \longmapsto_l a{<}\nu x{>} \prec_l Q'$ **by**(*rule earlyLateBoundOutput*)
**moreover have** $x \sharp P$ **by** *fact*
**ultimately show** *?case*
**by**(*blast intro*: *Late-Semantics.Close1*)
**next**
**case**(*Close2 P a x P' Q Q'*)
**have** $P \longmapsto_e a{<}\nu x{>} \prec_e P'$ **by** *fact*
**hence** *PTrans*: $P \longmapsto_l a{<}\nu x{>} \prec_l P'$ **by**(*rule earlyLateBoundOutput*)

**have** $Q \longmapsto_e a{<}x{>} \prec_e Q'$ **and** $x \sharp Q$ **by** *fact+*
**then obtain** $Q''$ $y$ **where** $Q \longmapsto_l a{<}x{>} \prec Q''$ **and** $Q' = Q''[x{::=}x]$
**by**(*blast dest*: *earlyLateInput*)
**moreover have** $x \sharp Q$ **by** *fact*
**ultimately show** *?case* **using** *PTrans*
**by**(*blast intro*: *Late-Semantics.Close2*)
**next**
**case**(*Res P P' x*)
**have** $P \longmapsto_l \tau \prec_l P'$ **by** *fact*
**thus** *?case* **by**(*force intro*: *Late-Semantics.ResF*)
**next**
**case**(*Bang P P'*)
**have** $P \parallel !P \longmapsto_l \tau \prec_l P'$ **by** *fact*
**thus** *?case* **by**(*force intro*: *Late-Semantics.Bang*)
**qed**

**lemma** *tauEq*:
**fixes** $P$ :: *pi*
**and** $P'$ :: *pi*

**shows** $P \longmapsto_e(\textit{Early-Semantics.FreeR Early-Semantics.TauR } P') = P \longmapsto \tau \prec_l P'$

**by**(*auto intro*: *earlyLateTau lateEarlyTau*)

**abbreviation** *simLate-judge* (‹- $\rightsquigarrow_l$[-] -› [*80*, *80*, *80*] *80*) **where** $P \rightsquigarrow_l[Rel]\ Q \equiv$ *Strong-Late-Sim.simulation P Rel Q*
**abbreviation** *simEarly-judge* (‹- $\rightsquigarrow_e$[-] -› [*80*, *80*, *80*] *80*) **where** $P \rightsquigarrow_e[Rel]\ Q$ $\equiv$ *Strong-Early-Sim.strongSimEarly P Rel Q*

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

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

**shows** $P \rightsquigarrow_e[Rel]\ Q$
**proof**(*induct rule*: *Strong-Early-Sim.simCases*)
**case**(*Bound a x Q'*)
**have** $Q \longmapsto_e a{<}\nu x{>} \prec_e Q'$ **by** *fact*
**hence** $Q \longmapsto_l a{<}\nu x{>} \prec_l Q'$ **by**(*rule earlyLateBoundOutput*)
**moreover have** $x \sharp P$ **by** *fact*
**ultimately obtain** $P'$ **where** *PTrans*: $P \longmapsto_l a{<}\nu x{>} \prec_l P'$ **and** *P'RelQ'*: $(P', Q') \in Rel$ **using** *PSimQ*
**by**(*force dest*: *Strong-Late-Sim.simE simp add*: *derivative-def*)
**from** *PTrans* **have** $P \longmapsto_e a{<}\nu x{>} \prec_e P'$ **by**(*rule lateEarlyBoundOutput*)
**with** *P'RelQ'* **show** *?case* **by** *blast*
**next**
**case**(*Free* $\alpha$ *Q'*)
**have** $Q \longmapsto_e$ *Early-Semantics.residual.FreeR* $\alpha$ $Q'$ **by** *fact*
**thus** *?case*
**proof**(*nominal-induct* $\alpha$ *rule*: *freeRes.strong-induct*)
**case**(*InputR a u*)
**obtain** *x*::*name* **where** $x \sharp Q$ **and** $x \sharp P$ **by**(*generate-fresh name*) *auto*
**with** ‹$Q \longmapsto_e a{<}u{>} \prec_e Q'$› **obtain** $Q''$ **where** *QTrans*: $Q \longmapsto_l a{<}x{>} \prec_l Q''$ **and** *Q'eqQ''*: $Q' = Q''[x{::=}u]$
**by**(*blast dest*: *earlyLateInput*)
**from** *PSimQ QTrans* ‹$x \sharp P$› **obtain** $P'$ **where** *PTrans*: $P \longmapsto_l a{<}x{>} \prec P'$
**and** *P'RelQ'*: $(P'[x{::=}u], Q''[x{::=}u]) \in Rel$
**by**(*force dest*: *Strong-Late-Sim.simE simp add*: *derivative-def*)
**from** *PTrans* **have** $P \longmapsto_e a{<}u{>} \prec_e P'[x{::=}u]$ **by**(*rule lateEarlyInput*)
**with** *P'RelQ' Q'eqQ''* **show** $\exists P'.\ P \longmapsto_e a{<}u{>} \prec_e P' \wedge (P', Q') \in Rel$ **by** *blast*
**next**
**case**(*OutputR a b*)
**from** ‹$Q \longmapsto_e a[b] \prec_e Q'$› **have** $Q \longmapsto_l a[b] \prec_l Q'$ **by**(*rule earlyLateOutput*)
**with** *PSimQ* **obtain** $P'$ **where** *PTrans*: $P \longmapsto_l a[b] \prec_l P'$ **and** *P'RelQ'*: $(P', Q') \in Rel$
**by**(*blast dest*: *Strong-Late-Sim.simE*)

**from** *PTrans* **have** $P \longmapsto_e a[b] \prec_e P'$ **by**(*rule lateEarlyOutput*)
**with** *P'RelQ'* **show** $\exists P'. P \longmapsto_e a[b] \prec_e P' \wedge (P', Q') \in Rel$ **by** *blast*
**next**
**case** *TauR*
**from** ‹$Q \longmapsto_e \tau \prec_e Q'$› **have** $Q \longmapsto_l \tau \prec_l Q'$ **by**(*rule earlyLateTau*)
**with** *PSimQ* **obtain** $P'$ **where** *PTrans*: $P \longmapsto_l \tau \prec_l P'$ **and** *P'RelQ'*: $(P', Q') \in Rel$
**by**(*blast dest*: *Strong-Late-Sim.simE*)
**from** *PTrans* **have** $P \longmapsto_e \tau \prec_e P'$ **by**(*rule lateEarlyTau*)
**with** *P'RelQ'* **show** $\exists P'. P \longmapsto_e \tau \prec_e P' \wedge (P', Q') \in Rel$ **by** *blast*
**qed**
**qed**

**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** $P \sim_e Q$
**using** *assms*
**by**(*coinduct rule*: *Strong-Early-Bisim.weak-coinduct*)
(*auto dest*: *Strong-Late-Bisim.bisimE Strong-Late-Bisim.symmetric intro*: *lateEarlySim*)

**abbreviation** *congLate-judge* (‹- $\sim^s{}_l$ -› [*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** $P \sim^s{}_e Q$
**using** *assms*
**by**(*auto simp add*: *substClosed-def intro*: *lateEarlyBisim*)

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

  **assumes** $P \equiv_s Q$

  **shows** $P \sim^s{}_e Q$
**using** *assms lateEarlyCong bisimSubstStructCong*
**by** *blast*

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

  **assumes** $P \equiv_s Q$

  **shows** $P \sim_e Q$
**using** *assms lateEarlyBisim structCongBisim*
**by** *blast*

**end**

**theory** *Strong-Early-Bisim-SC*
  **imports** *Strong-Early-Bisim Strong-Late-Bisim-SC Strong-Early-Late-Comp*
**begin**

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

  **shows** $<\nu a><\nu b>P \sim_e <\nu b><\nu a>P$
**proof** −
  **have** $<\nu a><\nu b>P \sim_l <\nu b><\nu a>P$ **by**(*rule Strong-Late-Bisim-SC.resComm*)
  **thus** *?thesis* **by**(*rule lateEarlyBisim*)
**qed**

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

  **shows** $[a \frown a]P \sim_e P$
**proof** −
  **have** $[a \frown a]P \sim_l P$ **by**(*rule Strong-Late-Bisim-SC.matchId*)

**thus** *?thesis* **by**(*rule lateEarlyBisim*)
**qed**

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

**assumes** $a \neq b$

**shows** $[a{\neq}b]P \sim_e P$
**proof** −
**from** *assms* **have** $[a{\neq}b]P \sim_l P$ **by**(*rule Strong-Late-Bisim-SC.mismatchId*)
**thus** *?thesis* **by**(*rule lateEarlyBisim*)
**qed**

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

**shows** $[a{\neq}a]P \sim_e \mathbf{0}$
**proof** −
**have** $[a{\neq}a]P \sim_l \mathbf{0}$ **by**(*rule Strong-Late-Bisim-SC.mismatchNil*)
**thus** *?thesis* **by**(*rule lateEarlyBisim*)
**qed**

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

**shows** $P \oplus Q \sim_e Q \oplus P$
**proof** −
**have** $P \oplus Q \sim_l Q \oplus P$ **by**(*rule Strong-Late-Bisim-SC.sumSym*)
**thus** *?thesis* **by**(*rule lateEarlyBisim*)
**qed**

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

**shows** $(P \oplus Q) \oplus R \sim_e P \oplus (Q \oplus R)$
**proof** −
**have** $(P \oplus Q) \oplus R \sim_l P \oplus (Q \oplus R)$ **by**(*rule Strong-Late-Bisim-SC.sumAssoc*)
**thus** *?thesis* **by**(*rule lateEarlyBisim*)

**qed**

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

  **shows** $P \oplus \mathbf{0} \sim_e P$
**proof** −
  **have** $P \oplus \mathbf{0} \sim_l P$ **by**(*rule Strong-Late-Bisim-SC.sumZero*)
  **thus** *?thesis* **by**(*rule lateEarlyBisim*)
**qed**

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

  **shows** $P \parallel \mathbf{0} \sim_e P$
**proof** −
  **have** $P \parallel \mathbf{0} \sim_l P$ **by**(*rule Strong-Late-Bisim-SC.parZero*)
  **thus** *?thesis* **by**(*rule lateEarlyBisim*)
**qed**

**lemma** *parSym*:
  **fixes** $P$ :: *pi*
  **and** $Q$ :: *pi*

  **shows** $P \parallel Q \sim_e Q \parallel P$
**proof** −
  **have** $P \parallel Q \sim_l Q \parallel P$ **by**(*rule Strong-Late-Bisim-SC.parSym*)
  **thus** *?thesis* **by**(*rule lateEarlyBisim*)
**qed**

**lemma** *scopeExtPar*:
  **fixes** $P$ :: *pi*
  **and** $Q$ :: *pi*
  **and** $x$ :: *name*

  **assumes** $x \sharp P$

  **shows** $<\nu x>(P \parallel Q) \sim_e P \parallel <\nu x>Q$
**proof** −
  **from** *assms* **have** $<\nu x>(P \parallel Q) \sim_l P \parallel <\nu x>Q$ **by**(*rule Strong-Late-Bisim-SC.scopeExtPar*)
  **thus** *?thesis* **by**(*rule lateEarlyBisim*)
**qed**

**lemma** *scopeExtPar'*:
  **fixes** $P$ :: *pi*
  **and** $Q$ :: *pi*
  **and** $x$ :: *name*

**assumes** *xFreshQ*: $x \sharp Q$

**shows** $<\nu x>(P \parallel Q) \sim_e (<\nu x>P) \parallel Q$
**proof** −
**from** *assms* **have** $<\nu x>(P \parallel Q) \sim_l (<\nu x>P) \parallel Q$ **by**(*rule Strong-Late-Bisim-SC.scopeExtPar'*)
**thus** *?thesis* **by**(*rule lateEarlyBisim*)
**qed**

**lemma** *parAssoc*:
**fixes** $P :: pi$
**and** $Q :: pi$
**and** $R :: pi$

**shows** $(P \parallel Q) \parallel R \sim_e P \parallel (Q \parallel R)$
**proof** −
**have** $(P \parallel Q) \parallel R \sim_l P \parallel (Q \parallel R)$ **by**(*rule Strong-Late-Bisim-SC.parAssoc*)
**thus** *?thesis* **by**(*rule lateEarlyBisim*)
**qed**

**lemma** *freshRes*:
**fixes** $P :: pi$
**and** $a :: name$

**assumes** *aFreshP*: $a \sharp P$

**shows** $<\nu a>P \sim_e P$
**proof** −
**from** *aFreshP* **have** $<\nu a>P \sim_l P$ **by**(*rule Strong-Late-Bisim-SC.scopeFresh*)
**thus** *?thesis* **by**(*rule lateEarlyBisim*)
**qed**

**lemma** *scopeExtSum*:
**fixes** $P :: pi$
**and** $Q :: pi$
**and** $x :: name$

**assumes** $x \sharp P$

**shows** $<\nu x>(P \oplus Q) \sim_e P \oplus <\nu x>Q$
**proof** −
**from** ‹$x \sharp P$› **have** $<\nu x>(P \oplus Q) \sim_l P \oplus <\nu x>Q$ **by**(*rule Strong-Late-Bisim-SC.scopeExtSum*)
**thus** *?thesis* **by**(*rule lateEarlyBisim*)
**qed**

**lemma** *bangSC*:
**fixes** $P$

**shows** $!P \sim_e P \parallel !P$

**proof** −
  **have** $!P \sim_l P \parallel !P$ **by**(*rule Strong-Late-Bisim-SC.bangSC*)
  **thus** *?thesis* **by**(*rule lateEarlyBisim*)
**qed**

**end**

**theory** *Weak-Early-Bisim-SC*
  **imports** *Weak-Early-Bisim Strong-Early-Bisim-SC*
**begin**

**lemma** *weakBisimStructCong*:
  **fixes** $P :: pi$
  **and** $Q :: pi$

  **assumes** $P \equiv_s Q$

  **shows** $P \approx Q$
**using** *assms*
**by**(*metis earlyBisimStructCong strongBisimWeakBisim*)

**lemma** *matchId*:
  **fixes** $a :: name$
  **and** $P :: pi$

  **shows** $[a\frown a]P \approx P$
**proof** −
  **have** $[a\frown a]P \sim_e P$ **by**(*rule Strong-Early-Bisim-SC.matchId*)
  **thus** *?thesis* **by**(*rule strongBisimWeakBisim*)
**qed**

**lemma** *mismatchId*:
  **fixes** $a :: name$
  **and** $b :: name$
  **and** $P :: pi$

  **assumes** $a \neq b$

  **shows** $[a\neq b]P \approx P$
**proof** −
  **from** ‹$a \neq b$› **have** $[a\neq b]P \sim_e P$ **by**(*rule Strong-Early-Bisim-SC.mismatchId*)
  **thus** *?thesis* **by**(*rule strongBisimWeakBisim*)
**qed**

**lemma** *mismatchNil*:
  **fixes** $a :: name$
  **and** $P :: pi$

**shows** $[a{\neq}a]P \approx \mathbf{0}$
**proof** $-$
**have** $[a{\neq}a]P \sim_e \mathbf{0}$ **by**(*rule Strong-Early-Bisim-SC.mismatchNil*)
**thus** *?thesis* **by**(*rule strongBisimWeakBisim*)
**qed**

**lemma** *resComm*:
**fixes** $P$ :: *pi*

**shows** $<\nu a><\nu b>P \approx <\nu b><\nu a>P$
**proof** $-$
**have** $<\nu a><\nu b>P \sim_e <\nu b><\nu a>P$ **by**(*rule Strong-Early-Bisim-SC.resComm*)
**thus** *?thesis* **by**(*rule strongBisimWeakBisim*)
**qed**

**lemma** *sumSym*:
**fixes** $P$ :: *pi*
**and** $Q$ :: *pi*

**shows** $P \oplus Q \approx Q \oplus P$
**proof** $-$
**have** $P \oplus Q \sim_e Q \oplus P$ **by**(*rule Strong-Early-Bisim-SC.sumSym*)
**thus** *?thesis* **by**(*rule strongBisimWeakBisim*)
**qed**

**lemma** *sumAssoc*:
**fixes** $P$ :: *pi*
**and** $Q$ :: *pi*
**and** $R$ :: *pi*

**shows** $(P \oplus Q) \oplus R \approx P \oplus (Q \oplus R)$
**proof** $-$
**have** $(P \oplus Q) \oplus R \sim_e P \oplus (Q \oplus R)$ **by**(*rule Strong-Early-Bisim-SC.sumAssoc*)
**thus** *?thesis* **by**(*rule strongBisimWeakBisim*)
**qed**

**lemma** *sumZero*:
**fixes** $P$ :: *pi*

**shows** $P \oplus \mathbf{0} \approx P$
**proof** $-$
**have** $P \oplus \mathbf{0} \sim_e P$ **by**(*rule Strong-Early-Bisim-SC.sumZero*)
**thus** *?thesis* **by**(*rule strongBisimWeakBisim*)
**qed**

**lemma** *parZero*:
  **fixes** $P$ :: *pi*

  **shows** $P \parallel \mathbf{0} \approx P$
**proof** −
  **have** $P \parallel \mathbf{0} \sim_e P$ **by**(*rule Strong-Early-Bisim-SC.parZero*)
  **thus** *?thesis* **by**(*rule strongBisimWeakBisim*)
**qed**

**lemma** *parSym*:
  **fixes** $P$ :: *pi*
  **and** $Q$ :: *pi*

  **shows** $P \parallel Q \approx Q \parallel P$
**proof** −
  **have** $P \parallel Q \sim_e Q \parallel P$ **by**(*rule Strong-Early-Bisim-SC.parSym*)
  **thus** *?thesis* **by**(*rule strongBisimWeakBisim*)
**qed**

**lemma** *scopeExtPar*:
  **fixes** $P$ :: *pi*
  **and** $Q$ :: *pi*
  **and** $x$ :: *name*

  **assumes** $x \sharp P$

  **shows** $<\nu x>(P \parallel Q) \approx P \parallel <\nu x>Q$
**proof** −
  **from** ‹$x \sharp P$› **have** $<\nu x>(P \parallel Q) \sim_e P \parallel <\nu x>Q$ **by**(*rule Strong-Early-Bisim-SC.scopeExtPar*)
  **thus** *?thesis* **by**(*rule strongBisimWeakBisim*)
**qed**

**lemma** *scopeExtPar'*:
  **fixes** $P$ :: *pi*
  **and** $Q$ :: *pi*
  **and** $x$ :: *name*

  **assumes** $x \sharp Q$

  **shows** $<\nu x>(P \parallel Q) \approx (<\nu x>P) \parallel Q$
**proof** −
  **from** ‹$x \sharp Q$› **have** $<\nu x>(P \parallel Q) \sim_e (<\nu x>P) \parallel Q$ **by**(*rule Strong-Early-Bisim-SC.scopeExtPar'*)
  **thus** *?thesis* **by**(*rule strongBisimWeakBisim*)
**qed**

**lemma** *parAssoc*:

**fixes** $P :: pi$
**and** $Q :: pi$
**and** $R :: pi$

**shows** $(P \parallel Q) \parallel R \approx P \parallel (Q \parallel R)$
**proof** −
**have** $(P \parallel Q) \parallel R \sim_e P \parallel (Q \parallel R)$ **by**(*rule Strong-Early-Bisim-SC.parAssoc*)
**thus** *?thesis* **by**(*rule strongBisimWeakBisim*)
**qed**

**lemma** *freshRes*:
**fixes** $P :: pi$
**and** $a :: name$

**assumes** $a \sharp P$

**shows** $<\nu a>P \approx P$
**proof** −
**from** ‹$a \sharp P$› **have** $<\nu a>P \sim_e P$ **by**(*rule Strong-Early-Bisim-SC.freshRes*)
**thus** *?thesis* **by**(*rule strongBisimWeakBisim*)
**qed**

**lemma** *scopeExtSum*:
**fixes** $P :: pi$
**and** $Q :: pi$
**and** $x :: name$

**assumes** $x \sharp P$

**shows** $<\nu x>(P \oplus Q) \approx P \oplus <\nu x>Q$
**proof** −
**from** ‹$x \sharp P$› **have** $<\nu x>(P \oplus Q) \sim_e P \oplus <\nu x>Q$ **by**(*rule Strong-Early-Bisim-SC.scopeExtSum*)
**thus** *?thesis* **by**(*rule strongBisimWeakBisim*)
**qed**

**lemma** *bangSC*:
**fixes** $P$

**shows** $!P \approx P \parallel !P$
**proof** −
**have** $!P \sim_e P \parallel !P$ **by**(*rule Strong-Early-Bisim-SC.bangSC*)
**thus** *?thesis* **by**(*rule strongBisimWeakBisim*)
**qed**

**end**

**theory** *Weak-Early-Bisim-Pres*
**imports** *Strong-Early-Bisim-Pres Weak-Early-Sim-Pres Weak-Early-Bisim-SC Weak-Early-Bisim*

**begin**

**lemma** *tauPres*:
  **fixes** $P$ :: *pi*
  **and** $Q$ :: *pi*

  **assumes** $P \approx Q$

  **shows** $\tau.(P) \approx \tau.(Q)$
**proof** −
  **let** $?X = \{(\tau.(P), \tau.(Q)) \mid P\ Q.\ P \approx Q\}$
  **from** ‹$P \approx Q$› **have** $(\tau.(P), \tau.(Q)) \in ?X$ **by** *auto*
  **thus** *?thesis*
  **proof**(*coinduct rule*: *weakBisimCoinduct*)
    **case**(*cSim P Q*)
    **thus** *?case*
      **by**(*force intro*: *Weak-Early-Sim-Pres.tauPres*)
  **next**
    **case**(*cSym P Q*)
    **thus** *?case* **by**(*force dest*: *Weak-Early-Bisim.symetric simp add*: *pi.inject*)
  **qed**
**qed**

**lemma** *outputPres*:
  **fixes** $P$ :: *pi*
  **and** $Q$ :: *pi*
  **and** $a$ :: *name*
  **and** $b$ :: *name*

  **assumes** $P \approx Q$

  **shows** $a\{b\}.P \approx a\{b\}.Q$
**proof** −
  **let** $?X = \{(a\{b\}.(P), a\{b\}.(Q)) \mid P\ Q\ a\ b.\ P \approx Q\}$
  **from** ‹$P \approx Q$› **have** $(a\{b\}.(P), a\{b\}.(Q)) \in ?X$ **by** *auto*
  **thus** *?thesis*
  **proof**(*coinduct rule*: *weakBisimCoinduct*)
    **case**(*cSim P Q*)
    **thus** *?case*
      **by**(*force intro*: *Weak-Early-Sim-Pres.outputPres*)
  **next**
    **case**(*cSym P Q*)
    **thus** *?case* **by**(*force dest*: *Weak-Early-Bisim.symetric simp add*: *pi.inject*)
  **qed**
**qed**

**lemma** *inputPres*:

**fixes** *P* :: *pi*
**and** *Q* :: *pi*
**and** *a* :: *name*
**and** *x* :: *name*

**assumes** *PSimQ*: ∀ *y*. *P*[*x*::=*y*] ≈ *Q*[*x*::=*y*]

**shows** *a*<*x*>.*P* ≈ *a*<*x*>.*Q*
**proof** −
**let** *?X* = {(*a*<*x*>.*P*, *a*<*x*>.*Q*) | *a x P Q*. ∀ *y*. *P*[*x*::=*y*] ≈ *Q*[*x*::=*y*]}
**{**
**fix** *axP axQ p*
**assume** (*axP*, *axQ*) ∈ *?X*
**then obtain** *a x P Q* **where** *A*: ∀ *y*. *P*[*x*::=*y*] ≈ *Q*[*x*::=*y*] **and** *B*: *axP* = *a*<*x*>.*P* **and** *C*: *axQ* = *a*<*x*>.*Q*
**by** *auto*
**have** ⋀*y*. ((*p*::*name prm*) · *P*)[(*p* · *x*)::=*y*] ≈ (*p* · *Q*)[(*p* · *x*)::=*y*]
**proof** −
**fix** *y*
**from** *A* **have** *P*[*x*::=(*rev p* · *y*)] ≈ *Q*[*x*::=(*rev p* · *y*)]
**by** *blast*
**hence** (*p* · (*P*[*x*::=(*rev p* · *y*)])) ≈ *p* · (*Q*[*x*::=(*rev p* · *y*)])
**by**(*rule eqvts*)
**thus** (*p* · *P*)[(*p* · *x*)::=*y*] ≈ (*p* · *Q*)[(*p* · *x*)::=*y*]
**by**(*simp add*: *eqvts pt-pi-rev*[*OF pt-name-inst*, *OF at-name-inst*])
**qed**
**hence** ((*p*::*name prm*) · *axP*, *p* · *axQ*) ∈ *?X* **using** *B C*
**by** *auto*
**}**
**hence** *eqvt ?X* **by**(*simp add*: *eqvt-def*)

**from** *PSimQ* **have** (*a*<*x*>.*P*, *a*<*x*>.*Q*) ∈ *?X* **by** *auto*
**thus** *?thesis*
**proof**(*coinduct rule*: *weakBisimCoinduct*)
**case**(*cSim P Q*)
**thus** *?case* **using** ‹*eqvt ?X*›
**by**(*force intro*: *Weak-Early-Sim-Pres.inputPres*)
**next**
**case**(*cSym P Q*)
**thus** *?case*
**by**(*blast dest*: *weakBisimE*)
**qed**
**qed**

**lemma** *resPres*:
**fixes** *P* :: *pi*
**and** *Q* :: *pi*
**and** *x* :: *name*

**assumes** $P \approx Q$

**shows** $<\nu x>P \approx <\nu x>Q$
**proof** −
**let** $?X = \{(<\nu x>P, <\nu x>Q) \mid x\ P\ Q.\ P \approx Q\}$
**from** ‹$P \approx Q$› **have** $(<\nu x>P, <\nu x>Q) \in ?X$ **by** *blast*
**thus** *?thesis*
**proof**(*coinduct rule*: *weakBisimCoinduct*)
**case**(*cSim xP xQ*)
**{**
**fix** $P\ Q\ x$
**assume** $P \approx Q$
**hence** $P \leadsto<weakBisim>\ Q$ **by**(*rule weakBisimE*)
**moreover have** $\bigwedge P\ Q\ x.\ P \approx Q \Longrightarrow (<\nu x>P, <\nu x>Q) \in ?X \cup weakBisim$
**by** *blast*
**moreover have** $weakBisim \subseteq ?X \cup weakBisim$ **by** *blast*
**moreover have** *eqvt weakBisim* **by** *simp*
**moreover have** *eqvt* $(?X \cup weakBisim)$
**by**(*auto simp add*: *eqvt-def dest*: *Weak-Early-Bisim.eqvtI*)+
**ultimately have** $<\nu x>P \leadsto<(?X \cup weakBisim)> <\nu x>Q$
**by**(*rule Weak-Early-Sim-Pres.resPres*)
**}**
**with** ‹$(xP, xQ) \in ?X$› **show** *?case* **by** *blast*
**next**
**case**(*cSym xP xQ*)
**thus** *?case* **by**(*blast dest*: *Weak-Early-Bisim.symetric*)
**qed**
**qed**

**lemma** *matchPres*:
**fixes** $P :: pi$
**and** $Q :: pi$
**and** $a :: name$
**and** $b :: name$

**assumes** $P \approx Q$

**shows** $[a \frown b]P \approx [a \frown b]Q$
**proof** −
**let** $?X = \{([a \frown b]P, [a \frown b]Q) \mid a\ b\ P\ Q.\ P \approx Q\}$
**from** ‹$P \approx Q$› **have** $([a \frown b]P, [a \frown b]Q) \in ?X$ **by** *blast*
**thus** *?thesis*
**proof**(*coinduct rule*: *weakBisimCoinduct*)
**case**(*cSim abP abQ*)
**{**
**fix** $P\ Q\ a\ b$
**assume** $P \approx Q$
**hence** $P \leadsto<weakBisim>\ Q$ **by**(*rule weakBisimE*)
**moreover have** $weakBisim \subseteq (?X \cup weakBisim)$ **by** *blast*

**moreover have** $\bigwedge P\ Q\ a.\ P \approx Q \Longrightarrow [a \frown a]P \approx Q$
**by** (*metis* (*full-types*) *strongBisimWeakBisim Strong-Early-Bisim-SC.matchId Weak-Early-Bisim.transitive*)
**ultimately have**$[a \frown b]P \rightsquigarrow<(?X \cup weakBisim)> [a \frown b]Q$
**by**(*rule Weak-Early-Sim-Pres.matchPres*)
**}**
**with** ‹$(abP, abQ) \in ?X$› **show** *?case* **by** *blast*
**next**
**case**(*cSym abP abQ*)
**thus** *?case* **by**(*blast dest*: *Weak-Early-Bisim.symetric*)
**qed**
**qed**

**lemma** *mismatchPres*:
**fixes** $P :: pi$
**and** $Q :: pi$
**and** $a :: name$
**and** $b :: name$

**assumes** $P \approx Q$

**shows** $[a \neq b]P \approx [a \neq b]Q$
**proof** −
**let** $?X = \{([a \neq b]P, [a \neq b]Q) |\ a\ b\ P\ Q.\ P \approx Q\}$
**from** ‹$P \approx Q$› **have** $([a \neq b]P, [a \neq b]Q) \in ?X$ **by** *blast*
**thus** *?thesis*
**proof**(*coinduct rule*: *weakBisimCoinduct*)
**case**(*cSim abP abQ*)
**{**
**fix** $P\ Q\ a\ b$
**assume** $P \approx Q$
**hence** $P \rightsquigarrow<weakBisim> Q$ **by**(*rule weakBisimE*)
**moreover have** $weakBisim \subseteq (?X \cup weakBisim)$ **by** *blast*
**moreover have** $\bigwedge P\ Q\ a\ b.\ [\![P \approx Q;\ a \neq b]\!] \Longrightarrow [a \neq b]P \approx Q$
**by** (*metis* (*full-types*) *strongBisimWeakBisim Strong-Early-Bisim-SC.mismatchId Weak-Early-Bisim.transitive*)
**ultimately have** $[a \neq b]P \rightsquigarrow<(?X \cup weakBisim)> [a \neq b]Q$
**by**(*rule Weak-Early-Sim-Pres.mismatchPres*)
**}**
**with** ‹$(abP, abQ) \in ?X$› **show** *?case* **by** *blast*
**next**
**case**(*cSym abP abQ*)
**thus** *?case* **by**(*blast dest*: *Weak-Early-Bisim.symetric*)
**qed**
**qed**

**lemma** *parPres*:
**fixes** $P :: pi$
**and** $Q :: pi$

**and** $R :: pi$

**assumes** $P \approx Q$

**shows** $P \parallel R \approx Q \parallel R$
**proof** $-$
**let** $?X = \{(resChain\ lst\ (P \parallel R),\ resChain\ lst\ (Q \parallel R)) \mid lst\ P\ R\ Q.\ P \approx Q\}$
**have** $BC$: $\bigwedge P\ Q.\ P \parallel Q = resChain\ []\ (P \parallel Q)$ **by** *auto*
**from** ‹$P \approx Q$› **have** $(P \parallel R,\ Q \parallel R) \in\ ?X$ **by**(*blast intro*: $BC$)
**thus** *?thesis*
**proof**(*coinduct rule*: *weakBisimCoinduct*)
**case**(*cSym PR QR*)
**{**
**fix** $P\ Q\ R\ lst$
**assume** $P \approx Q$
**moreover hence** $P \leadsto<weakBisim>\ Q$ **by**(*rule weakBisimE*)
**moreover have** $\bigwedge P\ Q\ R.\ P \approx Q \Longrightarrow (P \parallel R,\ Q \parallel R) \in\ ?X$ **using** $BC$
**by** *blast*
**moreover {**
**fix** $PR\ QR\ x$
**assume** $(PR,\ QR) \in\ ?X$
**then obtain** $lst\ P\ Q\ R$ **where** $P \approx Q$ **and** $A$: $PR = resChain\ lst\ (P \parallel R)$
**and** $B$: $QR = resChain\ lst\ (Q \parallel R)$
**by** *auto*
**from** $A$ **have** $<\nu x>PR = resChain\ (x\#lst)\ (P \parallel R)$ **by** *auto*
**moreover from** $B$ **have** $<\nu x>QR = resChain\ (x\#lst)\ (Q \parallel R)$ **by** *auto*
**ultimately have** $(<\nu x>PR,\ <\nu x>QR) \in\ ?X$ **using** ‹$P \approx Q$›
**by** *blast*
**}**
**note** $Res = this$
**ultimately have** $P \parallel R \leadsto<?X>\ Q \parallel R$
**by**(*rule-tac Weak-Early-Sim-Pres.parPres*)
**moreover have** *eqvt ?X*
**by**(*auto simp add*: *eqvt-def*) (*blast intro*: *eqvts*)
**ultimately have** $resChain\ lst\ (P \parallel R) \leadsto<?X>\ resChain\ lst\ (Q \parallel R)$ **using**
*Res*
**by**(*rule-tac Weak-Early-Sim-Pres.resChainI*)
**hence** $resChain\ lst\ (P \parallel R) \leadsto<(?X \cup weakBisim)>\ resChain\ lst\ (Q \parallel R)$
**by**(*force intro*: *Weak-Early-Sim.monotonic*)
**}**
**with** ‹$(PR,\ QR) \in\ ?X$› **show** $PR \leadsto<(?X \cup weakBisim)>\ QR$
**by** *blast*
**next**
**case**(*cSym PR QR*)
**thus** *?case* **by**(*blast dest*: *Weak-Early-Bisim.symetric*)
**qed**
**qed**

**lemma** *bangPres*:

```
  fixes P :: pi
  and   Q :: pi

  assumes PBisimQ: P ≈ Q

  shows !P ≈ !Q
proof -
  let ?X = (bangRel weakBisim)
  let ?Y = Strong-Early-Bisim.bisim O (bangRel weakBisim) O Strong-Early-Bisim.bisim

  from Weak-Early-Bisim.eqvt Strong-Early-Bisim.eqvt have eqvtY: eqvt ?Y by(blast
intro: eqvtBangRel)
  have XsubY: ?X ⊆ ?Y by(auto intro: Strong-Early-Bisim.reflexive)

  have RelStay: ⋀P Q. (P ∥ !P, Q) ∈ ?Y ⟹ (!P, Q) ∈ ?Y
  proof(auto)
    fix P Q R T
    assume PBisimQ: P ∥ !P ∼e Q
      and QBRR: (Q, R) ∈ bangRel weakBisim
      and RBisimT: R ∼e T
    have !P ∼e Q
    proof -
      have !P ∼e P ∥ !P by(rule Strong-Early-Bisim-SC.bangSC)
      thus ?thesis using PBisimQ by(rule Strong-Early-Bisim.transitive)
    qed
    with QBRR RBisimT show (!P, T) ∈ ?Y by blast
  qed

  have ParCompose: ⋀P Q R T. ⟦P ≈ Q; (R, T) ∈ ?Y⟧ ⟹ (P ∥ R, Q ∥ T) ∈
?Y
  proof -
    fix P Q R T
    assume PBisimQ: P ≈ Q
      and RYT:     (R, T) ∈ ?Y
    thus (P ∥ R, Q ∥ T) ∈ ?Y
    proof(auto)
      fix T' R'
      assume T'BisimT: T' ∼e T and RBisimR': R ∼e R'
        and R'BRT': (R', T') ∈ bangRel weakBisim
      have P ∥ R ∼e P ∥ R'
      proof -
      from RBisimR' have R ∥ P ∼e R' ∥ P by(rule Strong-Early-Bisim-Pres.parPres)
          moreover have P ∥ R ∼e R ∥ P and R' ∥ P ∼e P ∥ R' by(rule
Strong-Early-Bisim-SC.parSym)+
        ultimately show ?thesis by(blast intro: Strong-Early-Bisim.transitive)
      qed
    moreover from PBisimQ R'BRT' have (P ∥ R', Q ∥ T') ∈ bangRel weakBisim
by(rule BRPar)
      moreover have Q ∥ T' ∼e Q ∥ T
```

**proof** −
**from** $T'BisimT$ **have** $T' \parallel Q \sim_e T \parallel Q$ **by**(*rule Strong-Early-Bisim-Pres.parPres*)
**moreover have** $Q \parallel T' \sim_e T' \parallel Q$ **and** $T \parallel Q \sim_e Q \parallel T$ **by**(*rule Strong-Early-Bisim-SC.parSym*)+
**ultimately show** *?thesis* **by**(*blast intro*: *Strong-Early-Bisim.transitive*)
**qed**
**ultimately show** *?thesis* **by** *blast*
**qed**
**qed**

**have** *ResCong*: $\bigwedge P\ Q\ x.\ (P,\ Q) \in ?Y \Longrightarrow (<\nu x>P,\ <\nu x>Q) \in ?Y$
**by**(*auto intro*: *BRRes Strong-Early-Bisim-Pres.resPres transitive*)

**have** *Sim*: $\bigwedge P\ Q.\ (P,\ Q) \in ?X \Longrightarrow P \rightsquigarrow<?Y>\ Q$
**proof** −
**fix** $P\ Q$
**assume** $(P,\ Q) \in ?X$
**thus** $P \rightsquigarrow<?Y>\ Q$
**proof**(*induct*)
**case**(*BRBang P Q*)
**have** $P \approx Q$ **by** *fact*
**moreover hence** $P \rightsquigarrow<weakBisim>\ Q$ **by**(*blast dest*: *weakBisimE*)
**moreover have** $\bigwedge P\ Q.\ P \approx Q \Longrightarrow P \rightsquigarrow<weakBisim>\ Q$ **by**(*blast dest*: *weakBisimE*)
**moreover from** *Strong-Early-Bisim.eqvt Weak-Early-Bisim.eqvt* **have** *eqvt ?Y* **by**(*blast intro*: *eqvtBangRel*)

**ultimately show** $!P \rightsquigarrow<?Y>\ !Q$ **using** *ParCompose ResCong RelStay XsubY*
**by**(*rule-tac Weak-Early-Sim-Pres.bangPres*, *simp-all*)
**next**
**case**(*BRPar P Q R T*)
**have** *PBiSimQ*: $P \approx Q$ **by** *fact*
**moreover have** *RBangRelT*: $(R,\ T) \in ?X$ **by** *fact*
**have** *RSimT*: $R \rightsquigarrow<?Y>\ T$ **by** *fact*
**moreover from** *PBiSimQ* **have** $P \rightsquigarrow<weakBisim>\ Q$ **by**(*blast dest*: *weakBisimE*)
**moreover from** *RBangRelT* **have** $(R,\ T) \in ?Y$ **by**(*blast intro*: *Strong-Early-Bisim.reflexive*)
**ultimately show** $P \parallel R \rightsquigarrow<?Y>\ Q \parallel T$ **using** *ParCompose ResCong eqvt eqvtY*
**by**(*rule-tac Weak-Early-Sim-Pres.parCompose*)
**next**
**case**(*BRRes P Q x*)
**have** $P \rightsquigarrow<?Y>\ Q$ **by** *fact*
**thus** $<\nu x>P \rightsquigarrow<?Y>\ <\nu x>Q$ **using** *ResCong eqvtY XsubY*
**by**(*rule-tac Weak-Early-Sim-Pres.resPres*, *simp-all*)
**qed**
**qed**

**from** *PBisimQ* **have** $(!P,\ !Q) \in ?X$ **by**(*rule BRBang*)

**moreover from** *Weak-Early-Bisim.eqvt* **have** *eqvt* (*bangRel weakBisim*) **by**(*rule eqvtBangRel*)
**ultimately show** *?thesis*
**apply**(*coinduct rule*: *Weak-Early-Bisim.transitive-coinduct-weak*)
**apply**(*blast intro*: *Sim*)
**by**(*blast dest*: *bangRelSymetric Weak-Early-Bisim.symetric intro*: *Strong-Early-Bisim.reflexive*)
**qed**

**lemma** *bangRelSubWeakBisim*:
**shows** *bangRel weakBisim* $\subseteq$ *weakBisim*
**proof**(*auto*)
**fix** *a b*
**assume** $(a, b) \in$ *bangRel weakBisim*
**thus** $a \approx b$
**proof**(*induct*)
**fix** *P Q*
**assume** $P \approx Q$
**thus** $!P \approx !Q$ **by**(*rule bangPres*)
**next**
**fix** *P Q R T*
**assume** $R \approx T$ **and** $P \approx Q$
**thus** $R \parallel P \approx T \parallel Q$ **by**(*metis parPres parSym Weak-Early-Bisim.transitive*)
**next**
**fix** *P Q*
**fix** *a*::*name*
**assume** $P \approx Q$
**thus** $<\nu a>P \approx <\nu a>Q$ **by**(*rule resPres*)
**qed**
**qed**

**end**

**theory** *Weak-Early-Cong-Pres*
**imports** *Weak-Early-Cong Weak-Early-Step-Sim-Pres Weak-Early-Bisim-Pres*
**begin**

**lemma** *tauPres*:
**fixes** *P* :: *pi*
**and** *Q* :: *pi*

**assumes** $P \simeq Q$

**shows** $\tau.(P) \simeq \tau.(Q)$
**proof** −
**from** *assms* **have** $P \approx Q$ **by**(*rule congruenceWeakBisim*)
**thus** *?thesis* **by**(*force intro*: *Weak-Early-Step-Sim-Pres.tauPres simp add*: *weak-Congruence-def dest*: *weakBisimE*(*2*))
**qed**

**lemma** *outputPres*:
  **fixes** $P :: pi$
  **and** $Q :: pi$

  **assumes** $P \simeq Q$

  **shows** $a\{b\}.P \simeq a\{b\}.Q$
**proof** −
  **from** *assms* **have** $P \approx Q$ **by**(*rule congruenceWeakBisim*)
  **thus** *?thesis* **by**(*force intro*: *Weak-Early-Step-Sim-Pres.outputPres simp add*: *weakCongruence-def dest*: *weakBisimE*(*2*))
**qed**

**lemma** *matchPres*:
  **fixes** $P :: pi$
  **and** $Q :: pi$
  **and** $a :: name$
  **and** $b :: name$

  **assumes** $P \simeq Q$

  **shows** $[a\frown b]P \simeq [a\frown b]Q$
**using** *assms*
**by**(*auto simp add*: *weakCongruence-def intro*: *Weak-Early-Step-Sim-Pres.matchPres*)

**lemma** *mismatchPres*:
  **fixes** $P :: pi$
  **and** $Q :: pi$
  **and** $a :: name$
  **and** $b :: name$

  **assumes** $P \simeq Q$

  **shows** $[a\neq b]P \simeq [a\neq b]Q$
**using** *assms*
**by**(*auto simp add*: *weakCongruence-def intro*: *Weak-Early-Step-Sim-Pres.mismatchPres*)

**lemma** *sumPres*:
  **fixes** $P :: pi$
  **and** $Q :: pi$
  **and** $R :: pi$

  **assumes** $P \simeq Q$

  **shows** $P \oplus R \simeq Q \oplus R$
**using** *assms*
**by**(*auto simp add*: *weakCongruence-def intro*: *Weak-Early-Step-Sim-Pres.sumPres Weak-Early-Bisim.reflexive*)

**lemma** *parPres*:
**fixes** $P$ :: *pi*
**and** $Q$ :: *pi*
**and** $R$ :: *pi*

**assumes** $P \simeq Q$

**shows** $P \parallel R \simeq Q \parallel R$
**proof** −
**have** $\bigwedge P\ Q\ R.\ [\![P \leadsto«weakBisim»\ Q;\ P \approx Q]\!] \Longrightarrow P \parallel R \leadsto«weakBisim»\ Q \parallel R$
**proof** −
**fix** $P\ Q\ R$
**assume** $P \leadsto«weakBisim»\ Q$ **and** $P \approx Q$
**thus** $P \parallel R \leadsto«weakBisim»\ Q \parallel R$
**using** *Weak-Early-Bisim-Pres.parPres Weak-Early-Bisim-Pres.resPres Weak-Early-Bisim.reflexive Weak-Early-Bisim.eqvt*
**by**(*blast intro*: *Weak-Early-Step-Sim-Pres.parPres*)
**qed**
**moreover from** *assms* **have** $P \approx Q$ **by**(*rule congruenceWeakBisim*)
**ultimately show** *?thesis* **using** *assms*
**by**(*auto simp add*: *weakCongruence-def dest*: *weakBisimE*)
**qed**

**lemma** *resPres*:
**fixes** $P$ :: *pi*
**and** $Q$ :: *pi*
**and** $x$ :: *name*

**assumes** *PeqQ*: $P \simeq Q$

**shows** $<\nu x>P \simeq <\nu x>Q$
**proof** −
**have** $\bigwedge P\ Q\ x.\ P \leadsto«weakBisim»\ Q \Longrightarrow <\nu x>P \leadsto«weakBisim»\ <\nu x>Q$
**proof** −
**fix** $P\ Q\ x$
**assume** $P \leadsto«weakBisim»\ Q$
**with** *Weak-Early-Bisim.eqvt Weak-Early-Bisim-Pres.resPres* **show** $<\nu x>P \leadsto«weakBisim»\ <\nu x>Q$
**by**(*blast intro*: *Weak-Early-Step-Sim-Pres.resPres*)
**qed**
**with** *assms* **show** *?thesis* **by**(*simp add*: *weakCongruence-def*)
**qed**

**lemma** *bangPres*:
**fixes** $P$ :: *pi*
**and** $Q$ :: *pi*

**assumes** $P \simeq Q$

**shows** $!P \simeq !Q$
**using** *assms*
**proof**(*induct rule*: *weakCongISym2*)
**case**(*cSim P Q*)
**let** *?X* = $\{(P, Q) \mid P\ Q.\ P \simeq Q\}$
**from** ‹$P \simeq Q$› **have** $(P, Q) \in$ *?X* **by** *auto*
**moreover have** $\bigwedge P\ Q.\ (P, Q) \in$ *?X* $\Longrightarrow P \rightsquigarrow$«*weakBisim*» $Q$ **by**(*auto simp add*: *weakCongruence-def*)
**moreover from** *congruenceWeakBisim* **have** *?X* $\subseteq$ *weakBisim* **by** *auto*
**ultimately have** $!P \rightsquigarrow$«*bangRel weakBisim*» $!Q$ **using** *Weak-Early-Bisim.eqvt*
**by**(*rule Weak-Early-Step-Sim-Pres.bangPres*)
**moreover have** *bangRel weakBisim* $\subseteq$ *weakBisim* **by**(*rule bangRelSubWeakBisim*)
**ultimately show** $!P \rightsquigarrow$«*weakBisim*» $!Q$
**by**(*rule Weak-Early-Step-Sim.monotonic*)
**qed**

**end**

**theory** *Weak-Early-Cong-Subst-Pres*
**imports** *Weak-Early-Cong-Subst Weak-Early-Cong-Pres*
**begin**

**lemma** *weakCongStructCong*:
**fixes** $P$ :: *pi*
**and** $Q$ :: *pi*

**assumes** $P \equiv_s Q$

**shows** $P \simeq^s Q$
**using** *assms*
**by**(*metis earlyCongStructCong strongEqWeakCong*)

**lemma** *tauPres*:
**fixes** $P$ :: *pi*
**and** $Q$ :: *pi*

**assumes** $P \simeq^s Q$

**shows** $\tau.(P) \simeq^s \tau.(Q)$
**using** *assms*
**by**(*auto simp add*: *weakCongruenceSubst-def intro*: *Weak-Early-Cong-Pres.tauPres*)

**lemma** *inputPres*:
**fixes** $P$ :: *pi*
**and** $Q$ :: *pi*
**and** $a$ :: *name*
**and** $x$ :: *name*

**assumes** *PeqQ*: $P \simeq^s Q$

**shows** $a{<}x{>}.P \simeq^s a{<}x{>}.Q$
**proof**(*auto simp add*: *weakCongruenceSubst-def*)
**fix** *s*::(*name* × *name*) *list*

**from** *congruenceWeakBisim* **have** *Input*: $\bigwedge P\ Q\ a\ x\ s.\ [\![P[{<}s{>}] \simeq^s Q[{<}s{>}];\ x \sharp s]\!] \Longrightarrow (a{<}x{>}.P)[{<}s{>}] \simeq (a{<}x{>}.Q)[{<}s{>}]$
**apply**(*auto simp add*: *weakCongruenceSubst-def weakCongruence-def*)
**apply**(*rule Weak-Early-Step-Sim-Pres.inputPres*, *auto*)
**apply**(*erule-tac x*=[(*x*, *y*)] **in** *allE*, *auto*)
**apply**(*rule Weak-Early-Step-Sim-Pres.inputPres*, *auto*)
**by**(*erule-tac x*=[(*x*, *y*)] **in** *allE*, *auto*)

**then obtain** *c*::*name* **where** *cFreshP*: $c \sharp P$ **and** *cFreshQ*: $c \sharp Q$ **and** *cFreshs*: $c \sharp s$
**by**(*force intro*: *name-exists-fresh*[*of* (*P*, *Q*, *s*)])

**from** *PeqQ* **have** $P[{<}([(x, c)] \cdot s){>}] \simeq^s Q[{<}([(x, c)] \cdot s){>}]$ **by**(*rule partUnfold*)
**hence** $([(x, c)] \cdot P[{<}([(x, c)] \cdot s){>}]) \simeq^s ([(x, c)] \cdot Q[{<}([(x, c)] \cdot s){>}])$ **by**(*rule Weak-Early-Cong-Subst.eqvtI*)
**hence** $([(x, c)] \cdot P)[{<}s{>}] \simeq^s ([(x, c)] \cdot Q)[{<}s{>}]$ **by** *simp*
**hence** $(a{<}c{>}.([(x, c)] \cdot P))[{<}s{>}] \simeq (a{<}c{>}.([(x, c)] \cdot Q))[{<}s{>}]$ **using** *cFreshs* **by**(*rule Input*)

**moreover from** *cFreshP cFreshQ* **have** $a{<}x{>}.P = a{<}c{>}.([(x, c)] \cdot P)$ **and** $a{<}x{>}.Q = a{<}c{>}.([(x, c)] \cdot Q)$
**by**(*simp add*: *Agent.alphaInput*)+

**ultimately show** $(a{<}x{>}.P)[{<}s{>}] \simeq (a{<}x{>}.Q)[{<}s{>}]$ **by** *simp*
**qed**

**lemma** *outputPres*:
**fixes** *P* :: *pi*
**and** *Q* :: *pi*

**assumes** $P \simeq^s Q$

**shows** $a\{b\}.P \simeq^s a\{b\}.Q$
**using** *assms*
**by**(*auto simp add*: *weakCongruenceSubst-def intro*: *Weak-Early-Cong-Pres.outputPres*)

**lemma** *matchPres*:
**fixes** *P* :: *pi*
**and** *Q* :: *pi*
**and** *a* :: *name*
**and** *b* :: *name*

**assumes** $P \simeq^s Q$

**shows** $[a \frown b]P \simeq^s [a \frown b]Q$
**using** *assms*
**by**(*auto simp add*: *weakCongruenceSubst-def intro*: *Weak-Early-Cong-Pres.matchPres*)

**lemma** *mismatchPres*:
**fixes** $P :: pi$
**and** $Q :: pi$
**and** $a :: name$
**and** $b :: name$

**assumes** $P \simeq^s Q$

**shows** $[a \neq b]P \simeq^s [a \neq b]Q$
**using** *assms*
**by**(*auto simp add*: *weakCongruenceSubst-def intro*: *Weak-Early-Cong-Pres.mismatchPres*)

**lemma** *sumPres*:
**fixes** $P :: pi$
**and** $Q :: pi$
**and** $R :: pi$

**assumes** $P \simeq^s Q$

**shows** $P \oplus R \simeq^s Q \oplus R$
**using** *assms*
**by**(*auto simp add*: *weakCongruenceSubst-def intro*: *Weak-Early-Cong-Pres.sumPres*)

**lemma** *parPres*:
**fixes** $P :: pi$
**and** $Q :: pi$
**and** $R :: pi$

**assumes** $P \simeq^s Q$

**shows** $P \parallel R \simeq^s Q \parallel R$
**using** *assms*
**by**(*auto simp add*: *weakCongruenceSubst-def intro*: *Weak-Early-Cong-Pres.parPres*)

**lemma** *resPres*:
**fixes** $P :: pi$
**and** $Q :: pi$
**and** $x :: name$

**assumes** *PeqQ*: $P \simeq^s Q$

**shows** $<\nu x>P \simeq^s <\nu x>Q$
**proof**(*auto simp add*: *weakCongruenceSubst-def*)

**fix** *s*::(*name* × *name*) *list*

**have** *Goal*: $\bigwedge P\ Q\ x\ s.\ [\![P[<s>] \rightsquigarrow«weakBisim»\ Q[<s>];\ x \sharp s]\!] \Longrightarrow (<\nu x>P)[<s>] \rightsquigarrow«weakBisim»\ (<\nu x>Q)[<s>]$
**by**(*force intro*: *Weak-Early-Step-Sim-Pres.resPres Weak-Early-Bisim-Pres.resPres Weak-Early-Bisim.eqvt*)

**then obtain** *c*::*name* **where** *cFreshP*: $c \sharp P$ **and** *cFreshQ*: $c \sharp Q$ **and** *cFreshs*: $c \sharp s$
**by**(*force intro*: *name-exists-fresh*[*of* (*P*, *Q*, *s*)])

**from** *PeqQ* **have** $P[<([(x, c)] \cdot s)>] \rightsquigarrow«weakBisim»\ Q[<([(x, c)] \cdot s)>]$ **and** $Q[<([(x, c)] \cdot s)>] \rightsquigarrow«weakBisim»\ P[<([(x, c)] \cdot s)>]$
**by**(*force simp add*: *weakCongruenceSubst-def weakCongruence-def*)+

**hence** $([(x, c)] \cdot (P[<([(x, c)] \cdot s)>])) \rightsquigarrow«weakBisim»\ ([(x, c)] \cdot (Q[<([(x, c)] \cdot s)>]))$ **and**
$([(x, c)] \cdot (Q[<([(x, c)] \cdot s)>])) \rightsquigarrow«weakBisim»\ ([(x, c)] \cdot (P[<([(x, c)] \cdot s)>]))$
**by**(*blast intro*: *Weak-Early-Step-Sim.eqvtI Weak-Early-Bisim.eqvt*)+

**hence** $([(x, c)] \cdot P)[<s>] \rightsquigarrow«weakBisim»\ ([(x, c)] \cdot Q)[<s>]$ **and**
$([(x, c)] \cdot Q)[<s>] \rightsquigarrow«weakBisim»\ ([(x, c)] \cdot P)[<s>]$ **by** *simp*+

**with** *cFreshs* **have** $(<\nu c>([(x, c)] \cdot P))[<s>] \rightsquigarrow«weakBisim»\ (<\nu c>([(x, c)] \cdot Q))[<s>]$ **and**
$(<\nu c>([(x, c)] \cdot Q))[<s>] \rightsquigarrow«weakBisim»\ (<\nu c>([(x, c)] \cdot P))[<s>]$
**by**(*blast intro*: *Goal*)+

**moreover from** *cFreshP cFreshQ* **have** $<\nu x>P = <\nu c>([(x, c)] \cdot P)$ **and** $<\nu x>Q = <\nu c>([(x, c)] \cdot Q)$
**by**(*simp add*: *alphaRes*)+

**ultimately show** $(<\nu x>P)[<s>] \simeq (<\nu x>Q)[<s>]$
**by**(*simp add*: *weakCongruence-def*)
**qed**

**lemma** *bangPres*:
**fixes** *P* :: *pi*
**and** *Q* :: *pi*

**assumes** $P \simeq^s Q$

**shows** $!P \simeq^s !Q$
**using** *assms*
**by**(*auto simp add*: *weakCongruenceSubst-def intro*: *Weak-Early-Cong-Pres.bangPres*)

**end**

**theory** *Strong-Late-Expansion-Law*
  **imports** *Strong-Late-Bisim-SC*
**begin**

**nominal-primrec** *summands* :: *pi* $\Rightarrow$ *pi set* **where**
  *summands* $\mathbf{0} = \{\}$
| *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* $(!P) = \{\}$
**apply**(*auto simp add*: *fresh-singleton name-fresh-abs fresh-set-empty fresh-singleton pi.fresh*)
**apply**(*finite-guess*)+
**by**(*fresh-guess*)+

**lemma** *summandsInput*[*simp*]:
  **fixes** $a$ :: *name*
  **and** $x$ :: *name*
  **and** $P$ :: *pi*

  **shows** *summands* $(a<x>.P) = \{a<x>.P\}$
**proof** −
  **obtain** $y$ **where** *yineqa*: $y \neq a$ **and** *yFreshP*: $y \sharp P$
    **by**(*force intro*: *name-exists-fresh*[*of* $(a, P)$] *simp add*: *fresh-prod*)
  **from** *yFreshP* **have** $a<x>.P = a<y>.([(x, y)] \cdot P)$ **by**(*simp add*: *alphaInput*)
  **with** *yineqa* **show** *?thesis* **by** *simp*
**qed**

**lemma** *finiteSummands*:
  **fixes** $P$ :: *pi*

  **shows** *finite*(*summands* $P$)
**by**(*induct* $P$ *rule*: *pi.induct*) *auto*

**lemma** *boundSummandDest*[*dest*]:
  **fixes** $x$ :: *name*
  **and** $y$ :: *name*
  **and** $P'$ :: *pi*
  **and** $P$ :: *pi*

  **assumes** $<\nu x>x\{y\}.P' \in$ *summands* $P$

  **shows** *False*

**using** *assms*
**by**(*induct P rule*: *pi.induct*, *auto simp add*: *if-split pi.inject name-abs-eq name-calc*)

**lemma** *summandFresh*:
  **fixes** $P :: pi$
  **and** $Q :: pi$
  **and** $x :: name$

  **assumes** $P \in summands\ Q$
  **and** $x \sharp Q$

  **shows** $x \sharp P$
**using** *assms*
**by**(*nominal-induct Q avoiding*: *P rule*: *pi.strong-induct*, *auto simp add*: *if-split*)

**nominal-primrec** $hnf :: pi \Rightarrow bool$ **where**
  $hnf\ \mathbf{0} = True$
| $hnf\ (\tau.(P)) = True$
| $x \sharp a \Longrightarrow hnf\ (a{<}x{>}.P) = True$
| $hnf\ (a\{b\}.P) = True$
| $hnf\ ([a{\frown}b]P) = False$
| $hnf\ ([a{\neq}b]P) = False$
| $hnf\ (P \oplus Q) = ((hnf\ P) \wedge (hnf\ Q) \wedge P \neq \mathbf{0} \wedge Q \neq \mathbf{0})$
| $hnf\ (P \parallel Q) = False$
| $hnf\ ({<}\nu x{>}P) = (\exists\, a\ P'.\ a \neq x \wedge P = a\{x\}.P')$
| $hnf\ (!P) = False$
**apply**(*auto simp add*: *fresh-bool*)
**apply**(*finite-guess*)+
**by**(*fresh-guess*)+

**lemma** *hnfInput*[*simp*]:
  **fixes** $a :: name$
  **and** $x :: name$
  **and** $P :: pi$

  **shows** $hnf\ (a{<}x{>}.P)$
**proof** −
  **obtain** $y$ **where** *yineqa*: $y \neq a$ **and** *yFreshP*: $y \sharp P$
    **by**(*force intro*: *name-exists-fresh*[*of* (*a*, *P*)] *simp add*: *fresh-prod*)
  **from** *yFreshP* **have** $a{<}x{>}.P = a{<}y{>}.([(x, y)] \cdot P)$ **by**(*simp add*: *alphaInput*)
  **with** *yineqa* **show** *?thesis* **by** *simp*
**qed**

**lemma** *summandTransition*:
  **fixes** $P :: pi$
  **and** $a :: name$
  **and** $x :: name$
  **and** $b :: name$
  **and** $P' :: pi$

**assumes** *hnf P*

**shows** $P \longmapsto\tau \prec P' = (\tau.(P') \in \mathit{summands}\ P)$
**and** $P \longmapsto a{<}x{>} \prec P' = (a{<}x{>}.P' \in \mathit{summands}\ P)$
**and** $P \longmapsto a[b] \prec P' = (a\{b\}.P' \in \mathit{summands}\ P)$
**and** $a \neq x \Longrightarrow P \longmapsto a{<}\nu x{>} \prec P' = ({<}\nu x{>}a\{x\}.P' \in \mathit{summands}\ P)$
**proof** −
**from** *assms* **show** $P \longmapsto\tau \prec P' = (\tau.(P') \in \mathit{summands}\ P)$
**proof**(*induct P rule*: *pi.induct*)
**case** *PiNil*
**show** *?case* **by** *auto*
**next**
**case**(*Output a b P*)
**show** *?case* **by** *auto*
**next**
**case**(*Tau P*)
**have** $\tau.(P) \longmapsto\tau \prec P' \Longrightarrow \tau.(P') \in \mathit{summands}\ (\tau.(P))$
**by**(*auto elim*: *tauCases simp add*: *pi.inject residual.inject*)
**moreover have** $\tau.(P') \in \mathit{summands}\ (\tau.(P)) \Longrightarrow \tau.(P) \longmapsto\tau \prec P'$
**by**(*auto simp add*: *pi.inject intro*: *transitions.Tau*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*Input a x P*)
**show** *?case* **by** *auto*
**next**
**case**(*Match a b P*)
**have** *hnf* $([a \frown b]P)$ **by** *fact*
**hence** *False* **by** *simp*
**thus** *?case* **by** *simp*
**next**
**case**(*Mismatch a b P*)
**have** *hnf* $([a \neq b]P)$ **by** *fact*
**hence** *False* **by** *simp*
**thus** *?case* **by** *simp*
**next**
**case**(*Sum P Q*)
**have** *hnf* $(P \oplus Q)$ **by** *fact*
**hence** *Phnf*: *hnf P* **and** *Qhnf*: *hnf Q* **by** *simp+*

**have** *IHP*: $P \longmapsto\tau \prec P' = (\tau.(P') \in \mathit{summands}\ P)$
**proof** −
**have** $\mathit{hnf}\ P \Longrightarrow P \longmapsto\tau \prec P' = (\tau.(P') \in \mathit{summands}\ P)$ **by** *fact*
**with** *Phnf* **show** *?thesis* **by** *simp*
**qed**

**have** *IHQ*: $Q \longmapsto\tau \prec P' = (\tau.(P') \in \mathit{summands}\ Q)$
**proof** −
**have** $\mathit{hnf}\ Q \Longrightarrow Q \longmapsto\tau \prec P' = (\tau.(P') \in \mathit{summands}\ Q)$ **by** *fact*

**with** *Qhnf* **show** *?thesis* **by** *simp*
**qed**

**from** *IHP IHQ* **have** $P \oplus Q \longmapsto \tau \prec P' \Longrightarrow \tau.(P') \in$ *summands* $(P \oplus Q)$
**by**(*erule-tac sumCases*, *auto*)
**moreover from** *IHP IHQ* **have** $\tau.(P') \in$ *summands* $(P \oplus Q) \Longrightarrow P \oplus Q \longmapsto \tau \prec P'$
**by**(*auto dest*: *Sum1 Sum2*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*Par P Q*)
**have** *hnf* $(P \parallel Q)$ **by** *fact*
**hence** *False* **by** *simp*
**thus** *?case* **by** *simp*
**next**
**case**(*Res x P*)
**thus** *?case* **by**(*auto elim*: *resCasesF*)
**next**
**case**(*Bang P*)
**have** *hnf* $(!P)$ **by** *fact*
**hence** *False* **by** *simp*
**thus** *?case* **by** *simp*
**qed**
**next**
**from** *assms* **show** $P \longmapsto a{<}x{>} \prec P' = (a{<}x{>}.P' \in$ *summands* $P)$
**proof**(*induct P rule*: *pi.induct*)
**case** *PiNil*
**show** *?case* **by** *auto*
**next**
**case**(*Output c b P*)
**show** *?case* **by** *auto*
**next**
**case**(*Tau P*)
**show** *?case* **by** *auto*
**next**
**case**(*Input b y P*)
**have** $b{<}y{>}.P \longmapsto a{<}x{>} \prec P' \Longrightarrow a{<}x{>}.P' \in$ *summands* $(b{<}y{>}.P)$
**by**(*auto elim*: *inputCases'* *simp add*: *pi.inject residual.inject*)
**moreover have** $a{<}x{>}.P' \in$ *summands* $(b{<}y{>}.P) \Longrightarrow b{<}y{>}.P \longmapsto a{<}x{>} \prec P'$
**apply**(*auto simp add*: *pi.inject name-abs-eq intro*: *Late-Semantics.Input*)
**apply**(*subgoal-tac* $b{<}x{>} \prec [(x, y)] \cdot P = (b{<}y{>} \prec [(x, y)] \cdot [(x, y)] \cdot P)$)
**apply**(*auto intro*: *Late-Semantics.Input*)
**by**(*simp add*: *alphaBoundResidual name-swap*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*Match a b P*)
**have** *hnf* $([a \frown b]P)$ **by** *fact*
**hence** *False* **by** *simp*

**thus** *?case* **by** *simp*
**next**
**case**(*Mismatch a b P*)
**have** *hnf* $([a{\neq}b]P)$ **by** *fact*
**hence** *False* **by** *simp*
**thus** *?case* **by** *simp*
**next**
**case**(*Sum P Q*)
**have** *hnf* $(P \oplus Q)$ **by** *fact*
**hence** *Phnf*: *hnf P* **and** *Qhnf*: *hnf Q* **by** *simp+*

**have** *IHP*: $P \longmapsto a{<}x{>} \prec P' = (a{<}x{>}.P' \in \mathit{summands}\ P)$
**proof** −
**have** $\mathit{hnf}\ P \Longrightarrow P \longmapsto a{<}x{>} \prec P' = (a{<}x{>}.P' \in \mathit{summands}\ P)$ **by** *fact*
**with** *Phnf* **show** *?thesis* **by** *simp*
**qed**

**have** *IHQ*: $Q \longmapsto a{<}x{>} \prec P' = (a{<}x{>}.P' \in \mathit{summands}\ Q)$
**proof** −
**have** $\mathit{hnf}\ Q \Longrightarrow Q \longmapsto a{<}x{>} \prec P' = (a{<}x{>}.P' \in \mathit{summands}\ Q)$ **by** *fact*
**with** *Qhnf* **show** *?thesis* **by** *simp*
**qed**

**from** *IHP IHQ* **have** $P \oplus Q \longmapsto a{<}x{>} \prec P' \Longrightarrow a{<}x{>}.P' \in \mathit{summands}\ (P \oplus Q)$
**by**(*erule-tac sumCases*, *auto*)
**moreover from** *IHP IHQ* **have** $a{<}x{>}.P' \in \mathit{summands}\ (P \oplus Q) \Longrightarrow P \oplus Q \longmapsto a{<}x{>} \prec P'$
**by**(*auto dest*: *Sum1 Sum2*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*Par P Q*)
**have** *hnf* $(P \parallel Q)$ **by** *fact*
**hence** *False* **by** *simp*
**thus** *?case* **by** *simp*
**next**
**case**(*Res y P*)
**have** $\mathit{hnf}({<}\nu y{>}P)$ **by** *fact*
**thus** *?case* **by**(*auto simp add*: *if-split*)
**next**
**case**(*Bang P*)
**have** *hnf* $(!P)$ **by** *fact*
**hence** *False* **by** *simp*
**thus** *?case* **by** *simp*
**qed**
**next**
**from** *assms* **show** $P \longmapsto a[b] \prec P' = (a\{b\}.P' \in \mathit{summands}\ P)$
**proof**(*induct P rule*: *pi.induct*)
**case** *PiNil*

**show** *?case* **by** *auto*
**next**
**case**(*Output c d P*)
**have** $c\{d\}.P \longmapsto a[b] \prec P' \Longrightarrow a\{b\}.P' \in \textit{summands}\ (c\{d\}.P)$
**by**(*auto elim*: *outputCases simp add*: *residual.inject pi.inject*)
**moreover have** $a\{b\}.P' \in \textit{summands}\ (c\{d\}.P) \Longrightarrow c\{d\}.P \longmapsto a[b] \prec P'$
**by**(*auto simp add*: *pi.inject intro*: *transitions.Output*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*Tau P*)
**show** *?case* **by** *auto*
**next**
**case**(*Input c x P*)
**show** *?case* **by** *auto*
**next**
**case**(*Match a b P*)
**have** *hnf* $([a \frown b]P)$ **by** *fact*
**hence** *False* **by** *simp*
**thus** *?case* **by** *simp*
**next**
**case**(*Mismatch a b P*)
**have** *hnf* $([a \neq b]P)$ **by** *fact*
**hence** *False* **by** *simp*
**thus** *?case* **by** *simp*
**next**
**case**(*Sum P Q*)
**have** *hnf* $(P \oplus Q)$ **by** *fact*
**hence** *Phnf*: *hnf P* **and** *Qhnf*: *hnf Q* **by** *simp+*

**have** *IHP*: $P \longmapsto a[b] \prec P' = (a\{b\}.P' \in \textit{summands}\ P)$
**proof** −
**have** $\textit{hnf}\ P \Longrightarrow P \longmapsto a[b] \prec P' = (a\{b\}.P' \in \textit{summands}\ P)$ **by** *fact*
**with** *Phnf* **show** *?thesis* **by** *simp*
**qed**

**have** *IHQ*: $Q \longmapsto a[b] \prec P' = (a\{b\}.P' \in \textit{summands}\ Q)$
**proof** −
**have** $\textit{hnf}\ Q \Longrightarrow Q \longmapsto a[b] \prec P' = (a\{b\}.P' \in \textit{summands}\ Q)$ **by** *fact*
**with** *Qhnf* **show** *?thesis* **by** *simp*
**qed**

**from** *IHP IHQ* **have** $P \oplus Q \longmapsto a[b] \prec P' \Longrightarrow a\{b\}.P' \in \textit{summands}\ (P \oplus Q)$
**by**(*erule-tac sumCases*, *auto*)
**moreover from** *IHP IHQ* **have** $a\{b\}.P' \in \textit{summands}\ (P \oplus Q) \Longrightarrow P \oplus Q \longmapsto a[b] \prec P'$
**by**(*auto dest*: *Sum1 Sum2*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*Par P Q*)

**have** *hnf* $(P \parallel Q)$ **by** *fact*
**hence** *False* **by** *simp*
**thus** *?case* **by** *simp*
**next**
**case**(*Res x P*)
**have** *hnf* $(<\nu x>P)$ **by** *fact*
**thus** *?case* **by**(*force elim*: *resCasesF outputCases simp add*: *if-split residual.inject*)
**next**
**case**(*Bang P*)
**have** *hnf* $(!P)$ **by** *fact*
**hence** *False* **by** *simp*
**thus** *?case* **by** *simp*
**qed**
**next**
**assume** $a \neq x$
**with** *assms* **show** $P \longmapsto a<\nu x> \prec P' = (<\nu x>a\{x\}.P' \in \mathit{summands}\ P)$
**proof**(*nominal-induct P avoiding*: $x\ P'$ *rule*: *pi.strong-induct*)
**case** *PiNil*
**show** *?case* **by** *auto*
**next**
**case**(*Output a b P*)
**show** *?case* **by** *auto*
**next**
**case**(*Tau P*)
**show** *?case* **by** *auto*
**next**
**case**(*Input a x P*)
**show** *?case* **by** *auto*
**next**
**case**(*Match a b P*)
**have** *hnf* $([a \frown b]P)$ **by** *fact*
**hence** *False* **by** *simp*
**thus** *?case* **by** *simp*
**next**
**case**(*Mismatch a b P*)
**have** *hnf* $([a \neq b]P)$ **by** *fact*
**hence** *False* **by** *simp*
**thus** *?case* **by** *simp*
**next**
**case**(*Sum P Q*)
**have** *hnf* $(P \oplus Q)$ **by** *fact*
**hence** *Phnf*: *hnf P* **and** *Qhnf*: *hnf Q* **by** *simp+*
**have** *aineqx*: $a \neq x$ **by** *fact*

**have** *IHP*: $P \longmapsto a<\nu x> \prec P' = (<\nu x>a\{x\}.P' \in \mathit{summands}\ P)$
**proof** −
**have** $\bigwedge x\ P'.\ [\![\mathit{hnf}\ P;\ a \neq x]\!] \Longrightarrow P \longmapsto a<\nu x> \prec P' = (<\nu x>a\{x\}.P' \in \mathit{summands}\ P)$ **by** *fact*

**with** *Phnf aineqx* **show** *?thesis* **by** *simp*
**qed**

**have** *IHQ*: $Q \longmapsto a{<}\nu x{>} \prec P' = ({<}\nu x{>}a\{x\}.P' \in summands\ Q)$
**proof** −
**have** $\bigwedge x\ Q'.\ [\![hnf\ Q;\ a \neq x]\!] \Longrightarrow Q \longmapsto a{<}\nu x{>} \prec P' = ({<}\nu x{>}a\{x\}.P' \in summands\ Q)$ **by** *fact*
**with** *Qhnf aineqx* **show** *?thesis* **by** *simp*
**qed**

**from** *IHP IHQ* **have** $P \oplus Q \longmapsto a{<}\nu x{>} \prec P' \Longrightarrow {<}\nu x{>}a\{x\}.P' \in summands\ (P \oplus Q)$
**by**(*erule-tac sumCases*, *auto*)
**moreover from** *IHP IHQ* **have** ${<}\nu x{>}a\{x\}.P' \in summands\ (P \oplus Q) \Longrightarrow P \oplus Q \longmapsto a{<}\nu x{>} \prec P'$
**by**(*auto dest*: *Sum1 Sum2*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*Par P Q*)
**have** *hnf* $(P \parallel Q)$ **by** *fact*
**hence** *False* **by** *simp*
**thus** *?case* **by** *simp*
**next**
**case**(*Res y P*)
**have** *Phnf*: *hnf* $({<}\nu y{>}P)$ **by** *fact*
**then obtain** $b\ P''$ **where** *bineqy*: $b \neq y$ **and** *PeqP''*: $P = b\{y\}.P''$
**by** *auto*
**have** $y \sharp x$ **by** *fact* **hence** *xineqy*: $x \neq y$ **by** *simp*
**have** *yFreshP'*: $y \sharp P'$ **by** *fact*
**have** *aineqx*: $a \neq x$ **by** *fact*
**have** ${<}\nu y{>}P \longmapsto a{<}\nu x{>} \prec P' \Longrightarrow ({<}\nu x{>}a\{x\}.P' \in summands\ ({<}\nu y{>}P))$
**proof** −
**assume** *Trans*: ${<}\nu y{>}P \longmapsto a{<}\nu x{>} \prec P'$
**hence** *aeqb*: $a = b$ **using** *xineqy bineqy PeqP''*
**by**(*induct rule*: *resCasesB'*, *auto elim*: *outputCases simp add*: *residual.inject alpha' abs-fresh pi.inject*)

**have** *Goal*: $\bigwedge x\ P'.\ [\![{<}\nu y{>}b\{y\}.P'' \longmapsto b{<}\nu x{>} \prec P';\ x \neq y;\ x \neq b;\ x \sharp P'']\!] \Longrightarrow$
${<}\nu x{>}b\{x\}.P' \in summands({<}\nu y{>}b\{y\}.P'')$
**proof** −
**fix** $x\ P'$
**assume** *xFreshP''*: $(x{::}name) \sharp P''$ **and** *xineqb*: $x \neq b$
**assume** ${<}\nu y{>}b\{y\}.P'' \longmapsto b{<}\nu x{>} \prec P'$ **and** *xineqy*: $x \neq y$
**moreover from** $\langle x \neq b\rangle\ \langle x \sharp P''\rangle\ \langle x \neq y\rangle$ **have** $x \sharp b\{y\}.P''$ **by** *simp*
**ultimately show** ${<}\nu x{>}b\{x\}.P' \in summands\ ({<}\nu y{>}b\{y\}.P'')$
**proof**(*induct rule*: *resCasesB*)
**case**(*cOpen a P'''*)
**have** *BoundOutputS b* = *BoundOutputS a* **by** *fact* **hence** *beqa*: $b = a$ **by**

*simp*
**have** *Trans*: $b\{y\}.P'' \longmapsto a[y] \prec P'''$ **by** *fact*
**with** *PeqP''* **have** *P''eqP'''*: $P'' = P'''$
**by**(*force elim*: *outputCases simp add*: *residual.inject*)
**with** *bineqy xineqy xFreshP''* **have** $y \sharp b\{x\}.([(x, y)] \cdot P''')$
**by**(*simp add*: *name-fresh-abs name-calc name-fresh-left*)
**with** *bineqy Phnf PeqP'' P''eqP''' xineqb* **show** *?case*
**by**(*simp only*: *alphaRes*, *simp add*: *name-calc*)
**next**
**case**(*cRes P'''*)
**have** $b\{y\}.P'' \longmapsto b<\nu x> \prec P'''$ **by** *fact*
**hence** *False* **by** *auto*
**thus** *?case* **by** *simp*
**qed**
**qed**
**obtain** $z$ **where** *zineqx*: $z \neq x$ **and** *zineqy*: $z \neq y$ **and** *zFreshP''*: $z \sharp P''$
**and** *zineqb*: $z \neq b$ **and** *zFreshP'*: $z \sharp P'$
**by**(*force intro*: *name-exists-fresh*[*of* $(x, y, b, P'', P')$] *simp add*: *fresh-prod*)

**from** *zFreshP' aeqb PeqP'' Trans* **have** *Trans'*: $<\nu y>b\{y\}.P'' \longmapsto b<\nu z> \prec [(z, x)] \cdot P'$
**by**(*simp add*: *alphaBoundResidual name-swap*)
**hence** $<\nu z>b\{z\}.([(z, x)] \cdot P') \in$ *summands* $(<\nu y>b\{y\}.P'')$ **using** *zineqy zineqb zFreshP''*
**by**(*rule Goal*)
**moreover from** *bineqy zineqx zFreshP' aineqx aeqb* **have** $x \sharp b\{z\}.([(z, x)] \cdot P')$
**by**(*simp add*: *name-fresh-left name-calc*)
**ultimately have** $<\nu x>b\{x\}.P' \in$ *summands* $(<\nu y>b\{y\}.P'')$ **using** *zineqb*
**by**(*simp add*: *alphaRes name-calc*)
**with** *aeqb PeqP''* **show** *?thesis* **by** *blast*
**qed**
**moreover have** $<\nu x>a\{x\}.P' \in$ *summands*$(<\nu y>P) \Longrightarrow <\nu y>P \longmapsto a<\nu x> \prec P'$
**proof** −
**assume** $<\nu x>a\{x\}.P' \in$ *summands*$(<\nu y>P)$
**with** *PeqP''* **have** *Summ*: $<\nu x>a\{x\}.P' \in$ *summands*$(<\nu y>b\{y\}.P'')$ **by** *simp*
**moreover with** *bineqy xineqy* **have** *aeqb*: $a = b$
**by**(*auto simp add*: *if-split pi.inject name-abs-eq name-fresh-fresh*)
**from** *bineqy xineqy yFreshP'* **have** $y \sharp b\{x\}.P'$ **by**(*simp add*: *name-calc*)
**with** *Summ aeqb bineqy aineqx* **have** $<\nu y>b\{y\}.([(x, y)] \cdot P') \in$ *summands*$(<\nu y>b\{y\}.P'')$
**by**(*simp only*: *alphaRes*, *simp add*: *name-calc*)
**with** *aeqb PeqP''* **have** $<\nu y>P \longmapsto a<\nu y> \prec [(x, y)] \cdot P'$
**by**(*auto intro*: *Open Output simp add*: *if-split pi.inject name-abs-eq*)
**moreover from** *yFreshP'* **have** $x \sharp [(x, y)] \cdot P'$ **by**(*simp add*: *name-fresh-left name-calc*)
**ultimately show** *?thesis* **by**(*simp add*: *alphaBoundResidual name-swap*)

**qed**
**ultimately show** *?case* **by** *blast*
**next**
**case**$(Bang\ P)$
**have** *hnf* $(!P)$ **by** *fact*
**hence** *False* **by** *simp*
**thus** *?case* **by** *simp*
**qed**
**qed**

**definition** *expandSet* :: $pi \Rightarrow pi \Rightarrow pi\ set$ **where**
$expandSet\ P\ Q \equiv \{\tau.(P' \parallel Q) \mid P'.\ \tau.(P') \in summands\ P\} \cup$
$\{\tau.(P \parallel Q') \mid Q'.\ \tau.(Q') \in summands\ Q\} \cup$
$\{a\{b\}.(P' \parallel Q) \mid a\ b\ P'.\ a\{b\}.P' \in summands\ P\} \cup$
$\{a\{b\}.(P \parallel Q') \mid a\ b\ Q'.\ a\{b\}.Q' \in summands\ Q\} \cup$
$\{a<x>.(P' \parallel Q) \mid a\ x\ P'.\ a<x>.P' \in summands\ P \wedge x \sharp Q\} \cup$
$\{a<x>.(P \parallel Q') \mid a\ x\ Q'.\ a<x>.Q' \in summands\ Q \wedge x \sharp P\} \cup$
$\{<\nu x>a\{x\}.(P' \parallel Q) \mid a\ x\ P'.\ <\nu x>a\{x\}.P' \in summands\ P \wedge x \sharp Q\} \cup$
$\{<\nu x>a\{x\}.(P \parallel Q') \mid a\ x\ Q'.\ <\nu x>a\{x\}.Q' \in summands\ Q \wedge x \sharp P\} \cup$
$\{\tau.(P'[x::=b] \parallel Q') \mid x\ P'\ b\ Q'.\ \exists a.\ a<x>.P' \in summands\ P \wedge a\{b\}.Q' \in summands\ Q\} \cup$
$\{\tau.(P' \parallel (Q'[x::=b])) \mid b\ P'\ x\ Q'.\ \exists a.\ a\{b\}.P' \in summands\ P \wedge a<x>.Q' \in summands\ Q\} \cup$
$\{\tau.(<\nu y>(P'[x::=y] \parallel Q')) \mid x\ P'\ y\ Q'.\ \exists a.\ a<x>.P' \in summands\ P \wedge <\nu y>a\{y\}.Q' \in summands\ Q \wedge y \sharp P\} \cup$
$\{\tau.(<\nu y>(P' \parallel (Q'[x::=y]))) \mid y\ P'\ x\ Q'.\ \exists a.\ <\nu y>a\{y\}.P' \in summands\ P \wedge a<x>.Q' \in summands\ Q \wedge y \sharp Q\}$

**lemma** *finiteExpand*:
**fixes** $P :: pi$
**and** $Q :: pi$

**shows** *finite*$(expandSet\ P\ Q)$
**proof** −
**have** *finite* $\{\tau.(P' \parallel Q) \mid P'.\ \tau.(P') \in summands\ P\}$
**by**(*induct P rule*: *pi.induct*, *auto simp add*: *pi.inject Collect-ex-eq conj-disj-distribL Collect-disj-eq UN-Un-distrib*)
**moreover have** *finite* $\{\tau.(P \parallel Q') \mid Q'.\ \tau.(Q') \in summands\ Q\}$
**by**(*induct Q rule*: *pi.induct*, *auto simp add*: *pi.inject Collect-ex-eq conj-disj-distribL Collect-disj-eq UN-Un-distrib*)
**moreover have** *finite* $\{a\{b\}.(P' \parallel Q) \mid a\ b\ P'.\ a\{b\}.P' \in summands\ P\}$
**by**(*induct P rule*: *pi.induct*, *auto simp add*: *pi.inject Collect-ex-eq conj-disj-distribL Collect-disj-eq UN-Un-distrib*)
**moreover have** *finite* $\{a\{b\}.(P \parallel Q') \mid a\ b\ Q'.\ a\{b\}.Q' \in summands\ Q\}$
**by**(*induct Q rule*: *pi.induct*, *auto simp add*: *pi.inject Collect-ex-eq conj-disj-distribL*

*Collect-disj-eq UN-Un-distrib*)
**moreover have** *finite* $\{a{<}x{>}.(P' \parallel Q) \mid a\ x\ P'.\ a{<}x{>}.P' \in summands\ P \wedge x \sharp Q\}$
**proof** −
**have** *Aux*: $\bigwedge a\ x\ P\ Q.\ (x{::}name) \sharp Q \Longrightarrow \{a'{<}x'{>}.(P' \parallel Q) \mid a'\ x'\ P'.\ a'{<}x'{>}.P' = a{<}x{>}.P \wedge x' \sharp Q\} = \{a{<}x{>}.(P \parallel Q)\}$
**by**(*auto simp add*: *pi.inject name-abs-eq name-fresh-fresh*)
**thus** *?thesis*
**by**(*nominal-induct P avoiding*: *Q rule*: *pi.strong-induct*,
*auto simp add*: *Collect-ex-eq conj-disj-distribL conj-disj-distribR*
*Collect-disj-eq UN-Un-distrib*)
**qed**
**moreover have** *finite* $\{a{<}x{>}.(P \parallel Q') \mid a\ x\ Q'.\ a{<}x{>}.Q' \in summands\ Q \wedge x \sharp P\}$
**proof** −
**have** *Aux*: $\bigwedge a\ x\ P\ Q.\ (x{::}name) \sharp P \Longrightarrow \{a'{<}x'{>}.(P \parallel Q') \mid a'\ x'\ Q'.\ a'{<}x'{>}.Q' = a{<}x{>}.Q \wedge x' \sharp P\} = \{a{<}x{>}.(P \parallel Q)\}$
**by**(*auto simp add*: *pi.inject name-abs-eq name-fresh-fresh*)
**thus** *?thesis*
**by**(*nominal-induct Q avoiding*: *P rule*: *pi.strong-induct*,
*auto simp add*: *Collect-ex-eq conj-disj-distribL conj-disj-distribR*
*Collect-disj-eq UN-Un-distrib*)
**qed**
**moreover have** *finite* $\{{<}\nu x{>}a\{x\}.(P' \parallel Q) \mid a\ x\ P'.\ {<}\nu x{>}a\{x\}.P' \in summands\ P \wedge x \sharp Q\}$
**proof** −
**have** *Aux*: $\bigwedge a\ x\ P\ Q.\ [\![x \sharp Q;\ a \neq x]\!] \Longrightarrow \{{<}\nu x'{>}a'\{x'\}.(P' \parallel Q) \mid a'\ x'\ P'.\ {<}\nu x'{>}a'\{x'\}.P' = {<}\nu x{>}a\{x\}.P \wedge x' \sharp Q\} =$
$\{{<}\nu x{>}a\{x\}.(P \parallel Q)\}$
**by**(*auto simp add*: *pi.inject name-abs-eq name-fresh-fresh*)
**thus** *?thesis*
**by**(*nominal-induct P avoiding*: *Q rule*: *pi.strong-induct*,
*auto simp add*: *Collect-ex-eq conj-disj-distribL conj-disj-distribR*
*Collect-disj-eq UN-Un-distrib*)
**qed**
**moreover have** *finite* $\{{<}\nu x{>}a\{x\}.(P \parallel Q') \mid a\ x\ Q'.\ {<}\nu x{>}a\{x\}.Q' \in summands\ Q \wedge x \sharp P\}$
**proof** −
**have** *Aux*: $\bigwedge a\ x\ P\ Q.\ [\![x \sharp P;\ a \neq x]\!] \Longrightarrow \{{<}\nu x'{>}a'\{x'\}.(P \parallel Q') \mid a'\ x'\ Q'.\ {<}\nu x'{>}a'\{x'\}.Q' = {<}\nu x{>}a\{x\}.Q \wedge x' \sharp P\} =$
$\{{<}\nu x{>}a\{x\}.(P \parallel Q)\}$
**by**(*auto simp add*: *pi.inject name-abs-eq name-fresh-fresh*)
**thus** *?thesis*
**by**(*nominal-induct Q avoiding*: *P rule*: *pi.strong-induct*,
*auto simp add*: *Collect-ex-eq conj-disj-distribL conj-disj-distribR*
*Collect-disj-eq UN-Un-distrib*)
**qed**
**moreover have** *finite* $\{\tau.(P'[x{::=}b] \parallel Q') \mid x\ P'\ b\ Q'.\ \exists\, a.\ a{<}x{>}.P' \in summands\ P \wedge a\{b\}.Q' \in summands\ Q\}$

**proof** −
**have** *Aux*: ⋀a x P b Q. {τ.(P′[x′::=b′] ‖ Q′) | a′ x′ P′ b′ Q′. a′<x′>.P′ = a<x>.P ∧ a′{b′}.Q′ = a{b}.Q} = {τ.(P[x::=b] ‖ Q)}
**by**(*auto simp add*: *name-abs-eq pi.inject renaming*)
**have** ⋀a x P Q b::′a::{}. *finite* {τ.(P′[x′::=b] ‖ Q′) | a′ x′ P′ b Q′. a′<x′>.P′ = a<x>.P ∧ a′{b}.Q′ ∈ *summands* Q}
**apply**(*induct rule*: *pi.induct*, *simp-all*)
**apply**(*case-tac a=name1*)
**apply**(*simp add*: *Aux*)
**apply**(*simp add*: *pi.inject*)
**by**(*simp add*: *Collect-ex-eq conj-disj-distribL conj-disj-distribR*
*Collect-disj-eq UN-Un-distrib*)
**hence** *finite* {τ.(P′[x::=b] ‖ Q′) | a x P′ b Q′. a<x>.P′ ∈ *summands* P ∧ a{b}.Q′ ∈ *summands* Q}
**by**(*nominal-induct P avoiding*: *Q rule*: *pi.strong-induct*,
*auto simp add*: *Collect-ex-eq conj-disj-distribL conj-disj-distribR*
*Collect-disj-eq UN-Un-distrib name-abs-eq*)
**thus** *?thesis*
**apply**(*rule-tac finite-subset*)
**defer**
**by** *blast*+
**qed**
**moreover have** *finite* {τ.(P′ ‖ (Q′[x::=b])) | b P′ x Q′. ∃ a. a{b}.P′ ∈ *summands* P ∧ a<x>.Q′ ∈ *summands* Q}
**proof** −
**have** *Aux*: ⋀a x P b Q. {τ.(P′ ‖ (Q′[x′::=b′])) | a′ b′ P′ x′ Q′. a′{b′}.P′ = a{b}.P ∧ a′<x′>.Q′ = a<x>.Q} = {τ.(P ‖ (Q[x::=b]))}
**by**(*auto simp add*: *name-abs-eq pi.inject renaming*)
**have** ⋀a b P Q x::′a::{}. *finite* {τ.(P′ ‖ (Q′[x::=b′])) | a′ b′ P′ x Q′. a′{b′}.P′ = a{b}.P ∧ a′<x>.Q′ ∈ *summands* Q}
**apply**(*induct rule*: *pi.induct*, *simp-all*)
**apply**(*case-tac a=name1*)
**apply**(*simp add*: *Aux*)
**apply**(*simp add*: *pi.inject*)
**by**(*simp add*: *Collect-ex-eq conj-disj-distribL conj-disj-distribR*
*Collect-disj-eq UN-Un-distrib*)
**hence** *finite* {τ.(P′ ‖ (Q′[x::=b])) | a b P′ x Q′. a{b}.P′ ∈ *summands* P ∧ a<x>.Q′ ∈ *summands* Q}
**by**(*nominal-induct P avoiding*: *Q rule*: *pi.strong-induct*,
*auto simp add*: *Collect-ex-eq conj-disj-distribL conj-disj-distribR*
*Collect-disj-eq UN-Un-distrib name-abs-eq*)
**thus** *?thesis*
**apply**(*rule-tac finite-subset*) **defer by** *blast*+
**qed**
**moreover have** *finite* {τ.(<νy>(P′[x::=y] ‖ Q′)) | x P′ y Q′. ∃ a. a<x>.P′ ∈ *summands* P ∧ <νy>a{y}.Q′ ∈ *summands* Q ∧ y ♯ P}
**proof** −
**have** *Aux*: ⋀a x P y Q. y ♯ P ∧ y ≠ a ⟹ {τ.(<νy′>(P′[x′::=y′] ‖ Q′)) | a′ x′ P′ y′ Q′. a′<x′>.P′ = a<x>.P ∧ <νy′>a′{y′}.Q′ = <νy>a{y}.Q ∧ y′ ♯

$a{<}x{>}.P\} = \{\tau.({<}\nu y{>}(P[x{::=}y] \parallel Q))\}$
**apply**(*auto simp add*: *pi.inject name-abs-eq name-fresh-abs name-calc fresh-fact2 fresh-fact1 eqvts forget*)
**apply**(*subst name-swap*, *simp add*: *injPermSubst fresh-fact1 fresh-fact2*)+
**by**(*simp add*: *name-swap injPermSubst*)+

**have** *BC*: $\bigwedge a\ x\ P\ Q.$ *finite* $\{\tau.({<}\nu y{>}(P'[x'{::=}y] \parallel Q')) \mid a'\ x'\ P'\ y\ Q'.\ a'{<}x'{>}.P' = a{<}x{>}.P \wedge {<}\nu y{>}a'\{y\}.Q' \in \mathit{summands}\ Q \wedge y \sharp a{<}x{>}.P\}$
**proof** −
**fix** $a\ x\ P\ Q$
**show** *finite* $\{\tau.({<}\nu y{>}(P'[x'{::=}y] \parallel Q')) \mid a'\ x'\ P'\ y\ Q'.\ a'{<}x'{>}.P' = a{<}x{>}.P \wedge {<}\nu y{>}a'\{y\}.Q' \in \mathit{summands}\ Q \wedge y \sharp a{<}x{>}.P\}$
**apply**(*nominal-induct Q avoiding*: *a P rule*: *pi.strong-induct*, *simp-all*)
**apply**(*simp add*: *Collect-ex-eq conj-disj-distribL conj-disj-distribR Collect-disj-eq UN-Un-distrib*)
**apply**(*clarsimp*)
**apply**(*case-tac a=aa*)
**apply**(*insert Aux*, *auto*)
**by**(*simp add*: *pi.inject name-abs-eq name-calc*)
**qed**

**have** *IH*: $\bigwedge P\ P'\ Q.\ \{\tau.({<}\nu y{>}(P''[x{::=}y] \parallel Q')) \mid a\ x\ P''\ y\ Q'.\ (a{<}x{>}.P'' \in \mathit{summands}\ P \vee a{<}x{>}.P'' \in \mathit{summands}\ P') \wedge {<}\nu y{>}a\{y\}.Q' \in \mathit{summands}\ Q \wedge y \sharp P \wedge y \sharp P'\} = \{\tau.({<}\nu y{>}(P''[x{::=}y] \parallel Q')) \mid a\ x\ P''\ y\ Q'.\ a{<}x{>}.P'' \in \mathit{summands}\ P \wedge {<}\nu y{>}a\{y\}.Q' \in \mathit{summands}\ Q \wedge y \sharp P \wedge y \sharp P'\} \cup \{\tau.({<}\nu y{>}(P''[x{::=}y] \parallel Q')) \mid a\ x\ P''\ y\ Q'.\ a{<}x{>}.P'' \in \mathit{summands}\ P' \wedge {<}\nu y{>}a\{y\}.Q' \in \mathit{summands}\ Q \wedge y \sharp P \wedge y \sharp P'\}$
**by** *blast*
**have** *IH'*: $\bigwedge P\ Q\ P'.\ \{\tau.({<}\nu y{>}(P''[x{::=}y] \parallel Q')) \mid a\ x\ P''\ y\ Q'.\ a{<}x{>}.P'' \in \mathit{summands}\ P \wedge {<}\nu y{>}a\{y\}.Q' \in \mathit{summands}\ Q \wedge y \sharp P \wedge y \sharp P'\} \subseteq \{\tau.({<}\nu y{>}(P''[x{::=}y] \parallel Q')) \mid a\ x\ P''\ y\ Q'.\ a{<}x{>}.P'' \in \mathit{summands}\ P \wedge {<}\nu y{>}a\{y\}.Q' \in \mathit{summands}\ Q \wedge y \sharp P\}$
**by** *blast*
**have** *IH''*: $\bigwedge P\ Q\ P'.\ \{\tau.({<}\nu y{>}(P''[x{::=}y] \parallel Q')) \mid a\ x\ P''\ y\ Q'.\ a{<}x{>}.P'' \in \mathit{summands}\ P' \wedge {<}\nu y{>}a\{y\}.Q' \in \mathit{summands}\ Q \wedge y \sharp P \wedge y \sharp P'\} \subseteq \{\tau.({<}\nu y{>}(P''[x{::=}y] \parallel Q')) \mid a\ x\ P''\ y\ Q'.\ a{<}x{>}.P'' \in \mathit{summands}\ P' \wedge {<}\nu y{>}a\{y\}.Q' \in \mathit{summands}\ Q \wedge y \sharp P'\}$
**by** *blast*
**have** *finite* $\{\tau.({<}\nu y{>}(P'[x{::=}y] \parallel Q')) \mid a\ x\ P'\ y\ Q'.\ a{<}x{>}.P' \in \mathit{summands}\ P \wedge {<}\nu y{>}a\{y\}.Q' \in \mathit{summands}\ Q \wedge y \sharp P\}$
**apply**(*nominal-induct P avoiding*: *Q rule*: *pi.strong-induct*, *simp-all*)
**apply**(*insert BC*, *force*)
**apply**(*insert IH*, *auto*)
**apply**(*blast intro*: *finite-subset*[*OF IH'*])
**by**(*blast intro*: *finite-subset*[*OF IH''*])
**thus** *?thesis*
**apply**(*rule-tac finite-subset*) **defer by**(*blast*)+
**qed**
**moreover have** *finite* $\{\tau.({<}\nu y{>}(P' \parallel (Q'[x{::=}y]))) \mid y\ P'\ x\ Q'.\ \exists a.\ {<}\nu y{>}a\{y\}.P'$

∈ *summands P* ∧ *a*<*x*>.*Q'* ∈ *summands Q* ∧ *y* ♯ *Q*}
**proof** −
**have** *Aux*: ⋀*a y P x Q*. ⟦*y* ♯ *Q*; *y* ≠ *a*⟧ ⟹ {τ.(<ν*y'*>(*P'* ∥ (*Q'*[*x'*::=*y'*]))) | *a' y' P' x' Q'*. <ν*y'*>*a'*{*y'*}.*P'* = <ν*y*>*a*{*y*}.*P* ∧ *a'*<*x'*>.*Q'* = *a*<*x*>.*Q* ∧ *y'* ♯ *a*<*x*>.*Q*} = {τ.(<ν*y*>(*P* ∥ (*Q*[*x*::=*y*])))}
**apply**(*auto simp add*: *pi.inject name-abs-eq name-fresh-abs name-calc fresh-fact2 fresh-fact1 forget eqvts fresh-left renaming*[*symmetric*])
**apply**(*subst name-swap*, *simp add*: *injPermSubst fresh-fact1 fresh-fact2*)+
**by**(*simp add*: *name-swap injPermSubst*)+

**have** *IH*: ⋀*P y a Q Q'*. {τ.(<ν*y'*>(*P'* ∥ (*Q''*[*x*::=*y'*]))) | *a' y' P' x Q''*. <ν*y'*>*a'*{*y'*}.*P'* = <ν*y*>*a*{*y*}.*P* ∧ (*a'*<*x*>.*Q''* ∈ *summands Q* ∨ *a'*<*x*>.*Q''* ∈ *summands Q'*) ∧ *y'* ♯ *Q* ∧ *y'* ♯ *Q'*} = {τ.(<ν*y'*>(*P'* ∥ (*Q''*[*x*::=*y'*]))) | *a' y' P' x Q''*. <ν*y'*>*a'*{*y'*}.*P'* = <ν*y*>*a*{*y*}.*P* ∧ *a'*<*x*>.*Q''* ∈ *summands Q* ∧ *y'* ♯ *Q* ∧ *y'* ♯ *Q'*} ∪ {τ.(<ν*y'*>(*P'* ∥ (*Q''*[*x*::=*y'*]))) | *a' y' P' x Q''*. <ν*y'*>*a'*{*y'*}.*P'* = <ν*y*>*a*{*y*}.*P* ∧ *a'*<*x*>.*Q''* ∈ *summands Q'* ∧ *y'* ♯ *Q* ∧ *y'* ♯ *Q'*}
**by** *blast*
**have** *IH'*: ⋀*a y P Q Q'*. {τ.(<ν*y'*>(*P'* ∥ (*Q''*[*x*::=*y'*]))) | *a' y' P' x Q''*. <ν*y'*>*a'*{*y'*}.*P'* = <ν*y*>*a*{*y*}.*P* ∧ *a'*<*x*>.*Q''* ∈ *summands Q* ∧ *y'* ♯ *Q* ∧ *y'* ♯ *Q'*} ⊆ {τ.(<ν*y'*>(*P'* ∥ (*Q''*[*x*::=*y'*]))) | *a' y' P' x Q''*. <ν*y'*>*a'*{*y'*}.*P'* = <ν*y*>*a*{*y*}.*P* ∧ *a'*<*x*>.*Q''* ∈ *summands Q* ∧ *y'* ♯ *Q*}
**by** *blast*
**have** *IH''*: ⋀*a y P Q Q'*. {τ.(<ν*y'*>(*P'* ∥ (*Q''*[*x*::=*y'*]))) | *a' y' P' x Q''*. <ν*y'*>*a'*{*y'*}.*P'* = <ν*y*>*a*{*y*}.*P* ∧ *a'*<*x*>.*Q''* ∈ *summands Q'* ∧ *y'* ♯ *Q* ∧ *y'* ♯ *Q'*} ⊆ {τ.(<ν*y'*>(*P'* ∥ (*Q''*[*x*::=*y'*]))) | *a' y' P' x Q''*. <ν*y'*>*a'*{*y'*}.*P'* = <ν*y*>*a*{*y*}.*P* ∧ *a'*<*x*>.*Q''* ∈ *summands Q'* ∧ *y'* ♯ *Q'*}
**by** *blast*

**have** *BC*: ⋀*a y P Q*. ⟦*y* ♯ *Q*; *y* ≠ *a*⟧ ⟹ *finite* {τ.(<ν*y'*>(*P'* ∥ (*Q'*[*x*::=*y'*]))) | *a' y' P' x Q'*. <ν*y'*>*a'*{*y'*}.*P'* = <ν*y*>*a*{*y*}.*P* ∧ *a'*<*x*>.*Q'* ∈ *summands Q* ∧ *y'* ♯ *Q*}
**proof** −
**fix** *a y P Q*
**assume** (*y*::*name*) ♯ (*Q*::*pi*) **and** *y* ≠ *a*
**thus** *finite* {τ.(<ν*y'*>(*P'* ∥ (*Q'*[*x*::=*y'*]))) | *a' y' P' x Q'*. <ν*y'*>*a'*{*y'*}.*P'* = <ν*y*>*a*{*y*}.*P* ∧ *a'*<*x*>.*Q'* ∈ *summands Q* ∧ *y'* ♯ *Q*}
**apply**(*nominal-induct Q avoiding*: *y rule*: *pi.strong-induct*, *simp-all*)
**apply**(*case-tac a*=*name1*)
**apply** *auto*
**apply**(*subgoal-tac ya* ♯ (*pi*::*pi*))
**apply**(*insert Aux*)
**apply** *auto*
**apply**(*simp add*: *name-fresh-abs*)
**apply**(*simp add*: *pi.inject name-abs-eq name-calc*)
**apply**(*insert IH*)
**apply** *auto*
**apply**(*blast intro*: *finite-subset*[*OF IH'*])
**by**(*blast intro*: *finite-subset*[*OF IH''*])
**qed**

**have** *finite* {τ.(<νy>(P′ ∥ (Q′[x::=y]))) | a y P′ x Q′. <νy>a{y}.P′ ∈ *summands* P ∧ a<x>.Q′ ∈ *summands* Q ∧ y ♯ Q}

**apply**(*nominal-induct P avoiding*: *Q rule*: *pi.strong-induct*, *simp-all*)
**apply**(*simp add*: *Collect-ex-eq conj-disj-distribL conj-disj-distribR name-fresh-abs Collect-disj-eq UN-Un-distrib*)
**by**(*auto intro*: *BC*)
**thus** *?thesis*
**apply**(*rule-tac finite-subset*) **defer by** *blast*+
**qed**

**ultimately show** *?thesis*
**by**(*simp add*: *expandSet-def*)
**qed**

**lemma** *expandHnf*:
**fixes** *P* :: *pi*
**and** *Q* :: *pi*

**shows** ∀ *R* ∈ (*expandSet P Q*). *hnf R*
**by**(*force simp add*: *expandSet-def*)

**inductive-set** *sumComposeSet* :: (*pi* × *pi set*) *set*
**where**
*empty*: (**0**, {}) ∈ *sumComposeSet*
| *insert*: ⟦Q ∈ S; (P, S − {Q}) ∈ *sumComposeSet*⟧ ⟹ (P ⊕ Q, S) ∈ *sumComposeSet*

**lemma** *expandAction*:
**fixes** *P* :: *pi*
**and** *Q* :: *pi*
**and** *S* :: *pi set*

**assumes** (*P*, *S*) ∈ *sumComposeSet*
**and** *Q* ∈ *S*
**and** *Q* ⟼ *Rs*

**shows** *P* ⟼ *Rs*
**using** *assms*
**proof**(*induct arbitrary*: *Q rule*: *sumComposeSet.induct*)
**case** *empty*
**have** *Q* ∈ {} **by** *fact*
**hence** *False* **by** *simp*
**thus** *?case* **by** *simp*
**next**
**case**(*insert Q′ S P Q*)
**have** *QTrans*: *Q* ⟼ *Rs* **by** *fact*
**show** *?case*
**proof**(*case-tac Q* = *Q′*)

**assume** $Q = Q'$
**with** *QTrans* **show** $P \oplus Q' \longmapsto Rs$ **by**(*blast intro*: *Sum2*)
**next**
**assume** *QineqQ'*: $Q \neq Q'$
**have** *IH*: $\bigwedge Q.\ [\![Q \in S - \{Q'\};\ Q \longmapsto Rs]\!] \Longrightarrow P \longmapsto Rs$ **by** *fact*
**have** *QinS*: $Q \in S$ **by** *fact*
**with** *QineqQ'* **have** $Q \in S - \{Q'\}$ **by** *simp*
**hence** $P \longmapsto Rs$ **using** *QTrans* **by**(*rule IH*)
**thus** *?case* **by**(*rule Sum1*)
**qed**
**qed**

**lemma** *expandAction'*:
**fixes** $P$ :: *pi*
**and** $Q$ :: *pi*
**and** $R$ :: *pi*

**assumes** $(R, S) \in$ *sumComposeSet*
**and** $R \longmapsto Rs$

**shows** $\exists P \in S.\ P \longmapsto Rs$
**using** *assms*
**proof**(*induct rule*: *sumComposeSet.induct*)
**case** *empty*
**have** $\mathbf{0} \longmapsto Rs$ **by** *fact*
**hence** *False* **by** *blast*
**thus** *?case* **by** *simp*
**next**
**case**(*insert Q S P*)
**have** *QinS*: $Q \in S$ **by** *fact*
**have** $P \oplus Q \longmapsto Rs$ **by** *fact*
**thus** *?case*
**proof**(*induct rule*: *sumCases*)
**case** *cSum1*
**have** $P \longmapsto Rs$ **by** *fact*
**moreover have** $P \longmapsto Rs \Longrightarrow \exists P \in (S - \{Q\}).\ P \longmapsto Rs$ **by** *fact*
**ultimately obtain** $P$ **where** *PinS*: $P \in (S - \{Q\})$ **and** *PTrans*: $P \longmapsto Rs$
**by** *blast*
**show** *?case*
**proof**(*case-tac* $P = Q$)
**assume** $P = Q$
**with** *PTrans QinS* **show** *?case* **by** *blast*
**next**
**assume** *PineqQ*: $P \neq Q$
**from** *PinS* **have** $P \in S$ **by** *simp*
**with** *PTrans* **show** *?thesis* **by** *blast*
**qed**
**next**
**case** *cSum2*

**have** $Q \longmapsto Rs$ **by** *fact*
**with** *QinS* **show** *?case* **by** *blast*
**qed**
**qed**

**lemma** *expandTrans*:
**fixes** $P :: pi$
**and** $Q :: pi$
**and** $R :: pi$
**and** $a :: name$
**and** $b :: name$
**and** $x :: name$

**assumes** *Exp*: $(R,\ expandSet\ P\ Q) \in sumComposeSet$
**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** −
**show** $P \parallel Q \longmapsto \tau \prec P' = R \longmapsto \tau \prec P'$
**proof**(*rule iffI*)
**assume** $P \parallel Q \longmapsto\tau \prec P'$
**thus** $R \longmapsto\tau \prec P'$
**proof**(*induct rule*: *parCasesF*[*of* - - - - - $(P,\ Q)$])
**case**(*cPar1* $P'$)
**have** $P \longmapsto\tau \prec P'$ **by** *fact*
**with** *Phnf* **have** $\tau.(P') \in summands\ P$ **by**(*simp add*: *summandTransition*)
**hence** $\tau.(P' \parallel Q) \in expandSet\ P\ Q$ **by**(*auto simp add*: *expandSet-def*)
**moreover have** $\tau.(P' \parallel Q) \longmapsto\tau \prec (P' \parallel Q)$ **by**(*rule Tau*)
**ultimately show** *?case* **using** *Exp* **by**(*blast intro*: *expandAction*)
**next**
**case**(*cPar2* $Q'$)
**have** $Q \longmapsto\tau \prec Q'$ **by** *fact*
**with** *Qhnf* **have** $\tau.(Q') \in summands\ Q$ **by**(*simp add*: *summandTransition*)
**hence** $\tau.(P \parallel Q') \in expandSet\ P\ Q$ **by**(*auto simp add*: *expandSet-def*)
**moreover have** $\tau.(P \parallel Q') \longmapsto\tau \prec (P \parallel Q')$ **by**(*rule Tau*)
**ultimately show** *?case* **using** *Exp* **by**(*blast intro*: *expandAction*)
**next**
**case**(*cComm1* $P'\ Q'\ a\ b\ x$)
**have** $P \longmapsto a{<}x{>} \prec P'$ **and** $Q \longmapsto a[b] \prec Q'$ **by** *fact+*
**with** *Phnf Qhnf* **have** $a{<}x{>}.P' \in summands\ P$ **and** $a\{b\}.Q' \in summands$
$Q$ **by**(*simp add*: *summandTransition*)+
**hence** $\tau.(P'[x::=b] \parallel Q') \in expandSet\ P\ Q$ **by**(*simp add*: *expandSet-def*, *blast*)
**moreover have** $\tau.(P'[x::=b] \parallel Q') \longmapsto\tau \prec (P'[x::=b] \parallel Q')$ **by**(*rule Tau*)
**ultimately show** *?case* **using** *Exp* **by**(*blast intro*: *expandAction*)
**next**

**case**(*cComm2* $P'$ $Q'$ *a b x*)
**have** $P \longmapsto a[b] \prec P'$ **and** $Q \longmapsto a{<}x{>} \prec Q'$ **by** *fact*+
**with** *Phnf Qhnf* **have** $a\{b\}.P' \in$ *summands P* **and** $a{<}x{>}.Q' \in$ *summands Q* **by**(*simp add*: *summandTransition*)+
**hence** $\tau.(P' \parallel (Q'[x{::=}b])) \in$ *expandSet P Q* **by**(*simp add*: *expandSet-def*, *blast*)
**moreover have** $\tau.(P' \parallel (Q'[x{::=}b])) \longmapsto\tau \prec (P' \parallel (Q'[x{::=}b]))$ **by**(*rule Tau*)
**ultimately show** *?case* **using** *Exp* **by**(*blast intro*: *expandAction*)
**next**
**case**(*cClose1* $P'$ $Q'$ *a x y*)
**have** $y \sharp (P, Q)$ **by** *fact*
**hence** *yFreshP*: $y \sharp P$ **by**(*simp add*: *fresh-prod*)
**have** *PTrans*: $P \longmapsto a{<}x{>} \prec P'$ **by** *fact*
**with** *Phnf* **have** *PSumm*: $a{<}x{>}.P' \in$ *summands P* **by**(*simp add*: *summandTransition*)
**have** $Q \longmapsto a{<}\nu y{>} \prec Q'$ **by** *fact*
**moreover from** *PTrans yFreshP* **have** $y \neq a$ **by**(*force dest*: *freshBoundDerivative*)
**ultimately have** ${<}\nu y{>}a\{y\}.Q' \in$ *summands Q* **using** *Qhnf* **by**(*simp add*: *summandTransition*)
**with** *PSumm yFreshP* **have** $\tau.({<}\nu y{>}(P'[x{::=}y] \parallel Q')) \in$ *expandSet P Q*
**by**(*auto simp add*: *expandSet-def*)
**moreover have** $\tau.({<}\nu y{>}(P'[x{::=}y] \parallel Q')) \longmapsto\tau \prec {<}\nu y{>}(P'[x{::=}y] \parallel Q')$ **by**(*rule Tau*)
**ultimately show** *?case* **using** *Exp* **by**(*blast intro*: *expandAction*)
**next**
**case**(*cClose2* $P'$ $Q'$ *a x y*)
**have** $y \sharp (P, Q)$ **by** *fact*
**hence** *yFreshQ*: $y \sharp Q$ **by**(*simp add*: *fresh-prod*)
**have** *QTrans*: $Q \longmapsto a{<}x{>} \prec Q'$ **by** *fact*
**with** *Qhnf* **have** *QSumm*: $a{<}x{>}.Q' \in$ *summands Q* **by**(*simp add*: *summandTransition*)
**have** $P \longmapsto a{<}\nu y{>} \prec P'$ **by** *fact*
**moreover from** *QTrans yFreshQ* **have** $y \neq a$ **by**(*force dest*: *freshBoundDerivative*)
**ultimately have** ${<}\nu y{>}a\{y\}.P' \in$ *summands P* **using** *Phnf* **by**(*simp add*: *summandTransition*)
**with** *QSumm yFreshQ* **have** $\tau.({<}\nu y{>}(P' \parallel (Q'[x{::=}y]))) \in$ *expandSet P Q*
**by**(*simp add*: *expandSet-def*, *blast*)
**moreover have** $\tau.({<}\nu y{>}(P' \parallel (Q'[x{::=}y]))) \longmapsto\tau \prec {<}\nu y{>}(P' \parallel (Q'[x{::=}y]))$ **by**(*rule Tau*)
**ultimately show** *?case* **using** *Exp* **by**(*blast intro*: *expandAction*)
**qed**
**next**
**assume** $R \longmapsto\tau \prec P'$
**with** *Exp* **obtain** *R* **where** $R \in$ *expandSet P Q* **and** $R \longmapsto\tau \prec P'$ **by**(*blast dest*: *expandAction'*)
**thus** $P \parallel Q \longmapsto\tau \prec P'$
**proof**(*auto simp add*: *expandSet-def*)

**fix** $P''$
**assume** $\tau.(P'') \in$ *summands* $P$
**with** *Phnf* **have** $P \longmapsto\tau \prec P''$ **by**(*simp add*: *summandTransition*)
**hence** *PQTrans*: $P \parallel Q \longmapsto\tau \prec P'' \parallel Q$ **by**(*rule Par1F*)
**assume** $\tau.(P'' \parallel Q) \longmapsto\tau \prec P'$
**hence** $P' = P'' \parallel Q$ **by**(*erule-tac tauCases*, *auto simp add*: *pi.inject residual.inject*)
**with** *PQTrans* **show** *?thesis* **by** *simp*
**next**
**fix** $Q'$
**assume** $\tau.(Q') \in$ *summands* $Q$
**with** *Qhnf* **have** $Q \longmapsto\tau \prec Q'$ **by**(*simp add*: *summandTransition*)
**hence** *PQTrans*: $P \parallel Q \longmapsto\tau \prec P \parallel Q'$ **by**(*rule Par2F*)
**assume** $\tau.(P \parallel Q') \longmapsto\tau \prec P'$
**hence** $P' = P \parallel Q'$ **by**(*erule-tac tauCases*, *auto simp add*: *pi.inject residual.inject*)
**with** *PQTrans* **show** *?thesis* **by** *simp*
**next**
**fix** $a\ x\ P''\ b\ Q'$
**assume** $a$<$x$>$.P'' \in$ *summands* $P$ **and** $a\{b\}.Q' \in$ *summands* $Q$
**with** *Phnf Qhnf* **have** $P \longmapsto a$<$x$> $\prec P''$ **and** $Q \longmapsto a[b] \prec Q'$ **by**(*simp add*: *summandTransition*)+
**hence** *PQTrans*: $P \parallel Q \longmapsto\tau \prec P''[x::=b] \parallel Q'$ **by**(*rule Comm1*)
**assume** $\tau.(P''[x::=b] \parallel Q') \longmapsto\tau \prec P'$
**hence** $P' = P''[x::=b] \parallel Q'$ **by**(*erule-tac tauCases*, *auto simp add*: *pi.inject residual.inject*)
**with** *PQTrans* **show** *?thesis* **by** *simp*
**next**
**fix** $a\ b\ P''\ x\ Q'$
**assume** $a\{b\}.P'' \in$ *summands* $P$ **and** $a$<$x$>$.Q' \in$ *summands* $Q$
**with** *Phnf Qhnf* **have** $P \longmapsto a[b] \prec P''$ **and** $Q \longmapsto a$<$x$> $\prec Q'$ **by**(*simp add*: *summandTransition*)+
**hence** *PQTrans*: $P \parallel Q \longmapsto\tau \prec P'' \parallel (Q'[x::=b])$ **by**(*rule Comm2*)
**assume** $\tau.(P'' \parallel (Q'[x::=b])) \longmapsto\tau \prec P'$
**hence** $P' = P'' \parallel (Q'[x::=b])$ **by**(*erule-tac tauCases*, *auto simp add*: *pi.inject residual.inject*)
**with** *PQTrans* **show** *?thesis* **by** *simp*
**next**
**fix** $a\ x\ P''\ y\ Q'$
**assume** *yFreshP*: $(y::name) \sharp P$
**assume** $a$<$x$>$.P'' \in$ *summands* $P$
**with** *Phnf* **have** *PTrans*: $P \longmapsto a$<$x$> $\prec P''$ **by**(*simp add*: *summandTransition*)
**assume** <$\nu y$>$a\{y\}.Q' \in$ *summands* $Q$
**moreover from** *yFreshP PTrans* **have** $y \neq a$ **by**(*force dest*: *freshBoundDerivative*)
**ultimately have** $Q \longmapsto a$<$\nu y$> $\prec Q'$ **using** *Qhnf* **by**(*simp add*: *summandTransition*)
**with** *PTrans* **have** *PQTrans*: $P \parallel Q \longmapsto\tau \prec$ <$\nu y$>$(P''[x::=y] \parallel Q')$ **using**

*yFreshP* **by**(*rule Close1*)
**assume** $\tau$.(<$\nu y$>($P''[x::=y] \parallel Q'$)) $\longmapsto\tau \prec P'$
**hence** $P' =$ <$\nu y$>($P''[x::=y] \parallel Q'$) **by**(*erule-tac tauCases*, *auto simp add*: *pi.inject residual.inject*)
**with** *PQTrans* **show** *?thesis* **by** *simp*
**next**
**fix** *a y* $P''$ *x* $Q'$
**assume** *yFreshQ*: (*y*::*name*) $\sharp$ *Q*
**assume** *a*<*x*>.$Q'$ $\in$ *summands Q*
**with** *Qhnf* **have** *QTrans*: *Q* $\longmapsto$*a*<*x*> $\prec Q'$ **by**(*simp add*: *summandTransition*)
**assume** <$\nu y$>*a*{*y*}.$P''$ $\in$ *summands P*
**moreover from** *yFreshQ QTrans* **have** $y \neq a$ **by**(*force dest*: *freshBoundDerivative*)
**ultimately have** *P* $\longmapsto$*a*<$\nu y$> $\prec P''$ **using** *Phnf* **by**(*simp add*: *summandTransition*)
**hence** *PQTrans*: $P \parallel Q \longmapsto\tau \prec$ <$\nu y$>($P'' \parallel Q'[x::=y]$) **using** *QTrans yFreshQ* **by**(*rule Close2*)
**assume** $\tau$.(<$\nu y$>($P'' \parallel Q'[x::=y]$)) $\longmapsto\tau \prec P'$
**hence** $P' =$ <$\nu y$>($P'' \parallel Q'[x::=y]$) **by**(*erule-tac tauCases*, *auto simp add*: *pi.inject residual.inject*)
**with** *PQTrans* **show** *?thesis* **by** *simp*
**qed**
**qed**
**next**
**show** $P \parallel Q \longmapsto a[b] \prec P' = R \longmapsto a[b] \prec P'$
**proof**(*rule iffI*)
**assume** $P \parallel Q \longmapsto a[b] \prec P'$
**thus** $R \longmapsto a[b] \prec P'$
**proof**(*induct rule*: *parCasesF*[**where** *C*=()])
**case**(*cPar1* $P'$)
**have** $P \longmapsto a[b] \prec P'$ **by** *fact*
**with** *Phnf* **have** *a*{*b*}.$P'$ $\in$ *summands P* **by**(*simp add*: *summandTransition*)
**hence** *a*{*b*}.($P' \parallel Q$) $\in$ *expandSet P Q* **by**(*auto simp add*: *expandSet-def*)
**moreover have** *a*{*b*}.($P' \parallel Q$) $\longmapsto a[b] \prec$ ($P' \parallel Q$) **by**(*rule Output*)
**ultimately show** *?case* **using** *Exp* **by**(*blast intro*: *expandAction*)
**next**
**case**(*cPar2* $Q'$)
**have** $Q \longmapsto a[b] \prec Q'$ **by** *fact*
**with** *Qhnf* **have** *a*{*b*}.$Q'$ $\in$ *summands Q* **by**(*simp add*: *summandTransition*)
**hence** *a*{*b*}.($P \parallel Q'$) $\in$ *expandSet P Q* **by**(*simp add*: *expandSet-def*, *blast*)
**moreover have** *a*{*b*}.($P \parallel Q'$) $\longmapsto a[b] \prec$ ($P \parallel Q'$) **by**(*rule Output*)
**ultimately show** *?case* **using** *Exp* **by**(*blast intro*: *expandAction*)
**next**
**case** *cComm1*
**thus** *?case* **by** *auto*
**next**
**case** *cComm2*
**thus** *?case* **by** *auto*

**next**
**case** *cClose1*
**thus** *?case* **by** *auto*
**next**
**case** *cClose2*
**thus** *?case* **by** *auto*
**qed**
**next**
**assume** $R \longmapsto a[b] \prec P'$
**with** *Exp* **obtain** $R$ **where** $R \in$ *expandSet* $P$ $Q$ **and** $R \longmapsto a[b] \prec P'$ **by**(*blast dest*: *expandAction'*)
**thus** $P \parallel Q \longmapsto a[b] \prec P'$
**proof**(*auto simp add*: *expandSet-def*)
**fix** $a'$ $b'$ $P''$
**assume** $a'\{b'\}.P'' \in$ *summands* $P$
**with** *Phnf* **have** $P \longmapsto a'[b'] \prec P''$ **by**(*simp add*: *summandTransition*)
**hence** *PQTrans*: $P \parallel Q \longmapsto a'[b'] \prec P'' \parallel Q$ **by**(*rule Par1F*)
**assume** $a'\{b'\}.(P'' \parallel Q) \longmapsto a[b] \prec P'$
**hence** $P' = P'' \parallel Q$ **and** $a = a'$ **and** $b = b'$
**by**(*erule-tac outputCases*, *auto simp add*: *pi.inject residual.inject*)+
**with** *PQTrans* **show** *?thesis* **by** *simp*
**next**
**fix** $a'$ $b'$ $Q'$
**assume** $a'\{b'\}.Q' \in$ *summands* $Q$
**with** *Qhnf* **have** $Q \longmapsto a'[b'] \prec Q'$ **by**(*simp add*: *summandTransition*)
**hence** *PQTrans*: $P \parallel Q \longmapsto a'[b'] \prec P \parallel Q'$ **by**(*rule Par2F*)
**assume** $a'\{b'\}.(P \parallel Q') \longmapsto a[b] \prec P'$
**hence** $P' = P \parallel Q'$ **and** $a = a'$ **and** $b = b'$
**by**(*erule-tac outputCases*, *auto simp add*: *pi.inject residual.inject*)+
**with** *PQTrans* **show** *?thesis* **by** *simp*
**qed**
**qed**
**next**
**show** $P \parallel Q \longmapsto a{<}x{>} \prec P' = R \longmapsto a{<}x{>} \prec P'$
**proof**(*rule iffI*)
**{**
**fix** $x$ $P'$
**assume** $P \parallel Q \longmapsto a{<}x{>} \prec P'$ **and** $x \sharp P$ **and** $x \sharp Q$
**hence** $R \longmapsto a{<}x{>} \prec P'$
**proof**(*induct rule*: *parCasesB*)
**case**(*cPar1* $P'$)
**have** $P \longmapsto a{<}x{>} \prec P'$ **by** *fact*
**with** *Phnf* **have** $a{<}x{>}.P' \in$ *summands* $P$ **by**(*simp add*: *summandTransition*)
**moreover have** $x \sharp Q$ **by** *fact*
**ultimately have** $a{<}x{>}.(P' \parallel Q) \in$ *expandSet* $P$ $Q$ **by**(*auto simp add*: *expandSet-def*)
**moreover have** $a{<}x{>}.(P' \parallel Q) \longmapsto a{<}x{>} \prec (P' \parallel Q)$ **by**(*rule Input*)
**ultimately show** *?case* **using** *Exp* **by**(*blast intro*: *expandAction*)
**next**

**case**(*cPar2 Q′*)
**have** $Q \longmapsto a{<}x{>} \prec Q'$ **by** *fact*
**with** *Qhnf* **have** $a{<}x{>}.Q' \in$ *summands Q* **by**(*simp add*: *summandTransition*)
**moreover have** $x \sharp P$ **by** *fact*
**ultimately have** $a{<}x{>}.(P \parallel Q') \in$ *expandSet P Q* **by**(*simp add*: *expandSet-def*, *blast*)
**moreover have** $a{<}x{>}.(P \parallel Q') \longmapsto a{<}x{>} \prec (P \parallel Q')$ **by**(*rule Input*)
**ultimately show** *?case* **using** *Exp* **by**(*blast intro*: *expandAction*)
**qed**
**}**
**moreover obtain** $y$::*name* **where** $y \sharp P$ **and** $y \sharp Q$ **and** $y \sharp P'$
**by**(*generate-fresh name*) *auto*
**assume** $P \parallel Q \longmapsto a{<}x{>} \prec P'$
**with** ‹$y \sharp P'$› **have** $P \parallel Q \longmapsto a{<}y{>} \prec ([(x, y)] \cdot P')$
**by**(*simp add*: *alphaBoundResidual*)
**ultimately have** $R \longmapsto a{<}y{>} \prec ([(x, y)] \cdot P')$ **using** ‹$y \sharp P$› ‹$y \sharp Q$›
**by** *auto*
**thus** $R \longmapsto a{<}x{>} \prec P'$ **using** ‹$y \sharp P'$› **by**(*simp add*: *alphaBoundResidual*)
**next**
**assume** $R \longmapsto a{<}x{>} \prec P'$
**with** *Exp* **obtain** $R$ **where** $R \in$ *expandSet P Q* **and** $R \longmapsto a{<}x{>} \prec P'$ **by**(*blast dest*: *expandAction′*)
**thus** $P \parallel Q \longmapsto a{<}x{>} \prec P'$
**proof**(*auto simp add*: *expandSet-def*)
**fix** $a'$ $y$ $P''$
**assume** $a'{<}y{>}.P'' \in$ *summands P*
**with** *Phnf* **have** $P \longmapsto a'{<}y{>} \prec P''$ **by**(*simp add*: *summandTransition*)
**moreover assume** $y \sharp Q$
**ultimately have** *PQTrans*: $P \parallel Q \longmapsto a'{<}y{>} \prec P'' \parallel Q$ **by**(*rule Par1B*)
**assume** $a'{<}y{>}.(P'' \parallel Q) \longmapsto a{<}x{>} \prec P'$
**hence** $a{<}x{>} \prec P' = a'{<}y{>} \prec P'' \parallel Q$ **and** $a = a'$
**by**(*erule-tac inputCases′*, *auto simp add*: *pi.inject residual.inject*)+
**with** *PQTrans* **show** *?thesis* **by** *simp*
**next**
**fix** $a'$ $y$ $Q'$
**assume** $a'{<}y{>}.Q' \in$ *summands Q*
**with** *Qhnf* **have** $Q \longmapsto (a'$::*name*$){<}y{>} \prec Q'$ **by**(*simp add*: *summandTransition*)
**moreover assume** $y \sharp P$
**ultimately have** *PQTrans*: $P \parallel Q \longmapsto a'{<}y{>} \prec P \parallel Q'$ **by**(*rule Par2B*)
**assume** $a'{<}y{>}.(P \parallel Q') \longmapsto a{<}x{>} \prec P'$
**hence** $a{<}x{>} \prec P' = a'{<}y{>} \prec P \parallel Q'$ **and** $a = a'$
**by**(*erule-tac inputCases′*, *auto simp add*: *pi.inject residual.inject*)+
**with** *PQTrans* **show** *?thesis* **by** *simp*
**qed**
**qed**
**next**
**have** *Goal*: $\bigwedge P\ Q\ a\ x\ P'\ R.\ [\![(R,$ *expandSet P Q*$) \in$ *sumComposeSet*; *hnf P*; *hnf Q*; $a \neq x]\!] \Longrightarrow P \parallel Q \longmapsto a{<}\nu x{>} \prec P' = R \longmapsto a{<}\nu x{>} \prec P'$

**proof** −
**fix** $P\ Q\ a\ x\ P'\ R$
**assume** *aineqx*: $(a{::}name) \neq x$
**assume** *Exp*: $(R,\ expandSet\ P\ Q) \in sumComposeSet$
**assume** *Phnf*: *hnf* $P$
**assume** *Qhnf*: *hnf* $Q$
**show** $P \parallel Q \longmapsto a{<}\nu x{>} \prec P' = R \longmapsto a{<}\nu x{>} \prec P'$
**proof**(*rule iffI*)
**{**
**fix** $x\ P'$
**assume** $P \parallel Q \longmapsto a{<}\nu x{>} \prec P'$ **and** $x \sharp P$ **and** $x \sharp Q$ **and** $a \neq x$
**hence** $R \longmapsto a{<}\nu x{>} \prec P'$
**proof**(*induct rule*: *parCasesB*)
**case**(*cPar1* $P'$)
**have** $P \longmapsto a{<}\nu x{>} \prec P'$ **by** *fact*
**with** *Phnf* ‹$a \neq x$› **have** $<\nu x>a\{x\}.P' \in summands\ P$ **by**(*simp add*: *summandTransition*)
**moreover have** $x \sharp Q$ **by** *fact*
**ultimately have** $<\nu x>a\{x\}.(P' \parallel Q) \in expandSet\ P\ Q$ **by**(*auto simp add*: *expandSet-def*)
**moreover have** $<\nu x>a\{x\}.(P' \parallel Q) \longmapsto a{<}\nu x{>} \prec (P' \parallel Q)$ **using** ‹$a \neq x$›
**by**(*blast intro*: *Open Output*)
**ultimately show** *?case* **using** *Exp* **by**(*blast intro*: *expandAction*)
**next**
**case**(*cPar2* $Q'$)
**have** $Q \longmapsto a{<}\nu x{>} \prec Q'$ **by** *fact*
**with** *Qhnf* ‹$a \neq x$› **have** $<\nu x>a\{x\}.Q' \in summands\ Q$ **by**(*simp add*: *summandTransition*)
**moreover have** $x \sharp P$ **by** *fact*
**ultimately have** $<\nu x>a\{x\}.(P \parallel Q') \in expandSet\ P\ Q$ **by**(*simp add*: *expandSet-def*, *blast*)
**moreover have** $<\nu x>a\{x\}.(P \parallel Q') \longmapsto a{<}\nu x{>} \prec (P \parallel Q')$ **using** ‹$a \neq x$›
**by**(*blast intro*: *Open Output*)
**ultimately show** *?case* **using** *Exp* **by**(*blast intro*: *expandAction*)
**qed**
**}**
**moreover obtain** $y{::}name$ **where** $y \sharp P$ **and** $y \sharp Q$ **and** $y \sharp P'$ **and** $y \neq a$
**by**(*generate-fresh name*) *auto*
**assume** $P \parallel Q \longmapsto a{<}\nu x{>} \prec P'$
**with** ‹$y \sharp P'$› **have** $P \parallel Q \longmapsto a{<}\nu y{>} \prec ([(x,\ y)] \cdot P')$
**by**(*simp add*: *alphaBoundResidual*)
**ultimately have** $R \longmapsto a{<}\nu y{>} \prec ([(x,\ y)] \cdot P')$ **using** ‹$y \sharp P$› ‹$y \sharp Q$› ‹$y \neq a$›
**by** *auto*
**thus** $R \longmapsto a{<}\nu x{>} \prec P'$ **using** ‹$y \sharp P'$› **by**(*simp add*: *alphaBoundResidual*)
**next**
**{**

**fix** $R\ x\ P'$
**assume** $R \longmapsto a<\nu x> \prec P'$ **and** $R \in$ *expandSet* $P\ Q$ **and** $x \sharp R$ **and** $x \sharp P$ **and** $x \sharp Q$
**hence** $P \parallel Q \longmapsto a<\nu x> \prec P'$
**proof**(*auto simp add*: *expandSet-def*)
**fix** $a'\ y\ P''$
**assume** $<\nu y>a'\{y\}.P'' \in$ *summands* $P$
**moreover hence** $a' \neq y$ **by** *auto*
**ultimately have** $P \longmapsto a'<\nu y> \prec P''$ **using** *Phnf* **by**(*simp add*: *summandTransition*)
**moreover assume** $y \sharp Q$
**ultimately have** *PQTrans*: $P \parallel Q \longmapsto a'<\nu y> \prec P'' \parallel Q$ **by**(*rule Par1B*)
**assume** *ResTrans*: $<\nu y>a'\{y\}.(P'' \parallel Q) \longmapsto a<\nu x> \prec P'$ **and** $x \sharp [y].a'\{y\}.(P'' \parallel Q)$
**with** *ResTrans* ‹$a' \neq y$› ‹$x \sharp P$› ‹$x \sharp Q$› **have** $a<\nu x> \prec P' = a'<\nu y> \prec P'' \parallel Q$
**apply**(*case-tac x=y*)
**defer**
**apply**(*erule-tac resCasesB*)
**apply** *simp*
**apply**(*simp add*: *abs-fresh*)
**apply**(*auto simp add*: *residual.inject alpha' calc-atm fresh-left abs-fresh elim*: *outputCases*)
**apply**(*ind-cases* $<\nu y>a'\{y\}.(P'' \parallel Q) \longmapsto a<\nu y> \prec P'$)
**apply**(*simp add*: *pi.inject alpha' residual.inject abs-fresh eqvts calc-atm*)
**apply**(*auto elim*: *outputCases*)
**apply**(*simp add*: *pi.inject residual.inject alpha' calc-atm*)
**apply** *auto*
**apply**(*ind-cases* $<\nu y>a'\{y\}.(P'' \parallel Q) \longmapsto a<\nu y> \prec P'$)
**apply**(*auto simp add*: *pi.inject alpha' residual.inject abs-fresh eqvts calc-atm*)
**apply**(*auto elim*: *outputCases*)
**apply**(*erule-tac outputCases*)
**apply**(*auto simp add*: *freeRes.inject*)
**apply** *hypsubst-thin*
**apply**(*drule-tac pi*=$[(b,\ y)]$ **in** *pt-bij3*)
**by** *simp*
**with** *PQTrans* **show** *?thesis* **by** *simp*
**next**
**fix** $a'\ y\ Q'$
**assume** $<\nu y>a'\{y\}.Q' \in$ *summands* $Q$
**moreover hence** $a' \neq y$ **by** *auto*
**ultimately have** $Q \longmapsto a'<\nu y> \prec Q'$ **using** *Qhnf* **by**(*simp add*: *summandTransition*)
**moreover assume** $y \sharp P$
**ultimately have** *PQTrans*: $P \parallel Q \longmapsto a'<\nu y> \prec P \parallel Q'$ **by**(*rule Par2B*)
**assume** *ResTrans*: $<\nu y>a'\{y\}.(P \parallel Q') \longmapsto a<\nu x> \prec P'$ **and** $x \sharp [y].a'\{y\}.(P \parallel Q')$
**with** *ResTrans* ‹$a' \neq y$› **have** $a<\nu x> \prec P' = a'<\nu y> \prec P \parallel Q'$

**apply**(*case-tac x=y*)
**defer**
**apply**(*erule-tac resCasesB*)
**apply** *simp*
**apply**(*simp add*: *abs-fresh*)
**apply**(*auto simp add*: *residual.inject alpha′ calc-atm fresh-left abs-fresh elim*: *outputCases*)
**apply**(*ind-cases* $<\nu y>a'\{y\}.(P \parallel Q') \longmapsto a<\nu y> \prec P'$)
**apply**(*simp add*: *pi.inject alpha′ residual.inject abs-fresh eqvts calc-atm*)
**apply**(*auto elim*: *outputCases*)
**apply**(*simp add*: *pi.inject residual.inject alpha′ calc-atm*)
**apply** *auto*
**apply**(*ind-cases* $<\nu y>a'\{y\}.(P \parallel Q') \longmapsto a<\nu y> \prec P'$)
**apply**(*auto simp add*: *pi.inject alpha′ residual.inject abs-fresh eqvts calc-atm*)
**apply**(*auto elim*: *outputCases*)
**apply**(*erule-tac outputCases*)
**apply**(*auto simp add*: *freeRes.inject*)
**apply** *hypsubst-thin*
**apply**(*drule-tac pi*=$[(b, y)]$ **in** *pt-bij3*)
**by** *simp*
**with** *PQTrans* **show** *?thesis* **by** *simp*
**qed**
**}**
**moreover assume** $R \longmapsto a<\nu x> \prec P'$
**with** *Exp* **obtain** $R$ **where** $R \in$ *expandSet* $P$ $Q$ **and** $R \longmapsto a<\nu x> \prec P'$
**apply**(*drule-tac expandAction′*) **by** *auto*
**moreover obtain** $y$::*name* **where** $y \sharp P$ **and** $y \sharp Q$ **and** $y \sharp R$ **and** $y \sharp P'$
**by**(*generate-fresh name*) *auto*
**moreover with** ‹$y \sharp P'$› ‹$R \longmapsto a<\nu x> \prec P'$› **have** $R \longmapsto a<\nu y> \prec ([(x, y)] \cdot P')$ **by**(*simp add*: *alphaBoundResidual*)
**ultimately have** $P \parallel Q \longmapsto a<\nu y> \prec ([(x, y)] \cdot P')$ **by** *auto*
**thus** $P \parallel Q \longmapsto a<\nu x> \prec P'$ **using** ‹$y \sharp P'$› **by**(*simp add*: *alphaBoundResidual*)
**qed**
**qed**

**obtain** $y$ **where** *yineqx*: $a \neq y$ **and** *yFreshP′*: $y \sharp P'$
**by**(*force intro*: *name-exists-fresh*[*of* $(a, P')$] *simp add*: *fresh-prod*)
**from** *Exp Phnf Qhnf yineqx* **have** $(P \parallel Q \longmapsto a<\nu y> \prec [(x, y)] \cdot P') = (R \longmapsto a<\nu y> \prec [(x, y)] \cdot P')$
**by**(*rule Goal*)
**moreover with** *yFreshP′* **have** $x \sharp [(x, y)] \cdot P'$ **by**(*simp add*: *name-fresh-left name-calc*)
**ultimately show** $(P \parallel Q \longmapsto a<\nu x> \prec P') = (R \longmapsto a<\nu x> \prec P')$
**by**(*simp add*: *alphaBoundResidual name-swap*)
**qed**

**lemma** *expandLeft*:
**fixes** $P$ :: *pi*

**and** $Q$ :: *pi*
**and** $R$ :: *pi*
**and** *Rel* :: (*pi* × *pi*) *set*

**assumes** *Exp*: ($R$, *expandSet* $P$ $Q$) ∈ *sumComposeSet*
**and** *Phnf*: *hnf* $P$
**and** *Qhnf*: *hnf* $Q$
**and** *Id*: *Id* ⊆ *Rel*

**shows** $P \parallel Q \rightsquigarrow[Rel]\ R$
**proof**(*induct rule*: *simCases*)
**case**(*Bound a x* $R'$)
**have** $R \longmapsto a«x» \prec R'$ **by** *fact*
**with** *Exp Phnf Qhnf* **have** $P \parallel Q \longmapsto a«x» \prec R'$ **by**(*cases a*, *auto simp add*: *expandTrans*)
**moreover from** *Id* **have** *derivative* $R'$ $R'$ *a x Rel* **by**(*cases a*, *auto simp add*: *derivative-def*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*Free* $\alpha$ $R'$)
**have** $R \longmapsto \alpha \prec R'$ **by** *fact*
**with** *Exp Phnf Qhnf* **have** $P \parallel Q \longmapsto \alpha \prec R'$ **by**(*cases* $\alpha$, *auto simp add*: *expandTrans*)
**moreover from** *Id* **have** ($R'$, $R'$) ∈ *Rel* **by** *blast*
**ultimately show** *?case* **by** *blast*
**qed**

**lemma** *expandRight*:
**fixes** $P$ :: *pi*
**and** $Q$ :: *pi*
**and** $R$ :: *pi*
**and** *Rel* :: (*pi* × *pi*) *set*

**assumes** *Exp*: ($R$, *expandSet* $P$ $Q$) ∈ *sumComposeSet*
**and** *Phnf*: *hnf* $P$
**and** *Qhnf*: *hnf* $Q$
**and** *Id*: *Id* ⊆ *Rel*

**shows** $R \rightsquigarrow[Rel]\ P \parallel Q$
**proof**(*induct rule*: *simCases*)
**case**(*Bound a x* $R'$)
**have** $P \parallel Q \longmapsto a«x» \prec R'$ **by** *fact*
**with** *Exp Phnf Qhnf* **have** $R \longmapsto a«x» \prec R'$ **by**(*cases a*, *auto simp add*: *expandTrans*)
**moreover from** *Id* **have** *derivative* $R'$ $R'$ *a x Rel* **by**(*cases a*, *auto simp add*: *derivative-def*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*Free* $\alpha$ $R'$)

**have** $P \parallel Q \longmapsto\alpha \prec R'$ **by** *fact*
**with** *Exp Phnf Qhnf* **have** $R \longmapsto\alpha \prec R'$ **by**(*cases* $\alpha$, *auto simp add*: *expandTrans*)
**moreover from** *Id* **have** $(R', R') \in$ *Rel* **by** *blast*
**ultimately show** *?case* **by** *blast*
**qed**

**lemma** *expandSC*:
**fixes** $P$ :: *pi*
**and** $Q$ :: *pi*
**and** $R$ :: *pi*

**assumes** $(R,$ *expandSet* $P\ Q) \in$ *sumComposeSet*
**and** *hnf* $P$
**and** *hnf* $Q$

**shows** $P \parallel Q \sim R$
**proof** −
**let** $?X = \{(P \parallel Q, R) \mid P\ Q\ R.\ (R,$ *expandSet* $P\ Q) \in$ *sumComposeSet* $\wedge$ *hnf* $P$ $\wedge$ *hnf* $Q\} \cup \{(R, P \parallel Q) \mid P\ Q\ R.\ (R,$ *expandSet* $P\ Q) \in$ *sumComposeSet* $\wedge$ *hnf* $P$ $\wedge$ *hnf* $Q\}$
**from** *assms* **have** $(P \parallel Q, R) \in\ ?X$ **by** *auto*
**thus** *?thesis*
**proof**(*coinduct rule*: *bisimCoinduct*)
**case**(*cSim P Q*)
**thus** *?case*
**by**(*blast intro*: *reflexive expandLeft expandRight*)
**next**
**case**(*cSym P Q*)
**thus** *?case* **by** *auto*
**qed**
**qed**

**end**

**theory** *Strong-Late-Axiomatisation*
**imports** *Strong-Late-Expansion-Law*
**begin**

**lemma** *inputSuppPres*:
**fixes** $P$ :: *pi*
**and** $Q$ :: *pi*
**and** $x$ :: *name*
**and** *Rel* :: $(pi \times pi)$ *set*

**assumes** *PRelQ*: $\bigwedge y.\ y \in supp(P, Q, x) \Longrightarrow (P[x::=y], Q[x::=y]) \in Rel$
**and** *Eqvt*: *eqvt Rel*

**shows** $a{<}x{>}.P \rightsquigarrow[Rel]\ a{<}x{>}.Q$
**proof** −

```
  from Eqvt show ?thesis
  proof(induct rule: simCasesCont[where C=(x, a, Q, P)])
    case(Bound b y Q′)
    have x ∈ supp(P, Q, x) by(simp add: supp-prod supp-atm)
    with PRelQ have (P, Q) ∈ Rel by fastforce
    have QTrans: a<x>.Q ⟼ b«y» ≺ Q′ by fact
    have y ♯ (x, a, Q, P) by fact
   hence y ≠ a and yineqx: y ≠ x and y ♯ Q and y ♯ P by(simp add: fresh-prod)+
    with QTrans show ?case
    proof(induct rule: inputCases)
      have a<y>.([(x, y)] • P) ⟼a<y> ≺ ([(x, y)] • P) by(rule Input)
        hence a<x>.P ⟼a<y> ≺ ([(x, y)] • P) using ‹y ♯ P› by(simp add:
alphaInput)
      moreover have derivative ([(x, y)] • P) ([(x, y)] • Q) (InputS a) y Rel
      proof(auto simp add: derivative-def)
        fix u
        have x ∈ supp(P, Q, x) by(simp add: supp-prod supp-atm)
        have (P[x::=u], Q[x::=u]) ∈ Rel
        proof(cases u ∈ supp(P, Q, x))
          case True
          with PRelQ show ?thesis by auto
        next
          case False
          hence u ♯ P and u ♯ Q by(auto simp add: fresh-def supp-prod)
           moreover from ‹eqvt Rel› ‹(P, Q) ∈ Rel› have ([(x, u)] • P, [(x, u)] •
Q) ∈ Rel
            by(rule eqvtRelI)
          ultimately show ?thesis by(simp only: injPermSubst)
        qed
         with ‹y ♯ P› ‹y ♯ Q› show (([(x, y)] • P)[y::=u], ([(x, y)] • Q)[y::=u]) ∈
Rel
          by(simp add: renaming)
      qed
      ultimately show ∃P′. a<x>.P ⟼ a<y> ≺ P′ ∧ derivative P′ ([(x, y)] •
Q) (InputS a) y Rel
        by blast
    qed
  next
    case(Free α Q′)
    have a<x>.Q ⟼ α ≺ Q′ by fact
    hence False by auto
    thus ?case by blast
  qed
qed

lemma inputSuppPresBisim:
  fixes P :: pi
  and   Q :: pi
  and   x :: name
```

```
  assumes PSimQ: ⋀y. y ∈ supp(P, Q, x) ⟹ P[x::=y] ∼ Q[x::=y]

  shows a<x>.P ∼ a<x>.Q
proof −
  let ?X = {(a<x>.P, a<x>.Q) | a x P Q. ∀ y ∈ supp(P, Q, x). P[x::=y] ∼
Q[x::=y]}
  have eqvt ?X
    apply(auto simp add: eqvt-def)
    apply(rule-tac x=perm · aa in exI)
    apply(rule-tac x=perm · x in exI)
    apply(rule-tac x=perm · P in exI)
    apply auto
    apply(rule-tac x=perm · Q in exI)
    apply auto
   apply(drule-tac pi=rev perm in pt-set-bij2[OF pt-name-inst, OF at-name-inst])
    apply(simp add: eqvts pt-rev-pi[OF pt-name-inst, OF at-name-inst])
    apply(erule-tac x=rev perm · y in ballE)
    apply auto
    apply(drule-tac p=perm in bisimClosed)
    by(simp add: eqvts pt-pi-rev[OF pt-name-inst, OF at-name-inst])
  from assms have (a<x>.P, a<x>.Q) ∈ ?X by fastforce
  thus ?thesis
  proof(coinduct rule: bisimCoinduct)
    case(cSim P Q)
    thus ?case using ‹eqvt ?X›
      by(fastforce intro: inputSuppPres)
  next
    case(cSym P Q)
    thus ?case by(fastforce simp add: supp-prod dest: symmetric)
  qed
qed

inductive equiv :: pi ⇒ pi ⇒ bool (infixr ‹≡e› 80)
where
  Refl:          P ≡e P
| Sym:           P ≡e Q ⟹ Q ≡e P
| Trans:         ⟦P ≡e Q; Q ≡e R⟧ ⟹ P ≡e R

| Match:         [a⌢a]P ≡e P
| Match':        a ≠ b ⟹ [a⌢b]P ≡e 0

| Mismatch:      a ≠ b ⟹ [a≠b]P ≡e P
| Mismatch':     [a≠a]P ≡e 0

| SumSym:        P ⊕ Q ≡e Q ⊕ P
| SumAssoc:      (P ⊕ Q) ⊕ R ≡e P ⊕ (Q ⊕ R)
| SumZero:       P ⊕ 0 ≡e P
| SumIdemp:      P ⊕ P ≡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** −
  **have** $P \equiv_e P \oplus P$ **by**(*blast intro*: *Sym SumIdemp*)
  **moreover from** *assms* **have** $P \oplus P \equiv_e P' \oplus P$ **by**(*rule SumPres*)
  **moreover have** $P' \oplus P \equiv_e P \oplus P'$ **by**(*rule SumSym*)
  **ultimately have** $P \equiv_e P \oplus P'$ **by**(*blast intro*: *Trans*)
  **thus** *?thesis* **by**(*rule Sym*)
**qed**

**lemma** *SumPres'*:
  **fixes** $P$ :: *pi*
  **and** $P'$ :: *pi*
  **and** $Q$ :: *pi*
  **and** $Q'$ :: *pi*

  **assumes** *PeqP'*: $P \equiv_e P'$
  **and** *QeqQ'*: $Q \equiv_e Q'$

  **shows** $P \oplus Q \equiv_e P' \oplus Q'$
**proof** −
  **from** *PeqP'* **have** $P \oplus Q \equiv_e P' \oplus Q$ **by**(*rule SumPres*)

**moreover have** $P' \oplus Q \equiv_e Q \oplus P'$ **by**(*rule SumSym*)
**moreover from** *QeqQ'* **have** $Q \oplus P' \equiv_e Q' \oplus P'$ **by**(*rule SumPres*)
**moreover have** $Q' \oplus P' \equiv_e P' \oplus Q'$ **by**(*rule SumSym*)
**ultimately show** *?thesis* **by**(*blast intro*: *Trans*)
**qed**

**lemma** *sound*:
**fixes** $P :: pi$
**and** $Q :: pi$

**assumes** $P \equiv_e Q$

**shows** $P \sim Q$
**using** *assms*
**proof**(*induct*)
**case**(*Refl P*)
**show** *?case* **by**(*rule reflexive*)
**next**
**case**(*Sym P Q*)
**have** $P \sim Q$ **by** *fact*
**thus** *?case* **by**(*rule symmetric*)
**next**
**case**(*Trans P Q R*)
**have** $P \sim Q$ **and** $Q \sim R$ **by** *fact+*
**thus** *?case* **by**(*rule transitive*)
**next**
**case**(*Match a P*)
**show** *?case* **by**(*rule matchId*)
**next**
**case**(*Match' a b P*)
**have** $a \neq b$ **by** *fact*
**thus** *?case* **by**(*rule matchNil*)
**next**
**case**(*Mismatch a b P*)
**have** $a \neq b$ **by** *fact*
**thus** *?case* **by**(*rule mismatchId*)
**next**
**case**(*Mismatch' a P*)
**show** *?case* **by**(*rule mismatchNil*)
**next**
**case**(*SumSym P Q*)
**show** *?case* **by**(*rule sumSym*)
**next**
**case**(*SumAssoc P Q R*)
**show** *?case* **by**(*rule sumAssoc*)
**next**
**case**(*SumZero P*)
**show** *?case* **by**(*rule sumZero*)
**next**

**case**(*SumIdemp P*)
**show** *?case* **by**(*rule sumIdemp*)
**next**
**case**(*SumRes x P Q*)
**show** *?case* **by**(*rule sumRes*)
**next**
**case**(*ResNil x*)
**show** *?case* **by**(*rule nilRes*)
**next**
**case**(*ResInput x a y P*)
**have** $x \neq a$ **and** $x \neq y$ **by** *fact+*
**thus** *?case* **by**(*rule resInput*)
**next**
**case**(*ResInput' x y P*)
**show** *?case* **by**(*rule resNil*)
**next**
**case**(*ResOutput x a b P*)
**have** $x \neq a$ **and** $x \neq b$ **by** *fact+*
**thus** *?case* **by**(*rule resOutput*)
**next**
**case**(*ResOutput' x b P*)
**show** *?case* **by**(*rule resNil*)
**next**
**case**(*ResTau x P*)
**show** *?case* **by**(*rule resTau*)
**next**
**case**(*ResComm x P*)
**show** *?case* **by**(*rule resComm*)
**next**
**case**(*ResFresh x P*)
**have** $x \sharp P$ **by** *fact*
**thus** *?case* **by**(*rule scopeFresh*)
**next**
**case**(*Expand R P Q*)
**have** (*R*, *expandSet P Q*) $\in$ *sumComposeSet* **and** *hnf P* **and** *hnf Q* **by** *fact+*
**thus** *?case* **by**(*rule expandSC*)
**next**
**case**(*SumPres P Q R*)
**from** $\langle P \sim Q \rangle$ **show** *?case* **by**(*rule sumPres*)
**next**
**case**(*ParPres P P' Q Q'*)
**have** $P \sim P'$ **and** $Q \sim Q'$ **by** *fact+*
**thus** *?case* **by**(*metis transitive symmetric parPres parSym*)
**next**
**case**(*ResPres P Q x*)
**from** $\langle P \sim Q \rangle$ **show** *?case* **by**(*rule resPres*)
**next**
**case**(*TauPres P Q*)
**from** $\langle P \sim Q \rangle$ **show** *?case* **by**(*rule tauPres*)

**next**
  **case**(*OutputPres P Q a b*)
  **from** ‹$P \sim Q$› **show** *?case* **by**(*rule outputPres*)
**next**
  **case**(*InputPres P Q x a*)
  **have** $\forall\, y \in supp(P,\ Q,\ x).\ P[x{::=}y] \equiv_e Q[x{::=}y] \wedge P[x{::=}y] \sim Q[x{::=}y]$ **by** *fact*
  **hence** $\forall\, y \in supp(P,\ Q,\ x).\ P[x{::=}y] \sim Q[x{::=}y]$ **by** *blast*
  **thus** *?case* **by**(*rule-tac inputSuppPresBisim*) *auto*
**qed**

**lemma** *zeroDest*[*dest*]:
  **fixes** $a$ :: *name*
  **and** $b$ :: *name*
  **and** $x$ :: *name*
  **and** $P$ :: *pi*

  **shows** $(a\{b\}.P) \equiv_e \mathbf{0} \Longrightarrow False$
  **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** $\mathbf{0} \equiv_e \tau.(P) \Longrightarrow False$
**by**(*auto dest*: *sound*)

**lemma** *eq-eqvt*:
  **fixes** *pi*::*name prm*
  **and** $x$::*'a*::*pt-name*
  **shows** $pi{\cdot}(x{=}y) = ((pi{\cdot}x){=}(pi{\cdot}y))$
**by**(*simp add*: *perm-bool perm-bij*)

**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* $(!P) = (depth\ P)$
**apply**(*auto simp add*: *fresh-nat*)
**apply**(*finite-guess*)+
**by**(*fresh-guess*)+

**lemma** *depthEqvt*[*simp*]:
  **fixes** $P$ :: *pi*
  **and** $p$ :: *name prm*

**shows** *depth*(*p* • *P*) = *depth P*
**by**(*nominal-induct P rule*: *pi.strong-induct*, *auto simp add*: *name-bij*)

**lemma** *depthInput*[*simp*]:
**fixes** *a* :: *name*
**and** *x* :: *name*
**and** *P* :: *pi*

**shows** *depth* (*a*<*x*>.*P*) = *1* + (*depth P*)
**proof** −
**obtain** *y* **where** *yineqa*: *y* ≠ *a* **and** *yFreshP*: *y* ♯ *P*
**by**(*force intro*: *name-exists-fresh*[*of* (*a*, *P*)] *simp add*: *fresh-prod*)
**from** *yFreshP* **have** *a*<*x*>.*P* = *a*<*y*>.([(*x*, *y*)] • *P*) **by**(*simp add*: *alphaInput*)
**with** *yineqa* **show** *?thesis* **by** *simp*
**qed**

**nominal-primrec** *valid* :: *pi* ⇒ *bool* **where**
*valid* **0** = *True*
| *valid* (τ.(*P*)) = *valid P*
| *x* ♯ *a* ⟹ *valid* (*a*<*x*>.*P*) = *valid P*
| *valid* (*a*{*b*}.*P*) = *valid P*
| *valid* ([*a*⌢*b*]*P*) = *valid P*
| *valid* ([*a*≠*b*]*P*) = *valid P*
| *valid* (*P* ⊕ *Q*) = ((*valid P*) ∧ (*valid Q*))
| *valid* (*P* ∥ *Q*) = ((*valid P*) ∧ (*valid Q*))
| *valid* (<ν*x*>*P*) = *valid P*
| *valid* (!*P*) = *False*
**apply**(*auto simp add*: *fresh-bool*)
**apply**(*finite-guess*)+
**by**(*fresh-guess*)+

**lemma** *validEqvt*[*simp*]:
**fixes** *P* :: *pi*
**and** *p* :: *name prm*

**shows** *valid*(*p* • *P*) = *valid P*
**by**(*nominal-induct P rule*: *pi.strong-induct*, *auto simp add*: *name-bij*)

**lemma** *validInput*[*simp*]:
**fixes** *a* :: *name*
**and** *x* :: *name*
**and** *P* :: *pi*

**shows** *valid* (*a*<*x*>.*P*) = *valid P*
**proof** −
**obtain** *y* **where** *yineqa*: *y* ≠ *a* **and** *yFreshP*: *y* ♯ *P*
**by**(*force intro*: *name-exists-fresh*[*of* (*a*, *P*)] *simp add*: *fresh-prod*)
**from** *yFreshP* **have** *a*<*x*>.*P* = *a*<*y*>.([(*x*, *y*)] • *P*) **by**(*simp add*: *alphaInput*)

**with** *yineqa* **show** *?thesis* **by** *simp*
**qed**

**lemma** *depthMin*[*intro*]:
**fixes** $P$

**shows** $0 \leq depth\ P$
**by**(*induct P rule*: *pi.induct*, *auto*)

**lemma** *hnfTransition*:
**fixes** $P :: pi$

**assumes** *hnf P*
**and** $P \neq \mathbf{0}$

**shows** $\exists Rs.\ P \longmapsto Rs$
**using** *assms*
**by**(*induct rule*: *pi.induct*, *auto intro*: *Output Tau Input Sum1 Open*)

**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** $P \oplus Q' \equiv_e P$
**using** *assms*
**proof**(*nominal-induct P arbitrary*: *Q rule*: *pi.strong-inducts*)
**case**(*PiNil Q*)
**have** $Q \in summands\ \mathbf{0}$ **by** *fact*
**hence** *False* **by** *simp*
**thus** *?case* **by** *simp*
**next**
**case**(*Output a b P Q*)
**have** $Q \equiv_e Q'$ **by** *fact*
**hence** $a\{b\}.P \oplus Q' \equiv_e a\{b\}.P \oplus Q$ **by**(*blast intro*: *SumPres' Refl Sym*)
**moreover have** $Q = a\{b\}.P$
**proof** −
**have** $Q \in summands\ (a\{b\}.P)$ **by** *fact*
**thus** *?thesis* **by** *simp*
**qed**
**ultimately show** *?case* **by**(*blast intro*: *SumIdemp Trans*)
**next**
**case**(*Tau P Q*)

**have** $Q \equiv_e Q'$ **by** *fact*
**hence** $\tau.(P) \oplus Q' \equiv_e \tau.(P) \oplus Q$ **by**(*blast intro*: *SumPres' Refl Sym*)
**moreover have** $Q = \tau.(P)$
**proof** −
**have** $Q \in$ *summands* $(\tau.(P))$ **by** *fact*
**thus** *?thesis* **by** *simp*
**qed**
**ultimately show** *?case* **by**(*blast intro*: *SumIdemp Trans*)
**next**
**case**(*Input a x P Q*)
**have** $Q \equiv_e Q'$ **by** *fact*
**hence** $a{<}x{>}.P \oplus Q' \equiv_e a{<}x{>}.P \oplus Q$ **by**(*blast intro*: *SumPres' Refl Sym*)
**moreover have** $Q = a{<}x{>}.P$
**proof** −
**have** $Q \in$ *summands* $(a{<}x{>}.P)$ **by** *fact*
**thus** *?thesis* **by** *simp*
**qed**
**ultimately show** *?case* **by**(*blast intro*: *SumIdemp Trans*)
**next**
**case**(*Match a b P Q*)
**have** $Q \in$ *summands* $([a \frown b]P)$ **by** *fact*
**hence** *False* **by** *simp*
**thus** *?case* **by** *simp*
**next**
**case**(*Mismatch a b P Q*)
**have** $Q \in$ *summands* $([a \neq b]P)$ **by** *fact*
**hence** *False* **by** *simp*
**thus** *?case* **by** *simp*
**next**
**case**(*Sum P Q R*)
**have** *IHP*: $\bigwedge P'.\ [\![P' \in \textit{summands}\ P;\ P' \equiv_e Q']\!] \Longrightarrow P \oplus Q' \equiv_e P$ **by** *fact*
**have** *IHQ*: $\bigwedge Q''.\ [\![Q'' \in \textit{summands}\ Q;\ Q'' \equiv_e Q']\!] \Longrightarrow Q \oplus Q' \equiv_e Q$ **by** *fact*
**have** *ReqQ'*: $R \equiv_e Q'$ **by** *fact*
**have** $R \in$ *summands*$(P \oplus Q)$ **by** *fact*
**hence** $R \in$ *summands* $P \vee R \in$ *summands* $Q$ **by** *simp*
**thus** *?case*
**proof**(*rule disjE*)
**assume** $R \in$ *summands* $P$
**hence** *PQ'eqP*: $P \oplus Q' \equiv_e P$ **using** *ReqQ'* **by**(*rule IHP*)

**have** $(P \oplus Q) \oplus Q' \equiv_e P \oplus (Q \oplus Q')$ **by**(*rule SumAssoc*)
**moreover have** $P \oplus (Q \oplus Q') \equiv_e P \oplus (Q' \oplus Q)$ **by**(*blast intro*: *Refl SumSym SumPres'*)
**moreover have** $P \oplus (Q' \oplus Q) \equiv_e (P \oplus Q') \oplus Q$ **by**(*blast intro*: *SumAssoc Sym*)
**moreover from** *PQ'eqP* **have** $(P \oplus Q') \oplus Q \equiv_e P \oplus Q$ **by**(*blast intro*: *SumPres' Refl*)
**ultimately show** *?case* **by**(*blast intro*: *Trans*)
**next**

```
    assume R ∈ summands Q
    hence QQ'eqQ: Q ⊕ Q' ≡e Q using ReqQ' by(rule IHQ)

    have (P ⊕ Q) ⊕ Q' ≡e P ⊕ (Q ⊕ Q') by(rule SumAssoc)
    moreover from QQ'eqQ have P ⊕ (Q ⊕ Q') ≡e P ⊕ Q by(blast intro: Refl
SumPres')
    ultimately show ?case by(rule Trans)
  qed
next
  case(Par P Q R)
  have R ∈ summands (P ∥ Q) by fact
  hence False by simp
  thus ?case by simp
next
  case(Res x P Q)
  have Q ≡e Q' by fact
  hence (<νx>P) ⊕ Q' ≡e (<νx>P) ⊕ Q by(blast intro: SumPres' Refl Sym)
  moreover have Q = <νx>P
  proof -
    have Q ∈ summands (<νx>P) by fact
    thus ?thesis by(auto simp add: if-split)
  qed
  ultimately show ?case by(blast intro: SumIdemp Trans)
next
  case(Bang P Q)
  have Q ∈ summands(!P) by fact
  hence False by simp
  thus ?case by simp
qed

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 ∃R. uhnf R ∧ valid R ∧ P ⊕ Q ≡e R ∧ (depth R) ≤ (depth (P ⊕ Q))
using assms
proof(nominal-induct P arbitrary: Q rule: pi.strong-inducts)
  case(PiNil Q)
  have uhnf Q by fact
  moreover have valid Q by fact
  moreover have 0 ⊕ Q ≡e Q by(blast intro: SumZero SumSym Trans)
  moreover have depth Q ≤ depth(0 ⊕ Q) by auto
  ultimately show ?case by blast
next
```

**case**(*Output a b P Q*)
**show** *?case*
**proof**(*case-tac* $Q = \mathbf{0}$)
  **assume** $Q = \mathbf{0}$
  **moreover have** *uhnf* $(a\{b\}.P)$ **by**(*simp add*: *uhnf-def*)
  **moreover have** *valid* $(a\{b\}.P)$ **by** *fact*
  **moreover have** $a\{b\}.P \oplus \mathbf{0} \equiv_e a\{b\}.P$ **by**(*rule SumZero*)
  **moreover have** $depth(a\{b\}.P) \leq depth(a\{b\}.P \oplus \mathbf{0})$ **by** *simp*
  **ultimately show** *?case* **by** *blast*
**next**
  **assume** *QineqNil*: $Q \neq \mathbf{0}$
  **have** *Qhnf*: *uhnf Q* **by** *fact*
  **have** *validQ*: *valid Q* **by** *fact*
  **have** *validP*: *valid*$(a\{b\}.P)$ **by** *fact*
  **show** *?case*
  **proof**(*case-tac* $\exists\, Q' \in summands\ Q.\ Q' \equiv_e a\{b\}.P$)
    **assume** $\exists\, Q' \in summands\ Q.\ Q' \equiv_e a\{b\}.P$
    **then obtain** $Q'$ **where** $Q' \in summands\ Q$ **and** $Q' \equiv_e a\{b\}.P$ **by** *blast*
    **hence** $Q \oplus a\{b\}.P \equiv_e Q$ **by**(*rule summandsIdemp*)
    **moreover have** $depth\ Q \leq depth(Q \oplus a\{b\}.P)$ **by** *simp*
    **ultimately show** *?case* **using** *Qhnf validQ* **by**(*force intro*: *SumSym Trans*)
  **next**
    **assume** $\neg(\exists\, Q' \in summands\ Q.\ Q' \equiv_e a\{b\}.P)$
    **hence** $\forall\, Q' \in summands\ Q.\ \neg(Q' \equiv_e a\{b\}.P)$ **by** *simp*
    **with** *Qhnf QineqNil* **have** *uhnf* $(a\{b\}.P \oplus Q)$
      **by**(*force dest*: *Sym simp add*: *uhnf-def*)
    **moreover from** *validQ validP* **have** *valid*$(a\{b\}.P \oplus Q)$ **by** *simp*
    **moreover have** $a\{b\}.P \oplus Q \equiv_e a\{b\}.P \oplus Q$ **by**(*rule Refl*)
    **moreover have** $depth(a\{b\}.P \oplus Q) \leq depth(a\{b\}.P \oplus Q)$ **by** *simp*
    **ultimately show** *?case* **by** *blast*
  **qed**
**qed**
**next**
**case**(*Tau P Q*)
**show** *?case*
**proof**(*case-tac* $Q = \mathbf{0}$)
  **assume** $Q = \mathbf{0}$
  **moreover have** *uhnf* $(\tau.(P))$ **by**(*simp add*: *uhnf-def*)
  **moreover have** *valid* $(\tau.(P))$ **by** *fact*
  **moreover have** $\tau.(P) \oplus \mathbf{0} \equiv_e \tau.(P)$ **by**(*rule SumZero*)
  **moreover have** $depth(\tau.(P)) \leq depth(\tau.(P) \oplus \mathbf{0})$ **by** *simp*
  **ultimately show** *?case* **by** *blast*
**next**
  **assume** *QineqNil*: $Q \neq \mathbf{0}$
  **have** *Qhnf*: *uhnf Q* **by** *fact*
  **have** *validP*: *valid*$(\tau.(P))$ **and** *validQ*: *valid Q* **by** *fact+*
  **show** *?case*
  **proof**(*case-tac* $\exists\, Q' \in summands\ Q.\ Q' \equiv_e \tau.(P)$)
    **assume** $\exists\, Q' \in summands\ Q.\ Q' \equiv_e \tau.(P)$

**then obtain** $Q'$ **where** $Q' \in \mathit{summands}\ Q$ **and** $Q' \equiv_e \tau.(P)$ **by** *blast*
**hence** $Q \oplus \tau.(P) \equiv_e Q$ **by**(*rule summandsIdemp*)
**moreover have** $\mathit{depth}\ Q \leq \mathit{depth}(Q \oplus \tau.(P))$ **by** *simp*
**ultimately show** *?case* **using** *Qhnf validQ* **by**(*force intro*: *SumSym Trans*)
**next**
**assume** $\neg(\exists\, Q' \in \mathit{summands}\ Q.\ Q' \equiv_e \tau.(P))$
**hence** $\forall\, Q' \in \mathit{summands}\ Q.\ \neg(Q' \equiv_e \tau.(P))$ **by** *simp*
**with** *Qhnf QineqNil* **have** $\mathit{uhnf}\ (\tau.(P) \oplus Q)$
**by**(*force dest*: *Sym simp add*: *uhnf-def*)
**moreover from** *validP validQ* **have** $\mathit{valid}(\tau.(P) \oplus Q)$ **by** *simp*
**moreover have** $\tau.(P) \oplus Q \equiv_e \tau.(P) \oplus Q$ **by**(*rule Refl*)
**moreover have** $\mathit{depth}(\tau.(P) \oplus Q) \leq \mathit{depth}(\tau.(P) \oplus Q)$ **by** *simp*
**ultimately show** *?case* **by** *blast*
**qed**
**qed**
**next**
**case**(*Input a x P Q*)
**show** *?case*
**proof**(*case-tac* $Q = \mathbf{0}$)
**assume** $Q = \mathbf{0}$
**moreover have** $\mathit{uhnf}\ (a{<}x{>}.P)$ **by**(*simp add*: *uhnf-def*)
**moreover have** $\mathit{valid}\ (a{<}x{>}.P)$ **by** *fact*
**moreover have** $a{<}x{>}.P \oplus \mathbf{0} \equiv_e a{<}x{>}.P$ **by**(*rule SumZero*)
**moreover have** $\mathit{depth}(a{<}x{>}.P) \leq \mathit{depth}(a{<}x{>}.P \oplus \mathbf{0})$ **by** *simp*
**ultimately show** *?case* **by** *blast*
**next**
**assume** *QineqNil*: $Q \neq \mathbf{0}$
**have** *validP*: $\mathit{valid}(a{<}x{>}.P)$ **and** *validQ*: $\mathit{valid}\ Q$ **by** *fact+*
**have** *Qhnf*: $\mathit{uhnf}\ Q$ **by** *fact*
**show** *?case*
**proof**(*case-tac* $\exists\, Q' \in \mathit{summands}\ Q.\ Q' \equiv_e a{<}x{>}.P$)
**assume** $\exists\, Q' \in \mathit{summands}\ Q.\ Q' \equiv_e a{<}x{>}.P$
**then obtain** $Q'$ **where** $Q' \in \mathit{summands}\ Q$ **and** $Q' \equiv_e a{<}x{>}.P$ **by** *blast*
**hence** $Q \oplus a{<}x{>}.P \equiv_e Q$ **by**(*rule summandsIdemp*)
**moreover have** $\mathit{depth}\ Q \leq \mathit{depth}(Q \oplus a{<}x{>}.P)$ **by** *simp*
**ultimately show** *?case* **using** *Qhnf validQ* **by**(*force intro*: *SumSym Trans*)
**next**
**assume** $\neg(\exists\, Q' \in \mathit{summands}\ Q.\ Q' \equiv_e a{<}x{>}.P)$
**hence** $\forall\, Q' \in \mathit{summands}\ Q.\ \neg(Q' \equiv_e a{<}x{>}.P)$ **by** *simp*
**with** *Qhnf QineqNil* **have** $\mathit{uhnf}\ (a{<}x{>}.P \oplus Q)$
**by**(*force dest*: *Sym simp add*: *uhnf-def*)
**moreover from** *validP validQ* **have** $\mathit{valid}(a{<}x{>}.P \oplus Q)$ **by** *simp*
**moreover have** $a{<}x{>}.P \oplus Q \equiv_e a{<}x{>}.P \oplus Q$ **by**(*rule Refl*)
**moreover have** $\mathit{depth}(a{<}x{>}.P \oplus Q) \leq \mathit{depth}(a{<}x{>}.P \oplus Q)$ **by** *simp*
**ultimately show** *?case* **by** *blast*
**qed**
**qed**
**next**
**case**(*Match a b P Q*)

**have** *uhnf* $([a\frown b]P)$ **by** *fact*
**hence** *False* **by**(*simp add*: *uhnf-def*)
**thus** *?case* **by** *simp*
**next**
**case**(*Mismatch a b P Q*)
**have** *uhnf* $([a\neq b]P)$ **by** *fact*
**hence** *False* **by**(*simp add*: *uhnf-def*)
**thus** *?case* **by** *simp*
**next**
**case**(*Sum P Q R*)
**have** *Rhnf*: *uhnf R* **by** *fact*
**have** *validR*: *valid R* **by** *fact*
**have** *PQhnf*: *uhnf* $(P \oplus Q)$ **by** *fact*
**have** *validPQ*: *valid*$(P \oplus Q)$ **by** *fact*
**have** $\exists\, T.$ *uhnf T* $\wedge$ *valid T* $\wedge$ $Q \oplus R \equiv_e T \wedge$ *depth T* $\leq$ *depth* $(Q \oplus R)$
**proof** $-$
**from** *PQhnf* **have** *uhnf Q* **by**(*simp add*: *uhnf-def*)
**moreover from** *validPQ* **have** *valid Q* **by** *simp+*
**moreover have** $\bigwedge R.$ ⟦*uhnf Q*; *uhnf R*; *valid Q*; *valid R*⟧ $\Longrightarrow \exists\, T.$ *uhnf T* $\wedge$ *valid T* $\wedge$ $Q \oplus R \equiv_e T \wedge$ *depth T* $\leq$ *depth*$(Q \oplus R)$ **by** *fact*
**ultimately show** *?thesis* **using** *Rhnf validR* **by** *blast*
**qed**
**then obtain** *T* **where** *Thnf*: *uhnf T* **and** *QReqT*: $Q \oplus R \equiv_e T$ **and** *validT*: *valid T*
**and** *Tdepth*: *depth T* $\leq$ *depth*$(Q \oplus R)$ **by** *blast*

**have** $\exists\, S.$ *uhnf S* $\wedge$ *valid S* $\wedge$ $P \oplus T \equiv_e S \wedge$ *depth S* $\leq$ *depth*$(P \oplus T)$
**proof** $-$
**from** *PQhnf* **have** *uhnf P* **by**(*simp add*: *uhnf-def*)
**moreover from** *validPQ* **have** *valid P* **by** *simp*
**moreover have** $\bigwedge T.$ ⟦*uhnf P*; *uhnf T*; *valid P*; *valid T*⟧ $\Longrightarrow \exists\, S.$ *uhnf S* $\wedge$ *valid S* $\wedge$ $P \oplus T \equiv_e S \wedge$ *depth S* $\leq$ *depth*$(P \oplus T)$ **by** *fact*
**ultimately show** *?thesis* **using** *Thnf validT* **by** *blast*
**qed**
**then obtain** *S* **where** *Shnf*: *uhnf S* **and** *PTeqS*: $P \oplus T \equiv_e S$ **and** *validS*: *valid S*
**and** *Sdepth*: *depth S* $\leq$ *depth*$(P \oplus T)$ **by** *blast*

**have** $(P \oplus Q) \oplus R \equiv_e S$
**proof** $-$
**have** $(P \oplus Q) \oplus R \equiv_e P \oplus (Q \oplus R)$ **by**(*rule SumAssoc*)
**moreover from** *QReqT* **have** $P \oplus (Q \oplus R) \equiv_e P \oplus T$
**by**(*blast intro*: *Refl SumPres'*)
**ultimately show** *?thesis* **using** *PTeqS* **by**(*blast intro*: *Trans*)
**qed**
**moreover from** *Tdepth Sdepth* **have** *depth S* $\leq$ *depth*$((P \oplus Q) \oplus R)$ **by** *auto*

**ultimately show** *?case* **using** *Shnf validS* **by** *blast*
**next**

**case**($Par\ P\ Q\ R$)
**have** *uhnf* $(P \parallel Q)$ **by** *fact*
**hence** *False* **by**(*simp add*: *uhnf-def*)
**thus** *?case* **by** *simp*
**next**
**case**($Res\ x\ P\ Q$)
**show** *?case*
**proof**(*case-tac* $Q = \mathbf{0}$)
**assume** $Q = \mathbf{0}$
**moreover have** *uhnf* $(<\nu x>P)$ **by** *fact*
**moreover have** *valid* $(<\nu x>P)$ **by** *fact*
**moreover have** $<\nu x>P \oplus \mathbf{0} \equiv_e <\nu x>P$ **by**(*rule SumZero*)
**moreover have** $depth(<\nu x>P) \leq depth((<\nu x>P) \oplus \mathbf{0})$ **by** *simp*
**ultimately show** *?case* **by** *blast*
**next**
**assume** *QineqNil*: $Q \neq \mathbf{0}$
**have** *Qhnf*: *uhnf Q* **by** *fact*
**have** *validP*: $valid(<\nu x>P)$ **and** *validQ*: *valid Q* **by** *fact+*
**show** *?case*
**proof**(*case-tac* $\exists\, Q' \in summands\ Q.\ Q' \equiv_e <\nu x>P$)
**assume** $\exists\, Q' \in summands\ Q.\ Q' \equiv_e <\nu x>P$
**then obtain** $Q'$ **where** $Q' \in summands\ Q$ **and** $Q' \equiv_e <\nu x>P$ **by** *blast*
**hence** $Q \oplus <\nu x>P \equiv_e Q$ **by**(*rule summandsIdemp*)
**moreover have** $depth\ Q \leq depth(Q \oplus <\nu x>P)$ **by** *simp*
**ultimately show** *?case* **using** *Qhnf validQ* **by**(*force intro*: *SumSym Trans*)
**next**
**assume** $\neg(\exists\, Q' \in summands\ Q.\ Q' \equiv_e <\nu x>P)$
**hence** $\forall\, Q' \in summands\ Q.\ \neg(Q' \equiv_e <\nu x>P)$ **by** *simp*
**moreover have** *uhnf* $(<\nu x>P)$ **by** *fact*
**ultimately have** *uhnf* $(<\nu x>P \oplus Q)$ **using** *Qhnf QineqNil*
**by**(*force dest*: *Sym simp add*: *uhnf-def*)
**moreover from** *validP validQ* **have** $valid(<\nu x>P \oplus Q)$ **by** *simp*
**moreover have** $(<\nu x>P) \oplus Q \equiv_e (<\nu x>P) \oplus Q$ **by**(*rule Refl*)
**moreover have** $depth((<\nu x>P) \oplus Q) \leq depth((<\nu x>P) \oplus Q)$ **by** *simp*
**ultimately show** *?case* **by** *blast*
**qed**
**qed**
**next**
**case**($Bang\ P\ Q$)
**have** *uhnf* $(!P)$ **by** *fact*
**hence** *False* **by**(*simp add*: *uhnf-def*)
**thus** *?case* **by** *simp*
**qed**

**lemma** *uhnfRes*:
**fixes** $x :: name$
**and** $P :: pi$

**assumes** *Phnf*: *uhnf P*

**and** *validP*: *valid P*

**shows** $\exists P'.\ uhnf\ P' \wedge valid\ P' \wedge {<}\nu x{>}P \equiv_e P' \wedge depth\ P' \leq depth({<}\nu x{>}P)$
**using** *assms*
**proof**(*nominal-induct P avoiding*: *x rule*: *pi.strong-inducts*)
**case**(*PiNil x*)
**have** *uhnf* **0** **by**(*simp add*: *uhnf-def*)
**moreover have** *valid* **0** **by** *simp*
**moreover have** $<\nu x>\mathbf{0} \equiv_e \mathbf{0}$ **by**(*rule ResNil*)
**moreover have** $depth\ \mathbf{0} \leq depth(<\nu x>\mathbf{0})$ **by** *simp*
**ultimately show** *?case* **by** *blast*
**next**
**case**(*Output a b P*)
**have** $valid(a\{b\}.P)$ **by** *fact*
**hence** *validP*: *valid P* **by** *simp*
**show** *?case*
**proof**(*case-tac* $x=a$)
**assume** $x = a$
**moreover have** *uhnf* **0** **by**(*simp add*: *uhnf-def*)
**moreover have** *valid* **0** **by** *simp*
**moreover have** $\mathbf{0} \equiv_e <\nu x>x\{b\}.P$ **by**(*blast intro*: *ResOutput′ Sym*)
**moreover have** $depth\ \mathbf{0} \leq depth(<\nu x>x\{b\}.P)$ **by** *simp*
**ultimately show** *?case* **by**(*blast intro*: *Sym*)
**next**
**assume** *xineqa*: $x \neq a$
**show** *?case*
**proof**(*case-tac* $x=b$)
**assume** $x=b$
**moreover from** *xineqa* **have** $uhnf(<\nu x>a\{x\}.P)$ **by**(*force simp add*: *uhnf-def*)
**moreover from** *validP* **have** $valid(<\nu x>a\{x\}.P)$ **by** *simp*
**moreover have** $<\nu x>a\{x\}.P \equiv_e <\nu x>a\{x\}.P$ **by**(*rule Refl*)
**moreover have** $depth(<\nu x>a\{x\}.P) \leq depth(<\nu x>a\{x\}.P)$ **by** *simp*
**ultimately show** *?case* **by** *blast*
**next**
**assume** *xineqb*: $x \neq b$
**have** $uhnf(a\{b\}.(<\nu x>P))$ **by**(*simp add*: *uhnf-def*)
**moreover from** *validP* **have** $valid(a\{b\}.(<\nu x>P))$ **by** *simp*
**moreover from** *xineqa xineqb* **have** $a\{b\}.(<\nu x>P) \equiv_e <\nu x>a\{b\}.P$ **by**(*blast intro*: *ResOutput Sym*)
**moreover have** $depth(a\{b\}.(<\nu x>P)) \leq depth(<\nu x>a\{b\}.P)$ **by** *simp*
**ultimately show** *?case* **by**(*blast intro*: *Sym*)
**qed**
**qed**
**next**
**case**(*Tau P*)
**have** $valid(\tau.(P))$ **by** *fact*
**hence** *validP*: *valid P* **by** *simp*

**have** $uhnf(\tau.(<\nu x>P))$ **by**(*simp add*: *uhnf-def*)

**moreover from** *validP* **have** $valid(\tau.(<\nu x>P))$ **by** *simp*
**moreover have** $\tau.(<\nu x>P) \equiv_e <\nu x>\tau.(P)$ **by**(*blast intro*: *ResTau Sym*)
**moreover have** $depth(\tau.(<\nu x>P)) \leq depth(<\nu x>\tau.(P))$ **by** *simp*
**ultimately show** *?case* **by**(*blast intro*: *Sym*)
**next**
**case**(*Input a y P x*)
**have** $valid(a<y>.P)$ **by** *fact*
**hence** *validP*: *valid P* **by** *simp*
**have** $y \sharp x$ **by** *fact* **hence** *yineqx*: $y \neq x$ **by** *simp*
**show** *?case*
**proof**(*case-tac* $x=a$)
**assume** $x = a$
**moreover have** *uhnf* **0** **by**(*simp add*: *uhnf-def*)
**moreover have** *valid* **0** **by** *simp*
**moreover have** $\mathbf{0} \equiv_e <\nu x>x<y>.P$ **by**(*blast intro*: *ResInput′ Sym*)
**moreover have** $depth\ \mathbf{0} \leq depth(<\nu x>x<y>.P)$ **by** *simp*
**ultimately show** *?case* **by**(*blast intro*: *Sym*)
**next**
**assume** *xineqa*: $x \neq a$
**have** $uhnf(a<y>.(<\nu x>P))$ **by**(*simp add*: *uhnf-def*)
**moreover from** *validP* **have** $valid(a<y>.(<\nu x>P))$ **by** *simp*
**moreover from** *xineqa yineqx* **have** $a<y>.(<\nu x>P) \equiv_e <\nu x>a<y>.P$ **by**(*blast intro*: *ResInput Sym*)
**moreover have** $depth(a<y>.(<\nu x>P)) \leq depth(<\nu x>a<y>.P)$ **by** *simp*
**ultimately show** *?case* **by**(*blast intro*: *Sym*)
**qed**
**next**
**case**(*Match a b P x*)
**have** $uhnf([a \frown b]P)$ **by** *fact*
**hence** *False* **by**(*simp add*: *uhnf-def*)
**thus** *?case* **by** *simp*
**next**
**case**(*Mismatch a b P x*)
**have** $uhnf([a \neq b]P)$ **by** *fact*
**hence** *False* **by**(*simp add*: *uhnf-def*)
**thus** *?case* **by** *simp*
**next**
**case**(*Sum P Q x*)
**have** $valid(P \oplus Q)$ **by** *fact*
**hence** *validP*: *valid P* **and** *validQ*: *valid Q* **by** *simp+*
**have** $uhnf(P \oplus Q)$ **by** *fact*
**hence** *Phnf*: *uhnf P* **and** *Qhnf*: *uhnf Q* **by**(*auto simp add*: *uhnf-def*)

**have** $\exists P'.\ uhnf\ P' \wedge valid\ P' \wedge P' \equiv_e <\nu x>P \wedge (depth\ P') \leq (depth(<\nu x>P))$
**proof** −
**have** $[\![uhnf\ P;\ valid\ P]\!] \Longrightarrow \exists P'.\ uhnf\ P' \wedge valid\ P' \wedge <\nu x>P \equiv_e P' \wedge (depth\ P') \leq (depth\ (<\nu x>P))$ **by** *fact*
**with** *validP Phnf* **show** *?thesis* **by**(*blast intro*: *Sym*)
**qed**

**then obtain** $P'$ **where** $P'hnf$: $uhnf\ P'$ **and** $P'eqP$: $P' \equiv_e <\nu x>P$ **and** $validP'$: $valid\ P'$
**and** $P'depth$: $(depth\ P') \leq (depth(<\nu x>P))$ **by** *blast*

**have** $\exists\, Q'.\ uhnf\ Q' \wedge valid\ Q' \wedge Q' \equiv_e <\nu x>Q \wedge (depth\ Q') \leq (depth(<\nu x>Q))$
**proof** −
**have** $[\![uhnf\ Q;\ valid\ Q]\!] \Longrightarrow \exists\, Q'.\ uhnf\ Q' \wedge valid\ Q' \wedge <\nu x>Q \equiv_e Q' \wedge (depth\ Q') \leq (depth(<\nu x>Q))$ **by** *fact*
**with** *validQ Qhnf* **show** *?thesis* **by**(*blast intro*: *Sym*)
**qed**

**then obtain** $Q'$ **where** $Q'hnf$: $uhnf\ Q'$ **and** $Q'eqQ$: $Q' \equiv_e <\nu x>Q$ **and** $validQ'$: $valid\ Q'$
**and** $Q'depth$: $(depth\ Q') \leq (depth(<\nu x>Q))$ **by** *blast*

**from** $P'hnf\ Q'hnf\ validP'\ validQ'$ **obtain** $R$ **where** $Rhnf$: $uhnf\ R$ **and** $validR$: $valid\ R$
**and** $P'Q'eqR$: $P' \oplus Q' \equiv_e R$
**and** $Rdepth$: $depth\ R \leq depth(P' \oplus Q')$
**apply**(*drule-tac uhnfSum*) **apply** *assumption+* **by** *blast*

**from** $P'eqP\ Q'eqQ\ P'Q'eqR$ **have** $<\nu x>(P \oplus Q) \equiv_e R$ **by**(*blast intro*: *Sym SumPres' SumRes Trans*)
**moreover from** $Rdepth\ P'depth\ Q'depth$ **have** $depth\ R \leq depth(<\nu x>(P \oplus Q))$ **by** *auto*
**ultimately show** *?case* **using** *validR Rhnf* **by**(*blast intro*: *Sym*)
**next**
**case**($Par\ P\ Q$)
**have** $uhnf(P \parallel Q)$ **by** *fact*
**hence** *False* **by**(*simp add*: *uhnf-def*)
**thus** *?case* **by** *simp*
**next**
**case**($Res\ y\ P\ x$)
**have** $valid(<\nu y>P)$ **by** *fact* **hence** $validP$: $valid\ P$ **by** *simp*
**have** $uhnf(<\nu y>P)$ **by** *fact*
**then obtain** $a\ P'$ **where** $aineqy$: $a \neq y$ **and** $PeqP'$: $P = a\{y\}.P'$
**by**(*force simp add*: *uhnf-def*)

**show** *?case*
**proof**(*case-tac* $x{=}y$)
**assume** $x{=}y$
**moreover from** *aineqy* **have** $uhnf(<\nu y>a\{y\}.P')$ **by**(*force simp add*: *uhnf-def*)
**moreover from** $validP\ PeqP'$ **have** $valid(<\nu y>a\{y\}.P')$ **by** *simp*
**moreover have** $<\nu y><\nu y>a\{y\}.P' \equiv_e <\nu y>a\{y\}.P'$
**proof** −
**have** $y \sharp <\nu y>a\{y\}.P'$ **by**(*simp add*: *name-fresh-abs*)
**hence** $<\nu y><\nu y>a\{y\}.P' \equiv_e <\nu y>a\{y\}.P'$ **by**(*rule ResFresh*)
**thus** *?thesis* **by**(*blast intro*: *Trans*)
**qed**

**moreover have** $depth(<\nu y>a\{y\}.P') \leq depth(<\nu y><\nu y>a\{y\}.P')$ **by** *simp*
**ultimately show** *?case* **using** *PeqP'* **by** *blast*
**next**
**assume** *xineqy*: $x \neq y$
**show** *?case*
**proof**(*case-tac* $x=a$)
**assume** $x=a$
**moreover have** *uhnf* **0** **by**(*simp add*: *uhnf-def*)
**moreover have** *valid* **0** **by** *simp*
**moreover have** $<\nu a><\nu y>a\{y\}.P' \equiv_e \mathbf{0}$
**proof** −
**have** $<\nu a><\nu y>a\{y\}.P' \equiv_e <\nu y><\nu a>a\{y\}.P'$ **by**(*rule ResComm*)
**moreover have** $<\nu y><\nu a>a\{y\}.P' \equiv_e \mathbf{0}$
**by**(*blast intro*: *ResOutput' ResNil ResPres Trans*)
**ultimately show** *?thesis* **by**(*blast intro*: *Trans*)
**qed**
**moreover have** $depth\ \mathbf{0} \leq depth(<\nu a><\nu y>a\{y\}.P')$ **by** *simp*
**ultimately show** *?case* **using** *PeqP'* **by** *blast*
**next**
**assume** *xineqa*: $x \neq a$
**from** *aineqy* **have** $uhnf(<\nu y>a\{y\}.(<\nu x>P'))$ **by**(*force simp add*: *uhnf-def*)
**moreover from** *validP PeqP'* **have** $valid(<\nu y>a\{y\}.(<\nu x>P'))$ **by** *simp*
**moreover have** $<\nu x><\nu y>a\{y\}.P' \equiv_e <\nu y>a\{y\}.(<\nu x>P')$
**proof** −
**have** $<\nu x><\nu y>a\{y\}.P' \equiv_e <\nu y><\nu x>a\{y\}.P'$ **by**(*rule ResComm*)
**moreover from** *xineqa xineqy* **have** $<\nu y><\nu x>a\{y\}.P' \equiv_e <\nu y>a\{y\}.(<\nu x>P')$
**by**(*blast intro*: *ResOutput ResPres Trans*)
**ultimately show** *?thesis* **by**(*blast intro*: *Trans*)
**qed**
**moreover have** $depth(<\nu y>a\{y\}.(<\nu x>P')) \leq depth(<\nu x><\nu y>a\{y\}.P')$
**by** *simp*
**ultimately show** *?case* **using** *PeqP'* **by** *blast*
**qed**
**qed**
**next**
**case**(*Bang P x*)
**have** $valid(!P)$ **by** *fact*
**hence** *False* **by** *simp*
**thus** *?case* **by** *simp*
**qed**

**lemma** *expandHnf*:
**fixes** $P :: pi$
**and** $S :: pi\ set$

**assumes** $(P, S) \in sumComposeSet$
**and** $\forall P \in S.\ uhnf\ P \wedge valid\ P$

**shows** $\exists P'.\ uhnf\ P' \wedge valid\ P' \wedge P \equiv_e P' \wedge depth\ P' \leq depth\ P$

**using** *assms*
**proof**(*induct rule*: *sumComposeSet.induct*)
  **case** *empty*
  **have** *uhnf* **0** **by**(*simp add*: *uhnf-def*)
  **moreover have** *valid* **0** **by** *simp*
  **moreover have** **0** $\equiv_e$ **0** **by**(*rule Refl*)
  **moreover have** *depth* **0** $\leq$ *depth* **0** **by** *simp*
  **ultimately show** *?case* **by** *blast*
**next**
  **case**(*insert Q S P*)
  **have** *Shnf*: $\forall P \in S$. *uhnf P* $\wedge$ *valid P* **by** *fact*
  **hence** $\forall P \in (S - \{(Q)\})$. *uhnf P* $\wedge$ *valid P* **by** *simp*
  **moreover have** $\forall P \in (S - \{(Q)\})$. *uhnf P* $\wedge$ *valid P* $\Longrightarrow \exists P'$. *uhnf* $P' \wedge$ *valid* $P' \wedge P \equiv_e P' \wedge$ *depth* $P' \leq$ *depth P* **by** *fact*
  **ultimately obtain** $P'$ **where** $P'hnf$: *uhnf* $P'$ **and** $validP'$: *valid* $P'$
      **and** $PeqP'$: $P \equiv_e P'$ **and** $PP'depth$: *depth* $P' \leq$ *depth P*
    **by** *blast*

  **have** $Q \in S$ **by** *fact*
  **with** *Shnf* **have** *uhnf Q* **and** *valid Q* **by** *simp+*
  **with** $P'hnf$ $validP'$ **obtain** *R* **where** *Rhnf*: *uhnf R* **and** *validR*: *valid R*
      **and** $P'QeqR$: $P' \oplus Q \equiv_e R$ **and** $P'QRdepth$: *depth* $R \leq$ *depth* $(P' \oplus Q)$
    **by**(*auto dest*: *uhnfSum*)

  **from** $PeqP'$ $P'QeqR$ **have** $P \oplus Q \equiv_e R$ **by**(*blast intro*: *SumPres Trans*)
  **moreover from** $PP'depth$ $P'QRdepth$ **have** *depth* $R \leq$ *depth*$(P \oplus Q)$ **by** *simp*
  **ultimately show** *?case* **using** *Rhnf validR* **by** *blast*
**qed**

**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*) $- \{P\}$
**using** *assms*
**by**(*auto intro*: *Refl simp add*: *uhnf-def*)

**lemma** *pullSummand*:
  **fixes** $P$ :: *pi*
  **and** $Q$ :: *pi*

  **assumes** *PsummQ*: $P \in$ *summands Q*
  **and** *Qhnf*: *uhnf Q*

**shows** $\exists Q'. P \oplus Q' \equiv_e Q \wedge (\textit{summands } Q') = ((\textit{summands } Q) - \{x. \exists P'. x = P' \wedge P' \in (\textit{summands } Q) \wedge P' \equiv_e P\}) \wedge \textit{uhnf } Q'$
**proof** $-$
**have** *SumGoal*: $\bigwedge P\ Q\ R.\ [\![P \in \textit{summands } Q; \textit{uhnf}(Q \oplus R);$
$\bigwedge P.\ [\![P \in \textit{summands } Q]\!] \Longrightarrow \exists Q'. P \oplus Q' \equiv_e Q \wedge$
$(\textit{summands } Q') = ((\textit{summands } Q) - \{P' | P'. P' \in \textit{summands } Q \wedge P' \equiv_e P\}) \wedge \textit{uhnf } Q';$
$\bigwedge P.\ [\![P \in \textit{summands } R]\!] \Longrightarrow \exists R'. P \oplus R' \equiv_e R \wedge$
$(\textit{summands } R') = ((\textit{summands } R) - \{P' | P'. P' \in \textit{summands } R \wedge P' \equiv_e P\}) \wedge \textit{uhnf } R']\!]$
$\Longrightarrow \exists Q'. P \oplus Q' \equiv_e Q \oplus R \wedge$
$\textit{summands } Q' = \textit{summands } (\textit{pi.Sum } Q\ R) - \{P' | P'. P' \in \textit{summands } (Q \oplus R) \wedge P' \equiv_e P\} \wedge \textit{uhnf } Q'$
**proof** $-$
**fix** $P\ Q\ R$
**assume** *IHR*: $\bigwedge P.\ P \in \textit{summands } R \Longrightarrow \exists R'. P \oplus R' \equiv_e R \wedge$
$(\textit{summands } R') = ((\textit{summands } R) - \{P' | P'. P' \in \textit{summands } R \wedge P' \equiv_e P\}) \wedge \textit{uhnf } R'$
**assume** *PsummQ*: $P \in \textit{summands } Q$
**moreover assume** $\bigwedge P.\ P \in \textit{summands } Q \Longrightarrow \exists Q'. P \oplus Q' \equiv_e Q \wedge$
$(\textit{summands } Q') = ((\textit{summands } Q) - \{P' | P'. P' \in \textit{summands } Q \wedge P' \equiv_e P\}) \wedge \textit{uhnf } Q'$

**ultimately obtain** $Q'$ **where** *PQ'eqQ*: $P \oplus Q' \equiv_e Q$
**and** *Q'summQ*: $(\textit{summands } Q') = ((\textit{summands } Q) - \{P' | P'. P' \in \textit{summands } Q \wedge P' \equiv_e P\})$
**and** *Q'hnf*: $\textit{uhnf } Q'$
**by** *blast*
**assume** *QRhnf*: $\textit{uhnf}(Q \oplus R)$

**show** $\exists Q'. P \oplus Q' \equiv_e Q \oplus R \wedge$
$\textit{summands } Q' = \textit{summands } (\textit{pi.Sum } Q\ R) - \{P' | P'. P' \in \textit{summands } (Q \oplus R) \wedge P' \equiv_e P\} \wedge \textit{uhnf } Q'$
**proof**(*cases* $\exists P' \in \textit{summands } R.\ P' \equiv_e P$)
**assume** $\exists P' \in \textit{summands } R.\ P' \equiv_e P$
**then obtain** $P'$ **where** *P'summR*: $P' \in \textit{summands } R$ **and** *P'eqP*: $P' \equiv_e P$ **by** *blast*
**with** *IHR* **obtain** $R'$ **where** *PR'eqR*: $P' \oplus R' \equiv_e R$
**and** *R'summR*: $(\textit{summands } R') = ((\textit{summands } R) - \{P'' | P''. P'' \in \textit{summands } R \wedge P'' \equiv_e P'\})$
**and** *R'hnf*: $\textit{uhnf } R'$
**by** *blast*

**have** *L1*: $P \oplus (Q' \oplus R') \equiv_e Q \oplus R$
**proof** $-$
**from** *P'eqP* **have** $P \oplus (Q' \oplus R') \equiv_e (P \oplus P') \oplus (Q' \oplus R')$
**by**(*blast intro*: *SumIdemp' SumPres Sym*)
**moreover have** $(P \oplus P') \oplus (Q' \oplus R') \equiv_e P \oplus (P' \oplus (Q' \oplus R'))$ **by**(*rule SumAssoc*)

**moreover have** $P \oplus (P' \oplus (Q' \oplus R')) \equiv_e P \oplus (P' \oplus (R' \oplus Q'))$
**by**(*blast intro*: *Refl SumPres' SumSym*)
**moreover have** $P \oplus (P' \oplus (R' \oplus Q')) \equiv_e P \oplus (P' \oplus R') \oplus Q'$
**by**(*blast intro*: *Refl SumPres' Sym SumAssoc*)
**moreover have** $P \oplus (P' \oplus R') \oplus Q' \equiv_e (P \oplus Q') \oplus (P' \oplus R')$
**proof** −
**have** $P \oplus (P' \oplus R') \oplus Q' \equiv_e P \oplus Q' \oplus (P' \oplus R')$
**by**(*blast intro*: *Refl SumPres' SumSym*)
**thus** *?thesis* **by**(*blast intro*: *Sym SumAssoc Trans*)
**qed**
**moreover from** *PQ'eqQ PR'eqR* **have** $(P \oplus Q') \oplus (P' \oplus R') \equiv_e Q \oplus R$ **by**(*rule SumPres'*)
**ultimately show** *?thesis* **by**(*blast intro*!: *Trans*)
**qed**

**show** *?thesis*
**proof**(*cases* $Q' = \mathbf{0}$)
**assume** *Q'eqNil*: $Q' = \mathbf{0}$
**have** $P \oplus R' \equiv_e Q \oplus R$
**proof** −
**have** $P \oplus R' \equiv_e P \oplus (R' \oplus \mathbf{0})$ **by**(*blast intro*: *SumZero Refl Trans SumPres' Sym*)
**moreover have** $P \oplus (R' \oplus \mathbf{0}) \equiv_e P \oplus (\mathbf{0} \oplus R')$
**by**(*blast intro*: *SumSym Trans SumPres' Refl*)
**ultimately show** *?thesis* **using** *L1 Q'eqNil* **by**(*blast intro*: *Trans*)
**qed**
**moreover from** *R'summR Q'summQ P'eqP Q'eqNil* **have** *summands* $(R') = (\mathit{summands}\ (Q \oplus R) - \{P' \mid P'.\ P' \in \mathit{summands}(Q \oplus R) \wedge P' \equiv_e P\})$
**by**(*auto intro*: *Sym Trans*)
**ultimately show** *?thesis* **using** *R'hnf* **by** *blast*
**next**
**assume** *Q'ineqNil*: $Q' \neq \mathbf{0}$
**show** *?thesis*
**proof**(*case-tac* $R' = \mathbf{0}$)
**assume** *R'eqNil*: $R' = \mathbf{0}$
**have** $P \oplus Q' \equiv_e Q \oplus R$
**proof** −
**have** $P \oplus Q' \equiv_e P \oplus (Q' \oplus \mathbf{0})$ **by**(*blast intro*: *SumZero Refl Trans SumPres' Sym*)
**with** *L1 R'eqNil* **show** *?thesis* **by**(*blast intro*: *Trans*)
**qed**
**moreover from** *R'summR Q'summQ P'eqP R'eqNil* **have** *summands* $(Q') = (\mathit{summands}\ (Q \oplus R) - \{P' \mid P'.\ P' \in \mathit{summands}(Q \oplus R) \wedge P' \equiv_e P\})$
**by**(*auto intro*: *Sym Trans*)
**ultimately show** *?thesis* **using** *Q'hnf* **by** *blast*
**next**
**assume** *R'ineqNil*: $R' \neq \mathbf{0}$

**from** *R'summR Q'summQ P'eqP* **have** *summands* $(Q' \oplus R') = ($*summands*

$(Q \oplus R) - \{P' \mid P'. P' \in \textit{summands}(Q \oplus R) \wedge P' \equiv_e P\})$
**by**(*auto intro*: *Sym Trans*)
**moreover from** *QRhnf Q'hnf R'hnf R'summR Q'summQ Q'ineqNil R'ineqNil* **have** *uhnf*$(Q' \oplus R')$
**by**(*auto simp add*: *uhnf-def*)

**ultimately show** *?thesis* **using** *L1* **by** *blast*
**qed**
**qed**
**next**
**assume** $\neg(\exists P' \in \textit{summands } R.\ P' \equiv_e P)$
**hence** *Case*: $\forall P' \in \textit{summands } R.\ \neg(P' \equiv_e P)$ **by** *simp*
**show** *?thesis*
**proof**(*case-tac* $Q' = \mathbf{0}$)
**assume** *Q'eqNil*: $Q' = \mathbf{0}$
**have** $P \oplus R \equiv_e Q \oplus R$
**proof** −

**have** $P \oplus R \equiv_e (P \oplus \mathbf{0}) \oplus R$ **by**(*blast intro*: *SumZero Sym Trans SumPres*)
**moreover from** *PQ'eqQ* **have** $P \oplus (Q' \oplus R) \equiv_e Q \oplus R$
**by**(*blast intro*: *SumAssoc Trans Sym SumPres*)
**ultimately show** *?thesis* **using** *Q'eqNil* **by**(*blast intro*: *SumAssoc Trans*)
**qed**

**moreover from** *Q'summQ Q'eqNil Case* **have** *summands* $(R) = (\textit{summands}\ (Q \oplus R) - \{P' \mid P'. P' \in \textit{summands}(Q \oplus R) \wedge P' \equiv_e P\})$
**by** *auto*
**moreover from** *QRhnf* **have** *uhnf R* **by**(*simp add*: *uhnf-def*)

**ultimately show** *?thesis* **by** *blast*
**next**
**assume** *Q'ineqNil*: $Q' \neq \mathbf{0}$
**from** *PQ'eqQ* **have** $P \oplus (Q' \oplus R) \equiv_e Q \oplus R$
**by**(*blast intro*: *SumAssoc Trans Sym SumPres*)

**moreover from** *Q'summQ Case* **have** *summands* $(Q' \oplus R) = (\textit{summands}\ (Q \oplus R) - \{P' \mid P'. P' \in \textit{summands}(Q \oplus R) \wedge P' \equiv_e P\})$
**by** *auto*
**moreover from** *QRhnf Q'hnf Q'summQ Q'ineqNil* **have** *uhnf* $(Q' \oplus R)$
**by**(*auto simp add*: *uhnf-def*)
**ultimately show** *?thesis* **by** *blast*
**qed**
**qed**
**qed**

**from** *assms* **show** *?thesis*
**proof**(*nominal-induct Q arbitrary*: *P rule*: *pi.strong-inducts*)
**case** *PiNil*
**have** $P \in \textit{summands}\ \mathbf{0}$ **by** *fact*

**hence** *False* **by** *auto*
**thus** *?case* **by** *simp*
**next**
**case**(*Output a b Q*)
**have** $P \in$ *summands* $(a\{b\}.Q)$ **by** *fact*
**hence** *PeqQ*: $P = a\{b\}.Q$ **by** *simp*
**have** $P \oplus \mathbf{0} \equiv_e a\{b\}.Q$
**proof** −
**have** $P \oplus \mathbf{0} \equiv_e P$ **by**(*rule SumZero*)
**with** *PeqQ* **show** *?thesis* **by** *simp*
**qed**
**moreover have** (*summands* $\mathbf{0}$) = (*summands* $(a\{b\}.Q)$) − $\{P' \mid P'.\ P' \in$ *summands* $(a\{b\}.Q) \wedge P' \equiv_e P\}$
**proof** −
**have** $a\{b\}.Q \equiv_e a\{b\}.Q$ **by**(*rule Refl*)
**with** *PeqQ* **show** *?thesis* **by** *simp*
**qed**
**moreover have** *uhnf* $\mathbf{0}$ **by**(*simp add*: *uhnf-def*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*Tau Q*)
**have** $P \in$ *summands* $(\tau.(Q))$ **by** *fact*
**hence** *PeqQ*: $P = \tau.(Q)$ **by** *simp*
**have** $P \oplus \mathbf{0} \equiv_e \tau.(Q)$
**proof** −
**have** $P \oplus \mathbf{0} \equiv_e P$ **by**(*rule SumZero*)
**with** *PeqQ* **show** *?thesis* **by** *simp*
**qed**
**moreover have** (*summands* $\mathbf{0}$) = (*summands* $(\tau.(Q))$) − $\{P' \mid P'.\ P' \in$ *summands* $(\tau.(Q)) \wedge P' \equiv_e P\}$
**proof** −
**have** $\tau.(Q) \equiv_e \tau.(Q)$ **by**(*rule Refl*)
**with** *PeqQ* **show** *?thesis* **by** *simp*
**qed**
**moreover have** *uhnf* $\mathbf{0}$ **by**(*simp add*: *uhnf-def*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*Input a x Q*)
**have** $P \in$ *summands* $(a{<}x{>}.Q)$ **by** *fact*
**hence** *PeqQ*: $P = a{<}x{>}.Q$ **by** *simp*
**have** $P \oplus \mathbf{0} \equiv_e a{<}x{>}.Q$
**proof** −
**have** $P \oplus \mathbf{0} \equiv_e P$ **by**(*rule SumZero*)
**with** *PeqQ* **show** *?thesis* **by** *simp*
**qed**
**moreover have** (*summands* $\mathbf{0}$) = (*summands* $(a{<}x{>}.Q)$) − $\{P' \mid P'.\ P' \in$ *summands* $(a{<}x{>}.Q) \wedge P' \equiv_e P\}$
**proof** −
**have** $a{<}x{>}.Q \equiv_e a{<}x{>}.Q$ **by**(*rule Refl*)

**with** *PeqQ* **show** *?thesis* **by** *simp*
**qed**
**moreover have** *uhnf* **0** **by**(*simp add*: *uhnf-def*)
**ultimately show** *?case* **by** *blast*
**next**
**case**(*Match a b Q*)
**have** $P \in summands$ $([a \frown b]Q)$ **by** *fact*
**hence** *False* **by** *simp*
**thus** *?case* **by** *simp*
**next**
**case**(*Mismatch a b Q*)
**have** $P \in summands$ $([a \neq b]Q)$ **by** *fact*
**hence** *False* **by** *simp*
**thus** *?case* **by** *simp*
**next**
**case**(*Sum Q R*)
**have** *QRhnf*: *uhnf* $(Q \oplus R)$ **by** *fact*
**hence** *Qhnf*: *uhnf Q* **and** *Rhnf*: *uhnf R* **by**(*simp add*: *uhnf-def*)+
**have** $\bigwedge P.\ [\![P \in summands\ Q;\ uhnf\ Q]\!] \Longrightarrow \exists\, Q'.\ P \oplus Q' \equiv_e Q\ \wedge$
$(summands\ Q') = ((summands\ Q) - \{P' \mid P'.\ P'$
$\in summands\ Q \wedge P' \equiv_e P\}) \wedge uhnf\ Q'$
**by** *fact*
**with** *Qhnf* **have** *IHQ*: $\bigwedge P.\ P \in summands\ Q \Longrightarrow \exists\, Q'.\ P \oplus Q' \equiv_e Q\ \wedge$
$(summands\ Q') = ((summands\ Q) - \{P' \mid P'.\ P' \in$
$summands\ Q \wedge P' \equiv_e P\}) \wedge uhnf\ Q'$
**by** *simp*
**have** $\bigwedge P.\ [\![P \in summands\ R;\ uhnf\ R]\!] \Longrightarrow \exists\, R'.\ P \oplus R' \equiv_e R\ \wedge$
$(summands\ R') = ((summands\ R) - \{P' \mid P'.\ P'$
$\in summands\ R \wedge P' \equiv_e P\}) \wedge uhnf\ R'$
**by** *fact*
**with** *Rhnf* **have** *IHR*: $\bigwedge P.\ P \in summands\ R \Longrightarrow \exists\, R'.\ P \oplus R' \equiv_e R\ \wedge$
$(summands\ R') = ((summands\ R) - \{P' \mid P'.\ P' \in$
$summands\ R \wedge P' \equiv_e P\}) \wedge uhnf\ R'$
**by** *simp*
**have** $P \in summands$ $(Q \oplus R)$ **by** *fact*
**hence** $P \in summands\ Q \vee P \in summands\ R$ **by** *simp*
**thus** *?case*
**proof**(*rule disjE*)
**assume** $P \in summands\ Q$
**thus** *?case* **using** *QRhnf IHQ IHR* **by**(*rule SumGoal*)
**next**
**assume** $P \in summands\ R$
**moreover from** *QRhnf* **have** *uhnf* $(R \oplus Q)$ **by**(*auto simp add*: *uhnf-def*)
**ultimately have** $\exists\, Q'.\ (pi.Sum\ P\ Q') \equiv_e (pi.Sum\ R\ Q)\ \wedge$
$summands\ Q' = summands\ (pi.Sum\ R\ Q) - \{P' \mid P'.\ P' \in summands$
$(pi.Sum\ R\ Q) \wedge P' \equiv_e P\} \wedge uhnf\ Q'$ **using** *IHR IHQ*
**by**(*rule SumGoal*)
**thus** *?case*
**by**(*force intro*: *SumSym Trans*)

**qed**
**next**
**case**($Par\ Q\ R\ P$)
**have** $P \in summands\ (Q \parallel R)$ **by** *fact*
**hence** *False* **by** *simp*
**thus** *?case* **by** *simp*
**next**
**case**($Res\ x\ Q\ P$)
**have** $P \in summands\ (<\nu x>Q)$ **by** *fact*
**hence** *PeqQ*: $P = <\nu x>Q$ **by**(*simp add*: *if-split*)
**have** $P \oplus \mathbf{0} \equiv_e <\nu x>Q$
**proof** −
**have** $P \oplus \mathbf{0} \equiv_e P$ **by**(*rule SumZero*)
**with** *PeqQ* **show** *?thesis* **by** *simp*
**qed**
**moreover have** $(summands\ \mathbf{0}) = (summands\ (<\nu x>Q)) - \{P' \mid P'.\ P' \in summands\ (<\nu x>Q) \wedge P' \equiv_e P\}$
**proof** −
**have** $<\nu x>Q \equiv_e <\nu x>Q$ **by**(*rule Refl*)
**with** *PeqQ* **show** *?thesis* **by** *simp*
**qed**
**moreover have** *uhnf* $\mathbf{0}$ **by**(*simp add*: *uhnf-def*)
**ultimately show** *?case* **by** *blast*
**next**
**case**($Bang\ Q\ P$)
**have** $P \in summands\ (!Q)$ **by** *fact*
**hence** *False* **by** *simp*
**thus** *?case* **by** *simp*
**qed**
**qed**

**lemma** *nSym*:
**fixes** $P :: pi$
**and** $Q :: pi$

**assumes** $\neg(P \equiv_e Q)$

**shows** $\neg(Q \equiv_e P)$
**using** *assms*
**by**(*blast dest*: *Sym*)

**lemma** *summandsZero*:
**fixes** $P :: pi$

**assumes** $summands\ P = \{\}$
**and** *hnf P*

**shows** $P = \mathbf{0}$
**using** *assms*

**by**(*nominal-induct P rule*: *pi.strong-inducts*, *auto intro*: *Refl SumIdemp SumPres′ Trans*)

**lemma** *summandsZero′*:
  **fixes** $P$ :: *pi*

  **assumes** *summP*: *summands* $P = \{\}$
  **and** *Puhnf*: *uhnf P*

  **shows** $P = \mathbf{0}$
**proof** −
  **from** *Puhnf* **have** *hnf P* **by**(*simp add*: *uhnf-def*)
  **with** *summP* **show** *?thesis* **by**(*rule summandsZero*)
**qed**

**lemma** *summandEquiv*:
  **fixes** $P$ :: *pi*
  **and** $Q$ :: *pi*

  **assumes** *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** −
  **from** *finiteSummands assms* **show** *?thesis*
  **proof**(*induct F==summands P arbitrary*: *P Q rule*: *finite-induct*)
    **case**(*empty P Q*)
    **have** *PEmpty*: $\{\}$ = *summands P* **by** *fact*
    **moreover have** $\forall Q'\in$*summands Q*. $\exists P'\in$*summands P*. $Q' \equiv_e P'$ **by** *fact*
    **ultimately have** *QEmpty*: *summands* $Q = \{\}$ **by** *simp*

    **have** $P = \mathbf{0}$
    **proof** −
      **have** *uhnf P* **by** *fact*
      **with** *PEmpty* **show** *?thesis* **by**(*blast intro*: *summandsZero′*)
    **qed**
    **moreover have** $Q = \mathbf{0}$
    **proof** −
      **have** *uhnf Q* **by** *fact*
      **with** *QEmpty* **show** *?thesis* **by**(*blast intro*: *summandsZero′*)
    **qed**
    **ultimately show** *?case* **by**(*blast intro*: *Refl*)
  **next**
    **case**(*insert P′ F P Q*)
    **have** *Phnf*: *uhnf P* **by** *fact*
    **have** *Qhnf*: *uhnf Q* **by** *fact*

**have** *IH*: $\bigwedge P\ Q.\ [\![F = summands\ P;\ uhnf\ P;\ uhnf\ Q;\ \forall P' \in summands\ P.$
$\exists Q' \in summands\ Q.\ P' \equiv_e Q';$
$\forall Q' \in summands\ Q.\ \exists P' \in summands\ P.\ Q' \equiv_e P']\!] \Longrightarrow P \equiv_e Q$
**by** *fact*
**have** *PeqQ*: $\forall P' \in summands\ P.\ \exists Q' \in summands\ Q.\ P' \equiv_e Q'$ **by** *fact*
**have** *QeqP*: $\forall Q' \in summands\ Q.\ \exists P' \in summands\ P.\ Q' \equiv_e P'$ **by** *fact*

**have** *PSumm*: $insert\ P'\ F = summands\ P$ **by** *fact*
**hence** $P'SummP$: $P' \in summands\ P$ **by** *auto*

**with** *Phnf* **obtain** $P''$ **where** $P'P''eqP$: $P' \oplus P'' \equiv_e P$
**and** $P''Summ$: $summands\ P'' = summands\ P - \{P''\ |P''.$
$P'' \in summands\ P \wedge P'' \equiv_e P'\}$
**and** $P''hnf$: $uhnf\ P''$
**by**(*blast dest*: *pullSummand*)

**from** *PeqQ* $P'SummP$ **obtain** $Q'$ **where** $Q'SummQ$: $Q' \in summands\ Q$ **and**
$P'eqQ'$: $P' \equiv_e Q'$ **by** *blast*

**from** $Q'SummQ$ *Qhnf* **obtain** $Q''$ **where** $Q'Q''eqQ$: $Q' \oplus Q'' \equiv_e Q$
**and** $Q''Summ$: $summands\ Q'' = summands\ Q - \{Q''$
$|Q''.\ Q'' \in summands\ Q \wedge Q'' \equiv_e Q'\}$
**and** $Q''hnf$: $uhnf\ Q''$
**by**(*blast dest*: *pullSummand*)

**have** $FeqP''$: $F = summands\ P''$
**proof** $-$
**have** $P' \notin F$ **by** *fact*
**with** $P''Summ$ *PSumm* *hnfSummandsRemove*[*OF* $P'SummP$ *Phnf*] **show**
*?thesis* **by** *blast*
**qed**

**moreover have** $\forall P' \in summands\ P''.\ \exists Q' \in summands\ Q''.\ P' \equiv_e Q'$
**proof**(*rule ballI*)
**fix** $P'''$
**assume** $P'''Summ$: $P''' \in summands\ P''$
**with** $P''Summ$ **have** $P''' \in summands\ P$ **by** *simp*
**with** *PeqQ* **obtain** $Q'''$ **where** $Q'''Summ$: $Q''' \in summands\ Q$ **and** $P'''eqQ'''$:
$P''' \equiv_e Q'''$ **by** *blast*
**have** $Q''' \in summands\ Q''$
**proof** $-$
**from** $P'''Summ$ $P''Summ$ **have** $\neg(P''' \equiv_e P')$ **by** *simp*
**with** $P'eqQ'$ $P'''eqQ'''$ **have** $\neg(Q''' \equiv_e Q')$ **by**(*blast intro*: *Trans Sym*)
**with** $Q''Summ$ $Q'''Summ$ **show** *?thesis* **by** *simp*
**qed**

**with** $P'''eqQ'''$ **show** $\exists Q' \in summands\ Q''.\ P''' \equiv_e Q'$ **by** *blast*
**qed**

**moreover have** $\forall\, Q' \in \mathit{summands}\ Q''.\ \exists\, P' \in \mathit{summands}\ P''.\ Q' \equiv_e P'$
**proof**(*rule ballI*)
**fix** $Q'''$
**assume** $Q'''Summ$: $Q''' \in \mathit{summands}\ Q''$
**with** $Q''Summ$ **have** $Q''' \in \mathit{summands}\ Q$ **by** *simp*
**with** *QeqP* **obtain** $P'''$ **where** $P'''Summ$: $P''' \in \mathit{summands}\ P$
**and** $Q'''eqP'''$: $Q''' \equiv_e P'''$ **by** *blast*
**have** $P''' \in \mathit{summands}\ P''$
**proof** $-$
**from** $Q'''Summ\ Q''Summ$ **have** $\neg(Q''' \equiv_e Q')$ **by** *simp*
**with** $P'eqQ'\ Q'''eqP'''$ **have** $\neg(P''' \equiv_e P')$ **by**(*blast intro*: *Trans*)
**with** $P''Summ\ P'''Summ$ **show** *?thesis* **by** *simp*
**qed**
**with** $Q'''eqP'''$ **show** $\exists\, P' \in \mathit{summands}\ P''.\ Q''' \equiv_e P'$ **by** *blast*
**qed**

**ultimately have** $P''eqQ''$: $P'' \equiv_e Q''$ **using** $P''hnf\ Q''hnf$ **by**(*rule-tac IH*)

**from** $P'P''eqP$ **have** $P \equiv_e P' \oplus P''$ **by**(*rule Sym*)
**moreover from** $P'eqQ'\ P''eqQ''$ **have** $P' \oplus P'' \equiv_e Q' \oplus Q''$ **by**(*rule SumPres'*)
**ultimately show** *?case* **using** $Q'Q''eqQ$ **by**(*blast intro*: *Trans*)
**qed**
**qed**

**lemma** *validSubst*[*simp*]:
**fixes** $P$ :: *pi*
**and** $a$ :: *name*
**and** $b$ :: *name*
**and** $p$ :: *pi*

**shows** $\mathit{valid}(P[a::=b]) = \mathit{valid}\ P$
**by**(*nominal-induct P avoiding*: *a b rule*: *pi.strong-inducts*, *auto*)

**lemma** *validOutputTransition*:
**fixes** $P$ :: *pi*
**and** $a$ :: *name*
**and** $b$ :: *name*
**and** $P'$ :: *pi*

**assumes** $P \longmapsto a[b] \prec P'$
**and** *valid P*

**shows** *valid* $P'$
**using** *assms*
**by**(*nominal-induct rule*: *outputInduct*, *auto*)

**lemma** *validInputTransition*:
**fixes** $P$ :: *pi*

**and** $a$ :: *name*
**and** $x$ :: *name*
**and** $P'$ :: *pi*

**assumes** *PTrans*: $P \longmapsto a{<}x{>} \prec P'$
**and** *validP*: *valid* $P$

**shows** *valid* $P'$
**proof** −
**have** *Goal*: $\bigwedge P\ a\ x\ P'.\ [\![P \longmapsto a{<}x{>} \prec P';\ x \sharp P;\ \textit{valid}\ P]\!] \Longrightarrow \textit{valid}\ P'$
**proof** −
**fix** $P\ a\ x\ P'$
**assume** $P \longmapsto a{<}x{>} \prec P'$ **and** $x \sharp P$ **and** *valid* $P$
**thus** *valid* $P'$
**by**(*nominal-induct rule*: *inputInduct*, *auto*)
**qed**
**obtain** $y$::*name* **where** *yFreshP*: $y \sharp P$ **and** *yFreshP'*: $y \sharp P'$
**by**(*rule-tac name-exists-fresh*[*of* $(P, P')$], *auto simp add*: *fresh-prod*)
**from** *yFreshP' PTrans* **have** $P \longmapsto a{<}y{>} \prec [(x, y)] \cdot P'$ **by**(*simp add*: *alphaBoundResidual*)
**hence** *valid* $([(x, y)] \cdot P')$ **using** *yFreshP validP* **by**(*rule Goal*)
**thus** *valid* $P'$ **by** *simp*
**qed**

**lemma** *validBoundOutputTransition*:
**fixes** $P$ :: *pi*
**and** $a$ :: *name*
**and** $x$ :: *name*
**and** $P'$ :: *pi*

**assumes** *PTrans*: $P \longmapsto a{<}\nu x{>} \prec P'$
**and** *validP*: *valid* $P$

**shows** *valid* $P'$
**proof** −
**have** *Goal*: $\bigwedge P\ a\ x\ P'.\ [\![P \longmapsto a{<}\nu x{>} \prec P';\ x \sharp P;\ \textit{valid}\ P]\!] \Longrightarrow \textit{valid}\ P'$
**proof** −
**fix** $P\ a\ x\ P'$
**assume** $P \longmapsto a{<}\nu x{>} \prec P'$ **and** $x \sharp P$ **and** *valid* $P$
**thus** *valid* $P'$
**apply**(*nominal-induct rule*: *boundOutputInduct*, *auto*)
**proof** −
**fix** $P\ a\ x\ P'$
**assume** $P \longmapsto (a::\textit{name})[x] \prec P'$ **and** *valid* $P$
**thus** *valid* $P'$
**by**(*nominal-induct rule*: *outputInduct*, *auto*)
**qed**
**qed**
**obtain** $y$::*name* **where** *yFreshP*: $y \sharp P$ **and** *yFreshP'*: $y \sharp P'$

**by**(*rule-tac name-exists-fresh*[*of* $(P, P')$], *auto simp add*: *fresh-prod*)
**from** *yFreshP′ PTrans* **have** $P \longmapsto a{<}\nu y{>} \prec [(x, y)] \cdot P'$ **by**(*simp add*: *alphaBoundResidual*)
**hence** *valid* $([(x, y)] \cdot P')$ **using** *yFreshP validP* **by**(*rule Goal*)
**thus** *valid P′* **by** *simp*
**qed**

**lemma** *validTauTransition*:
**fixes** $P$ :: *pi*
**and** $P'$ :: *pi*

**assumes** *PTrans*: $P \longmapsto \tau \prec P'$
**and** *validP*: *valid P*

**shows** *valid P′*
**using** *assms*
**by**(*nominal-induct rule*: *tauInduct*, *auto dest*: *validOutputTransition validInputTransition validBoundOutputTransition*)

**lemmas** *validTransition* = *validInputTransition validOutputTransition validTauTransition validBoundOutputTransition*

**lemma** *validSummand*:
**fixes** $P$ :: *pi*
**and** $P'$ :: *pi*
**and** $a$ :: *name*
**and** $b$ :: *name*
**and** $x$ :: *name*

**assumes** *valid P*
**and** *hnf P*

**shows** $\tau.(P') \in$ *summands P* $\Longrightarrow$ *valid P′*
**and** $a\{b\}.P' \in$ *summands P* $\Longrightarrow$ *valid P′*
**and** $a{<}x{>}.P' \in$ *summands P* $\Longrightarrow$ *valid P′*
**and** $[\![a \neq x;\ {<}\nu x{>}a\{x\}.P' \in \textit{summands } P]\!] \Longrightarrow$ *valid P′*
**proof** −
**assume** $\tau.(P') \in$ *summands P*
**with** *assms* **show** *valid P′* **by**(*force intro*: *validTauTransition simp add*: *summandTransition*)
**next**
**assume** $a\{b\}.P' \in$ *summands P*
**with** *assms* **show** *valid P′* **by**(*force intro*: *validOutputTransition simp add*: *summandTransition*)
**next**
**assume** $a{<}x{>}.P' \in$ *summands P*
**with** *assms* **show** *valid P′* **by**(*force intro*: *validInputTransition simp add*: *summandTransition*)
**next**

**assume** $<\nu x>a\{x\}.P' \in$ *summands* $P$ **and** $a \neq x$
**with** *assms* **show** *valid* $P'$
**by**(*force intro*: *validBoundOutputTransition simp add*: *summandTransition*[*THEN sym*])
**qed**

**lemma** *validExpand*:
**fixes** $P$ :: *pi*
**and** $Q$ :: *pi*

**assumes** *valid* $P$
**and** *valid* $Q$
**and** *uhnf* $P$
**and** *uhnf* $Q$

**shows** $\forall R \in$ (*expandSet* $P$ $Q$). *uhnf* $R \wedge$ *valid* $R$
**proof** −
**from** *assms* **have** *hnf* $P$ **and** *hnf* $Q$ **by**(*simp add*: *uhnf-def*)+
**with** *assms* **show** *?thesis*
**apply**(*auto simp add*: *expandSet-def*)
**apply**(*force dest*: *validSummand simp add*: *uhnf-def*)
**apply**(*force dest*: *validSummand*)
**apply**(*force dest*: *validSummand simp add*: *uhnf-def*)
**apply**(*force dest*: *validSummand*)
**apply**(*force dest*: *validSummand simp add*: *uhnf-def*)
**apply**(*force dest*: *validSummand*)
**apply**(*force dest*: *validSummand simp add*: *uhnf-def*)
**apply**(*force dest*: *validSummand*)
**apply**(*force dest*: *validSummand simp add*: *uhnf-def*)
**apply**(*force dest*: *validSummand*)
**apply**(*force dest*: *validSummand simp add*: *uhnf-def*)
**apply**(*force dest*: *validSummand*)
**apply**(*subgoal-tac* $a \neq x$)
**apply**(*force dest*: *validSummand simp add*: *uhnf-def*)
**apply** *blast*
**apply**(*subgoal-tac* $a \neq x$)
**apply**(*drule-tac validSummand*(*4*)) **apply** *assumption*+
**apply** *blast*
**apply**(*subgoal-tac* $a \neq x$)
**apply**(*drule-tac validSummand*(*4*)[**where** $P=Q$]) **apply** *assumption*+
**apply**(*force dest*: *validSummand simp add*: *uhnf-def*)
**apply** *blast*
**apply**(*subgoal-tac* $a \neq x$)
**apply**(*drule-tac validSummand*(*4*)[**where** $P=Q$]) **apply** *assumption*+
**apply** *blast*
**apply**(*force dest*: *validSummand simp add*: *uhnf-def*)
**apply**(*force dest*: *validSummand*)
**apply**(*force dest*: *validSummand simp add*: *uhnf-def*)
**apply**(*force simp add*: *uhnf-def*)

**apply**(*force dest*: *validSummand*)
**apply**(*force dest*: *validSummand*)
**apply**(*force simp add*: *uhnf-def*)
**apply**(*force dest*: *validSummand*)
**apply**(*subgoal-tac* $a \neq y$)
**apply**(*drule-tac validSummand*(*4*)[**where** $P=Q$]) **apply** *assumption+*
**apply** *blast*
**apply**(*force dest*: *validSummand simp add*: *uhnf-def*)
**apply**(*subgoal-tac* $a \neq y$)
**apply**(*drule-tac validSummand*(*4*)) **apply** *assumption+*
**apply** *blast*
**by**(*force dest*: *validSummand*)
**qed**

**lemma** *expandComplete*:
**fixes** $F$ :: *pi set*

**assumes** *finite* $F$

**shows** $\exists P.\ (P,\ F) \in$ *sumComposeSet*
**using** *assms*
**proof**(*induct F rule*: *finite-induct*)
**case** *empty*
**have** $(\mathbf{0},\ \{\}) \in$ *sumComposeSet* **by**(*rule sumComposeSet.empty*)
**thus** *?case* **by** *blast*
**next**
**case**(*insert Q F*)
**have** $\exists P.\ (P,\ F) \in$ *sumComposeSet* **by** *fact*
**then obtain** $P$ **where** $(P,\ F) \in$ *sumComposeSet* **by** *blast*
**moreover have** $Q \in$ *insert Q F* **by** *simp*
**moreover have** $Q \notin F$ **by** *fact*
**ultimately have** $(P \oplus Q,$ *insert Q F*$) \in$ *sumComposeSet*
**by**(*force intro*: *sumComposeSet.insert*)
**thus** *?case* **by** *blast*
**qed**

**lemma** *expandDepth*:
**fixes** $F$ :: *pi set*
**and** $P$ :: *pi*
**and** $Q$ :: *pi*

**assumes** $(P,\ F) \in$ *sumComposeSet*
**and** $F \neq \{\}$

**shows** $\exists Q \in F.$ *depth* $P \leq$ *depth* $Q \wedge (\forall R \in F.$ *depth* $R \leq$ *depth* $Q)$
**using** *assms*
**proof**(*induct arbitrary*: *Q rule*: *sumComposeSet.induct*)
**case** *empty*
**have** $(\{\}$::*pi set*$) \neq \{\}$ **by** *fact*

**hence** *False* **by** *simp*
**thus** *?case* **by** *simp*
**next**
**case**(*insert Q S P*)
**have** *QinS*: $Q \in S$ **by** *fact*
**show** *?case*
**proof**(*case-tac* $(S - \{Q\}) = \{\}$)
**assume** $(S - \{Q\}) = \{\}$
**with** *QinS* **have** *SeqQ*: $S = \{Q\}$ **by** *auto*
**have** $(P, S - \{Q\}) \in sumComposeSet$ **by** *fact*
**with** *SeqQ* **have** $(P, \{\}) \in sumComposeSet$ **by** *simp*
**hence** $P = \mathbf{0}$ **apply** − **by**(*ind-cases* $(P, \{\}) \in sumComposeSet$, *auto*)
**with** *QinS SeqQ* **show** *?case* **by** *simp*
**next**
**assume** $(S - \{Q\}) \neq \{\}$
**moreover have** $(S - \{Q\}) \neq \{\} \Longrightarrow \exists\, Q' \in (S - \{Q\}).\ depth\ P \leq depth\ Q' \wedge (\forall\, R \in (S - \{Q\}).\ depth\ R \leq depth\ Q')$ **by** *fact*
**ultimately obtain** $Q'$ **where** *Q'inS*: $Q' \in S - \{Q\}$ **and** *PQ'depth*: $depth\ P \leq depth\ Q'$ **and** *All*: $\forall\, R \in (S - \{Q\}).\ depth\ R \leq depth\ Q'$ **by** *auto*
**show** *?case*
**proof**(*case-tac* $Q = Q'$)
**assume** $Q = Q'$
**with** *PQ'depth All QinS* **show** *?case* **by** *auto*
**next**
**assume** *QineqQ'*: $Q \neq Q'$
**show** *?case*
**proof**(*case-tac* $depth\ Q \leq depth\ Q'$)
**assume** $depth\ Q \leq depth\ Q'$
**with** *QineqQ' PQ'depth All Q'inS* **show** *?thesis* **by** *force*
**next**
**assume** $\neg\ depth\ Q \leq depth\ Q'$
**with** *QineqQ' PQ'depth All Q'inS QinS* **show** *?thesis* **apply** *auto*
**apply**(*rule-tac* $x{=}Q$ **in** *bexI*)
**apply** *auto*
**apply**(*case-tac* $R{=}Q$)
**apply** *auto*
**apply**(*erule-tac* $x{=}R$ **in** *ballE*)
**by** *auto*
**qed**
**qed**
**qed**
**qed**

**lemma** *depthSubst*[*simp*]:
**fixes** $P :: pi$
**and** $a :: name$
**and** $b :: name$

**shows** $depth(P[a{::=}b]) = depth\ P$

**by**(*nominal-induct P avoiding*: *a b rule*: *pi.strong-inducts*, *auto*)

**lemma** *depthTransition*:
  **fixes** $P$ :: *pi*
  **and** $a$ :: *name*
  **and** $b$ :: *name*
  **and** $P'$ :: *pi*

  **assumes** *Phnf*: *hnf P*

  **shows** $P \longmapsto a[b] \prec P' \Longrightarrow \mathit{depth}\ P' < \mathit{depth}\ P$
  **and** $P \longmapsto a{<}x{>} \prec P' \Longrightarrow \mathit{depth}\ P' < \mathit{depth}\ P$
  **and** $P \longmapsto \tau \prec P' \Longrightarrow \mathit{depth}\ P' < \mathit{depth}\ P$
  **and** $P \longmapsto a{<}\nu x{>} \prec P' \Longrightarrow \mathit{depth}\ P' < \mathit{depth}\ P$
**proof** −
  **assume** $P \longmapsto a[b] \prec P'$
  **thus** $\mathit{depth}\ P' < \mathit{depth}\ P$ **using** *assms*
    **by**(*nominal-induct rule*: *outputInduct*, *auto*)
**next**
  **assume** *Trans*: $P \longmapsto a{<}x{>} \prec P'$
  **have** *Goal*: $\bigwedge P\ a\ x\ P'.\ [\![P \longmapsto a{<}x{>} \prec P';\ x \sharp P;\ \mathit{hnf}\ P]\!] \Longrightarrow \mathit{depth}\ P' < \mathit{depth}$
$P$
  **proof** −
    **fix** $P\ a\ x\ P'$
    **assume** $P \longmapsto a{<}x{>} \prec P'$ **and** $x \sharp P$ **and** *hnf P*
    **thus** $\mathit{depth}\ P' < \mathit{depth}\ P$
      **by**(*nominal-induct rule*: *inputInduct*, *auto*)
  **qed**
  **obtain** *y*::*name* **where** *yFreshP*: $y \sharp P$ **and** *yFreshP'*: $y \sharp P'$
    **by**(*rule-tac name-exists-fresh*[*of* $(P, P')$], *auto simp add*: *fresh-prod*)
  **from** *yFreshP' Trans* **have** $P \longmapsto a{<}y{>} \prec [(x, y)] \cdot P'$ **by**(*simp add*: *alphaBound-Residual*)
  **hence** $\mathit{depth}\ ([(x, y)] \cdot P') < \mathit{depth}\ P$ **using** *yFreshP Phnf* **by**(*rule Goal*)
  **thus** $\mathit{depth}\ P' < \mathit{depth}\ P$ **by** *simp*
**next**
  **assume** $P \longmapsto \tau \prec P'$
  **thus** $\mathit{depth}\ P' < \mathit{depth}\ P$ **using** *assms*
    **by**(*nominal-induct rule*: *tauInduct*, *auto simp add*: *uhnf-def*)
**next**
  **assume** *Trans*: $P \longmapsto a{<}\nu x{>} \prec P'$
  **have** *Goal*: $\bigwedge P\ a\ x\ P'.\ [\![P \longmapsto a{<}\nu x{>} \prec P';\ x \sharp P;\ \mathit{hnf}\ P]\!] \Longrightarrow \mathit{depth}\ P' < \mathit{depth}$
$P$
  **proof** −
    **fix** $P\ a\ x\ P'$
    **assume** $P \longmapsto a{<}\nu x{>} \prec P'$ **and** $x \sharp P$ **and** *hnf P*
    **thus** $\mathit{depth}\ P' < \mathit{depth}\ P$
      **by**(*nominal-induct rule*: *boundOutputInduct*,
        *auto elim*: *outputCases simp add*: *residual.inject*)
  **qed**

**obtain** $y$::*name* **where** *yFreshP*: $y \sharp P$ **and** *yFreshP'*: $y \sharp P'$
**by**(*rule-tac name-exists-fresh*[*of* $(P, P')$], *auto simp add*: *fresh-prod*)
**from** *yFreshP' Trans* **have** $P \longmapsto a{<}\nu y{>} \prec [(x, y)] \cdot P'$ **by**(*simp add*: *alphaBoundResidual*)
**hence** *depth* $([(x, y)] \cdot P') <$ *depth* $P$ **using** *yFreshP Phnf* **by**(*rule Goal*)
**thus** *depth* $P' <$ *depth* $P$ **by** *simp*
**qed**

**lemma** *maxExpandDepth*:
**fixes** $P$ :: *pi*
**and** $Q$ :: *pi*
**and** $R$ :: *pi*

**assumes** $R \in$ *expandSet* $P$ $Q$
**and** *hnf* $P$
**and** *hnf* $Q$

**shows** *depth* $R \leq$ *depth*$(P \parallel Q)$
**using** *assms*
**apply**(*auto simp add*: *expandSet-def summandTransition*[*THEN sym*] *dest*: *depthTransition*)
**apply**(*subgoal-tac* $a \neq x$)
**apply**(*simp add*: *summandTransition*[*THEN sym*])
**apply**(*force dest*: *depthTransition*)
**apply** *blast*
**apply**(*subgoal-tac* $a \neq x$)
**apply**(*simp add*: *summandTransition*[*THEN sym*])
**apply**(*force dest*: *depthTransition*)
**apply** *blast*
**apply**(*force dest*: *depthTransition*)
**apply**(*force dest*: *depthTransition*)
**apply**(*subgoal-tac* $a \neq y$)
**apply**(*simp add*: *summandTransition*[*THEN sym*])
**apply**(*force dest*: *depthTransition*)
**apply** *blast*
**apply**(*subgoal-tac* $a \neq y$)
**apply**(*simp add*: *summandTransition*[*THEN sym*])
**apply**(*force dest*: *depthTransition*)
**by** *blast*

**lemma** *expandDepth'*:
**fixes** $P$ :: *pi*
**and** $Q$ :: *pi*

**assumes** *Phnf*: *hnf* $P$
**and** *Qhnf*: *hnf* $Q$

**shows** $\exists R$. $(R$, *expandSet* $P$ $Q) \in$ *sumComposeSet* $\wedge$ *depth* $R \leq$ *depth*$(P \parallel Q)$
**proof**(*case-tac* *expandSet* $P$ $Q = \{\}$)

**assume** *expandSet P Q* = {}
**with** *Phnf Qhnf* **show** *?thesis* **by**(*auto intro*: *sumComposeSet.empty*)
**next**
**assume** *expandSet P Q* $\neq$ {}

**moreover from** *Phnf Qhnf finiteExpand* **obtain** *R* **where** *TSC*: (*R*, *expandSet P Q*) $\in$ *sumComposeSet*
**by**(*blast dest*: *expandComplete*)
**ultimately obtain** *T* **where** *T* $\in$ *expandSet P Q*
**and** *depth R* $\leq$ *depth T*
**by**(*blast dest*: *expandDepth*)
**with** *Phnf Qhnf* **have** *depth R* $\leq$ *depth*(*P* $\parallel$ *Q*)
**by**(*force dest*: *maxExpandDepth*)
**with** *TSC* **show** *?thesis* **by** *blast*
**qed**

**lemma** *validToHnf*:
**fixes** *P* :: *pi*

**assumes** *valid P*

**shows** $\exists$ *Q*. *uhnf Q* $\wedge$ *valid Q* $\wedge$ *Q* $\equiv_e$ *P* $\wedge$ (*depth Q*) $\leq$ (*depth P*)
**proof** −
**have** *MatchGoal*: $\bigwedge$*a b P Q*. ⟦*uhnf Q*; *valid Q*; *Q* $\equiv_e$ *P*; *depth Q* $\leq$ *depth P*⟧ $\Longrightarrow$
$\exists$ *Q*. *uhnf Q* $\wedge$ *valid Q* $\wedge$ *Q* $\equiv_e$ [*a*⌢*b*]*P* $\wedge$ *depth Q* $\leq$ *depth* ([*a*⌢*b*]*P*)
**proof** −
**fix** *a b P Q*
**assume** *Qhnf*: *uhnf Q* **and** *validQ*: *valid Q* **and** *QeqP*: *Q* $\equiv_e$ *P*
**and** *QPdepth*: *depth Q* $\leq$ *depth P*
**show** $\exists$ *Q*. *uhnf Q* $\wedge$ *valid Q* $\wedge$ *Q* $\equiv_e$ [*a*⌢*b*]*P* $\wedge$ *depth Q* $\leq$ *depth* ([*a*⌢*b*]*P*)
**proof**(*case-tac a* = *b*)
**assume** *a* = *b*
**with** *QeqP* **have** *Q* $\equiv_e$ [*a*⌢*b*]*P* **by**(*blast intro*: *Sym Trans equiv.Match*)
**with** *Qhnf validQ QPdepth* **show** *?thesis* **by** *force*
**next**
**assume** *a* $\neq$ *b*
**hence** **0** $\equiv_e$ [*a*⌢*b*]*P* **by**(*blast intro*: *Sym Match'*)
**moreover have** *uhnf* **0** **by**(*simp add*: *uhnf-def*)
**ultimately show** *?thesis* **by** *force*
**qed**
**qed**

**from** *assms* **show** *?thesis*
**proof**(*nominal-induct P rule*: *pi.strong-inducts*)
**case** *PiNil*
**have** *uhnf* **0** **by**(*simp add*: *uhnf-def*)
**moreover have** *valid* **0** **by** *simp*
**moreover have** **0** $\equiv_e$ **0** **by**(*rule Refl*)

**moreover have** $(depth\ \mathbf{0}) \leq (depth\ \mathbf{0})$ **by** *simp*
**ultimately show** *?case* **by** *blast*
**next**
**case**$(Output\ a\ b\ P)$
**have** $uhnf\ (a\{b\}.P)$ **by**$(simp\ add$: $uhnf\text{-}def)$
**moreover have** $valid(a\{b\}.P)$ **by** *fact*
**moreover have** $a\{b\}.P \equiv_e a\{b\}.P$ **by**$(rule\ Refl)$
**moreover have** $(depth\ (a\{b\}.P)) \leq (depth\ (a\{b\}.P))$ **by** *simp*
**ultimately show** *?case* **by** *blast*
**next**
**case**$(Tau\ P)$
**have** $uhnf\ (\tau.(P))$ **by**$(simp\ add$: $uhnf\text{-}def)$
**moreover have** $valid\ (\tau.(P))$ **by** *fact*
**moreover have** $\tau.(P) \equiv_e \tau.(P)$ **by**$(rule\ Refl)$
**moreover have** $(depth\ (\tau.(P))) \leq (depth\ (\tau.(P)))$ **by** *simp*
**ultimately show** *?case* **by** *blast*
**next**
**case**$(Input\ a\ x\ P)$
**have** $uhnf\ (a<x>.P)$ **by**$(simp\ add$: $uhnf\text{-}def)$
**moreover have** $valid\ (a<x>.P)$ **by** *fact*
**moreover have** $a<x>.P \equiv_e a<x>.P$ **by**$(rule\ Refl)$
**moreover have** $(depth\ (a<x>.P)) \leq (depth\ (a<x>.P))$ **by** *simp*
**ultimately show** *?case* **by** *blast*
**next**
**case**$(Match\ a\ b\ P)$
**have** $valid\ ([a\frown b]P)$ **by** *fact*
**hence** $valid\ P$ **by** *simp*
**moreover have** $valid\ P \Longrightarrow \exists\, Q.\ uhnf\ Q \wedge valid\ Q \wedge Q \equiv_e P \wedge depth\ Q \leq depth\ P$ **by** *fact*
**ultimately obtain** $Q$ **where** $Qhnf$: $uhnf\ Q$ **and** $validQ$: $valid\ Q$ **and** $QeqP$: $Q \equiv_e P$
**and** $QPdepth$: $depth\ Q \leq depth\ P$ **by** *blast*
**thus** *?case* **by**$(rule\ MatchGoal)$
**next**
**case**$(Mismatch\ a\ b\ P)$
**have** $valid\ ([a{\neq}b]P)$ **by** *fact*
**hence** $valid\ P$ **by** *simp*
**moreover have** $valid\ P \Longrightarrow \exists\, Q.\ uhnf\ Q \wedge valid\ Q \wedge Q \equiv_e P \wedge depth\ Q \leq depth\ P$ **by** *fact*
**ultimately obtain** $Q$ **where** $Qhnf$: $uhnf\ Q$ **and** $validQ$: $valid\ Q$ **and** $QeqP$: $Q \equiv_e P$
**and** $QPdepth$: $depth\ Q \leq depth\ P$ **by** *blast*
**show** *?case*
**proof**$(case\text{-}tac\ a = b)$
**assume** $a = b$
**hence** $\mathbf{0} \equiv_e [a{\neq}b]P$ **by**$(blast\ intro$: $Sym\ Mismatch')$
**moreover have** $uhnf\ \mathbf{0}$ **by**$(simp\ add$: $uhnf\text{-}def)$
**ultimately show** *?case* **by** *force*
**next**

**assume** $a \neq b$
**with** *QeqP* **have** $Q \equiv_e [a{\neq}b]P$ **by**(*blast intro*: *Sym Trans equiv.Mismatch*)
**with** *Qhnf validQ QPdepth* **show** *?case* **by** *force*
**qed**
**next**
**case**(*Sum P Q*)
**have** *valid*$(P \oplus Q)$ **by** *fact*
**hence** *validP*: *valid P* **and** *validQ*: *valid Q* **by** *simp*+

**have** $\exists P'$. *uhnf* $P' \wedge$ *valid* $P' \wedge P' \equiv_e P \wedge$ (*depth* $P'$) $\leq$ (*depth* $P$)
**proof** −
**have** *valid* $P \Longrightarrow \exists P'$. *uhnf* $P' \wedge$ *valid* $P' \wedge P' \equiv_e P \wedge$ (*depth* $P'$) $\leq$ (*depth* $P$) **by** *fact*
**with** *validP* **show** *?thesis* **by** *simp*
**qed**
**then obtain** $P'$ **where** $P'$*hnf*: *uhnf* $P'$ **and** $P'$*eqP*: $P' \equiv_e P$ **and** *validP'*: *valid* $P'$
**and** $P'$*depth*: (*depth* $P'$) $\leq$ (*depth* $P$) **by** *blast*

**have** $\exists Q'$. *uhnf* $Q' \wedge$ *valid* $Q' \wedge Q' \equiv_e Q \wedge$ (*depth* $Q'$) $\leq$ (*depth* $Q$)
**proof** −
**have** *valid* $Q \Longrightarrow \exists Q'$. *uhnf* $Q' \wedge$ *valid* $Q' \wedge Q' \equiv_e Q \wedge$ (*depth* $Q'$) $\leq$ (*depth* $Q$) **by** *fact*
**with** *validQ* **show** *?thesis* **by** *simp*
**qed**

**then obtain** $Q'$ **where** $Q'$*hnf*: *uhnf* $Q'$ **and** $Q'$*eqQ*: $Q' \equiv_e Q$ **and** *validQ'*: *valid* $Q'$
**and** $Q'$*depth*: (*depth* $Q'$) $\leq$ (*depth* $Q$) **by** *blast*

**from** $P'$*hnf* $Q'$*hnf* *validP'* *validQ'* **obtain** $R$ **where** *Rhnf*: *uhnf* $R$ **and** *validR*: *valid* $R$
**and** $P'Q'$*eqR*: $P' \oplus Q' \equiv_e R$
**and** *Rdepth*: *depth* $R \leq$ *depth*$(P' \oplus Q')$
**apply**(*drule-tac uhnfSum*) **apply** *assumption*+ **by** *blast*

**from** *validP'* *validQ'* **have** *valid*$(P' \oplus Q')$ **by** *simp*
**from** $P'$*eqP* $Q'$*eqQ* $P'Q'$*eqR* **have** $P \oplus Q \equiv_e R$ **by**(*blast intro*: *Sym SumPres' Trans*)
**moreover from** *Rdepth* $P'$*depth* $Q'$*depth* **have** *depth* $R \leq$ *depth*$(P \oplus Q)$ **by** *auto*
**ultimately show** *?case* **using** *validR Rhnf* **by**(*blast intro*: *Sym*)
**next**
**case**(*Par P Q*)
**have** *valid*$(P \parallel Q)$ **by** *fact*

**hence** *validP*: *valid P* **and** *validQ*: *valid Q* **by** *simp*+
**have** $\exists P'$. *uhnf* $P' \wedge$ *valid* $P' \wedge P' \equiv_e P \wedge$ (*depth* $P'$) $\leq$ (*depth* $P$)
**proof** −

**have** *valid* $P \Longrightarrow \exists P'.$ *uhnf* $P' \wedge$ *valid* $P' \wedge P' \equiv_e P \wedge$ (*depth* $P'$) $\leq$ (*depth* $P$) **by** *fact*
**with** *validP* **show** *?thesis* **by** *simp*
**qed**
**then obtain** $P'$ **where** $P'$*hnf*: *uhnf* $P'$ **and** $P'$*eqP*: $P' \equiv_e P$ **and** *validP'*: *valid* $P'$
**and** $P'$*depth*: (*depth* $P'$) $\leq$ (*depth* $P$) **by** *blast*

**have** $\exists Q'.$ *uhnf* $Q' \wedge$ *valid* $Q' \wedge Q' \equiv_e Q \wedge$ (*depth* $Q'$) $\leq$ (*depth* $Q$)
**proof** −
**have** *valid* $Q \Longrightarrow \exists Q'.$ *uhnf* $Q' \wedge$ *valid* $Q' \wedge Q' \equiv_e Q \wedge$ (*depth* $Q'$) $\leq$ (*depth* $Q$) **by** *fact*
**with** *validQ* **show** *?thesis* **by** *simp*
**qed**

**then obtain** $Q'$ **where** $Q'$*hnf*: *uhnf* $Q'$ **and** $Q'$*eqQ*: $Q' \equiv_e Q$ **and** *validQ'*: *valid* $Q'$
**and** $Q'$*depth*: (*depth* $Q'$) $\leq$ (*depth* $Q$) **by** *blast*

**from** $P'$*hnf* $Q'$*hnf* **obtain** $R$ **where** *Exp*: ($R$, *expandSet* $P'$ $Q'$) $\in$ *sumComposeSet* **and** *Rdepth*: *depth* $R \leq$ *depth*($P' \parallel Q'$)
**by**(*force dest*: *expandDepth'* *simp add*: *uhnf-def*)

**from** *Exp* $P'$*hnf* $Q'$*hnf* **have** $P'Q'$*eqR*: $P' \parallel Q' \equiv_e R$ **by**(*force intro*: *Expand simp add*: *uhnf-def*)
**from** $P'$*hnf* $Q'$*hnf* *validP'* *validQ'* **have** $\forall P \in$ (*expandSet* $P'$ $Q'$). *uhnf* $P \wedge$ *valid* $P$ **by**(*blast dest*: *validExpand*)
**with** *Exp* **obtain** $R'$ **where** $R'$*hnf*: *uhnf* $R'$ **and** *validR'*: *valid* $R'$
**and** *ReqR'*: $R \equiv_e R'$
**and** $R'$*depth*: *depth* $R' \leq$ *depth* $R$
**by**(*blast dest*: *expandHnf*)
**from** $P'$*eqP* $Q'$*eqQ* $P'Q'$*eqR* *ReqR'* **have** $P \parallel Q \equiv_e R'$ **by**(*blast intro*: *Sym ParPres Trans*)
**moreover from** *Rdepth* $P'$*depth* $Q'$*depth* $R'$*depth* **have** *depth* $R' \leq$ *depth*($P \parallel Q$) **by** *auto*
**ultimately show** *?case* **using** *validR'* $R'$*hnf* **by**(*blast dest*: *Sym*)
**next**
**case**(*Res x P*)
**have** *valid* (<$\nu x$>$P$) **by** *fact*
**hence** *validP*: *valid* $P$ **by** *simp*
**moreover have** *valid* $P \Longrightarrow \exists Q.$ *uhnf* $Q \wedge$ *valid* $Q \wedge Q \equiv_e P \wedge$ *depth* $Q \leq$ *depth* $P$ **by** *fact*
**ultimately obtain** $Q$ **where** *Qhnf*: *uhnf* $Q$ **and** *validQ*: *valid* $Q$ **and** *QeqP*: $Q \equiv_e P$
**and** *QPDepth*: *depth* $Q \leq$ *depth* $P$ **by** *blast*

**from** *validP* **show** *?case*
**proof**(*nominal-induct P avoiding*: *x rule*: *pi.strong-inducts*)
**case** *PiNil*

**have** *uhnf* **0** **by**(*simp add*: *uhnf-def*)
**moreover have** *valid* **0** **by** *simp*
**moreover have** **0** $\equiv_e$ $<\nu x>$**0**
**proof** $-$
**have** $x \sharp$ **0** **by** *simp*
**thus** *?thesis* **by**(*blast intro*: *Sym ResFresh*)
**qed**
**moreover have** *depth* **0** $\leq$ *depth* ($<\nu x>$**0**) **by** *simp*
**ultimately show** *?case* **by** *blast*
**next**
**case**(*Output a b P*)
**have** *valid*($a\{b\}.P$) **by** *fact*
**hence** *validP*: *valid P* **by** *simp*
**show** *?case*
**proof**(*case-tac* $x=a$)
**assume** $x = a$
**moreover have** *uhnf* **0** **by**(*simp add*: *uhnf-def*)
**moreover have** *valid* **0** **by** *simp*
**moreover have** **0** $\equiv_e$ $<\nu x>x\{b\}.P$ **by**(*blast intro*: *ResOutput′ Sym*)
**moreover have** *depth* **0** $\leq$ *depth*($<\nu x>x\{b\}.P$) **by** *simp*
**ultimately show** *?case* **by** *blast*
**next**
**assume** *xineqa*: $x \neq a$
**show** *?case*
**proof**(*case-tac* $x=b$)
**assume** $x=b$
**moreover from** *xineqa* **have** *uhnf*($<\nu x>a\{x\}.P$) **by**(*force simp add*: *uhnf-def*)
**moreover from** *validP* **have** *valid*($<\nu x>a\{x\}.P$) **by** *simp*
**moreover have** $<\nu x>a\{x\}.P \equiv_e <\nu x>a\{x\}.P$ **by**(*rule Refl*)
**moreover have** *depth*($<\nu x>a\{x\}.P$) $\leq$ *depth*($<\nu x>a\{x\}.P$) **by** *simp*
**ultimately show** *?case* **by** *blast*
**next**
**assume** *xineqb*: $x \neq b$
**have** *uhnf*($a\{b\}.(<\nu x>P)$) **by**(*simp add*: *uhnf-def*)
**moreover from** *validP* **have** *valid*($a\{b\}.(<\nu x>P)$) **by** *simp*
**moreover from** *xineqa xineqb* **have** $a\{b\}.(<\nu x>P) \equiv_e <\nu x>a\{b\}.P$ **by**(*blast intro*: *ResOutput Sym*)
**moreover have** *depth*($a\{b\}.(<\nu x>P)$) $\leq$ *depth*($<\nu x>a\{b\}.P$) **by** *simp*
**ultimately show** *?case* **by** *blast*
**qed**
**qed**
**next**
**case**(*Tau P*)
**have** *valid*($\tau.(P)$) **by** *fact*
**hence** *validP*: *valid P* **by** *simp*

**have** *uhnf*($\tau.(<\nu x>P)$) **by**(*simp add*: *uhnf-def*)
**moreover from** *validP* **have** *valid*($\tau.(<\nu x>P)$) **by** *simp*

**moreover have** $\tau.(<\nu x>P) \equiv_e <\nu x>\tau.(P)$ **by**(*blast intro*: *ResTau Sym*)
**moreover have** $depth(\tau.(<\nu x>P)) \leq depth(<\nu x>\tau.(P))$ **by** *simp*
**ultimately show** *?case* **by** *blast*
**next**
**case**(*Input a y P*)
**have** $valid(a<y>.P)$ **by** *fact*
**hence** *validP*: *valid P* **by** *simp*
**have** $y \sharp x$ **by** *fact* **hence** *yineqx*: $y \neq x$ **by** *simp*
**show** *?case*
**proof**(*case-tac x=a*)
**assume** $x = a$
**moreover have** *uhnf* **0** **by**(*simp add*: *uhnf-def*)
**moreover have** *valid* **0** **by** *simp*
**moreover have** $\mathbf{0} \equiv_e <\nu x>x<y>.P$ **by**(*blast intro*: *ResInput' Sym*)
**moreover have** *depth* $\mathbf{0} \leq depth(<\nu x>x<y>.P)$ **by** *simp*
**ultimately show** *?case* **by** *blast*
**next**
**assume** *xineqa*: $x \neq a$
**have** $uhnf(a<y>.(<\nu x>P))$ **by**(*simp add*: *uhnf-def*)
**moreover from** *validP* **have** $valid(a<y>.(<\nu x>P))$ **by** *simp*
**moreover from** *xineqa yineqx* **have** $a<y>.(<\nu x>P) \equiv_e <\nu x>a<y>.P$
**by**(*blast intro*: *ResInput Sym*)
**moreover have** $depth(a<y>.(<\nu x>P)) \leq depth(<\nu x>a<y>.P)$ **by** *simp*
**ultimately show** *?case* **by** *blast*
**qed**
**next**
**case**(*Match a b P x*)
**have** $valid([a \frown b]P)$ **by** *fact* **hence** *valid P* **by** *simp*
**moreover have** $\bigwedge x.\ valid\ P \Longrightarrow \exists\, Q.\ uhnf\ Q \wedge valid\ Q \wedge Q \equiv_e <\nu x>P \wedge$
$depth\ Q \leq depth(<\nu x>P)$
**by** *fact*
**ultimately obtain** $Q$ **where** *Qhnf*: *uhnf Q* **and** *validQ*: *valid Q*
**and** *QeqP*: $Q \equiv_e (<\nu x>P)$
**and** *QPdepth*: $depth\ Q \leq depth(<\nu x>P)$
**by** *blast*
**show** *?case*
**proof**(*case-tac* $a = b$)
**assume** $a=b$
**moreover have** $Q \equiv_e <\nu x>[a \frown a]P$
**proof** −
**have** $P \equiv_e [a \frown a]P$ **by**(*blast intro*: *equiv.Match Sym*)
**hence** $<\nu x>P \equiv_e <\nu x>[a \frown a]P$ **by**(*rule ResPres*)
**with** *QeqP* **show** *?thesis* **by**(*blast intro*: *Trans*)
**qed**
**moreover from** *QPdepth* **have** $depth\ Q \leq depth(<\nu x>[a \frown a]P)$ **by** *simp*
**ultimately show** *?case* **using** *Qhnf validQ* **by** *blast*
**next**
**assume** *aineqb*: $a \neq b$
**have** *uhnf* **0** **by**(*simp add*: *uhnf-def*)

**moreover have** *valid* **0** **by** *simp*
**moreover have** **0** $\equiv_e$ $<\nu x>[a\frown b]P$
**proof** −
**from** *aineqb* **have** **0** $\equiv_e$ $[a\frown b]P$ **by**(*blast intro*: *Match′ Sym*)
**hence** $<\nu x>$**0** $\equiv_e$ $<\nu x>[a\frown b]P$ **by**(*rule ResPres*)
**thus** *?thesis* **by**(*blast intro*: *ResNil Trans Sym*)
**qed**
**moreover have** *depth* **0** $\leq$ *depth*$(<\nu x>[a\frown b]P)$ **by** *simp*
**ultimately show** *?case* **by** *blast*
**qed**
**next**
**case**(*Mismatch a b P x*)
**have** *valid*$([a\neq b]P)$ **by** *fact* **hence** *valid P* **by** *simp*
**moreover have** $\bigwedge x.$ *valid P* $\Longrightarrow \exists\, Q.$ *uhnf Q* $\wedge$ *valid Q* $\wedge$ $Q \equiv_e <\nu x>P \wedge$
*depth Q* $\leq$ *depth*$(<\nu x>P)$
**by** *fact*
**ultimately obtain** *Q* **where** *Qhnf*: *uhnf Q* **and** *validQ*: *valid Q*
**and** *QeqP*: $Q \equiv_e (<\nu x>P)$
**and** *QPdepth*: *depth Q* $\leq$ *depth*$(<\nu x>P)$
**by** *blast*
**show** *?case*
**proof**(*case-tac a = b*)
**assume** $a{=}b$
**moreover have** *uhnf* **0** **by**(*simp add*: *uhnf-def*)
**moreover have** *valid* **0** **by** *simp*
**moreover have** **0** $\equiv_e$ $<\nu x>[a\neq a]P$
**proof** −
**have** **0** $\equiv_e$ $[a\neq a]P$ **by**(*blast intro*: *Mismatch′ Sym*)
**hence** $<\nu x>$**0** $\equiv_e$ $<\nu x>[a\neq a]P$ **by**(*rule ResPres*)
**thus** *?thesis* **by**(*blast intro*: *ResNil Trans Sym*)
**qed**
**moreover have** *depth* **0** $\leq$ *depth*$(<\nu x>[a\neq a]P)$ **by** *simp*
**ultimately show** *?case* **by** *blast*
**next**
**assume** *aineqb*: $a\neq b$
**have** $Q \equiv_e <\nu x>[a\neq b]P$
**proof** −
**from** *aineqb* **have** $P \equiv_e [a\neq b]P$ **by**(*blast intro*: *equiv.Mismatch Sym*)
**hence** $<\nu x>P \equiv_e <\nu x>[a\neq b]P$ **by**(*rule ResPres*)
**with** *QeqP* **show** *?thesis* **by**(*blast intro*: *Trans*)
**qed**
**moreover from** *QPdepth* **have** *depth Q* $\leq$ *depth*$(<\nu x>[a\neq b]P)$ **by** *simp*
**ultimately show** *?case* **using** *Qhnf validQ* **by** *blast*
**qed**
**next**
**case**(*Sum P Q x*)
**have** *valid*$(P \oplus Q)$ **by** *fact*
**hence** *validP*: *valid P* **and** *validQ*: *valid Q* **by** *simp+*

**have** $\exists P'. \; uhnf \, P' \wedge valid \, P' \wedge P' \equiv_e <\nu x>P \wedge (depth \, P') \leq (depth(<\nu x>P))$
**proof** $-$
**have** $valid \, P \Longrightarrow \exists P'. \; uhnf \, P' \wedge valid \, P' \wedge P' \equiv_e <\nu x>P \wedge (depth \, P') \leq (depth \, (<\nu x>P))$ **by** *fact*
**with** *validP* **show** *?thesis* **by** *simp*
**qed**
**then obtain** $P'$ **where** $P'hnf$: $uhnf \, P'$ **and** $P'eqP$: $P' \equiv_e <\nu x>P$ **and** $validP'$: $valid \, P'$
**and** $P'depth$: $(depth \, P') \leq (depth(<\nu x>P))$ **by** *blast*

**have** $\exists Q'. \; uhnf \, Q' \wedge valid \, Q' \wedge Q' \equiv_e <\nu x>Q \wedge (depth \, Q') \leq (depth(<\nu x>Q))$
**proof** $-$
**have** $valid \, Q \Longrightarrow \exists Q'. \; uhnf \, Q' \wedge valid \, Q' \wedge Q' \equiv_e <\nu x>Q \wedge (depth \, Q') \leq (depth(<\nu x>Q))$ **by** *fact*
**with** *validQ* **show** *?thesis* **by** *simp*
**qed**

**then obtain** $Q'$ **where** $Q'hnf$: $uhnf \, Q'$ **and** $Q'eqQ$: $Q' \equiv_e <\nu x>Q$ **and** $validQ'$: $valid \, Q'$
**and** $Q'depth$: $(depth \, Q') \leq (depth(<\nu x>Q))$ **by** *blast*

**from** $P'hnf \; Q'hnf \; validP' \; validQ'$ **obtain** $R$ **where** $Rhnf$: $uhnf \, R$ **and** $validR$: $valid \, R$
**and** $P'Q'eqR$: $P' \oplus Q' \equiv_e R$
**and** $Rdepth$: $depth \, R \leq depth(P' \oplus Q')$
**apply**(*drule-tac uhnfSum*) **apply** *assumption+* **by** *blast*

**from** $P'eqP \; Q'eqQ \; P'Q'eqR$ **have** $<\nu x>(P \oplus Q) \equiv_e R$ **by**(*blast intro*: *Sym SumPres' SumRes Trans*)
**moreover from** $Rdepth \; P'depth \; Q'depth$ **have** $depth \, R \leq depth(<\nu x>(P \oplus Q))$ **by** *auto*
**ultimately show** *?case* **using** *validR Rhnf* **by**(*blast intro*: *Sym*)
**next**
**case**($Par \; P \; Q \; x$)
**have** $valid(P \parallel Q)$ **by** *fact*

**hence** *validP*: $valid \, P$ **and** *validQ*: $valid \, Q$ **by** *simp+*
**have** $\exists P'. \; uhnf \, P' \wedge valid \, P' \wedge P' \equiv_e P \wedge (depth \, P') \leq (depth \, P)$
**proof** $-$
**obtain** $x$::*name* **where** *xFreshP*: $x \sharp P$ **by**(*rule name-exists-fresh*)
**moreover have** $\bigwedge x. \; valid \, P \Longrightarrow \exists P'. \; uhnf \, P' \wedge valid \, P' \wedge P' \equiv_e (<\nu x>P) \wedge (depth \, P') \leq (depth(<\nu x>P))$ **by** *fact*
**with** *validP* **obtain** $P'$ **where** $uhnf \, P'$ **and** $valid \, P'$ **and** $P'eqP$: $P' \equiv_e (<\nu x>P)$ **and** $P'depth$: $(depth \, P') \leq (depth(<\nu x>P))$ **by** *blast*
**moreover from** $xFreshP \; P'eqP$ **have** $P' \equiv_e P$ **by**(*blast intro*: *Trans ResFresh*)
**moreover with** $P'depth$ **have** $depth \, P' \leq depth \, P$ **by** *simp*
**ultimately show** *?thesis* **by** *blast*
**qed**

**then obtain** $P'$ **where** $P'hnf$: $uhnf\ P'$ **and** $P'eqP$: $P' \equiv_e P$ **and** $validP'$: $valid\ P'$
**and** $P'depth$: $(depth\ P') \leq (depth\ P)$ **by** *blast*

**have** $\exists\, Q'.\ uhnf\ Q' \wedge valid\ Q' \wedge Q' \equiv_e Q \wedge (depth\ Q') \leq (depth\ Q)$
**proof** $-$
**obtain** $x{::}name$ **where** $xFreshQ$: $x \sharp Q$ **by**(*rule name-exists-fresh*)
**moreover have** $\bigwedge x.\ valid\ Q \Longrightarrow \exists\, Q'.\ uhnf\ Q' \wedge valid\ Q' \wedge Q' \equiv_e (<\nu x>Q) \wedge (depth\ Q') \leq (depth(<\nu x>Q))$ **by** *fact*
**with** $validQ$ **obtain** $Q'$ **where** $uhnf\ Q'$ **and** $valid\ Q'$ **and** $Q'eqQ$: $Q' \equiv_e (<\nu x>Q)$ **and** $Q'depth$: $(depth\ Q') \leq (depth(<\nu x>Q))$ **by** *blast*
**moreover from** $xFreshQ\ Q'eqQ$ **have** $Q' \equiv_e Q$ **by**(*blast intro*: *Trans ResFresh*)
**moreover with** $Q'depth$ **have** $depth\ Q' \leq depth\ Q$ **by** *simp*
**ultimately show** *?thesis* **by** *blast*
**qed**

**then obtain** $Q'$ **where** $Q'hnf$: $uhnf\ Q'$ **and** $Q'eqQ$: $Q' \equiv_e Q$ **and** $validQ'$: $valid\ Q'$
**and** $Q'depth$: $(depth\ Q') \leq (depth\ Q)$ **by** *blast*

**from** $P'hnf\ Q'hnf$ **obtain** $R$ **where** $Exp$: $(R,\ expandSet\ P'\ Q') \in sumComposeSet$ **and** $Rdepth$: $depth\ R \leq depth(P' \parallel Q')$
**by**(*force dest*: *expandDepth′ simp add*: *uhnf-def*)

**from** $Exp\ P'hnf\ Q'hnf$ **have** $P'Q'eqR$: $P' \parallel Q' \equiv_e R$ **by**(*force intro*: *Expand simp add*: *uhnf-def*)
**from** $P'hnf\ Q'hnf\ validP'\ validQ'$ **have** $\forall P \in (expandSet\ P'\ Q').\ uhnf\ P \wedge valid\ P$ **by**(*blast dest*: *validExpand*)
**with** $Exp$ **obtain** $R'$ **where** $R'hnf$: $uhnf\ R'$ **and** $validR'$: $valid\ R'$
**and** $ReqR'$: $R \equiv_e R'$
**and** $R'depth$: $depth\ R' \leq depth\ R$
**by**(*blast dest*: *expandHnf*)

**from** $P'eqP\ Q'eqQ\ P'Q'eqR\ ReqR'$ **have** $P \parallel Q \equiv_e R'$ **by**(*blast intro*: *Sym ParPres Trans*)
**hence** $ResTrans$: $<\nu x>(P \parallel Q) \equiv_e <\nu x>R'$ **by**(*rule ResPres*)
**from** $validR'\ R'hnf$ **obtain** $R''$ **where** $R''hnf$: $uhnf\ R''$ **and** $validR''$: $valid\ R''$ **and** $R'eqR''$: $<\nu x>R' \equiv_e R''$ **and** $R''depth$: $depth\ R'' \leq depth(<\nu x>R')$
**by**(*force dest*: *uhnfRes*)
**from** $ResTrans\ R'eqR''$ **have** $<\nu x>(P \parallel Q) \equiv_e R''$ **by**(*rule Trans*)
**moreover from** $Rdepth\ P'depth\ Q'depth\ R'depth\ R''depth$ **have** $depth\ R'' \leq depth(<\nu x>(P \parallel Q))$ **by** *auto*
**ultimately show** *?case* **using** $validR''\ R''hnf$ **by**(*blast dest*: *Sym*)
**next**
**case**(*Res y P x*)
**have** $valid(<\nu y>P)$ **by** *fact* **hence** $valid\ P$ **by** *simp*
**moreover have** $\bigwedge x.\ valid\ P \Longrightarrow \exists\, Q.\ uhnf\ Q \wedge valid\ Q \wedge Q \equiv_e <\nu x>P \wedge$

*depth* $Q \leq$ *depth*($<\nu x>P$)
by *fact*
ultimately obtain $Q$ where *Qhnf*: *uhnf* $Q$ and *validQ*: *valid* $Q$ and *QeqP*: $Q \equiv_e <\nu y>P$
and *QPdepth*: *depth* $Q \leq$ *depth*($<\nu y>P$) by *blast*

from *Qhnf validQ* obtain $Q'$ where $Q'hnf$: *uhnf* $Q'$ and $validQ'$: *valid* $Q'$ and $QeqQ'$: $<\nu x>Q \equiv_e Q'$
and $Q'Qdepth$: *depth* $Q' \leq$ *depth*($<\nu x>Q$)
by(*force dest*: *uhnfRes*)

from *QeqP* have $<\nu x>Q \equiv_e <\nu x><\nu y>P$ by(*rule ResPres*)
with $QeqQ'$ have $Q' \equiv_e <\nu x><\nu y>P$ by(*blast intro*: *Trans Sym*)
moreover from $Q'Qdepth$ *QPdepth* have *depth* $Q' \leq$ *depth*($<\nu x><\nu y>P$) by *simp*
ultimately show *?case* using $Q'hnf$ $validQ'$ by *blast*
next
case(*Bang P x*)
have *valid*($!P$) by *fact*
hence *False* by *simp*
thus *?case* by *simp*
qed
next
case(*Bang P*)
have *valid*($!P$) by *fact*
hence *False* by *simp*
thus *?case* by *simp*
qed
qed

lemma *depthZero*:
fixes $P$ :: *pi*

assumes *depth* $P = 0$
and *uhnf* $P$

shows $P = \mathbf{0}$
using *assms*
apply(*nominal-induct P rule*: *pi.strong-inducts*, *auto simp add*: *uhnf-def max-def if-split*)
apply(*case-tac depth pi1* $\leq$ *depth pi2*)
by *auto*

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** $P \equiv_e Q$
**using** *assms*
**proof**(*induct n arbitrary*: *P Q rule*: *nat.induct*)
**case**(*zero P Q*)
**have** *depth P* + *depth Q* $\leq$ *0* **by** *fact*
**hence** *Pdepth*: *depth P* = *0* **and** *Qdepth*: *depth Q* = *0* **by** *auto*
**moreover have** *uhnf P* **and** *uhnf Q* **by** *fact+*
**ultimately have** *P* = **0** **and** *Q* = **0** **by**(*blast intro*: *depthZero*)+
**thus** *?case* **by**(*blast intro*: *Refl*)
**next**
**case**(*Suc n P Q*)
**have** *validP*: *valid P* **and** *validQ*: *valid Q* **by** *fact+*
**have** *Phnf*: *uhnf P* **and** *Qhnf*: *uhnf Q* **by** *fact+*
**have** *PBisimQ*: $P \sim Q$ **by** *fact*
**have** *IH*: $\bigwedge P\ Q.\ \llbracket depth\ P + depth\ Q \leq n;\ valid\ P;\ valid\ Q;\ uhnf\ P;\ uhnf\ Q;\ P \sim Q\rrbracket \Longrightarrow P \equiv_e Q$
**by** *fact*
**have** *PQdepth*: *depth P* + *depth Q* $\leq$ *Suc n* **by** *fact*

**have** *Goal*: $\bigwedge P\ Q\ Q'.\ \llbracket depth\ P + depth\ Q \leq Suc\ n;\ valid\ P;\ valid\ Q;\ uhnf\ P;\ uhnf\ Q;$
$P \leadsto[bisim]\ Q;\ Q' \in summands\ Q\rrbracket \Longrightarrow \exists P' \in summands\ P.\ Q' \equiv_e P'$
**proof** −
**fix** $P\ Q\ Q'$
**assume** *PQdepth*: *depth P* + *depth Q* $\leq$ *Suc n*
**assume** *validP*: *valid P* **and** *validQ*: *valid Q*
**assume** *Phnf*: *uhnf P* **and** *Qhnf*: *uhnf Q*
**assume** *PSimQ*: $P \leadsto[bisim]\ Q$
**assume** *Q'inQ*: $Q' \in summands\ Q$

**thus** $\exists P' \in summands\ P.\ Q' \equiv_e P'$ **using** *PSimQ Phnf validP PQdepth*
**proof**(*nominal-induct* $Q'$ *avoiding*: *P rule*: *pi.strong-inducts*)
**case** *PiNil*
**have** $\mathbf{0} \in summands\ Q$ **by** *fact*
**hence** *False* **by**(*nominal-induct Q rule*: *pi.strong-inducts*, *auto simp add*: *if-split*)
**thus** *?case* **by** *simp*
**next**
**case**(*Output a b* $Q'$ *P*)
**have** *validP*: *valid P* **and** *Phnf*: *uhnf P* **and** *PSimQ*: $P \leadsto[bisim]\ Q$ **by** *fact+*
**have** *PQdepth*: *depth P* + *depth Q* $\leq$ *Suc n* **by** *fact*
**have** $a\{b\}.Q' \in summands\ Q$ **by** *fact*

**with** *Qhnf* **have** *QTrans*: $Q \longmapsto a[b] \prec Q'$ **by**(*simp add*: *summandTransition uhnf-def*)
**with** *PSimQ* **obtain** $P'$ **where** *PTrans*: $P \longmapsto a[b] \prec P'$ **and** *P'BisimQ'*: $P' \sim Q'$
**by**(*blast dest*: *simE*)

**from** *Phnf PTrans* **have** $a\{b\}.P' \in$ *summands* $P$ **by**(*simp add*: *summandTransition uhnf-def*)
**moreover have** $P' \equiv_e Q'$
**proof** −
**from** *validP PTrans* **have** *validP'*: *valid* $P'$ **by**(*blast intro*: *validTransition*)
**from** *validQ QTrans* **have** *validQ'*: *valid* $Q'$ **by**(*blast intro*: *validTransition*)

**from** *validP'* **obtain** $P''$ **where** *P''hnf*: *uhnf* $P''$ **and** *validP''*: *valid* $P''$
**and** *P''eqP'*: $P'' \equiv_e P'$ **and** *P''depth*: *depth* $P'' \leq$ *depth* $P'$
**by**(*blast dest*: *validToHnf*)

**from** *validQ'* **obtain** $Q''$ **where** *Q''hnf*: *uhnf* $Q''$ **and** *validQ''*: *valid* $Q''$
**and** *Q''eqQ'*: $Q'' \equiv_e Q'$ **and** *Q''depth*: *depth* $Q'' \leq$ *depth* $Q'$
**by**(*blast dest*: *validToHnf*)

**have** *depth* $P''$ + *depth* $Q'' \leq n$
**proof** −
**from** *Phnf PTrans* **have** *depth* $P'$ < *depth* $P$
**by**(*force intro*: *depthTransition simp add*: *uhnf-def*)
**moreover from** *Qhnf QTrans* **have** *depth* $Q'$ < *depth* $Q$
**by**(*force intro*: *depthTransition simp add*: *uhnf-def*)
**ultimately show** *?thesis* **using** *PQdepth P''depth Q''depth* **by** *simp*
**qed**

**moreover have** $P'' \sim Q''$
**proof** −
**from** *P''eqP'* **have** $P'' \sim P'$ **by**(*rule sound*)
**moreover from** *Q''eqQ'* **have** $Q'' \sim Q'$ **by**(*rule sound*)
**ultimately show** *?thesis* **using** *P'BisimQ'* **by**(*blast dest*: *transitive symmetric*)
**qed**
**ultimately have** $P'' \equiv_e Q''$ **using** *validP'' validQ'' P''hnf Q''hnf* **by**(*rule-tac IH*)
**with** *P''eqP' Q''eqQ'* **show** *?thesis* **by**(*blast intro*: *Sym Trans*)
**qed**
**ultimately show** *?case* **by**(*blast intro*: *Sym equiv.OutputPres*)
**next**
**case**(*Tau* $Q'$ $P$)
**have** *validP*: *valid* $P$ **and** *Phnf*: *uhnf* $P$ **and** *PSimQ*: $P \leadsto[bisim]\ Q$ **by** *fact+*
**have** *PQdepth*: *depth* $P$ + *depth* $Q \leq$ *Suc* $n$ **by** *fact*
**have** $\tau.(Q') \in$ *summands* $Q$ **by** *fact*

**with** *Qhnf* **have** *QTrans*: $Q \longmapsto \tau \prec Q'$ **by**(*simp add*: *summandTransition uhnf-def*)
**with** *PSimQ* **obtain** $P'$ **where** *PTrans*: $P \longmapsto \tau \prec P'$ **and** *P'BisimQ'*: $P' \sim Q'$
**by**(*blast dest*: *simE*)

**from** *Phnf PTrans* **have** $\tau.(P') \in$ *summands P* **by**(*simp add*: *summandTransition uhnf-def*)
**moreover have** $P' \equiv_e Q'$
**proof** −
**from** *validP PTrans* **have** *validP'*: *valid* $P'$ **by**(*blast intro*: *validTransition*)
**from** *validQ QTrans* **have** *validQ'*: *valid* $Q'$ **by**(*blast intro*: *validTransition*)

**from** *validP'* **obtain** $P''$ **where** *P''hnf*: *uhnf* $P''$ **and** *validP''*: *valid* $P''$
**and** *P''eqP'*: $P'' \equiv_e P'$ **and** *P''depth*: *depth* $P'' \leq$ *depth* $P'$
**by**(*blast dest*: *validToHnf*)

**from** *validQ'* **obtain** $Q''$ **where** *Q''hnf*: *uhnf* $Q''$ **and** *validQ''*: *valid* $Q''$
**and** *Q''eqQ'*: $Q'' \equiv_e Q'$ **and** *Q''depth*: *depth* $Q'' \leq$ *depth* $Q'$
**by**(*blast dest*: *validToHnf*)

**have** *depth* $P''$ + *depth* $Q'' \leq n$
**proof** −
**from** *Phnf PTrans* **have** *depth* $P' <$ *depth* $P$
**by**(*force intro*: *depthTransition simp add*: *uhnf-def*)
**moreover from** *Qhnf QTrans* **have** *depth* $Q' <$ *depth* $Q$
**by**(*force intro*: *depthTransition simp add*: *uhnf-def*)
**ultimately show** *?thesis* **using** *PQdepth P''depth Q''depth* **by** *simp*
**qed**

**moreover have** $P'' \sim Q''$
**proof** −
**from** *P''eqP'* **have** $P'' \sim P'$ **by**(*rule sound*)
**moreover from** *Q''eqQ'* **have** $Q'' \sim Q'$ **by**(*rule sound*)
**ultimately show** *?thesis* **using** *P'BisimQ'* **by**(*blast dest*: *transitive symmetric*)
**qed**
**ultimately have** $P'' \equiv_e Q''$ **using** *validP'' validQ'' P''hnf Q''hnf* **by**(*rule-tac IH*)
**with** *P''eqP' Q''eqQ'* **show** *?thesis* **by**(*blast intro*: *Sym Trans*)
**qed**
**ultimately show** *?case* **by**(*blast intro*: *Sym equiv.TauPres*)
**next**
**case**(*Input a x Q' P*)
**have** *validP*: *valid P* **and** *Phnf*: *uhnf P* **and** *PSimQ*: $P \leadsto$[*bisim*] $Q$ **and** *xFreshP*: $x \sharp P$ **by** *fact+*
**have** *PQdepth*: *depth P* + *depth Q* $\leq$ *Suc n* **by** *fact*

**have** $a{<}x{>}.Q' \in$ *summands* $Q$ **by** *fact*
**with** *Qhnf* **have** *QTrans*: $Q \longmapsto a{<}x{>} \prec Q'$ **by**(*simp add*: *summandTransition uhnf-def*)
**with** *PSimQ xFreshP* **obtain** $P'$ **where** *PTrans*: $P \longmapsto a{<}x{>} \prec P'$
**and** $P'derQ'$: *derivative* $P'$ $Q'$ (*InputS* $a$) $x$ *bisim*
**by**(*blast dest*: *simE*)

**from** *Phnf PTrans* **have** $a{<}x{>}.P' \in$ *summands* $P$ **by**(*simp add*: *summandTransition uhnf-def*)
**moreover have** $\forall y \in supp(P', Q', x).\ P'[x{::=}y] \equiv_e Q'[x{::=}y]$
**proof**(*rule ballI*)
**fix** $y{::}name$
**assume** *ysupp*: $y \in supp(P', Q', x)$
**have** $validP'$: $valid(P'[x{::=}y])$
**proof** −
**from** *validP PTrans* **have** $validP'$: *valid* $P'$ **by**(*blast intro*: *validTransition*)
**thus** *?thesis* **by** *simp*
**qed**
**have** $validQ'$: $valid(Q'[x{::=}y])$
**proof** −
**from** *validQ QTrans* **have** $validQ'$: *valid* $Q'$ **by**(*blast intro*: *validTransition*)
**thus** *?thesis* **by** *simp*
**qed**

**from** $validP'$ **obtain** $P''$ **where** $P''hnf$: *uhnf* $P''$ **and** $validP''$: *valid* $P''$
**and** $P''eqP'$: $P'' \equiv_e P'[x{::=}y]$ **and** $P''depth$: *depth* $P'' \leq depth(P'[x{::=}y])$
**by**(*blast dest*: *validToHnf*)

**from** $validQ'$ **obtain** $Q''$ **where** $Q''hnf$: *uhnf* $Q''$ **and** $validQ''$: *valid* $Q''$
**and** $Q''eqQ'$: $Q'' \equiv_e Q'[x{::=}y]$ **and** $Q''depth$: *depth* $Q'' \leq depth(Q'[x{::=}y])$
**by**(*blast dest*: *validToHnf*)

**have** *depth* $P''$ + *depth* $Q'' \leq n$
**proof** −
**from** *Phnf PTrans* **have** *depth* $P' <$ *depth* $P$
**by**(*force intro*: *depthTransition simp add*: *uhnf-def*)
**moreover from** *Qhnf QTrans* **have** *depth* $Q' <$ *depth* $Q$
**by**(*force intro*: *depthTransition simp add*: *uhnf-def*)
**ultimately show** *?thesis* **using** *PQdepth* $P''depth$ $Q''depth$ **by** *simp*
**qed**

**moreover have** $P'' \sim Q''$
**proof** −
**from** $P'derQ'$ **have** $P'BisimQ'$: $P'[x{::=}y] \sim Q'[x{::=}y]$
**by**(*auto simp add*: *derivative-def*)
**from** $P''eqP'$ **have** $P'' \sim P'[x{::=}y]$ **by**(*rule sound*)
**moreover from** $Q''eqQ'$ **have** $Q'' \sim Q'[x{::=}y]$ **by**(*rule sound*)

**ultimately show** *?thesis* **using** *P'BisimQ'* **by**(*blast dest*: *transitive symmetric*)
**qed**
**ultimately have** $P'' \equiv_e Q''$ **using** *validP'' validQ'' P''hnf Q''hnf* **by**(*rule-tac IH*)
**with** *P''eqP' Q''eqQ'* **show** $P'[x::=y] \equiv_e Q'[x::=y]$ **by**(*blast intro*: *Sym Trans*)
**qed**

**ultimately show** *?case*
**apply** −
**apply**(*rule-tac x=a<x>.P'* **in** *bexI*)
**apply**(*rule equiv.InputPres*)
**apply**(*rule ballI*)
**apply**(*erule-tac x=y* **in** *ballE*)
**apply**(*blast dest*: *Sym*)
**by**(*auto simp add*: *supp-prod*)
**next**
**case**(*Match a b P' P*)
**have** $[a \frown b]P' \in$ *summands Q* **by** *fact*
**hence** *False* **by**(*nominal-induct Q rule*: *pi.strong-inducts*, *auto simp add*: *if-split*)
**thus** *?case* **by** *simp*
**next**
**case**(*Mismatch a b P' P*)
**have** $[a \neq b]P' \in$ *summands Q* **by** *fact*
**hence** *False* **by**(*nominal-induct Q rule*: *pi.strong-inducts*, *auto simp add*: *if-split*)
**thus** *?case* **by** *simp*
**next**
**case**(*Sum P' Q' P*)
**have** $P' \oplus Q' \in$ *summands Q* **by** *fact*
**hence** *False* **by**(*nominal-induct Q rule*: *pi.strong-inducts*, *auto simp add*: *if-split*)
**thus** *?case* **by** *simp*
**next**
**case**(*Par P' Q' P*)
**have** $P' \parallel Q' \in$ *summands Q* **by** *fact*
**hence** *False* **by**(*nominal-induct Q rule*: *pi.strong-inducts*, *auto simp add*: *if-split*)
**thus** *?case* **by** *simp*
**next**
**case**(*Res x Q'' P*)
**have** *xFreshP*: $x \sharp P$ **by** *fact*
**have** *validP*: *valid P* **and** *Phnf*: *uhnf P* **and** *PSimQ*: $P \leadsto[bisim]\ Q$ **by** *fact+*
**have** *PQdepth*: *depth P + depth Q* $\leq$ *Suc n* **by** *fact*
**have** *Q''summQ*: $<\nu x>Q'' \in$ *summands Q* **by** *fact*
**hence** $\exists\, a\ Q'.\ a \neq x \wedge Q'' = a\{x\}.Q'$
**by**(*nominal-induct Q rule*: *pi.strong-inducts*, *auto simp add*: *if-split pi.inject*

*name-abs-eq name-calc*)
**then obtain** $a$ $Q'$ **where** *aineqx*: $a \neq x$ **and** $Q'eqQ''$: $Q'' = a\{x\}.Q'$
**by** *blast*
**with** *Qhnf* $Q''summQ$ **have** *QTrans*: $Q \longmapsto a{<}\nu x{>} \prec Q'$ **by**(*simp add*: *summandTransition uhnf-def*)
**with** *PSimQ xFreshP* **obtain** $P'$ **where** *PTrans*: $P \longmapsto a{<}\nu x{>} \prec P'$ **and** $P'BisimQ'$: $P' \sim Q'$
**by**(*force dest*: *simE simp add*: *derivative-def*)

**from** *Phnf PTrans aineqx* **have** $({<}\nu x{>}a\{x\}.P') \in$ *summands* $P$ **by**(*simp add*: *summandTransition uhnf-def*)
**moreover have** $a\{x\}.P' \equiv_e a\{x\}.Q'$
**proof** −
**have** $P' \equiv_e Q'$
**proof** −
**from** *validP PTrans* **have** $validP'$: *valid* $P'$ **by**(*blast intro*: *validTransition*)
**from** *validQ QTrans* **have** $validQ'$: *valid* $Q'$ **by**(*blast intro*: *validTransition*)

**from** $validP'$ **obtain** $P''$ **where** $P''hnf$: *uhnf* $P''$ **and** $validP''$: *valid* $P''$ **and** $P''eqP'$: $P'' \equiv_e P'$ **and** $P''depth$: *depth* $P'' \leq$ *depth* $P'$
**by**(*blast dest*: *validToHnf*)

**from** $validQ'$ **obtain** $Q''$ **where** $Q''hnf$: *uhnf* $Q''$ **and** $validQ''$: *valid* $Q''$ **and** $Q''eqQ'$: $Q'' \equiv_e Q'$ **and** $Q'''depth$: *depth* $Q'' \leq$ *depth* $Q'$
**by**(*blast dest*: *validToHnf*)

**have** *depth* $P''$ + *depth* $Q'' \leq n$
**proof** −
**from** *Phnf PTrans* **have** *depth* $P'$ < *depth* $P$
**by**(*force intro*: *depthTransition simp add*: *uhnf-def*)
**moreover from** *Qhnf QTrans* **have** *depth* $Q'$ < *depth* $Q$
**by**(*force intro*: *depthTransition simp add*: *uhnf-def*)
**ultimately show** *?thesis* **using** *PQdepth* $P''depth$ $Q'''depth$ **by** *simp*
**qed**

**moreover have** $P'' \sim Q''$
**proof** −
**from** $P''eqP'$ **have** $P'' \sim P'$ **by**(*rule sound*)
**moreover from** $Q''eqQ'$ **have** $Q'' \sim Q'$ **by**(*rule sound*)
**ultimately show** *?thesis* **using** $P'BisimQ'$ **by**(*blast dest*: *transitive symmetric*)
**qed**
**ultimately have** $P'' \equiv_e Q''$ **using** $validP''$ $validQ''$ $P''hnf$ $Q''hnf$ **by**(*rule-tac IH*)
**with** $P''eqP'$ $Q''eqQ'$ **show** *?thesis* **by**(*blast intro*: *Sym Trans*)
**qed**
**thus** *?thesis* **by**(*rule OutputPres*)

**qed**
**ultimately show** *?case* **using** $Q'eqQ''$ **by**(*blast intro*: *Sym equiv.ResPres*)
**next**
**case**(*Bang* $P'$ *P*)
**have** $!P' \in$ *summands Q* **by** *fact*
**hence** *False* **by**(*nominal-induct Q rule*: *pi.strong-inducts*, *auto simp add*: *if-split*)
**thus** *?case* **by** *simp*
**qed**
**qed**

**from** *Phnf Qhnf PQdepth validP validQ PBisimQ* **show** *?case*
**apply**(*rule-tac summandEquiv*, *auto*)
**apply**(*rule Goal*)
**apply** *auto*
**apply**(*blast dest*: *bisimE symmetric*)
**by**(*blast intro*: *Goal dest*: *bisimE*)
**qed**

**lemma** *complete*:
**fixes** $P :: pi$
**and** $Q :: pi$

**assumes** *validP*: *valid P*
**and** *validQ*: *valid Q*
**and** *PBisimQ*: $P \sim Q$

**shows** $P \equiv_e Q$
**proof** −
**from** *validP* **obtain** $P'$ **where** *validP'*: *valid* $P'$ **and** $P'hnf$: *uhnf* $P'$ **and** $P'eqP$: $P' \equiv_e P$
**by**(*blast dest*: *validToHnf*)
**from** *validQ* **obtain** $Q'$ **where** *validQ'*: *valid* $Q'$ **and** $Q'hnf$: *uhnf* $Q'$ **and** $Q'eqQ$: $Q' \equiv_e Q$
**by**(*blast dest*: *validToHnf*)

**have** $\exists n.\ depth\ P' + depth\ Q' \leq n$ **by** *auto*
**then obtain** *n* **where** $depth\ P' + depth\ Q' \leq n$ **by** *blast*
**moreover have** $P' \sim Q'$
**proof** −
**from** $P'eqP$ **have** $P' \sim P$ **by**(*rule sound*)
**moreover from** $Q'eqQ$ **have** $Q' \sim Q$ **by**(*rule sound*)
**ultimately show** *?thesis* **using** *PBisimQ* **by**(*blast intro*: *symmetric transitive*)
**qed**
**ultimately have** $P' \equiv_e Q'$ **using** *validP' validQ'* $P'hnf$ $Q'hnf$ **by**(*rule-tac completeAux*)
**with** $P'eqP$ $Q'eqQ$ **show** *?thesis* **by**(*blast intro*: *Sym Trans*)
**qed**

**end**

## References


[1] J. Bengtson. *Formalising process calculi*, volume 94. Uppsala Dissertations from the Faculty of Science and Technology, 2010.

[2] J. Bengtson and J. Parrow. A completeness proof for bisimulation in the pi-calculus using isabelle. *Electr. Notes Theor. Comput. Sci.*, 192(1):61–75, 2007.

[3] J. Bengtson and J. Parrow. Formalising the pi-calculus using nominal logic. *Logical Methods in Computer Science*, 5(2), 2009.