The Calculus of Communicating Systems

Jesper Bengtson

May 26, 2024

Abstract

We formalise a large portion of CCS as described in Milner's book 'Communication and Concurrency' using the nominal datatype package in Isabelle. Our results include many of the standard theorems of bisimulation equivalence and congruence, for both weak and strong versions. One main goal of this formalisation is to keep the machine-checked proofs as close to their pen-and-paper counterpart as possible.

Contents

1 Overview

These theories formalise the following results from Milner's book Communication and Concurrency.

- strong bisimilarity is a congruence
- strong bisimilarity respects the laws of structural congruence
- weak bisimilarity is preserved by all operators except sum
- weak congruence is a congruence
- all strongly bisimilar agents are also weakly congruent which in turn are weakly bisimilar. As a corollary, weak bisimilarity and weak congruence respect the laws of structural congruence.

The file naming convention is hopefully self explanatory, where the prefixes *Strong* and *Weak* denote that the file covers theories required to formalise properties of strong and weak bisimilarity respectively; if the file name contains *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 theories, please consult Bengtson's Ph. D. thesis [1].

2 Formalisation

```
theory Agent
  imports HOL−Nominal.Nominal
begin

atom-decl name

nominal-datatype act = actAction name      (⦇-⦈ 100)
                     | actCoAction name    (⟨-⟩ 100)
                     | actTau              (τ 100)

nominal-datatype ccs = CCSNil              (0 115)
                     | Action act ccs   (-.- [120, 110] 110)
                     | Sum ccs ccs      (infixl ⊕ 90)
                     | Par ccs ccs      (infixl ∥ 85)
                     | Res «name» ccs (⦇ν-⦈- [105, 100] 100)
                     | Bang ccs         (!- [95])

nominal-primrec coAction :: act ⇒ act
where
  coAction (⦇a⦈) = (⟨a⟩)
| coAction (⟨a⟩)= (⦇a⦈)
| coAction (τ) = τ
by(rule TrueI)+

lemma coActionEqvt[eqvt]:
  fixes p :: name prm
  and   a :: act

  shows (p · coAction a) = coAction(p · a)
by(nominal-induct a rule: act.strong-induct) (auto simp add: eqvts)

lemma coActionSimps[simp]:
  fixes a :: act

  shows coAction(coAction a) = a
  and   (coAction a = τ) = (a = τ)
by auto (nominal-induct rule: act.strong-induct, auto)+
```

lemma *coActSimp*[*simp*]: **shows** *coAction* $\alpha \neq \tau = (\alpha \neq \tau)$ **and** $(coAction\ \alpha = \tau) = (\alpha = \tau)$
by(*nominal-induct* α *rule*: *act.strong-induct*) *auto*

lemma *coActFresh*[*simp*]:
  fixes x :: *name*
  and a :: *act*

  shows $x \sharp coAction\ a = x \sharp a$
by(*nominal-induct a rule*: *act.strong-induct*) (*auto*)

lemma *alphaRes*:
  fixes y :: *name*
  and P :: *ccs*
  and x :: *name*

  assumes $y \sharp P$

  shows $(\!|\nu x|\!)P = (\!|\nu y|\!)([(x, y)] \cdot P)$
using *assms*
by(*auto simp add*: *ccs.inject alpha fresh-left calc-atm pt-swap-bij*[*OF pt-name-inst*, *OF at-name-inst*]*pt3*[*OF pt-name-inst*, *OF at-ds1*[*OF at-name-inst*]])

inductive *semantics* :: $ccs \Rightarrow act \Rightarrow ccs \Rightarrow bool$ (- $\longmapsto$- $\prec$ - [*80*, *80*, *80*] *80*)
where
  Action: $\alpha.(P) \longmapsto\alpha \prec P$
| *Sum1*: $P \longmapsto\alpha \prec P' \Longrightarrow P \oplus Q \longmapsto\alpha \prec P'$
| *Sum2*: $Q \longmapsto\alpha \prec Q' \Longrightarrow P \oplus Q \longmapsto\alpha \prec Q'$
| *Par1*: $P \longmapsto\alpha \prec P' \Longrightarrow P \parallel Q \longmapsto\alpha \prec P' \parallel Q$
| *Par2*: $Q \longmapsto\alpha \prec Q' \Longrightarrow P \parallel Q \longmapsto\alpha \prec P \parallel Q'$
| *Comm*: $[\![P \longmapsto a \prec P'; Q \longmapsto(coAction\ a) \prec Q'; a \neq \tau]\!] \Longrightarrow P \parallel Q \longmapsto\tau \prec P' \parallel Q'$
| *Res*: $[\![P \longmapsto\alpha \prec P'; x \sharp \alpha]\!] \Longrightarrow (\!|\nu x|\!)P \longmapsto\alpha \prec (\!|\nu x|\!)P'$
| *Bang*: $P \parallel !P \longmapsto\alpha \prec P' \Longrightarrow !P \longmapsto\alpha \prec P'$

equivariance *semantics*

nominal-inductive *semantics*
by(*auto simp add*: *abs-fresh*)

lemma *semanticsInduct*:
$[\![R \longmapsto\beta \prec R'; \bigwedge\alpha\ P\ \mathcal{C}.\ Prop\ \mathcal{C}\ (\alpha.(P))\ \alpha\ P;$
$\bigwedge P\ \alpha\ P'\ Q\ \mathcal{C}.\ [\![P \longmapsto\alpha \prec P'; \bigwedge\mathcal{C}.\ Prop\ \mathcal{C}\ P\ \alpha\ P']\!] \Longrightarrow Prop\ \mathcal{C}\ (ccs.Sum\ P\ Q)\ \alpha\ P';$
$\bigwedge Q\ \alpha\ Q'\ P\ \mathcal{C}.\ [\![Q \longmapsto\alpha \prec Q'; \bigwedge\mathcal{C}.\ Prop\ \mathcal{C}\ Q\ \alpha\ Q']\!] \Longrightarrow Prop\ \mathcal{C}\ (ccs.Sum\ P\ Q)\ \alpha\ Q';$
$\bigwedge P\ \alpha\ P'\ Q\ \mathcal{C}.\ [\![P \longmapsto\alpha \prec P'; \bigwedge\mathcal{C}.\ Prop\ \mathcal{C}\ P\ \alpha\ P']\!] \Longrightarrow Prop\ \mathcal{C}\ (P \parallel Q)\ \alpha\ (P' \parallel Q);$

$\bigwedge Q\ \alpha\ Q'\ P\ \mathcal{C}.\ [\![Q \longmapsto \alpha \prec Q';\ \bigwedge \mathcal{C}.\ Prop\ \mathcal{C}\ Q\ \alpha\ Q']\!] \Longrightarrow Prop\ \mathcal{C}\ (P \parallel Q)\ \alpha\ (P \parallel Q')$;
$\bigwedge P\ a\ P'\ Q\ Q'\ \mathcal{C}.$
$[\![P \longmapsto a \prec P';\ \bigwedge \mathcal{C}.\ Prop\ \mathcal{C}\ P\ a\ P';\ Q \longmapsto (coAction\ a) \prec Q';$
$\bigwedge \mathcal{C}.\ Prop\ \mathcal{C}\ Q\ (coAction\ a)\ Q';\ a \neq \tau]\!]$
$\Longrightarrow Prop\ \mathcal{C}\ (P \parallel Q)\ (\tau)\ (P' \parallel Q')$;
$\bigwedge P\ \alpha\ P'\ x\ \mathcal{C}.$
$[\![x \sharp \mathcal{C};\ P \longmapsto \alpha \prec P';\ \bigwedge \mathcal{C}.\ Prop\ \mathcal{C}\ P\ \alpha\ P';\ x \sharp \alpha]\!] \Longrightarrow Prop\ \mathcal{C}\ ((\!|\nu x|\!)P)\ \alpha\ ((\!|\nu x|\!)P')$;
$\bigwedge P\ \alpha\ P'\ \mathcal{C}.\ [\![P \parallel !P \longmapsto \alpha \prec P';\ \bigwedge \mathcal{C}.\ Prop\ \mathcal{C}\ (P \parallel !P)\ \alpha\ P']\!] \Longrightarrow Prop\ \mathcal{C}\ !P\ \alpha\ P']\!]$

$\Longrightarrow Prop\ (\mathcal{C}::'a::fs\text{-}name)\ R\ \beta\ R'$
by(*erule-tac z*=$\mathcal{C}$ **in** *semantics.strong-induct*) *auto*

lemma *NilTrans*[*dest*]:
shows $\mathbf{0} \longmapsto \alpha \prec P' \Longrightarrow False$
and $((\!|b|\!)).P \longmapsto \langle c\rangle \prec P' \Longrightarrow False$
and $((\!|b|\!)).P \longmapsto \tau \prec P' \Longrightarrow False$
and $(\langle b\rangle).P \longmapsto (\!|c|\!) \prec P' \Longrightarrow False$
and $(\langle b\rangle).P \longmapsto \tau \prec P' \Longrightarrow False$
apply(*ind-cases* $\mathbf{0} \longmapsto \alpha \prec P'$)
apply(*ind-cases* $((\!|b|\!)).P \longmapsto \langle c\rangle \prec P'$, *auto simp add*: *ccs.inject*)
apply(*ind-cases* $((\!|b|\!)).P \longmapsto \tau \prec P'$, *auto simp add*: *ccs.inject*)
apply(*ind-cases* $(\langle b\rangle).P \longmapsto (\!|c|\!) \prec P'$, *auto simp add*: *ccs.inject*)
apply(*ind-cases* $(\langle b\rangle).P \longmapsto \tau \prec P'$, *auto simp add*: *ccs.inject*)
done

lemma *freshDerivative*:
fixes P :: *ccs*
and a :: *act*
and P' :: *ccs*
and x :: *name*

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

shows $x \sharp \alpha$ **and** $x \sharp P'$
using *assms*
by(*nominal-induct rule*: *semantics.strong-induct*)
(*auto simp add*: *ccs.fresh abs-fresh*)

lemma *actCases*[*consumes 1*, *case-names cAct*]:
fixes α :: *act*
and P :: *ccs*
and β :: *act*
and P' :: *ccs*

assumes $\alpha.(P) \longmapsto \beta \prec P'$
and *Prop* α P

shows *Prop* β P'
using *assms*
by – (*ind-cases* $\alpha.(P) \longmapsto \beta \prec P'$, *auto simp add*: *ccs.inject*)

lemma *sumCases*[*consumes 1*, *case-names cSum1 cSum2*]:
fixes P :: *ccs*
and Q :: *ccs*
and α :: *act*
and R :: *ccs*

assumes $P \oplus Q \longmapsto \alpha \prec R$
and $\bigwedge P'. \ P \longmapsto \alpha \prec P' \Longrightarrow$ *Prop* P'
and $\bigwedge Q'. \ Q \longmapsto \alpha \prec Q' \Longrightarrow$ *Prop* Q'

shows *Prop* R
using *assms*
by – (*ind-cases* $P \oplus Q \longmapsto \alpha \prec R$, *auto simp add*: *ccs.inject*)

lemma *parCases*[*consumes 1*, *case-names cPar1 cPar2 cComm*]:
fixes P :: *ccs*
and Q :: *ccs*
and a :: *act*
and R :: *ccs*

assumes $P \parallel Q \longmapsto \alpha \prec R$
and $\bigwedge P'. \ P \longmapsto \alpha \prec P' \Longrightarrow$ *Prop* α $(P' \parallel Q)$
and $\bigwedge Q'. \ Q \longmapsto \alpha \prec Q' \Longrightarrow$ *Prop* α $(P \parallel Q')$
and $\bigwedge P' \ Q' \ a. \ [\![P \longmapsto a \prec P'; \ Q \longmapsto (coAction \ a) \prec Q'; \ a \neq \tau; \ \alpha = \tau]\!] \Longrightarrow$
Prop (τ) $(P' \parallel Q')$

shows *Prop* α R
using *assms*
by – (*ind-cases* $P \parallel Q \longmapsto \alpha \prec R$, *auto simp add*: *ccs.inject*)

lemma *resCases*[*consumes 1*, *case-names cRes*]:
fixes x :: *name*
and P :: *ccs*
and α :: *act*
and P' :: *ccs*

assumes $(\!|\nu x|\!)P \longmapsto \alpha \prec P'$
and $\bigwedge P'. \ [\![P \longmapsto \alpha \prec P'; \ x \sharp \alpha]\!] \Longrightarrow$ *Prop* $((\!|\nu x|\!)P')$

shows *Prop* P'
proof –
from ‹$(\!|\nu x|\!)P \longmapsto \alpha \prec P'$› **have** $x \sharp \alpha$ **and** $x \sharp P'$
by(*auto intro*: *freshDerivative simp add*: *abs-fresh*)+
with *assms* **show** *?thesis*

```
    by(cases rule: semantics.strong-cases[of - - - - x])
      (auto simp add: abs-fresh ccs.inject alpha)
qed

inductive bangPred :: ccs ⇒ ccs ⇒ bool
where
  aux1: bangPred P (!P)
| aux2: bangPred P (P ∥ !P)

lemma bangInduct[consumes 1, case-names cPar1 cPar2 cComm cBang]:
  fixes P  :: ccs
  and   α  :: act
  and   P' :: ccs
  and   𝒞  :: 'a::fs-name

  assumes !P ⟼α ≺ P'
  and     rPar1: ⋀α P' 𝒞. ⟦P ⟼α ≺ P'⟧ ⟹ Prop 𝒞 (P ∥ !P) α (P' ∥ !P)
  and     rPar2: ⋀α P' 𝒞. ⟦!P ⟼α ≺ P'; ⋀𝒞. Prop 𝒞 (!P) α P'⟧ ⟹ Prop 𝒞 (P
∥ !P) α (P ∥ P')
  and     rComm: ⋀a P' P'' 𝒞. ⟦P ⟼a ≺ P'; !P ⟼(coAction a) ≺ P''; ⋀𝒞.
Prop 𝒞 (!P) (coAction a) P''; a ≠ τ⟧ ⟹ Prop 𝒞 (P ∥ !P) (τ) (P' ∥ P'')
  and     rBang: ⋀α P' 𝒞. ⟦P ∥ !P ⟼α ≺ P'; ⋀𝒞. Prop 𝒞 (P ∥ !P) α P'⟧ ⟹
Prop 𝒞 (!P) α P'


  shows Prop 𝒞 (!P) α P'
proof −
  {
    fix X α P'
    assume X ⟼α ≺ P' and bangPred P X
    hence Prop 𝒞 X α P'
    proof(nominal-induct avoiding: 𝒞 rule: semantics.strong-induct)
      case(Action α Pa)
      thus ?case
        by − (ind-cases bangPred P (α.(Pa)))
    next
      case(Sum1 Pa α P' Q)
      thus ?case
        by − (ind-cases bangPred P (Pa ⊕ Q))
    next
      case(Sum2 Q α Q' Pa)
      thus ?case
        by − (ind-cases bangPred P (Pa ⊕ Q))
    next
      case(Par1 Pa α P' Q)
      thus ?case
        apply −
        by(ind-cases bangPred P (Pa ∥ Q), auto intro: rPar1 simp add: ccs.inject)
    next
```

case(*Par2 Q* α *P'* *Pa*)
thus *?case*
apply −
by(*ind-cases bangPred P* (*Pa* $\|$ *Q*), *auto intro*: *rPar2 aux1 simp add*: *ccs.inject*)
next
case(*Comm Pa a P' Q Q' C*)
thus *?case*
apply −
by(*ind-cases bangPred P* (*Pa* $\|$ *Q*), *auto intro*: *rComm aux1 simp add*: *ccs.inject*)
next
case(*Res Pa* α *P' x*)
thus *?case*
by − (*ind-cases bangPred P* ((|νx|)*Pa*))
next
case(*Bang Pa* α *P'*)
thus *?case*
apply −
by(*ind-cases bangPred P* (!*Pa*), *auto intro*: *rBang aux2 simp add*: *ccs.inject*)
qed
}
with ‹!*P* $\longmapsto\alpha \prec P'$› **show** *?thesis* **by**(*force intro*: *bangPred.aux1*)
qed

inductive-set *bangRel* :: (*ccs* × *ccs*) *set* ⇒ (*ccs* × *ccs*) *set*
for *Rel* :: (*ccs* × *ccs*) *set*
where
BRBang: (*P*, *Q*) ∈ *Rel* ⟹ (!*P*, !*Q*) ∈ *bangRel Rel*
| *BRPar*: (*R*, *T*) ∈ *Rel* ⟹ (*P*, *Q*) ∈ (*bangRel Rel*) ⟹ (*R* $\|$ *P*, *T* $\|$ *Q*) ∈ (*bangRel Rel*)

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

assumes (*P*, !*Q*) ∈ *bangRel Rel*
and ⋀*P*. (*P*, *Q*) ∈ *Rel* ⟹ *F* (!*P*)

shows *F P*
using *assms*
by − (*ind-cases* (*P*, !*Q*) ∈ *bangRel Rel*, *auto simp add*: *ccs.inject*)

lemma *BRParCases*[*consumes 1*, *case-names BRPar*]:
fixes *P* :: *ccs*
and *Q* :: *ccs*
and *Rel* :: (*ccs* × *ccs*) *set*

and F :: *ccs* ⇒ *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 − (*ind-cases* $(P, Q \parallel !Q) \in$ *bangRel Rel*, *auto simp add*: *ccs.inject*)

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

end

theory *Tau-Chain*
imports *Agent*
begin

definition *tauChain* :: *ccs* ⇒ *ccs* ⇒ *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 *tauChainInduct*[*consumes 1*, *case-names Base Step*]:
assumes $P \Longrightarrow_\tau P'$
and *Prop P*
and $\bigwedge P'\ P''.\ [\![P \Longrightarrow_\tau P'; P' \longmapsto \tau \prec P''; Prop\ P']\!] \Longrightarrow Prop\ P''$

shows *Prop P′*
using *assms*
by(*auto simp add*: *tauChain-def elim*: *rtrancl-induct*)

lemma *tauChainRefl*[*simp*]:
fixes *P* :: *ccs*

shows $P \Longrightarrow_\tau P$
by(*auto simp add*: *tauChain-def*)

lemma *tauChainCons*[*dest*]:
fixes *P* :: *ccs*
and *P′* :: *ccs*
and *P″* :: *ccs*

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

shows $P \Longrightarrow_\tau P''$
using *assms*
by(*auto simp add*: *tauChain-def*) (*blast dest*: *rtrancl-trans*)

lemma *tauChainCons2*[*dest*]:
fixes P :: *ccs*
and P' :: *ccs*
and P'' :: *ccs*

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

shows $P \Longrightarrow_\tau P''$
using *assms*
by(*auto simp add*: *tauChain-def*) (*blast dest*: *rtrancl-trans*)

lemma *tauChainAppend*[*dest*]:
fixes P :: *ccs*
and P' :: *ccs*
and P'' :: *ccs*

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

shows $P \Longrightarrow_\tau P''$
using ‹$P' \Longrightarrow_\tau P''$› ‹$P \Longrightarrow_\tau P'$›
by(*induct rule*: *tauChainInduct*) *auto*

lemma *tauChainSum1*:
fixes P :: *ccs*
and P' :: *ccs*
and Q :: *ccs*

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

shows $P \oplus Q \Longrightarrow_\tau P'$
using *assms*
proof(*induct rule*: *tauChainInduct*)
case *Base*
thus *?case* **by** *simp*
next
case(*Step* P' P'')
thus *?case*
by(*case-tac* $P{=}P'$) (*auto intro*: *Sum1 simp add*: *tauChain-def*)

qed

lemma *tauChainSum2*:
  fixes P :: *ccs*
  and P' :: *ccs*
  and Q :: *ccs*

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

  shows $P \oplus Q \Longrightarrow_\tau Q'$
using *assms*
proof(*induct rule*: *tauChainInduct*)
  case *Base*
  thus *?case* **by** *simp*
next
  case(*Step* Q' Q'')
  thus *?case*
    by(*case-tac* Q=Q') (*auto intro*: *Sum2 simp add*: *tauChain-def*)
qed

lemma *tauChainPar1*:
  fixes P :: *ccs*
  and P' :: *ccs*
  and Q :: *ccs*

  assumes $P \Longrightarrow_\tau P'$

  shows $P \parallel Q \Longrightarrow_\tau P' \parallel Q$
using *assms*
by(*induct rule*: *tauChainInduct*) (*auto intro*: *Par1*)

lemma *tauChainPar2*:
  fixes Q :: *ccs*
  and Q' :: *ccs*
  and P :: *ccs*

  assumes $Q \Longrightarrow_\tau Q'$

  shows $P \parallel Q \Longrightarrow_\tau P \parallel Q'$
using *assms*
by(*induct rule*: *tauChainInduct*) (*auto intro*: *Par2*)

lemma *tauChainRes*:
  fixes P :: *ccs*
  and P' :: *ccs*
  and x :: *name*

  assumes $P \Longrightarrow_\tau P'$

shows $(\!|\nu x|\!) P \Longrightarrow_\tau (\!|\nu x|\!) P'$
using *assms*
by(*induct rule*: *tauChainInduct*) (*auto dest*: *Res*)

lemma *tauChainRepl*:
fixes P :: *ccs*

assumes $P \parallel !P \Longrightarrow_\tau P'$
and $P' \neq P \parallel !P$

shows $!P \Longrightarrow_\tau P'$
using *assms*
apply(*induct rule*: *tauChainInduct*)
apply *auto*
apply(*case-tac* $P' \neq P \parallel !P$)
apply *auto*
apply(*drule Bang*)
apply(*simp add*: *tauChain-def*)
by *auto*

end

theory *Weak-Cong-Semantics*
imports *Tau-Chain*
begin

definition *weakCongTrans* :: *ccs* ⇒ *act* ⇒ *ccs* ⇒ *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 *weakCongTransE*:
fixes P :: *ccs*
and α :: *act*
and P' :: *ccs*

assumes $P \Longrightarrow\alpha \prec P'$

obtains $P''\ P'''$ **where** $P \Longrightarrow_\tau P''$ **and** $P'' \longmapsto\alpha \prec P'''$ **and** $P''' \Longrightarrow_\tau P'$
using *assms*
by(*auto simp add*: *weakCongTrans-def*)

lemma *weakCongTransI*:
fixes P :: *ccs*
and P'' :: *ccs*
and α :: *act*
and P''' :: *ccs*
and P' :: *ccs*

assumes $P \Longrightarrow_\tau P''$
and $P'' \longmapsto \alpha \prec P'''$
and $P''' \Longrightarrow_\tau P'$

shows $P \Longrightarrow \alpha \prec P'$
using *assms*
by(*auto simp add*: *weakCongTrans-def*)

lemma *transitionWeakCongTransition*:
fixes P :: *ccs*
and α :: *act*
and P' :: *ccs*

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

shows $P \Longrightarrow \alpha \prec P'$
using *assms*
by(*force simp add*: *weakCongTrans-def*)

lemma *weakCongAction*:
fixes a :: *name*
and P :: *ccs*

shows $\alpha.(P) \Longrightarrow \alpha \prec P$
by(*auto simp add*: *weakCongTrans-def*)
(*blast intro*: *Action tauChainRefl*)

lemma *weakCongSum1*:
fixes P :: *ccs*
and α :: *act*
and P' :: *ccs*
and Q :: *ccs*

assumes $P \Longrightarrow \alpha \prec P'$

shows $P \oplus Q \Longrightarrow \alpha \prec P'$
using *assms*
apply(*auto simp add*: *weakCongTrans-def*)
apply(*case-tac* $P=P''$)
apply(*force simp add*: *tauChain-def dest*: *Sum1*)
by(*force intro*: *tauChainSum1*)

lemma *weakCongSum2*:
fixes Q :: *ccs*
and α :: *act*
and Q' :: *ccs*
and P :: *ccs*

assumes $Q \Longrightarrow \alpha \prec Q'$

shows $P \oplus Q \Longrightarrow\alpha \prec Q'$
using *assms*
apply(*auto simp add*: *weakCongTrans-def*)
apply(*case-tac* $Q=P''$)
apply(*force simp add*: *tauChain-def dest*: *Sum2*)
by(*force intro*: *tauChainSum2*)

lemma *weakCongPar1*:
fixes P :: *ccs*
and α :: *act*
and P' :: *ccs*
and Q :: *ccs*

assumes $P \Longrightarrow\alpha \prec P'$

shows $P \parallel Q \Longrightarrow\alpha \prec P' \parallel Q$
using *assms*
by(*auto simp add*: *weakCongTrans-def*)
(*blast dest*: *tauChainPar1 Par1*)

lemma *weakCongPar2*:
fixes Q :: *ccs*
and α :: *act*
and Q' :: *ccs*
and P :: *ccs*

assumes $Q \Longrightarrow\alpha \prec Q'$

shows $P \parallel Q \Longrightarrow\alpha \prec P \parallel Q'$
using *assms*
by(*auto simp add*: *weakCongTrans-def*)
(*blast dest*: *tauChainPar2 Par2*)

lemma *weakCongSync*:
fixes P :: *ccs*
and α :: *act*
and P' :: *ccs*
and Q :: *ccs*

assumes $P \Longrightarrow\alpha \prec P'$
and $Q \Longrightarrow(coAction\ \alpha) \prec Q'$
and $\alpha \neq \tau$

shows $P \parallel Q \Longrightarrow\tau \prec P' \parallel Q'$
using *assms*
apply(*auto simp add*: *weakCongTrans-def*)
apply(*rule-tac* $x= P'' \parallel P''a$ **in** *exI*)
apply *auto*

```
apply(blast dest: tauChainPar1 tauChainPar2)
apply(rule-tac x=P''' ∥ P'''a in exI)
apply auto
apply(rule Comm)
apply auto
apply(rule-tac P'=P' ∥ P'''a in tauChainAppend)
by(blast dest: tauChainPar1 tauChainPar2)+

lemma weakCongRes:
  fixes P  :: ccs
  and   α  :: act
  and   P' :: ccs
  and   x  :: name

  assumes P ⟹α ≺ P'
  and     x ♯ α

  shows ⦇νx⦈P ⟹α ≺ ⦇νx⦈P'
using assms
by(auto simp add: weakCongTrans-def)
  (blast dest: tauChainRes Res)

lemma weakCongRepl:
  fixes P  :: ccs
  and   α  :: act
  and   P' :: ccs

  assumes P ∥ !P ⟹α ≺ P'

  shows !P ⟹α ≺ P'
using assms
apply(auto simp add: weakCongTrans-def)
apply(case-tac P'' = P ∥ !P)
apply auto
apply(force intro: Bang simp add: tauChain-def)
by(force intro: tauChainRepl)

end

theory Weak-Semantics
  imports Weak-Cong-Semantics
begin

definition weakTrans :: ccs ⇒ act ⇒ ccs ⇒ bool (- ⟹^ - ≺ - [80, 80, 80] 80)
  where P ⟹^α ≺ P' ≡ P ⟹α ≺ P' ∨ (α = τ ∧ P = P')

lemma weakEmptyTrans[simp]:
  fixes P :: ccs
```

shows $P \Longrightarrow\hat{}\ \tau \prec P$
by(*auto simp add*: *weakTrans-def*)

lemma *weakTransCases*[*consumes 1*, *case-names Base Step*]:
fixes P :: *ccs*
and α :: *act*
and P' :: *ccs*

assumes $P \Longrightarrow\hat{}\ \alpha \prec P'$
and $[\![\alpha = \tau; P = P']\!] \Longrightarrow Prop\ (\tau)\ P$
and $P \Longrightarrow\alpha \prec P' \Longrightarrow Prop\ \alpha\ P'$

shows $Prop\ \alpha\ P'$
using *assms*
by(*auto simp add*: *weakTrans-def*)

lemma *weakCongTransitionWeakTransition*:
fixes P :: *ccs*
and α :: *act*
and P' :: *ccs*

assumes $P \Longrightarrow\alpha \prec P'$

shows $P \Longrightarrow\hat{}\ \alpha \prec P'$
using *assms*
by(*auto simp add*: *weakTrans-def*)

lemma *transitionWeakTransition*:
fixes P :: *ccs*
and α :: *act*
and P' :: *ccs*

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

shows $P \Longrightarrow\hat{}\ \alpha \prec P'$
using *assms*
by(*auto dest*: *transitionWeakCongTransition weakCongTransitionWeakTransition*)

lemma *weakAction*:
fixes a :: *name*
and P :: *ccs*

shows $\alpha.(P) \Longrightarrow\hat{}\ \alpha \prec P$
by(*auto simp add*: *weakTrans-def intro*: *weakCongAction*)

lemma *weakSum1*:
fixes P :: *ccs*
and α :: *act*
and P' :: *ccs*

and Q :: *ccs*

assumes $P \Longrightarrow\hat{}\ \alpha \prec P'$
and $P \neq P'$

shows $P \oplus Q \Longrightarrow\hat{}\ \alpha \prec P'$
using *assms*
by(*auto simp add*: *weakTrans-def intro*: *weakCongSum1*)

lemma *weakSum2*:
fixes Q :: *ccs*
and α :: *act*
and Q' :: *ccs*
and P :: *ccs*

assumes $Q \Longrightarrow\hat{}\ \alpha \prec Q'$
and $Q \neq Q'$

shows $P \oplus Q \Longrightarrow\hat{}\ \alpha \prec Q'$
using *assms*
by(*auto simp add*: *weakTrans-def intro*: *weakCongSum2*)

lemma *weakPar1*:
fixes P :: *ccs*
and α :: *act*
and P' :: *ccs*
and Q :: *ccs*

assumes $P \Longrightarrow\hat{}\ \alpha \prec P'$

shows $P \parallel Q \Longrightarrow\hat{}\ \alpha \prec P' \parallel Q$
using *assms*
by(*auto simp add*: *weakTrans-def intro*: *weakCongPar1*)

lemma *weakPar2*:
fixes Q :: *ccs*
and α :: *act*
and Q' :: *ccs*
and P :: *ccs*

assumes $Q \Longrightarrow\hat{}\ \alpha \prec Q'$

shows $P \parallel Q \Longrightarrow\hat{}\ \alpha \prec P \parallel Q'$
using *assms*
by(*auto simp add*: *weakTrans-def intro*: *weakCongPar2*)

lemma *weakSync*:
fixes P :: *ccs*
and α :: *act*

and P' :: *ccs*
and Q :: *ccs*

assumes $P \Longrightarrow\hat{}\ \alpha \prec P'$
and $Q \Longrightarrow\hat{}\ (coAction\ \alpha) \prec Q'$
and $\alpha \neq \tau$

shows $P \parallel Q \Longrightarrow\hat{}\ \tau \prec P' \parallel Q'$
using *assms*
by(*auto simp add*: *weakTrans-def intro*: *weakCongSync*)

lemma *weakRes*:
fixes P :: *ccs*
and α :: *act*
and P' :: *ccs*
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 simp add*: *weakTrans-def intro*: *weakCongRes*)

lemma *weakRepl*:
fixes P :: *ccs*
and α :: *act*
and P' :: *ccs*

assumes $P \parallel !P \Longrightarrow\hat{}\ \alpha \prec P'$
and $P' \neq P \parallel !P$

shows $!P \Longrightarrow \alpha \prec P'$
using *assms*
by(*auto simp add*: *weakTrans-def intro*: *weakCongRepl*)

end

theory *Strong-Sim*
imports *Agent*
begin

definition *simulation* :: *ccs* ⇒ (*ccs* × *ccs*) *set* ⇒ *ccs* ⇒ *bool* (- ⇝[-] - [*80*, *80*, *80*] *80*)
where
$P \leadsto[Rel]\ Q \equiv \forall a\ Q'.\ Q \longmapsto a \prec Q' \longrightarrow (\exists P'.\ P \longmapsto a \prec P' \wedge (P', Q') \in Rel)$

lemma *simI*[*case-names Sim*]:
fixes P :: *ccs*

and Rel :: $(ccs \times ccs)$ set
and Q :: ccs

assumes $\bigwedge \alpha\ Q'.\ Q \longmapsto \alpha \prec Q' \Longrightarrow \exists P'.\ P \longmapsto \alpha \prec P' \wedge (P',\ Q') \in Rel$

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

lemma *simE*:
fixes P :: ccs
and Rel :: $(ccs \times ccs)$ set
and Q :: ccs
and α :: act
and Q' :: ccs

assumes $P \leadsto [Rel]\ Q$
and $Q \longmapsto \alpha \prec Q'$

obtains P' **where** $P \longmapsto \alpha \prec P'$ **and** $(P',\ Q') \in Rel$
using *assms*
by(*auto simp add*: *simulation-def*)

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

assumes $Id \subseteq Rel$

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

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

assumes $P \leadsto [Rel]\ Q$
and $Q \leadsto [Rel']\ R$
and $Rel\ O\ Rel' \subseteq Rel''$

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

end

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

definition *weakSimulation* :: *ccs* ⇒ (*ccs* × *ccs*) *set* ⇒ *ccs* ⇒ *bool*   (- ⇝^<-> - [*80*, *80*, *80*] *80*)
where
  P ⇝^<*Rel*> *Q* ≡ ∀ *a* *Q*′. *Q* ⟼*a* ≺ *Q*′ ⟶ (∃ *P*′. *P* ⟹^ *a* ≺ *P*′ ∧ (*P*′, *Q*′) ∈ *Rel*)

lemma *weakSimI*[*case-names Sim*]:
  fixes *P*   :: *ccs*
  and   *Rel* :: (*ccs* × *ccs*) *set*
  and   *Q*   :: *ccs*

  assumes ⋀*α* *Q*′. *Q* ⟼*α* ≺ *Q*′ ⟹ ∃ *P*′. *P* ⟹^ *α* ≺ *P*′ ∧ (*P*′, *Q*′) ∈ *Rel*

  shows *P* ⇝^<*Rel*> *Q*
using *assms*
by(*auto simp add*: *weakSimulation-def*)

lemma *weakSimE*:
  fixes *P*   :: *ccs*
  and   *Rel* :: (*ccs* × *ccs*) *set*
  and   *Q*   :: *ccs*
  and   *α*   :: *act*
  and   *Q*′ :: *ccs*

  assumes *P* ⇝^<*Rel*> *Q*
  and   *Q* ⟼*α* ≺ *Q*′

  obtains *P*′ **where** *P* ⟹^ *α* ≺ *P*′ **and** (*P*′, *Q*′) ∈ *Rel*
using *assms*
by(*auto simp add*: *weakSimulation-def*)

lemma *simTauChain*:
  fixes *P*   :: *ccs*
  and   *Rel* :: (*ccs* × *ccs*) *set*
  and   *Q*   :: *ccs*
  and   *Q*′ :: *ccs*

  assumes *Q* $\Longrightarrow_\tau$ *Q*′
  and   (*P*, *Q*) ∈ *Rel*
  and   *Sim*: ⋀*R* *S*. (*R*, *S*) ∈ *Rel* ⟹ *R* ⇝^<*Rel*> *S*

  obtains *P*′ **where** *P* $\Longrightarrow_\tau$ *P*′ **and** (*P*′, *Q*′) ∈ *Rel*
using ‹*Q* $\Longrightarrow_\tau$ *Q*′› ‹(*P*, *Q*) ∈ *Rel*›

proof(*induct arbitrary*: *thesis rule*: *tauChainInduct*)
case *Base*
from $\langle(P,\ Q) \in Rel\rangle$ **show** *?case*
by(*force intro*: *Base*)
next
case(*Step* Q'' Q')
from $\langle(P,\ Q) \in Rel\rangle$ **obtain** P'' **where** $P \Longrightarrow_\tau P''$ **and** $(P'',\ Q'') \in Rel$
by(*blast intro*: *Step*)
from $\langle(P'',\ Q'') \in Rel\rangle$ **have** $P'' \leadsto\hat{}<Rel> Q''$ **by**(*rule Sim*)
then obtain P' **where** $P'' \Longrightarrow\hat{}\ \tau \prec P'$ **and** $(P',\ Q') \in Rel$ **using** $\langle Q'' \longmapsto \tau \prec Q'\rangle$ **by**(*rule weakSimE*)
with $\langle P \Longrightarrow_\tau P''\rangle$ **show** *thesis*
by(*force simp add*: *weakTrans-def weakCongTrans-def intro*: *Step*)
qed

lemma *simE2*:
fixes P :: *ccs*
and Rel :: (*ccs* × *ccs*) *set*
and Q :: *ccs*
and α :: *act*
and Q' :: *ccs*

assumes $(P,\ Q) \in Rel$
and $Q \Longrightarrow\hat{}\ \alpha \prec Q'$
and *Sim*: $\bigwedge R\ S.\ (R,\ S) \in Rel \Longrightarrow R \leadsto\hat{}<Rel> S$

obtains P' **where** $P \Longrightarrow\hat{}\ \alpha \prec P'$ **and** $(P',\ Q') \in Rel$
proof −
assume *Goal*: $\bigwedge P'.\ [\![P \Longrightarrow\hat{}\ \alpha \prec P';\ (P',\ Q') \in Rel]\!] \Longrightarrow thesis$
moreover from $\langle Q \Longrightarrow\hat{}\ \alpha \prec Q'\rangle$ **have** $\exists P'.\ P \Longrightarrow\hat{}\ \alpha \prec P' \wedge (P',\ Q') \in Rel$
proof(*induct rule*: *weakTransCases*)
case *Base*
from $\langle(P,\ Q) \in Rel\rangle$ **show** *?case* **by** *force*
next
case *Step*
from $\langle Q \Longrightarrow \alpha \prec Q'\rangle$ **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(*rule weakCongTransE*)
from *QChain* $\langle(P,\ Q) \in Rel\rangle$ *Sim* **obtain** P''' **where** *PChain*: $P \Longrightarrow_\tau P'''$ **and** $(P''',\ Q''') \in Rel$
by(*rule simTauChain*)
from $\langle(P''',\ Q''') \in Rel\rangle$ **have** $P''' \leadsto\hat{}<Rel> Q'''$ **by**(*rule Sim*)
then obtain P'' **where** P'''*Trans*: $P''' \Longrightarrow\hat{}\ \alpha \prec P''$ **and** $(P'',\ Q'') \in Rel$ **using** Q'''*Trans* **by**(*rule weakSimE*)
from Q''*Chain* $\langle(P'',\ Q'') \in Rel\rangle$ *Sim* **obtain** P' **where** P''*Chain*: $P'' \Longrightarrow_\tau P'$ **and** $(P',\ Q') \in Rel$
by(*rule simTauChain*)
from P'''*Trans* P''*Chain* *Step* **show** *?thesis*

```
    proof(induct rule: weakTransCases)
      case Base
      from PChain ‹P''' ⟹τ P'› have P ⟹^τ ≺ P'
      proof(induct rule: tauChainInduct)
        case Base
        from ‹P ⟹τ P'› show ?case
        proof(induct rule: tauChainInduct)
          case Base
          show ?case by simp
        next
          case(Step P' P'')
          thus ?case by(fastforce simp add: weakTrans-def weakCongTrans-def)
        qed
      next
        case(Step P''' P'')
        thus ?case by(fastforce simp add: weakTrans-def weakCongTrans-def)
      qed
      with ‹(P', Q') ∈ Rel› show ?case by blast
    next
      case Step
      thus ?case using ‹(P', Q') ∈ Rel› PChain
      by(rule-tac x=P' in exI) (force simp add: weakTrans-def weakCongTrans-def)
    qed
  qed
  ultimately show ?thesis
    by blast
qed

lemma reflexive:
  fixes P   :: ccs
  and   Rel :: (ccs × ccs) set

  assumes Id ⊆ Rel

  shows P ↝^<Rel> P
using assms
by(auto simp add: weakSimulation-def intro: transitionWeakCongTransition weak-
CongTransitionWeakTransition)

lemma transitive:
  fixes P     :: ccs
  and   Rel   :: (ccs × ccs) set
  and   Q     :: ccs
  and   Rel'  :: (ccs × ccs) set
  and   R     :: ccs
  and   Rel'' :: (ccs × ccs) set

  assumes (P, Q) ∈ Rel
  and     Q ↝^<Rel'> R
```

and *Rel O Rel′ ⊆ Rel′′*
and ⋀*S T*. (*S*, *T*) ∈ *Rel* ⟹ *S* ⇝^<*Rel*> *T*

shows *P* ⇝^<*Rel′′*> *R*
proof(*induct rule*: *weakSimI*)
case(*Sim α R′*)
thus *?case* **using** *assms*
apply(*drule-tac Q*=*R* **in** *weakSimE*, *auto*)
by(*drule-tac Q*=*Q* **in** *simE2*, *auto*)
qed

lemma *weakMonotonic*:
fixes *P* :: *ccs*
and *A* :: (*ccs* × *ccs*) *set*
and *Q* :: *ccs*
and *B* :: (*ccs* × *ccs*) *set*

assumes *P* ⇝^<*A*> *Q*
and *A* ⊆ *B*

shows *P* ⇝^<*B*> *Q*
using *assms*
by(*fastforce simp add*: *weakSimulation-def*)

lemma *simWeakSim*:
fixes *P* :: *ccs*
and *Rel* :: (*ccs* × *ccs*) *set*
and *Q* :: *ccs*

assumes *P* ⇝[*Rel*] *Q*

shows *P* ⇝^<*Rel*> *Q*
using *assms*
by(*rule-tac weakSimI*, *auto*)
(*blast dest*: *simE transitionWeakTransition*)

end

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

definition *weakCongSimulation* :: *ccs* ⇒ (*ccs* × *ccs*) *set* ⇒ *ccs* ⇒ *bool* (- ⇝<-> - [*80*, *80*, *80*] *80*)
where
P ⇝<*Rel*> *Q* ≡ ∀ *a Q′*. *Q* ⟼*a* ≺ *Q′* ⟶ (∃ *P′*. *P* ⟹*a* ≺ *P′* ∧ (*P′*, *Q′*) ∈ *Rel*)

lemma *weakSimI*[*case-names Sim*]:
fixes *P* :: *ccs*

and *Rel* :: (*ccs* × *ccs*) *set*
and *Q* :: *ccs*

assumes $\bigwedge\alpha\ Q'.\ Q \longmapsto\alpha \prec Q' \Longrightarrow \exists P'.\ P \Longrightarrow\alpha \prec P' \wedge (P', Q') \in Rel$

shows $P \leadsto<Rel>\ Q$
using *assms*
by(*auto simp add*: *weakCongSimulation-def*)

lemma *weakSimE*:
fixes *P* :: *ccs*
and *Rel* :: (*ccs* × *ccs*) *set*
and *Q* :: *ccs*
and α :: *act*
and Q' :: *ccs*

assumes $P \leadsto<Rel>\ Q$
and $Q \longmapsto\alpha \prec Q'$

obtains P' **where** $P \Longrightarrow\alpha \prec P'$ **and** $(P', Q') \in Rel$
using *assms*
by(*auto simp add*: *weakCongSimulation-def*)

lemma *simWeakSim*:
fixes *P* :: *ccs*
and *Rel* :: (*ccs* × *ccs*) *set*
and *Q* :: *ccs*

assumes $P \leadsto[Rel]\ Q$

shows $P \leadsto<Rel>\ Q$
using *assms*
by(*rule-tac weakSimI*, *auto*)
(*blast dest*: *simE transitionWeakCongTransition*)

lemma *weakCongSimWeakSim*:
fixes *P* :: *ccs*
and *Rel* :: (*ccs* × *ccs*) *set*
and *Q* :: *ccs*

assumes $P \leadsto<Rel>\ Q$

shows $P \leadsto\hat{}<Rel>\ Q$
using *assms*
by(*rule-tac Weak-Sim.weakSimI*, *auto*)
(*blast dest*: *weakSimE weakCongTransitionWeakTransition*)

lemma *test*:
assumes $P \Longrightarrow_\tau P'$

shows $P = P' \vee (\exists P''.\ P \longmapsto\tau \prec P'' \wedge P'' \Longrightarrow_\tau P')$
using *assms*
by(*induct rule*: *tauChainInduct*) *auto*

lemma *tauChainCasesSym*[*consumes 1*, *case-names cTauNil cTauStep*]:
assumes $P \Longrightarrow_\tau P'$
and *Prop P*
and $\bigwedge P''.\ [\![P \longmapsto\tau \prec P''; P'' \Longrightarrow_\tau P']\!] \Longrightarrow$ *Prop P'*

shows *Prop P'*
using *assms*
by(*blast dest*: *test*)

lemma *simE2*:
fixes P :: *ccs*
and *Rel* :: (*ccs* × *ccs*) *set*
and Q :: *ccs*
and α :: *act*
and Q' :: *ccs*

assumes $P \leadsto\hat{}<Rel> Q$
and $Q \Longrightarrow\alpha \prec Q'$
and *Sim*: $\bigwedge R\ S.\ (R, S) \in Rel \Longrightarrow R \leadsto\hat{}<Rel> S$

obtains P' **where** $P \Longrightarrow\alpha \prec P'$ **and** $(P', Q') \in Rel$
proof −
assume *Goal*: $\bigwedge P'.\ [\![P \Longrightarrow\alpha \prec P'; (P', Q') \in Rel]\!] \Longrightarrow$ *thesis*
from ‹$Q \Longrightarrow\alpha \prec Q'$› **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(*rule weakCongTransE*)
from *QChain* Q'''*Trans* **show** *?thesis*
proof(*induct rule*: *tauChainCasesSym*)
case *cTauNil*
from ‹$P \leadsto<Rel> Q$› ‹$Q \longmapsto\alpha \prec Q''$› **obtain** P''' **where** *PTrans*: $P \Longrightarrow\alpha \prec P'''$ **and** $(P''', Q'') \in Rel$
by(*blast dest*: *weakSimE*)
moreover from Q''*Chain* ‹$(P''', Q'') \in Rel$› *Sim* **obtain** P' **where** P''*Chain*: $P''' \Longrightarrow_\tau P'$ **and** $(P', Q') \in Rel$
by(*rule simTauChain*)
with *PTrans* P''*Chain* **show** *?thesis*
by(*force intro*: *Goal simp add*: *weakCongTrans-def weakTrans-def*)
next
case(*cTauStep* Q'''')
from ‹$P \leadsto<Rel> Q$› ‹$Q \longmapsto\tau \prec Q''''$› **obtain** P'''' **where** *PChain*: $P \Longrightarrow\tau \prec P''''$ **and** $(P'''', Q'''') \in Rel$
by(*drule-tac weakSimE*) *auto*
from ‹$Q'''' \Longrightarrow_\tau Q'''$› ‹$(P'''', Q'''') \in Rel$› *Sim* **obtain** P''' **where** P''''*Chain*:

$P'''' \Longrightarrow_{\tau} P'''$ **and** $(P''', Q''') \in Rel$
by(*rule simTauChain*)
from ‹$(P''', Q''') \in Rel$› **have** $P''' \leadsto\hat{}<Rel> Q'''$ **by**(*rule Sim*)
then obtain P'' **where** $P'''Trans$: $P''' \Longrightarrow\hat{}\alpha \prec P''$ **and** $(P'', Q'') \in Rel$ **using** $Q'''Trans$ **by**(*rule Weak-Sim.weakSimE*)
from $Q''Chain$ ‹$(P'', Q'') \in Rel$› *Sim* **obtain** P' **where** $P''Chain$: $P'' \Longrightarrow_{\tau} P'$ **and** $(P', Q') \in Rel$
by(*rule simTauChain*)
from *PChain* $P''''Chain$ $P'''Trans$ $P''Chain$
have $P \Longrightarrow\alpha \prec P'$
apply(*auto simp add*: *weakCongTrans-def weakTrans-def*)
apply(*rule-tac* $x=P''aa$ **in** *exI*)
apply *auto*
defer
apply *blast*
by(*auto simp add*: *tauChain-def*)

with ‹$(P', Q') \in Rel$› **show** *?thesis*
by(*force intro*: *Goal simp add*: *weakCongTrans-def weakTrans-def*)
qed
qed

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

assumes $Id \subseteq Rel$

shows $P \leadsto<Rel> P$
using *assms*
by(*auto simp add*: *weakCongSimulation-def intro*: *transitionWeakCongTransition*)

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

assumes $P \leadsto<Rel> Q$
and $Q \leadsto<Rel'> R$
and $Rel\ O\ Rel' \subseteq Rel''$
and $\bigwedge S\ T.\ (S, T) \in Rel \Longrightarrow S \leadsto\hat{}<Rel> T$

shows $P \leadsto<Rel''> R$
proof(*induct rule*: *weakSimI*)
case(*Sim* α R')
thus *?case* **using** *assms*

apply(*drule-tac* $Q=R$ **in** *weakSimE*, *auto*)
by(*drule-tac* $Q=Q$ **in** *simE2*) *auto*
qed

lemma *weakMonotonic*:
fixes P :: *ccs*
and A :: (*ccs* × *ccs*) *set*
and Q :: *ccs*
and B :: (*ccs* × *ccs*) *set*

assumes $P \leadsto$<A> Q
and $A \subseteq B$

shows $P \leadsto$<B> Q
using *assms*
by(*fastforce simp add*: *weakCongSimulation-def*)

end

theory *Strong-Sim-SC*
imports *Strong-Sim*
begin

lemma *resNilLeft*:
fixes x :: *name*

shows $(\!|\nu x|\!)\mathbf{0} \leadsto[Rel]\ \mathbf{0}$
by(*auto simp add*: *simulation-def*)

lemma *resNilRight*:
fixes x :: *name*

shows $\mathbf{0} \leadsto[Rel]\ (\!|\nu x|\!)\mathbf{0}$
by(*auto simp add*: *simulation-def elim*: *resCases*)

lemma *test*[*simp*]:
fixes x :: *name*
and P :: *ccs*

shows $x \sharp [x].P$
by(*auto simp add*: *abs-fresh*)

lemma *scopeExtSumLeft*:
fixes x :: *name*
and P :: *ccs*
and Q :: *ccs*

assumes $x \sharp P$
and *C1*: $\bigwedge y\ R.\ y \sharp R \Longrightarrow ((\!|\nu y|\!)R,\ R) \in Rel$

and $Id \subseteq Rel$

shows $(\!|\nu x|\!)(P \oplus Q) \leadsto[Rel]\ P \oplus (\!|\nu x|\!)Q$
using *assms*
apply(*auto simp add*: *simulation-def*)
by(*elim sumCases resCases*) (*blast intro*: *Res Sum1 Sum2 C1 dest*: *freshDerivative*)+

lemma *scopeExtSumRight*:
fixes $x :: name$
and $P :: ccs$
and $Q :: ccs$

assumes $x \sharp P$
and $C1: \bigwedge y\ R.\ y \sharp R \Longrightarrow (R, (\!|\nu y|\!)R) \in Rel$
and $Id \subseteq Rel$

shows $P \oplus (\!|\nu x|\!)Q \leadsto[Rel]\ (\!|\nu x|\!)(P \oplus Q)$
using *assms*
apply(*auto simp add*: *simulation-def*)
by(*elim sumCases resCases*) (*blast intro*: *Res Sum1 Sum2 C1 dest*: *freshDerivative*)+

lemma *scopeExtLeft*:
fixes $x :: name$
and $P :: ccs$
and $Q :: ccs$

assumes $x \sharp P$
and $C1: \bigwedge y\ R\ T.\ y \sharp R \Longrightarrow ((\!|\nu y|\!)(R \parallel T), R \parallel (\!|\nu y|\!)T) \in Rel$

shows $(\!|\nu x|\!)(P \parallel Q) \leadsto[Rel]\ P \parallel (\!|\nu x|\!)Q$
using *assms*
by(*fastforce elim*: *parCases resCases intro*: *Res C1 Par1 Par2 Comm dest*: *freshDerivative simp add*: *simulation-def*)

lemma *scopeExtRight*:
fixes $x :: name$
and $P :: ccs$
and $Q :: ccs$

assumes $x \sharp P$
and $C1: \bigwedge y\ R\ T.\ y \sharp R \Longrightarrow (R \parallel (\!|\nu y|\!)T, (\!|\nu y|\!)(R \parallel T)) \in Rel$

shows $P \parallel (\!|\nu x|\!)Q \leadsto[Rel]\ (\!|\nu x|\!)(P \parallel Q)$
using *assms*
by(*fastforce elim*: *parCases resCases intro*: *Res C1 Par1 Par2 Comm dest*: *freshDerivative simp add*: *simulation-def*)

lemma *sumComm*:
  fixes P :: *ccs*
  and Q :: *ccs*

  assumes $Id \subseteq Rel$

  shows $P \oplus Q \rightsquigarrow[Rel]\ Q \oplus P$
using *assms*
by(*force simp add*: *simulation-def elim*: *sumCases intro*: *Sum1 Sum2*)

lemma *sumAssocLeft*:
  fixes P :: *ccs*
  and Q :: *ccs*
  and R :: *ccs*

  assumes $Id \subseteq Rel$

  shows $(P \oplus Q) \oplus R \rightsquigarrow[Rel]\ P \oplus (Q \oplus R)$
using *assms*
by(*force simp add*: *simulation-def elim*: *sumCases intro*: *Sum1 Sum2*)

lemma *sumAssocRight*:
  fixes P :: *ccs*
  and Q :: *ccs*
  and R :: *ccs*

  assumes $Id \subseteq Rel$

  shows $P \oplus (Q \oplus R) \rightsquigarrow[Rel]\ (P \oplus Q) \oplus R$
using *assms*
by(*intro simI*, *elim sumCases*) (*blast intro*: *Sum1 Sum2*)+

lemma *sumIdLeft*:
  fixes P :: *ccs*
  and   *Rel* :: (*ccs* × *ccs*) *set*

  assumes $Id \subseteq Rel$

  shows $P \oplus \mathbf{0} \rightsquigarrow[Rel]\ P$
using *assms*
by(*auto simp add*: *simulation-def intro*: *Sum1*)

lemma *sumIdRight*:
  fixes P :: *ccs*
  and   *Rel* :: (*ccs* × *ccs*) *set*

  assumes $Id \subseteq Rel$

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

using *assms*
by(*fastforce simp add*: *simulation-def elim*: *sumCases*)

lemma *parComm*:
  fixes P :: *ccs*
  and Q :: *ccs*

  assumes *C1*: $\bigwedge R\ T.\ (R \parallel T,\ T \parallel R) \in Rel$

  shows $P \parallel Q \leadsto[Rel]\ Q \parallel P$
by(*fastforce simp add*: *simulation-def elim*: *parCases intro*: *Par1 Par2 Comm C1*)

lemma *parAssocLeft*:
  fixes P :: *ccs*
  and Q :: *ccs*
  and R :: *ccs*

  assumes *C1*: $\bigwedge S\ T\ U.\ ((S \parallel T) \parallel U,\ S \parallel (T \parallel U)) \in Rel$

  shows $(P \parallel Q) \parallel R \leadsto[Rel]\ P \parallel (Q \parallel R)$
by(*fastforce simp add*: *simulation-def elim*: *parCases intro*: *Par1 Par2 Comm C1*)

lemma *parAssocRight*:
  fixes P :: *ccs*
  and Q :: *ccs*
  and R :: *ccs*

  assumes *C1*: $\bigwedge S\ T\ U.\ (S \parallel (T \parallel U), (S \parallel T) \parallel U) \in Rel$

  shows $P \parallel (Q \parallel R) \leadsto[Rel]\ (P \parallel Q) \parallel R$
by(*fastforce simp add*: *simulation-def elim*: *parCases intro*: *Par1 Par2 Comm C1*)

lemma *parIdLeft*:
  fixes P :: *ccs*
  and *Rel* :: (*ccs* × *ccs*) *set*

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

  shows $P \parallel \mathbf{0} \leadsto[Rel]\ P$
using *assms*
by(*auto simp add*: *simulation-def intro*: *Par1*)

lemma *parIdRight*:
  fixes P :: *ccs*
  and *Rel* :: (*ccs* × *ccs*) *set*

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

  shows $P \leadsto[Rel]\ P \parallel \mathbf{0}$

using *assms*
by(*fastforce simp add*: *simulation-def elim*: *parCases*)

declare *fresh-atm*[*simp*]

lemma *resActLeft*:
 fixes $x :: name$
 and $\alpha :: act$
 and $P :: ccs$

 assumes $x \sharp \alpha$
 and $Id \subseteq Rel$

 shows $(\!|\nu x|\!)(\alpha.(P)) \leadsto[Rel]\ (\alpha.((\!|\nu x|\!)P))$
using *assms*
by(*fastforce simp add*: *simulation-def elim*: *actCases intro*: *Res Action*)

lemma *resActRight*:
 fixes $x :: name$
 and $\alpha :: act$
 and $P :: ccs$

 assumes $x \sharp \alpha$
 and $Id \subseteq Rel$

 shows $\alpha.((\!|\nu x|\!)P) \leadsto[Rel]\ (\!|\nu x|\!)(\alpha.(P))$
using *assms*
by(*fastforce simp add*: *simulation-def elim*: *resCases actCases intro*: *Action*)

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

 assumes $\bigwedge Q.\ ((\!|\nu x|\!)((\!|\nu y|\!)Q),\ (\!|\nu y|\!)((\!|\nu x|\!)Q)) \in Rel$

 shows $(\!|\nu x|\!)((\!|\nu y|\!)P) \leadsto[Rel]\ (\!|\nu y|\!)((\!|\nu x|\!)P)$
using *assms*
by(*fastforce simp add*: *simulation-def elim*: *resCases intro*: *Res*)

inductive-cases *bangCases*[*simplified ccs.distinct act.distinct*]: $!P \longmapsto\alpha \prec P'$

lemma *bangUnfoldLeft*:
 fixes $P :: ccs$

 assumes $Id \subseteq Rel$

 shows $P \parallel !P \leadsto[Rel]\ !P$
using *assms*

```
by(fastforce simp add: simulation-def ccs.inject elim: bangCases)

lemma bangUnfoldRight:
  fixes P :: ccs

  assumes Id ⊆ Rel

  shows !P ⇝[Rel] P ∥ !P
using assms
by(fastforce simp add: simulation-def ccs.inject intro: Bang)

end

theory Strong-Bisim
  imports Strong-Sim
begin

lemma monotonic:
  fixes P :: ccs
  and   A :: (ccs × ccs) set
  and   Q :: ccs
  and   B :: (ccs × ccs) set

  assumes P ⇝[A] Q
  and     A ⊆ B

  shows P ⇝[B] Q
using assms
by(fastforce simp add: simulation-def)

lemma monoCoinduct: ⋀x y xa xb P Q.
                      x ≤ y ⟹
                      (Q ⇝[{(xb, xa). x xb xa}] P) ⟶
                     (Q ⇝[{(xb, xa). y xb xa}] P)
apply auto
apply(rule monotonic)
by(auto dest: le-funE)

coinductive-set bisim :: (ccs × ccs) set
where
  ⟦P ⇝[bisim] Q; (Q, P) ∈ bisim⟧ ⟹ (P, Q) ∈ bisim
monos monoCoinduct

abbreviation
  bisimJudge (- ∼ - [70, 70] 65) where P ∼ Q ≡ (P, Q) ∈ bisim

lemma bisimCoinductAux[consumes 1]:
  fixes P :: ccs
  and   Q :: ccs
```

and $X :: (ccs \times ccs)\ set$

assumes $(P, Q) \in X$
and $\bigwedge P\ Q.\ (P, Q) \in X \Longrightarrow P \leadsto[(X \cup bisim)]\ Q \wedge (Q, P) \in X$

shows $P \sim Q$
proof −
have $X \cup bisim = \{(P, Q).\ (P, Q) \in X \vee (P, Q) \in bisim\}$ **by** *auto*
with *assms* **show** *?thesis*
by *coinduct simp*
qed

lemma *bisimCoinduct*[*consumes 1*, *case-names cSim cSym*]:
fixes $P :: ccs$
and $Q :: ccs$
and $X :: (ccs \times ccs)\ set$

assumes $(P, Q) \in X$
and $\bigwedge R\ S.\ (R, S) \in X \Longrightarrow R \leadsto[(X \cup bisim)]\ S$
and $\bigwedge R\ S.\ (R, S) \in X \Longrightarrow (S, R) \in X$

shows $P \sim Q$
proof −
have $X \cup bisim = \{(P, Q).\ (P, Q) \in X \vee (P, Q) \in bisim\}$ **by** *auto*
with *assms* **show** *?thesis*
by *coinduct simp*
qed

lemma *bisimWeakCoinductAux*[*consumes 1*]:
fixes $P :: ccs$
and $Q :: ccs$
and $X :: (ccs \times ccs)\ set$

assumes $(P, Q) \in X$
and $\bigwedge R\ S.\ (R, S) \in X \Longrightarrow R \leadsto[X]\ S \wedge (S, R) \in X$

shows $P \sim Q$
using *assms*
by(*coinduct rule*: *bisimCoinductAux*) (*blast intro*: *monotonic*)

lemma *bisimWeakCoinduct*[*consumes 1*, *case-names cSim cSym*]:
fixes $P :: ccs$
and $Q :: ccs$
and $X :: (ccs \times ccs)\ set$

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 $P \sim Q$
proof −
have $X \cup bisim = \{(P,\ Q).\ (P,\ Q) \in X \lor (P,\ Q) \in bisim\}$ **by** *auto*
with *assms* **show** *?thesis*
by(*coinduct rule*: *bisimCoinduct*) (*blast intro*: *monotonic*)+
qed

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

assumes $P \sim Q$

shows $P \leadsto[bisim]\ Q$
and $Q \sim P$
using *assms*
by(*auto simp add*: *intro*: *bisim.cases*)

lemma *bisimI*:
fixes P :: *ccs*
and Q :: *ccs*

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

shows $P \sim Q$
using *assms*
by(*auto intro*: *bisim.intros*)

lemma *reflexive*:
fixes P :: *ccs*

shows $P \sim P$
proof −
have $(P,\ P) \in Id$ **by** *blast*
thus *?thesis*
by(*coinduct rule*: *bisimCoinduct*) (*auto intro*: *reflexive*)
qed

lemma *symmetric*:
fixes P :: *ccs*
and Q :: *ccs*

assumes $P \sim Q$

shows $Q \sim P$
using *assms*
by(*rule bisimE*)

```
lemma transitive:
  fixes P :: ccs
  and   Q :: ccs
  and   R :: ccs

  assumes P ∼ Q
  and     Q ∼ R

  shows P ∼ R
proof −
  from assms have (P, R) ∈ bisim O bisim by auto
  thus ?thesis
    by(coinduct rule: bisimCoinduct) (auto intro: transitive dest: bisimE)
qed

lemma bisimTransCoinduct[consumes 1, case-names cSim cSym]:
  fixes P :: ccs
  and   Q :: ccs

  assumes (P, Q) ∈ X
  and     rSim: ⋀R S. (R, S) ∈ X ⟹ R ↝[(bisim O X O bisim)] S
  and     rSym: ⋀R S. (R, S) ∈ X ⟹ (S, R) ∈ X

  shows P ∼ Q
proof −
  from ‹(P, Q) ∈ X› have (P, Q) ∈ bisim O X O bisim
    by(auto intro: reflexive)
  thus ?thesis
  proof(coinduct rule: bisimWeakCoinduct)
    case(cSim P Q)
    from ‹(P, Q) ∈ bisim O X O bisim›
    obtain R S where P ∼ R and (R, S) ∈ X and S ∼ Q
      by auto
    from ‹P ∼ R› have P ↝[bisim] R by(rule bisimE)
    moreover from ‹(R, S) ∈ X› have R ↝[(bisim O X O bisim)] S
      by(rule rSim)
    moreover have bisim O (bisim O X O bisim) ⊆ bisim O X O bisim
      by(auto intro: transitive)
    ultimately have P ↝[(bisim O X O bisim)] S
      by(rule Strong-Sim.transitive)
    moreover from ‹S ∼ Q› have S ↝[bisim] Q by(rule bisimE)
    moreover have (bisim O X O bisim) O bisim ⊆ bisim O X O bisim
      by(auto intro: transitive)
    ultimately show ?case by(rule Strong-Sim.transitive)
  next
    case(cSym P Q)
    thus ?case by(auto dest: symmetric rSym)
  qed
qed
```

end

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

lemma *actPres*:
  fixes P :: *ccs*
  and Q :: *ccs*
  and *Rel* :: (*ccs* × *ccs*) *set*
  and a :: *name*
  and *Rel'* :: (*ccs* × *ccs*) *set*

  assumes $(P, Q) \in Rel$

  shows $\alpha.(P) \leadsto[Rel]\ \alpha.(Q)$
using *assms*
by(*fastforce simp add*: *simulation-def elim*: *actCases intro*: *Action*)

lemma *sumPres*:
  fixes P :: *ccs*
  and Q :: *ccs*
  and *Rel* :: (*ccs* × *ccs*) *set*

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

  shows $P \oplus R \leadsto[Rel']\ Q \oplus R$
using *assms*
by(*force simp add*: *simulation-def elim*: *sumCases intro*: *Sum1 Sum2*)

lemma *parPresAux*:
  fixes P :: *ccs*
  and Q :: *ccs*
  and *Rel* :: (*ccs* × *ccs*) *set*

  assumes $P \leadsto[Rel]\ Q$
  and $(P, Q) \in Rel$
  and $R \leadsto[Rel']\ T$
  and $(R, T) \in Rel'$
  and $C1$: $\bigwedge P'\ Q'\ R'\ T'.\ [\![(P', Q') \in Rel; (R', T') \in Rel']\!] \Longrightarrow (P' \parallel R', Q' \parallel T') \in Rel''$

  shows $P \parallel R \leadsto[Rel'']\ Q \parallel T$
proof(*induct rule*: *simI*)
  case(*Sim a QT*)
  from ‹$Q \parallel T \longmapsto a \prec QT$›

show *?case*
proof(*induct rule*: *parCases*)
case(*cPar1 Q'*)
from ‹$P \leadsto$[*Rel*] Q› ‹$Q \longmapsto a \prec Q'$› **obtain** P' **where** $P \longmapsto a \prec P'$ **and** (P', Q') ∈ *Rel*
by(*rule simE*)
from ‹$P \longmapsto a \prec P'$› **have** $P \parallel R \longmapsto a \prec P' \parallel R$ **by**(*rule Par1*)
moreover from ‹(P', Q') ∈ *Rel*› ‹(R, T) ∈ *Rel'*› **have** ($P' \parallel R$, $Q' \parallel T$) ∈ *Rel''* **by**(*rule C1*)
ultimately show *?case* **by** *blast*
next
case(*cPar2 T'*)
from ‹$R \leadsto$[*Rel'*] T› ‹$T \longmapsto a \prec T'$› **obtain** R' **where** $R \longmapsto a \prec R'$ **and** (R', T') ∈ *Rel'*
by(*rule simE*)
from ‹$R \longmapsto a \prec R'$› **have** $P \parallel R \longmapsto a \prec P \parallel R'$ **by**(*rule Par2*)
moreover from ‹(P, Q) ∈ *Rel*› ‹(R', T') ∈ *Rel'*› **have** ($P \parallel R'$, $Q \parallel T'$) ∈ *Rel''* **by**(*rule C1*)
ultimately show *?case* **by** *blast*
next
case(*cComm Q' T' a*)
from ‹$P \leadsto$[*Rel*] Q› ‹$Q \longmapsto a \prec Q'$› **obtain** P' **where** $P \longmapsto a \prec P'$ **and** (P', Q') ∈ *Rel*
by(*rule simE*)
from ‹$R \leadsto$[*Rel'*] T› ‹$T \longmapsto$(*coAction a*) $\prec T'$› **obtain** R' **where** $R \longmapsto$(*coAction a*) $\prec R'$ **and** (R', T') ∈ *Rel'*
by(*rule simE*)
from ‹$P \longmapsto a \prec P'$› ‹$R \longmapsto$(*coAction a*) $\prec R'$› ‹$a \neq \tau$› **have** $P \parallel R \longmapsto \tau \prec P' \parallel R'$ **by**(*rule Comm*)
moreover from ‹(P', Q') ∈ *Rel*› ‹(R', T') ∈ *Rel'*› **have** ($P' \parallel R'$, $Q' \parallel T'$) ∈ *Rel''* **by**(*rule C1*)
ultimately show *?case* **by** *blast*
qed
qed

lemma *parPres*:
fixes P :: *ccs*
and Q :: *ccs*
and *Rel* :: (*ccs* × *ccs*) *set*

assumes $P \leadsto$[*Rel*] Q
and (P, Q) ∈ *Rel*
and *C1*: $\bigwedge S\ T\ U.$ (S, T) ∈ *Rel* $\Longrightarrow$ ($S \parallel U$, $T \parallel U$) ∈ *Rel'*

shows $P \parallel R \leadsto$[*Rel'*] $Q \parallel R$
using *assms*
by(*rule-tac parPresAux*[**where** *Rel''*=*Rel'* **and** *Rel'*=*Id*]) (*auto intro*: *reflexive*)

lemma *resPres*:

fixes P :: *ccs*
and *Rel* :: (*ccs* × *ccs*) *set*
and Q :: *ccs*
and x :: *name*

assumes $P \leadsto[Rel]\ Q$
and $\bigwedge R\ S\ y.\ (R,\ S) \in Rel \Longrightarrow ((\!|\nu y|\!)R,\ (\!|\nu y|\!)S) \in Rel'$

shows $(\!|\nu x|\!)P \leadsto[Rel']\ (\!|\nu x|\!)Q$
using *assms*
by(*fastforce simp add*: *simulation-def elim*: *resCases intro*: *Res*)

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

assumes $(P,\ Q) \in Rel$
and *C1*: $\bigwedge R\ S.\ (R,\ S) \in Rel \Longrightarrow R \leadsto[Rel]\ S$

shows $!P \leadsto[bangRel\ Rel]\ !Q$
proof(*induct rule*: *simI*)
case(*Sim* α Q')
{
fix *Pa* α Q'
assume $!Q \longmapsto \alpha \prec Q'$ **and** $(Pa,\ !Q) \in bangRel\ Rel$
hence $\exists P'.\ Pa \longmapsto \alpha \prec P' \wedge (P',\ Q') \in bangRel\ Rel$
proof(*nominal-induct arbitrary*: *Pa rule*: *bangInduct*)
case(*cPar1* α Q')
from ‹$(Pa,\ Q \parallel\ !Q) \in bangRel\ Rel$›
show *?case*
proof(*induct rule*: *BRParCases*)
case(*BRPar P R*)
from ‹$(P,\ Q) \in Rel$› **have** $P \leadsto[Rel]\ Q$ **by**(*rule C1*)
with ‹$Q \longmapsto \alpha \prec Q'$› **obtain** P' **where** $P \longmapsto \alpha \prec P'$ **and** $(P',\ Q') \in Rel$
by(*blast dest*: *simE*)
from ‹$P \longmapsto \alpha \prec P'$› **have** $P \parallel R \longmapsto \alpha \prec P' \parallel R$ **by**(*rule Par1*)
moreover from ‹$(P',\ Q') \in Rel$› ‹$(R,\ !Q) \in bangRel\ Rel$› **have** $(P' \parallel R,$
$Q' \parallel\ !Q) \in bangRel\ Rel$
by(*rule bangRel.BRPar*)
ultimately show *?case* **by** *blast*
qed
next
case(*cPar2* α Q')
from ‹$(Pa,\ Q \parallel\ !Q) \in bangRel\ Rel$›
show *?case*
proof(*induct rule*: *BRParCases*)
case(*BRPar P R*)
from ‹$(R,\ !Q) \in bangRel\ Rel$› **obtain** R' **where** $R \longmapsto \alpha \prec R'$ **and** $(R',$

Q') $\in$ *bangRel Rel* **using** *cPar2*
by *blast*
from $\langle R \longmapsto \alpha \prec R' \rangle$ **have** $P \parallel R \longmapsto \alpha \prec P \parallel R'$ **by**(*rule Par2*)
moreover from $\langle (P, Q) \in Rel \rangle$ $\langle (R', Q') \in bangRel\ Rel \rangle$ **have** $(P \parallel R', Q \parallel Q') \in bangRel\ Rel$ **by**(*rule bangRel.BRPar*)
ultimately show *?case* **by** *blast*
qed
next
case(*cComm a* Q' Q'' *Pa*)
from $\langle (Pa, Q \parallel !Q) \in bangRel\ Rel \rangle$
show *?case*
proof(*induct rule*: *BRParCases*)
case(*BRPar P R*)
from $\langle (P, Q) \in Rel \rangle$ **have** $P \rightsquigarrow[Rel]\ Q$ **by**(*rule C1*)
with $\langle Q \longmapsto a \prec Q' \rangle$ **obtain** P' **where** $P \longmapsto a \prec P'$ **and** $(P', Q') \in Rel$
by(*blast dest*: *simE*)
from $\langle (R, !Q) \in bangRel\ Rel \rangle$ **obtain** R' **where** $R \longmapsto (coAction\ a) \prec R'$ **and** $(R', Q'') \in bangRel\ Rel$ **using** *cComm*
by *blast*
from $\langle P \longmapsto a \prec P' \rangle$ $\langle R \longmapsto (coAction\ a) \prec R' \rangle$ $\langle a \neq \tau \rangle$ **have** $P \parallel R \longmapsto \tau \prec P' \parallel R'$ **by**(*rule Comm*)
moreover from $\langle (P', Q') \in Rel \rangle$ $\langle (R', Q'') \in bangRel\ Rel \rangle$ **have** $(P' \parallel R', Q' \parallel Q'') \in bangRel\ Rel$ **by**(*rule bangRel.BRPar*)
ultimately show *?case* **by** *blast*
qed
next
case(*cBang* α Q' *Pa*)
from $\langle (Pa, !Q) \in bangRel\ Rel \rangle$
show *?case*
proof(*induct rule*: *BRBangCases*)
case(*BRBang P*)
from $\langle (P, Q) \in Rel \rangle$ **have** $(!P, !Q) \in bangRel\ Rel$ **by**(*rule bangRel.BRBang*)
with $\langle (P, Q) \in Rel \rangle$ **have** $(P \parallel !P, Q \parallel !Q) \in bangRel\ Rel$ **by**(*rule bangRel.BRPar*)
then obtain P' **where** $P \parallel !P \longmapsto \alpha \prec P'$ **and** $(P', Q') \in bangRel\ Rel$ **using** *cBang*
by *blast*
from $\langle P \parallel !P \longmapsto \alpha \prec P' \rangle$ **have** $!P \longmapsto \alpha \prec P'$ **by**(*rule Bang*)
thus *?case* **using** $\langle (P', Q') \in bangRel\ Rel \rangle$ **by** *blast*
qed
qed
}

moreover from $\langle (P, Q) \in Rel \rangle$ **have** $(!P, !Q) \in bangRel\ Rel$ **by**(*rule BRBang*)

ultimately show *?case* **using** $\langle !Q \longmapsto \alpha \prec Q' \rangle$ **by** *blast*
qed

end

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

lemma *actPres*:
  fixes P :: *ccs*
  and Q :: *ccs*
  and α :: *act*

  assumes $P \sim Q$

  shows $\alpha.(P) \sim \alpha.(Q)$
proof −
  let $?X = \{(\alpha.(P), \alpha.(Q)) \mid P\ Q.\ P \sim Q\}$
  from *assms* **have** $(\alpha.(P), \alpha.(Q)) \in ?X$ **by** *auto*
  thus *?thesis*
    by(*coinduct rule*: *bisimCoinduct*) (*auto dest*: *bisimE intro*: *actPres*)
qed

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

  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** *auto*
  thus *?thesis*
    by(*coinduct rule*: *bisimCoinduct*) (*auto intro*: *sumPres reflexive dest*: *bisimE*)
qed

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

  assumes $P \sim Q$

  shows $P \parallel R \sim Q \parallel R$
proof −
  let $?X = \{(P \parallel R, Q \parallel R) \mid P\ Q\ R.\ P \sim Q\}$
  from *assms* **have** $(P \parallel R, Q \parallel R) \in ?X$ **by** *blast*
  thus *?thesis*
    by(*coinduct rule*: *bisimCoinduct*, *auto*) (*blast intro*: *parPres dest*: *bisimE*)+
qed

```
lemma resPres:
  fixes P :: ccs
  and   Q :: ccs
  and   x :: name

  assumes P ∼ Q

  shows ⦇νx⦈P ∼ ⦇νx⦈Q
proof −
  let ?X = {(⦇νx⦈P, ⦇νx⦈Q) | x P Q. P ∼ Q}
  from assms have (⦇νx⦈P, ⦇νx⦈Q) ∈ ?X by auto
  thus ?thesis
    by(coinduct rule: bisimCoinduct) (auto intro: resPres dest: bisimE)
qed

lemma bangPres:
  fixes P :: ccs
  and   Q :: ccs

  assumes P ∼ Q

  shows !P ∼ !Q
proof −
  from assms have (!P, !Q) ∈ bangRel bisim
    by(auto intro: BRBang)
  thus ?thesis
  proof(coinduct rule: bisimWeakCoinduct)
    case(cSim P Q)
    from ‹(P, Q) ∈ bangRel bisim› show ?case
    proof(induct)
      case(BRBang P Q)
      note ‹P ∼ Q› bisimE(1)
      thus !P ↝[bangRel bisim] !Q by(rule bangPres)
    next
      case(BRPar R T P Q)
      from ‹R ∼ T› have R ↝[bisim] T by(rule bisimE)
      moreover note ‹R ∼ T› ‹P ↝[bangRel bisim] Q› ‹(P, Q) ∈ bangRel bisim›
bangRel.BRPar
      ultimately show ?case by(rule Strong-Sim-Pres.parPresAux)
    qed
  next
    case(cSym P Q)
    thus ?case
      by induct (auto dest: bisimE intro: BRPar BRBang)
  qed
qed

end
```

theory *Struct-Cong*
  imports *Agent*
begin

inductive *structCong* :: *ccs* ⇒ *ccs* ⇒ *bool* (- $\equiv_s$ -)
  where
  Refl: $P \equiv_s P$
| *Sym*: $P \equiv_s Q \Longrightarrow Q \equiv_s P$
| *Trans*: $[\![P \equiv_s Q; Q \equiv_s R]\!] \Longrightarrow P \equiv_s R$

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

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

| *ResNil*: $(\!|\nu x|\!)\mathbf{0} \equiv_s \mathbf{0}$
| *ScopeExtPar*: $x \sharp P \Longrightarrow (\!|\nu x|\!)(P \parallel Q) \equiv_s P \parallel (\!|\nu x|\!)Q$
| *ScopeExtSum*: $x \sharp P \Longrightarrow (\!|\nu x|\!)(P \oplus Q) \equiv_s P \oplus (\!|\nu x|\!)Q$
| *ScopeAct*: $x \sharp \alpha \Longrightarrow (\!|\nu x|\!)(\alpha.(P)) \equiv_s \alpha.((\!|\nu x|\!)P)$
| *ScopeCommAux*: $x \neq y \Longrightarrow (\!|\nu x|\!)((\!|\nu y|\!)P) \equiv_s (\!|\nu y|\!)((\!|\nu x|\!)P)$

| *BangUnfold*: $!P \equiv_s P \parallel !P$
equivariance *structCong*
nominal-inductive *structCong*
by(*auto simp add*: *abs-fresh*)

lemma *ScopeComm*:
  fixes *x* :: *name*
  and *y* :: *name*
  and *P* :: *ccs*

  shows $(\!|\nu x|\!)((\!|\nu y|\!)P) \equiv_s (\!|\nu y|\!)((\!|\nu x|\!)P)$
by(*cases x=y*) (*auto intro*: *Refl ScopeCommAux*)

end

theory *Strong-Bisim-SC*
  imports *Strong-Sim-SC Strong-Bisim-Pres Struct-Cong*
begin

lemma *resNil*:
  fixes *x* :: *name*

  shows $(\!|\nu x|\!)\mathbf{0} \sim \mathbf{0}$
proof −

have ($(\!|\nu x|\!)$**0**, **0**) ∈ {($(\!|\nu x|\!)$**0**, **0**), (**0**, $(\!|\nu x|\!)$**0**)} **by** *simp*
thus *?thesis*
by(*coinduct rule*: *bisimCoinduct*)
(*auto intro*: *resNilLeft resNilRight*)
qed

lemma *scopeExt*:
fixes *x* :: *name*
and *P* :: *ccs*
and *Q* :: *ccs*

assumes $x \sharp P$

shows $(\!|\nu x|\!)(P \parallel Q) \sim P \parallel (\!|\nu x|\!)Q$
proof −
let $?X = \{((\!|\nu x|\!)(P \parallel Q),\ P \parallel (\!|\nu x|\!)Q) \mid x\ P\ Q.\ x \sharp P\} \cup \{(P \parallel (\!|\nu x|\!)Q,\ (\!|\nu x|\!)(P \parallel Q)) \mid x\ P\ Q.\ x \sharp P\}$
from *assms* **have** $((\!|\nu x|\!)(P \parallel Q),\ P \parallel (\!|\nu x|\!)Q) \in\ ?X$ **by** *auto*
thus *?thesis*
by(*coinduct rule*: *bisimCoinduct*) (*force intro*: *scopeExtLeft scopeExtRight*)+
qed

lemma *sumComm*:
fixes *P* :: *ccs*
and *Q* :: *ccs*

shows $P \oplus Q \sim Q \oplus P$
proof −
have $(P \oplus Q,\ Q \oplus P) \in \{(P \oplus Q,\ Q \oplus P),\ (Q \oplus P,\ P \oplus Q)\}$ **by** *simp*
thus *?thesis*
by(*coinduct rule*: *bisimCoinduct*) (*auto intro*: *sumComm reflexive*)
qed

lemma *sumAssoc*:
fixes *P* :: *ccs*
and *Q* :: *ccs*
and *R* :: *ccs*

shows $(P \oplus Q) \oplus R \sim P \oplus (Q \oplus R)$
proof −
have $((P \oplus Q) \oplus R,\ P \oplus (Q \oplus R)) \in \{((P \oplus Q) \oplus R,\ P \oplus (Q \oplus R)),\ (P \oplus (Q \oplus R),\ (P \oplus Q) \oplus R)\}$ **by** *simp*
thus *?thesis*
by(*coinduct rule*: *bisimCoinduct*) (*auto intro*: *sumAssocLeft sumAssocRight reflexive*)
qed

lemma *sumId*:
fixes *P* :: *ccs*

shows $P \oplus \mathbf{0} \sim P$
proof −
have $(P \oplus \mathbf{0},\ P) \in \{(P \oplus \mathbf{0},\ P),\ (P,\ P \oplus \mathbf{0})\}$ **by** *simp*
thus *?thesis* **by**(*coinduct rule*: *bisimCoinduct*) (*auto intro*: *sumIdLeft sumIdRight reflexive*)
qed

lemma *parComm*:
fixes P :: *ccs*
and Q :: *ccs*

shows $P \parallel Q \sim Q \parallel P$
proof −
have $(P \parallel Q,\ Q \parallel P) \in \{(P \parallel Q,\ Q \parallel P) \mid P\ Q.\ \mathit{True}\} \cup \{(Q \parallel P,\ P \parallel Q) \mid P\ Q.\ \mathit{True}\}$ **by** *auto*
thus *?thesis*
by(*coinduct rule*: *bisimCoinduct*) (*auto intro*: *parComm*)
qed

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

shows $(P \parallel Q) \parallel R \sim P \parallel (Q \parallel R)$
proof −
have $((P \parallel Q) \parallel R,\ P \parallel (Q \parallel R)) \in \{((P \parallel Q) \parallel R,\ P \parallel (Q \parallel R)) \mid P\ Q\ R.\ \mathit{True}\} \cup$
$\{(P \parallel (Q \parallel R),\ (P \parallel Q) \parallel R) \mid P\ Q\ R.\ \mathit{True}\}$ **by** *auto*
thus *?thesis*
by(*coinduct rule*: *bisimCoinduct*) (*force intro*: *parAssocLeft parAssocRight*)+
qed

lemma *parId*:
fixes P :: *ccs*

shows $P \parallel \mathbf{0} \sim P$
proof −
have $(P \parallel \mathbf{0},\ P) \in \{(P \parallel \mathbf{0},\ P) \mid P.\ \mathit{True}\} \cup \{(P,\ P \parallel \mathbf{0}) \mid P.\ \mathit{True}\}$ **by** *simp*
thus *?thesis* **by**(*coinduct rule*: *bisimCoinduct*) (*auto intro*: *parIdLeft parIdRight*)
qed

lemma *scopeFresh*:
fixes x :: *name*
and P :: *ccs*

assumes $x \sharp P$

shows $(\!|\nu x|\!)P \sim P$
proof −
have $(\!|\nu x|\!)P \sim (\!|\nu x|\!)P \parallel \mathbf{0}$ **by**(*rule parId*[*THEN symmetric*])
moreover have $(\!|\nu x|\!)P \parallel \mathbf{0} \sim \mathbf{0} \parallel (\!|\nu x|\!)P$ **by**(*rule parComm*)
moreover have $\mathbf{0} \parallel (\!|\nu x|\!)P \sim (\!|\nu x|\!)(\mathbf{0} \parallel P)$ **by**(*rule scopeExt*[*THEN symmetric*]) *auto*
moreover have $(\!|\nu x|\!)(\mathbf{0} \parallel P) \sim (\!|\nu x|\!)(P \parallel \mathbf{0})$ **by**(*rule resPres*[*OF parComm*])
moreover from ‹$x \sharp P$› **have** $(\!|\nu x|\!)(P \parallel \mathbf{0}) \sim P \parallel (\!|\nu x|\!)\mathbf{0}$ **by**(*rule scopeExt*)
moreover have $P \parallel (\!|\nu x|\!)\mathbf{0} \sim (\!|\nu x|\!)\mathbf{0} \parallel P$ **by**(*rule parComm*)
moreover have $(\!|\nu x|\!)\mathbf{0} \parallel P \sim \mathbf{0} \parallel P$ **by**(*rule parPres*[*OF resNil*])
moreover have $\mathbf{0} \parallel P \sim P \parallel \mathbf{0}$ **by**(*rule parComm*)
moreover have $P \parallel \mathbf{0} \sim P$ **by**(*rule parId*)
ultimately show *?thesis* **by**(*metis transitive*)
qed

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

assumes $x \sharp P$

shows $(\!|\nu x|\!)(P \oplus Q) \sim P \oplus (\!|\nu x|\!)Q$
proof −
have $((\!|\nu x|\!)(P \oplus Q), P \oplus (\!|\nu x|\!)Q) \in \{((\!|\nu x|\!)(P \oplus Q), P \oplus (\!|\nu x|\!)Q), (P \oplus (\!|\nu x|\!)Q, (\!|\nu x|\!)(P \oplus Q))\}$
by *simp*
thus *?thesis* **using** ‹$x \sharp P$›
by(*coinduct rule*: *bisimCoinduct*)
(*auto intro*: *scopeExtSumLeft scopeExtSumRight reflexive scopeFresh scopeFresh*[*THEN symmetric*])
qed

lemma *resAct*:
fixes x :: *name*
and α :: *act*
and P :: *ccs*

assumes $x \sharp \alpha$

shows $(\!|\nu x|\!)(\alpha.(P)) \sim \alpha.((\!|\nu x|\!)P)$
proof −
have $((\!|\nu x|\!)(\alpha.(P)), \alpha.((\!|\nu x|\!)P)) \in \{((\!|\nu x|\!)(\alpha.(P)), \alpha.((\!|\nu x|\!)P)), (\alpha.((\!|\nu x|\!)P), (\!|\nu x|\!)(\alpha.(P)))\}$
by *simp*
thus *?thesis* **using** ‹$x \sharp \alpha$›
by(*coinduct rule*: *bisimCoinduct*) (*auto intro*: *resActLeft resActRight reflexive*)
qed

lemma *resComm*:

fixes x :: *name*
and y :: *name*
and P :: *ccs*

shows $(\!|\nu x|\!)((\!|\nu y|\!)P) \sim (\!|\nu y|\!)((\!|\nu x|\!)P)$
proof −
have $((\!|\nu x|\!)((\!|\nu y|\!)P), (\!|\nu y|\!)((\!|\nu x|\!)P)) \in \{((\!|\nu x|\!)((\!|\nu y|\!)P), (\!|\nu y|\!)((\!|\nu x|\!)P)) \mid x\ y\ P.$ *True*$\}$ **by** *auto*
thus *?thesis*
by(*coinduct rule*: *bisimCoinduct*) (*auto intro*: *resComm*)
qed

lemma *bangUnfold*:
fixes P

shows $!P \sim P \parallel !P$
proof −
have $(!P, P \parallel !P) \in \{(!P, P \parallel !P), (P \parallel !P, !P)\}$ **by** *auto*
thus *?thesis*
by(*coinduct rule*: *bisimCoinduct*) (*auto intro*: *bangUnfoldLeft bangUnfoldRight reflexive*)
qed

lemma *bisimStructCong*:
fixes P :: *ccs*
and Q :: *ccs*

assumes $P \equiv_s Q$

shows $P \sim Q$
using *assms*
apply(*nominal-induct rule*: *Struct-Cong.strong-induct*)
by(*auto intro*: *reflexive symmetric transitive parComm parAssoc parId sumComm sumAssoc sumId resNil scopeExt scopeExtSum resAct resComm bangUnfold*)

end

theory *Weak-Bisim*
imports *Weak-Sim Strong-Bisim-SC Struct-Cong*
begin

lemma *weakMonoCoinduct*: $\bigwedge x\ y\ xa\ xb\ P\ Q.$
$x \leq y \Longrightarrow$
$(Q \leadsto\hat{}<\{(xb, xa).\ x\ xb\ xa\}> P) \longrightarrow$
$(Q \leadsto\hat{}<\{(xb, xa).\ y\ xb\ xa\}> P)$
apply *auto*
apply(*rule weakMonotonic*)
by(*auto dest*: *le-funE*)

coinductive-set *weakBisimulation* :: (*ccs* × *ccs*) *set*
where
⟦*P* ⇝^<*weakBisimulation*> *Q*; (*Q*, *P*) ∈ *weakBisimulation*⟧ ⟹ (*P*, *Q*) ∈ *weakBisimulation*
monos *weakMonoCoinduct*

abbreviation
weakBisimJudge (- ≈ - [*70*, *70*] *65*) **where** *P* ≈ *Q* ≡ (*P*, *Q*) ∈ *weakBisimulation*

lemma *weakBisimulationCoinductAux*[*consumes 1*]:
fixes *P* :: *ccs*
and *Q* :: *ccs*
and *X* :: (*ccs* × *ccs*) *set*

assumes (*P*, *Q*) ∈ *X*
and ⋀*P Q*. (*P*, *Q*) ∈ *X* ⟹ *P* ⇝^<(*X* ∪ *weakBisimulation*)> *Q* ∧ (*Q*, *P*) ∈ *X*

shows *P* ≈ *Q*
proof −
have *X* ∪ *weakBisimulation* = {(*P*, *Q*). (*P*, *Q*) ∈ *X* ∨ (*P*, *Q*) ∈ *weakBisimulation*} **by** *auto*
with *assms* **show** *?thesis*
by *coinduct simp*
qed

lemma *weakBisimulationCoinduct*[*consumes 1*, *case-names cSim cSym*]:
fixes *P* :: *ccs*
and *Q* :: *ccs*
and *X* :: (*ccs* × *ccs*) *set*

assumes (*P*, *Q*) ∈ *X*
and ⋀*R S*. (*R*, *S*) ∈ *X* ⟹ *R* ⇝^<(*X* ∪ *weakBisimulation*)> *S*
and ⋀*R S*. (*R*, *S*) ∈ *X* ⟹ (*S*, *R*) ∈ *X*

shows *P* ≈ *Q*
proof −
have *X* ∪ *weakBisimulation* = {(*P*, *Q*). (*P*, *Q*) ∈ *X* ∨ (*P*, *Q*) ∈ *weakBisimulation*} **by** *auto*
with *assms* **show** *?thesis*
by *coinduct simp*
qed

lemma *weakBisimWeakCoinductAux*[*consumes 1*]:
fixes *P* :: *ccs*
and *Q* :: *ccs*
and *X* :: (*ccs* × *ccs*) *set*

assumes (*P*, *Q*) ∈ *X*

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

shows $P \approx Q$
using *assms*
by(*coinduct rule*: *weakBisimulationCoinductAux*) (*blast intro*: *weakMonotonic*)

lemma *weakBisimWeakCoinduct*[*consumes 1*, *case-names cSim cSym*]:
fixes P :: *ccs*
and Q :: *ccs*
and X :: (*ccs* × *ccs*) *set*

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

shows $P \approx Q$
proof −
have $X \cup$ *weakBisim* = $\{(P,\ Q).\ (P,\ Q) \in X \vee (P,\ Q) \in$ *weakBisim*$\}$ **by** *auto*
with *assms* **show** *?thesis*
by(*coinduct rule*: *weakBisimulationCoinduct*) (*blast intro*: *weakMonotonic*)+
qed

lemma *weakBisimulationE*:
fixes P :: *ccs*
and Q :: *ccs*

assumes $P \approx Q$

shows $P \leadsto\hat{}\ \text{<}$*weakBisimulation*$\text{>}\ Q$
and $Q \approx P$
using *assms*
by(*auto simp add*: *intro*: *weakBisimulation.cases*)

lemma *weakBisimulationI*:
fixes P :: *ccs*
and Q :: *ccs*

assumes $P \leadsto\hat{}\ \text{<}$*weakBisimulation*$\text{>}\ Q$
and $Q \approx P$

shows $P \approx Q$
using *assms*
by(*auto intro*: *weakBisimulation.intros*)

lemma *reflexive*:
fixes P :: *ccs*

shows $P \approx P$
proof −

have $(P, P) \in Id$ **by** *blast*
thus *?thesis*
by(*coinduct rule*: *weakBisimulationCoinduct*) (*auto intro*: *Weak-Sim.reflexive*)
qed

lemma *symmetric*:
fixes P :: *ccs*
and Q :: *ccs*

assumes $P \approx Q$

shows $Q \approx P$
using *assms*
by(*rule weakBisimulationE*)

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

assumes $P \approx Q$
and $Q \approx R$

shows $P \approx R$
proof −
from *assms* **have** $(P, R) \in$ *weakBisimulation O weakBisimulation* **by** *auto*
thus *?thesis*
proof(*coinduct rule*: *weakBisimulationCoinduct*)
case(*cSim P R*)
from ‹$(P, R) \in$ *weakBisimulation O weakBisimulation*›
obtain Q **where** $P \approx Q$ **and** $Q \approx R$ **by** *auto*
note ‹$P \approx Q$›
moreover from ‹$Q \approx R$› **have** $Q \rightsquigarrow\hat{}$ <*weakBisimulation*> R **by**(*rule weak-BisimulationE*)
moreover have *weakBisimulation O weakBisimulation* $\subseteq$ (*weakBisimulation O weakBisimulation*) $\cup$ *weakBisimulation*
by *auto*
moreover note *weakBisimulationE*(*1*)
ultimately show *?case* **by**(*rule Weak-Sim.transitive*)
next
case(*cSym P R*)
thus *?case* **by**(*blast dest*: *symmetric*)
qed
qed

lemma *bisimWeakBisimulation*:
fixes P :: *ccs*
and Q :: *ccs*

```
  assumes P ∼ Q

  shows P ≈ Q
using assms
by(coinduct rule: weakBisimWeakCoinduct[where X=bisim])
  (auto dest: bisimE simWeakSim)

lemma structCongWeakBisimulation:
  fixes P :: ccs
  and   Q :: ccs

  assumes P ≡s Q

  shows P ≈ Q
using assms

by(auto intro: bisimWeakBisimulation bisimStructCong)

lemma strongAppend:
  fixes P      :: ccs
  and   Q      :: ccs
  and   R      :: ccs
  and   Rel    :: (ccs × ccs) set
  and   Rel'   :: (ccs × ccs) set
  and   Rel''  :: (ccs × ccs) set

  assumes PSimQ: P ⇝^<Rel> Q
  and     QSimR: Q ⇝[Rel'] R
  and     Trans: Rel O Rel' ⊆ Rel''

  shows P ⇝^<Rel''> R
using assms
by(simp add: weakSimulation-def simulation-def) blast

lemma weakBisimWeakUpto[case-names cSim cSym, consumes 1]:
  assumes p: (P, Q) ∈ X
  and rSim: ⋀P Q. (P, Q) ∈ X ⟹ P ⇝^<(weakBisimulation O X O bisim)> Q
  and rSym: ⋀ P Q. (P, Q) ∈ X ⟹ (Q, P) ∈ X

  shows P ≈ Q
proof -
  let ?X = weakBisimulation O X O weakBisimulation
  let ?Y = weakBisimulation O X O bisim
  from ‹(P, Q) ∈ X› have (P, Q) ∈ ?X by(blast intro: Strong-Bisim.reflexive
reflexive)
  thus ?thesis
  proof(coinduct rule: weakBisimWeakCoinduct)
    case(cSim P Q)
    {
```

fix P P′ Q′ Q
assume P ≈ P′ **and** (P′, Q′) ∈ X **and** Q′ ≈ Q
from ‹(P′, Q′) ∈ X› **have** (P′, Q′) ∈ ?Y **by**(*blast intro*: *reflexive Strong-Bisim.reflexive*)
moreover from ‹Q′ ≈ Q› **have** Q′ ⇝^<weakBisimulation> Q **by**(*rule weakBisimulationE*)
moreover have ?Y O weakBisimulation ⊆ ?X **by**(*blast dest*: *bisimWeakBisimulation transitive*)
moreover {
fix P Q
assume (P, Q) ∈ ?Y
then obtain P′ Q′ **where** P ≈ P′ **and** (P′, Q′) ∈ X **and** Q′ ∼ Q **by** *auto*
from ‹(P′, Q′) ∈ X› **have** P′ ⇝^<?Y> Q′ **by**(*rule rSim*)
moreover from ‹Q′ ∼ Q› **have** Q′ ⇝[bisim] Q **by**(*rule bisimE*)
moreover have ?Y O bisim ⊆ ?Y **by**(*auto dest*: *Strong-Bisim.transitive*)
ultimately have P′ ⇝^<?Y> Q **by**(*rule strongAppend*)
moreover note ‹P ≈ P′›
moreover have weakBisimulation O ?Y ⊆ ?Y **by**(*blast dest*: *transitive*)
ultimately have P ⇝^<?Y> Q **using** *weakBisimulationE*(*1*)
by(*rule-tac Weak-Sim.transitive*)
}
ultimately have P′ ⇝^<?X> Q **by**(*rule Weak-Sim.transitive*)
moreover note ‹P ≈ P′›
moreover have weakBisimulation O ?X ⊆ ?X **by**(*blast dest*: *transitive*)
ultimately have P ⇝^<?X> Q **using** *weakBisimulationE*(*1*)
by(*rule-tac Weak-Sim.transitive*)
}
with ‹(P, Q) ∈ ?X› **show** ?case **by** *auto*
next
case(*cSym P Q*)
thus ?case
apply *auto*
by(*blast dest*: *bisimE rSym weakBisimulationE*)
qed
qed

lemma *weakBisimUpto*[*case-names cSim cSym*, *consumes 1*]:
assumes *p*: (P, Q) ∈ X
and *rSim*: ⋀R S. (R, S) ∈ X ⟹ R ⇝^<(weakBisimulation O (X ∪ weakBisimulation) O bisim)> S
and *rSym*: ⋀R S. (R, S) ∈ X ⟹ (S, R) ∈ X

shows P ≈ Q
proof −
from *p* **have** (P, Q) ∈ X ∪ weakBisimulation **by** *simp*
thus ?thesis
apply(*coinduct rule*: *weakBisimWeakUpto*)
apply(*auto dest*: *rSim rSym weakBisimulationE*)
apply(*rule weakMonotonic*)
apply(*blast dest*: *weakBisimulationE*)

apply(*auto simp add*: *relcomp-unfold*)
by(*metis reflexive Strong-Bisim.reflexive transitive*)
qed

end

theory *Weak-Cong*
imports *Weak-Cong-Sim Weak-Bisim Strong-Bisim-SC*
begin

definition *weakCongruence* :: *ccs* ⇒ *ccs* ⇒ *bool* (- ≅ - [*70*, *70*] *65*)
where
$P \cong Q \equiv P \rightsquigarrow$<*weakBisimulation*> $Q \wedge Q \rightsquigarrow$<*weakBisimulation*> P

lemma *weakCongruenceE*:
fixes *P* :: *ccs*
and *Q* :: *ccs*

assumes $P \cong Q$

shows $P \rightsquigarrow$<*weakBisimulation*> Q
and $Q \rightsquigarrow$<*weakBisimulation*> P
using *assms*
by(*auto simp add*: *weakCongruence-def*)

lemma *weakCongruenceI*:
fixes *P* :: *ccs*
and *Q* :: *ccs*

assumes $P \rightsquigarrow$<*weakBisimulation*> Q
and $Q \rightsquigarrow$<*weakBisimulation*> P

shows $P \cong Q$
using *assms*
by(*auto simp add*: *weakCongruence-def*)

lemma *weakCongISym*[*consumes 1*, *case-names cSym cSim*]:
fixes *P* :: *ccs*
and *Q* :: *ccs*

assumes *Prop P Q*
and $\bigwedge P\ Q$. *Prop P Q* $\Longrightarrow$ *Prop Q P*
and $\bigwedge P\ Q$. *Prop P Q* $\Longrightarrow$ $(F\ P) \rightsquigarrow$<*weakBisimulation*> $(F\ Q)$

shows $F\ P \cong F\ Q$
using *assms*
by(*auto simp add*: *weakCongruence-def*)

lemma *weakCongISym2*[*consumes 1*, *case-names cSim*]:

```
  fixes P :: ccs
  and   Q :: ccs

  assumes P ≅ Q
  and     ⋀P Q. P ≅ Q ⟹ (F P) ⇝<weakBisimulation> (F Q)

  shows F P ≅ F Q
using assms
by(auto simp add: weakCongruence-def)

lemma reflexive:
  fixes P :: ccs

  shows P ≅ P
by(auto intro: weakCongruenceI Weak-Bisim.reflexive Weak-Cong-Sim.reflexive)

lemma symmetric:
  fixes P :: ccs
  and   Q :: ccs

  assumes P ≅ Q

  shows Q ≅ P
using assms
by(auto simp add: weakCongruence-def)

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

  assumes P ≅ Q
  and     Q ≅ R

  shows P ≅ R
proof –
  let ?Prop = λP R. ∃ Q. P ≅ Q ∧ Q ≅ R
  from assms have ?Prop P R by auto
  thus ?thesis
  proof(induct rule: weakCongISym)
    case(cSym P R)
    thus ?case by(auto dest: symmetric)
  next
    case(cSim P R)
    from ‹?Prop P R› obtain Q where P ≅ Q and Q ≅ R
      by auto
    from ‹P ≅ Q› have P ⇝<weakBisimulation> Q by(rule weakCongruenceE)
   moreover from ‹Q ≅ R› have Q ⇝<weakBisimulation> R by(rule weakCon-
gruenceE)
```

moreover from *Weak-Bisim.transitive* **have** *weakBisimulation O weakBisimulation* ⊆ *weakBisimulation*
by *auto*
ultimately show *?case* **using** *weakBisimulationE*(*1*)
by(*rule Weak-Cong-Sim.transitive*)
qed
qed

lemma *bisimWeakCongruence*:
fixes *P* :: *ccs*
and *Q* :: *ccs*

assumes *P* ∼ *Q*

shows *P* ≅ *Q*
using *assms*
proof(*induct rule*: *weakCongISym*)
case(*cSym P Q*)
thus *?case* **by**(*rule bisimE*)
next
case(*cSim P Q*)
from ‹*P* ∼ *Q*› **have** *P* ⇝[*bisim*] *Q* **by**(*rule bisimE*)
hence *P* ⇝[*weakBisimulation*] *Q* **using** *bisimWeakBisimulation*
by(*rule-tac monotonic*) *auto*
thus *?case* **by**(*rule simWeakSim*)
qed

lemma *structCongWeakCongruence*:
fixes *P* :: *ccs*
and *Q* :: *ccs*

assumes *P* $\equiv_s$ *Q*

shows *P* ≅ *Q*
using *assms*
by(*auto intro*: *bisimWeakCongruence bisimStructCong*)

lemma *weakCongruenceWeakBisimulation*:
fixes *P* :: *ccs*
and *Q* :: *ccs*

assumes *P* ≅ *Q*

shows *P* ≈ *Q*
proof −
let *?X* = {(*P*, *Q*) | *P Q*. *P* ≅ *Q*}
from *assms* **have** (*P*, *Q*) ∈ *?X* **by** *auto*
thus *?thesis*
proof(*coinduct rule*: *weakBisimulationCoinduct*)

```
    case(cSim P Q)
    from ‹(P, Q) ∈ ?X› have P ≅ Q by auto
    hence P ↝<weakBisimulation> Q by(rule Weak-Cong.weakCongruenceE)
   hence P ↝<(?X ∪ weakBisimulation)> Q by(force intro: Weak-Cong-Sim.weakMonotonic)
    thus ?case by(rule weakCongSimWeakSim)
  next
    case(cSym P Q)
    from ‹(P, Q) ∈ ?X› show ?case by(blast dest: symmetric)
  qed
qed


end

theory Weak-Sim-Pres
  imports Weak-Sim
begin

lemma actPres:
  fixes P    :: ccs
  and   Q    :: ccs
  and   Rel  :: (ccs × ccs) set
  and   a    :: name
  and   Rel' :: (ccs × ccs) set

  assumes (P, Q) ∈ Rel

  shows α.(P) ↝^<Rel> α.(Q)
using assms
by(fastforce simp add: weakSimulation-def elim: actCases intro: weakAction)

lemma sumPres:
  fixes P   :: ccs
  and   Q   :: ccs
  and   Rel :: (ccs × ccs) set

  assumes P ↝^<Rel> Q
  and     Rel ⊆ Rel'
  and     Id ⊆ Rel'
  and     C1: ⋀S T U. (S, T) ∈ Rel ⟹ (S ⊕ U, T) ∈ Rel'

  shows P ⊕ R ↝^<Rel'> Q ⊕ R
proof(induct rule: weakSimI)
  case(Sim α QR)
  from ‹Q ⊕ R ⟼α ≺ QR› show ?case
  proof(induct rule: sumCases)
    case(cSum1 Q')
    from ‹P ↝^<Rel> Q› ‹Q ⟼α ≺ Q'›
    obtain P' where P ⟹^α ≺ P' and (P', Q') ∈ Rel
```

by(*blast dest*: *weakSimE*)
thus *?case*
proof(*induct rule*: *weakTransCases*)
case *Base*
have $P \oplus R \Longrightarrow\hat{}\ \tau \prec P \oplus R$ **by** *simp*
moreover from ‹$(P,\ Q') \in Rel$› **have** $(P \oplus R,\ Q') \in Rel'$ **by**(*rule C1*)
ultimately show *?case* **by** *blast*
next
case *Step*
from ‹$P \Longrightarrow\alpha \prec P'$› **have** $P \oplus R \Longrightarrow\alpha \prec P'$ **by**(*rule weakCongSum1*)
hence $P \oplus R \Longrightarrow\hat{}\ \alpha \prec P'$ **by**(*simp add*: *weakTrans-def*)
thus *?case* **using** ‹$(P',\ Q') \in Rel$› ‹$Rel \subseteq Rel'$› **by** *blast*
qed
next
case(*cSum2* R')
from ‹$R \longmapsto\alpha \prec R'$› **have** $R \Longrightarrow\alpha \prec R'$ **by**(*rule transitionWeakCongTransition*)
hence $P \oplus R \Longrightarrow\alpha \prec R'$ **by**(*rule weakCongSum2*)
hence $P \oplus R \Longrightarrow\hat{}\ \alpha \prec R'$ **by**(*simp add*: *weakTrans-def*)
thus *?case* **using** ‹$Id \subseteq Rel'$› **by** *blast*
qed
qed

lemma *parPresAux*:
fixes P :: *ccs*
and Q :: *ccs*
and R :: *ccs*
and T :: *ccs*
and Rel :: $(ccs \times ccs)$ *set*
and Rel' :: $(ccs \times ccs)$ *set*
and Rel'' :: $(ccs \times ccs)$ *set*

assumes $P \leadsto\hat{}<Rel> Q$
and $(P,\ Q) \in Rel$
and $R \leadsto\hat{}<Rel'> T$
and $(R,\ T) \in Rel'$
and *C1*: $\bigwedge P'\ Q'\ R'\ T'.\ [\![(P',\ Q') \in Rel;\ (R',\ T') \in Rel']\!] \Longrightarrow (P' \parallel R',\ Q' \parallel T') \in Rel''$

shows $P \parallel R \leadsto\hat{}<Rel''> Q \parallel T$
proof(*induct rule*: *weakSimI*)
case(*Sim* α *QT*)
from ‹$Q \parallel T \longmapsto\alpha \prec QT$›
show *?case*
proof(*induct rule*: *parCases*)
case(*cPar1* Q')
from ‹$P \leadsto\hat{}<Rel> Q$› ‹$Q \longmapsto\alpha \prec Q'$› **obtain** P' **where** $P \Longrightarrow\hat{}\ \alpha \prec P'$ **and** $(P',\ Q') \in Rel$
by(*rule weakSimE*)
from ‹$P \Longrightarrow\hat{}\ \alpha \prec P'$› **have** $P \parallel R \Longrightarrow\hat{}\ \alpha \prec P' \parallel R$ **by**(*rule weakPar1*)

moreover from ‹$(P', Q') \in Rel$› ‹$(R, T) \in Rel'$› **have** $(P' \parallel R, Q' \parallel T) \in Rel''$ **by**(*rule C1*)
  ultimately show *?case* **by** *blast*
 next
  case(*cPar2 T'*)
  from ‹$R \leadsto\hat{} <Rel'> T$› ‹$T \longmapsto\alpha \prec T'$› **obtain** R' **where** $R \Longrightarrow\hat{} \alpha \prec R'$ **and** $(R', T') \in Rel'$
   by(*rule weakSimE*)
  from ‹$R \Longrightarrow\hat{} \alpha \prec R'$› **have** $P \parallel R \Longrightarrow\hat{} \alpha \prec P \parallel R'$ **by**(*rule weakPar2*)
  moreover from ‹$(P, Q) \in Rel$› ‹$(R', T') \in Rel'$› **have** $(P \parallel R', Q \parallel T') \in Rel''$ **by**(*rule C1*)
  ultimately show *?case* **by** *blast*
 next
  case(*cComm Q' T' α*)
  from ‹$P \leadsto\hat{} <Rel> Q$› ‹$Q \longmapsto\alpha \prec Q'$› **obtain** P' **where** $P \Longrightarrow\hat{} \alpha \prec P'$ **and** $(P', Q') \in Rel$
   by(*rule weakSimE*)
  from ‹$R \leadsto\hat{} <Rel'> T$› ‹$T \longmapsto(coAction\ \alpha) \prec T'$› **obtain** R' **where** $R \Longrightarrow\hat{} (coAction\ \alpha) \prec R'$ **and** $(R', T') \in Rel'$
   by(*rule weakSimE*)
  from ‹$P \Longrightarrow\hat{} \alpha \prec P'$› ‹$R \Longrightarrow\hat{} (coAction\ \alpha) \prec R'$› ‹$\alpha \neq \tau$› **have** $P \parallel R \Longrightarrow\tau \prec P' \parallel R'$
   by(*auto intro*: *weakCongSync simp add*: *weakTrans-def*)
  hence $P \parallel R \Longrightarrow\hat{} \tau \prec P' \parallel R'$ **by**(*simp add*: *weakTrans-def*)
  moreover from ‹$(P', Q') \in Rel$› ‹$(R', T') \in Rel'$› **have** $(P' \parallel R', Q' \parallel T') \in Rel''$ **by**(*rule C1*)
  ultimately show *?case* **by** *blast*
 qed
qed

lemma *parPres*:
 fixes P :: *ccs*
 and Q :: *ccs*
 and R :: *ccs*
 and Rel :: $(ccs \times ccs)$ *set*
 and Rel' :: $(ccs \times ccs)$ *set*
 assumes $P \leadsto\hat{} <Rel> Q$
 and $(P, Q) \in Rel$
 and *C1*: $\bigwedge S\ T\ U.\ (S, T) \in Rel \Longrightarrow (S \parallel U, T \parallel U) \in Rel'$

 shows $P \parallel R \leadsto\hat{} <Rel'> Q \parallel R$
using *assms*
by(*rule-tac parPresAux*[**where** *Rel'=Id* **and** *Rel''=Rel'*]) (*auto intro*: *reflexive*)

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

assumes $P \leadsto\hat{}<Rel> Q$
and $\bigwedge R\ S\ y.\ (R, S) \in Rel \Longrightarrow ((\!|\nu y|\!)R, (\!|\nu y|\!)S) \in Rel'$

shows $(\!|\nu x|\!)P \leadsto\hat{}<Rel'> (\!|\nu x|\!)Q$
using *assms*
by(*fastforce simp add*: *weakSimulation-def elim*: *resCases intro*: *weakRes*)

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

assumes $(P, Q) \in Rel$
and *C1*: $\bigwedge R\ S.\ (R, S) \in Rel \Longrightarrow R \leadsto\hat{}<Rel> S$
and *Par*: $\bigwedge R\ S\ T\ U.\ [\![(R, S) \in Rel; (T, U) \in Rel']\!] \Longrightarrow (R \parallel T, S \parallel U) \in Rel'$
and *C2*: $bangRel\ Rel \subseteq Rel'$
and *C3*: $\bigwedge R\ S.\ (R \parallel !R, S) \in Rel' \Longrightarrow (!R, S) \in Rel'$

shows $!P \leadsto\hat{}<Rel'> !Q$
proof(*induct rule*: *weakSimI*)
case(*Sim* α Q')
{
fix $Pa\ \alpha\ Q'$
assume $!Q \longmapsto\alpha \prec Q'$ **and** $(Pa, !Q) \in bangRel\ Rel$
hence $\exists P'.\ Pa \Longrightarrow\hat{}\alpha \prec P' \wedge (P', Q') \in Rel'$
proof(*nominal-induct arbitrary*: *Pa rule*: *bangInduct*)
case(*cPar1* α Q')
from ‹$(Pa, Q \parallel !Q) \in bangRel\ Rel$›
show *?case*
proof(*induct rule*: *BRParCases*)
case(*BRPar P R*)
from ‹$(P, Q) \in Rel$› **have** $P \leadsto\hat{}<Rel> Q$ **by**(*rule C1*)
with ‹$Q \longmapsto\alpha \prec Q'$› **obtain** P' **where** $P \Longrightarrow\hat{}\alpha \prec P'$ **and** $(P', Q') \in Rel$
by(*blast dest*: *weakSimE*)
from ‹$P \Longrightarrow\hat{}\alpha \prec P'$› **have** $P \parallel R \Longrightarrow\hat{}\alpha \prec P' \parallel R$ **by**(*rule weakPar1*)
moreover from ‹$(P', Q') \in Rel$› ‹$(R, !Q) \in bangRel\ Rel$› *C2* **have** $(P' \parallel R, Q' \parallel !Q) \in Rel'$
by(*blast intro*: *Par*)
ultimately show *?case* **by** *blast*
qed
next
case(*cPar2* α Q')
from ‹$(Pa, Q \parallel !Q) \in bangRel\ Rel$›
show *?case*
proof(*induct rule*: *BRParCases*)
case(*BRPar P R*)
from ‹$(R, !Q) \in bangRel\ Rel$› **obtain** R' **where** $R \Longrightarrow\hat{}\alpha \prec R'$ **and** $(R',$

$Q')\in \mathit{Rel}'$ **using** *cPar2*
by *blast*
from $\langle R \Longrightarrow^{\wedge} \alpha \prec R'\rangle$ **have** $P \parallel R \Longrightarrow^{\wedge} \alpha \prec P \parallel R'$ **by**(*rule weakPar2*)
moreover from $\langle (P,\ Q) \in \mathit{Rel}\rangle$ $\langle (R',\ Q') \in \mathit{Rel}'\rangle$ **have** $(P \parallel R',\ Q \parallel Q')$ $\in \mathit{Rel}'$ **by**(*rule Par*)
ultimately show *?case* **by** *blast*
qed
next
case(*cComm a* Q' Q'' *Pa*)
from $\langle (\mathit{Pa},\ Q \parallel !Q) \in \mathit{bangRel\ Rel}\rangle$
show *?case*
proof(*induct rule*: *BRParCases*)
case(*BRPar P R*)
from $\langle (P,\ Q) \in \mathit{Rel}\rangle$ **have** $P \leadsto^{\wedge} \mathrm{<}\mathit{Rel}\mathrm{>}\ Q$ **by**(*rule C1*)
with $\langle Q \longmapsto a \prec Q'\rangle$ **obtain** P' **where** $P \Longrightarrow^{\wedge} a \prec P'$ **and** $(P',\ Q') \in \mathit{Rel}$
by(*blast dest*: *weakSimE*)
from $\langle (R,\ !Q) \in \mathit{bangRel\ Rel}\rangle$ **obtain** R' **where** $R \Longrightarrow^{\wedge} (\mathit{coAction}\ a) \prec R'$ **and** $(R',\ Q'') \in \mathit{Rel}'$ **using** *cComm*
by *blast*
from $\langle P \Longrightarrow^{\wedge} a \prec P'\rangle$ $\langle R \Longrightarrow^{\wedge} (\mathit{coAction}\ a) \prec R'\rangle$ $\langle a \neq \tau\rangle$ **have** $P \parallel R$ $\Longrightarrow^{\wedge} \tau \prec P' \parallel R'$
by(*auto intro*: *weakCongSync simp add*: *weakTrans-def*)
moreover from $\langle (P',\ Q') \in \mathit{Rel}\rangle$ $\langle (R',\ Q'') \in \mathit{Rel}'\rangle$ **have** $(P' \parallel R',\ Q' \parallel$ $Q'') \in \mathit{Rel}'$ **by**(*rule Par*)
ultimately show *?case* **by** *blast*
qed
next
case(*cBang* α Q' *Pa*)
from $\langle (\mathit{Pa},\ !Q) \in \mathit{bangRel\ Rel}\rangle$
show *?case*
proof(*induct rule*: *BRBangCases*)
case(*BRBang P*)
from $\langle (P,\ Q) \in \mathit{Rel}\rangle$ **have** $(!P,\ !Q) \in \mathit{bangRel\ Rel}$ **by**(*rule bangRel.BRBang*)
with $\langle (P,\ Q) \in \mathit{Rel}\rangle$ **have** $(P \parallel !P,\ Q \parallel !Q) \in \mathit{bangRel\ Rel}$ **by**(*rule bangRel.BRPar*)
then obtain P' **where** $P \parallel !P \Longrightarrow^{\wedge} \alpha \prec P'$ **and** $(P',\ Q') \in \mathit{Rel}'$ **using** *cBang*
by *blast*
from $\langle P \parallel !P \Longrightarrow^{\wedge} \alpha \prec P'\rangle$
show *?case*
proof(*induct rule*: *weakTransCases*)
case *Base*
have $!P \Longrightarrow^{\wedge} \tau \prec !P$ **by** *simp*
moreover from $\langle (P',\ Q') \in \mathit{Rel}'\rangle$ $\langle P \parallel !P = P'\rangle$ **have** $(!P,\ Q') \in \mathit{Rel}'$ **by**(*blast intro*: *C3*)
ultimately show *?case* **by** *blast*
next
case *Step*
from $\langle P \parallel !P \Longrightarrow \alpha \prec P'\rangle$ **have** $!P \Longrightarrow \alpha \prec P'$ **by**(*rule weakCongRepl*)

```
        hence !P ⟹^ α ≺ P' by(simp add: weakTrans-def)
        with ‹(P', Q') ∈ Rel'› show ?case by blast
      qed
    qed
  qed
}

moreover from ‹(P, Q) ∈ Rel› have (!P, !Q) ∈ bangRel Rel by(rule BRBang)

ultimately show ?case using ‹!Q ⟼ α ≺ Q'› by blast
qed

end

theory Weak-Bisim-Pres
  imports Weak-Bisim Weak-Sim-Pres Strong-Bisim-SC
begin

lemma actPres:
  fixes P :: ccs
  and   Q :: ccs
  and   α :: act

  assumes P ≈ Q

  shows α.(P) ≈ α.(Q)
proof −
  let ?X = {(α.(P), α.(Q)) | P Q. P ≈ Q}
  from assms have (α.(P), α.(Q)) ∈ ?X by auto
  thus ?thesis
    by(coinduct rule: weakBisimulationCoinduct) (auto dest: weakBisimulationE
intro: actPres)
qed

lemma parPres:
  fixes P :: ccs
  and   Q :: ccs
  and   R :: ccs

  assumes P ≈ Q

  shows P ∥ R ≈ Q ∥ R
proof −
  let ?X = {(P ∥ R, Q ∥ R) | P Q R. P ≈ Q}
  from assms have (P ∥ R, Q ∥ R) ∈ ?X by blast
  thus ?thesis
   by(coinduct rule: weakBisimulationCoinduct, auto)
     (blast intro: parPres dest: weakBisimulationE)+
qed
```

lemma *resPres*:
  fixes P :: *ccs*
  and Q :: *ccs*
  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 *assms* **have** $((\!|\nu x|\!)P, (\!|\nu x|\!)Q) \in ?X$ **by** *auto*
  thus *?thesis*
    by(*coinduct rule*: *weakBisimulationCoinduct*) (*auto intro*: *resPres dest*: *weak-BisimulationE*)
qed

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

  assumes $P \approx Q$

  shows $!P \approx !Q$
proof −
  let *?X* = *bangRel weakBisimulation*
  let *?Y* = *weakBisimulation O ?X O bisim*
  {
    fix $R\ T\ P\ Q$
    assume $R \approx T$ **and** $(P, Q) \in ?Y$
    from ‹$(P, Q) \in ?Y$› **obtain** P' Q' **where** $P \approx P'$ **and** $(P', Q') \in ?X$ **and** $Q' \sim Q$
      by *auto*
    from ‹$P \approx P'$› **have** $R \parallel P \approx R \parallel P'$
      by(*metis parPres bisimWeakBisimulation transitive parComm*)
    moreover from ‹$R \approx T$› ‹$(P', Q') \in ?X$› **have** $(R \parallel P', T \parallel Q') \in ?X$ **by**(*auto dest*: *BRPar*)
    moreover from ‹$Q' \sim Q$› **have** $T \parallel Q' \sim T \parallel Q$ **by**(*metis Strong-Bisim-Pres.parPres Strong-Bisim.transitive parComm*)
    ultimately have $(R \parallel P, T \parallel Q) \in ?Y$ **by** *auto*
  } **note** *BRParAux* = *this*

  from *assms* **have** $(!P, !Q) \in ?X$ **by**(*auto intro*: *BRBang*)
  thus *?thesis*
  proof(*coinduct rule*: *weakBisimWeakUpto*)
    case(*cSim P Q*)
    from ‹$(P, Q) \in$ *bangRel weakBisimulation*› **show** *?case*
    proof(*induct*)
      case(*BRBang P Q*)

    note ‹*P* ≈ *Q*› *weakBisimulationE*(*1*) *BRParAux*
    moreover have *?X* ⊆ *?Y* **by**(*auto intro*: *Strong-Bisim.reflexive reflexive*)
    moreover {
      fix *P Q*
      assume (*P* ∥ !*P*, *Q*) ∈ *?Y*
      hence (!*P*, *Q*) ∈ *?Y* **using** *bangUnfold*
        by(*blast dest*: *Strong-Bisim.transitive transitive bisimWeakBisimulation*)
    }
    ultimately show *?case* **by**(*rule bangPres*)
  next
    case(*BRPar R T P Q*)
  from ‹*R* ≈ *T*› **have** *R* ⇝^<*weakBisimulation*> *T* **by**(*rule weakBisimulationE*)
    moreover note ‹*R* ≈ *T*› ‹*P* ⇝^<*?Y*> *Q*›
  moreover from ‹(*P*, *Q*) ∈ *?X*› **have** (*P*, *Q*) ∈ *?Y* **by**(*blast intro*: *Strong-Bisim.reflexive reflexive*)
  ultimately show *?case* **using** *BRParAux* **by**(*rule Weak-Sim-Pres.parPresAux*)
  qed
next
  case(*cSym P Q*)
  thus *?case*
    by *induct* (*auto dest*: *weakBisimulationE intro*: *BRPar BRBang*)
qed
qed

end

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

lemma *actPres*:
  fixes *P* :: *ccs*
  and *Q* :: *ccs*
  and *Rel* :: (*ccs* × *ccs*) *set*
  and *a* :: *name*
  and *Rel'* :: (*ccs* × *ccs*) *set*

  assumes (*P*, *Q*) ∈ *Rel*

  shows α.(*P*) ⇝<*Rel*> α.(*Q*)
using *assms*
by(*fastforce simp add*: *weakCongSimulation-def elim*: *actCases intro*: *weakCongAction*)

lemma *sumPres*:
  fixes *P* :: *ccs*
  and *Q* :: *ccs*
  and *Rel* :: (*ccs* × *ccs*) *set*

assumes $P \rightsquigarrow$<*Rel*> Q
and $\quad$ $Rel \subseteq Rel'$
and $\quad$ $Id \subseteq Rel'$

shows $P \oplus R \rightsquigarrow$<*Rel'*> $Q \oplus R$
using *assms*
by(*force simp add*: *weakCongSimulation-def elim*: *sumCases intro*: *weakCongSum1 weakCongSum2 transitionWeakCongTransition*)

lemma *parPres*:
fixes P :: *ccs*
and Q :: *ccs*
and *Rel* :: (*ccs* × *ccs*) *set*

assumes $P \rightsquigarrow$<*Rel*> Q
and $\quad$ $(P, Q) \in Rel$
and $\quad$ *C1*: $\bigwedge S\ T\ U.\ (S, T) \in Rel \Longrightarrow (S \parallel U, T \parallel U) \in Rel'$

shows $P \parallel R \rightsquigarrow$<*Rel'*> $Q \parallel R$
proof(*induct rule*: *weakSimI*)
case(*Sim* α *QR*)
from ‹$Q \parallel R \longmapsto \alpha \prec QR$›
show *?case*
proof(*induct rule*: *parCases*)
case(*cPar1* Q')
from ‹$P \rightsquigarrow$<*Rel*> Q› ‹$Q \longmapsto \alpha \prec Q'$› **obtain** P' **where** $P \Longrightarrow \alpha \prec P'$ **and** $(P', Q') \in Rel$
by(*rule weakSimE*)
from ‹$P \Longrightarrow \alpha \prec P'$› **have** $P \parallel R \Longrightarrow \alpha \prec P' \parallel R$ **by**(*rule weakCongPar1*)
moreover from ‹$(P', Q') \in Rel$› **have** $(P' \parallel R, Q' \parallel R) \in Rel'$ **by**(*rule C1*)
ultimately show *?case* **by** *blast*
next
case(*cPar2* R')
from ‹$R \longmapsto \alpha \prec R'$› **have** $R \Longrightarrow \alpha \prec R'$ **by**(*rule transitionWeakCongTransition*)
hence $P \parallel R \Longrightarrow \alpha \prec P \parallel R'$ **by**(*rule weakCongPar2*)
moreover from ‹$(P, Q) \in Rel$› **have** $(P \parallel R', Q \parallel R') \in Rel'$ **by**(*rule C1*)
ultimately show *?case* **by** *blast*
next
case(*cComm* Q' R' α)
from ‹$P \rightsquigarrow$<*Rel*> Q› ‹$Q \longmapsto \alpha \prec Q'$› **obtain** P' **where** $P \Longrightarrow \alpha \prec P'$ **and** $(P', Q') \in Rel$
by(*rule weakSimE*)
from ‹$R \longmapsto (coAction\ \alpha) \prec R'$› **have** $R \Longrightarrow (coAction\ \alpha) \prec R'$
by(*rule transitionWeakCongTransition*)
with ‹$P \Longrightarrow \alpha \prec P'$› **have** $P \parallel R \Longrightarrow \tau \prec P' \parallel R'$ **using** ‹$\alpha \neq \tau$›
by(*rule weakCongSync*)
moreover from ‹$(P', Q') \in Rel$› **have** $(P' \parallel R', Q' \parallel R') \in Rel'$ **by**(*rule C1*)
ultimately show *?case* **by** *blast*
qed

qed

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

assumes $P \leadsto<Rel> Q$
and $\bigwedge R\ S\ y.\ (R, S) \in Rel \Longrightarrow (⦇\nu y⦈R, ⦇\nu y⦈S) \in Rel'$

shows $⦇\nu x⦈P \leadsto<Rel'> ⦇\nu x⦈Q$
using *assms*
by(*fastforce simp add*: *weakCongSimulation-def elim*: *resCases intro*: *weakCongRes*)

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

assumes $(P, Q) \in Rel$
and *C1*: $\bigwedge R\ S.\ (R, S) \in Rel \Longrightarrow R \leadsto<Rel'> S$
and *C2*: $Rel \subseteq Rel'$

shows $!P \leadsto<bangRel\ Rel'>\ !Q$
proof(*induct rule*: *weakSimI*)
case($Sim\ \alpha\ Q'$)
{
fix $Pa\ \alpha\ Q'$
assume $!Q \longmapsto\alpha \prec Q'$ **and** $(Pa, !Q) \in bangRel\ Rel$
hence $\exists P'.\ Pa \Longrightarrow\alpha \prec P' \wedge (P', Q') \in bangRel\ Rel'$
proof(*nominal-induct arbitrary*: *Pa rule*: *bangInduct*)
case($cPar1\ \alpha\ Q'$)
from $\langle(Pa, Q \parallel !Q) \in bangRel\ Rel\rangle$
show *?case*
proof(*induct rule*: *BRParCases*)
case($BRPar\ P\ R$)
from $\langle(P, Q) \in Rel\rangle$ **have** $P \leadsto<Rel'> Q$ **by**(*rule C1*)
with $\langle Q \longmapsto\alpha \prec Q'\rangle$ **obtain** P' **where** $P \Longrightarrow\alpha \prec P'$ **and** $(P', Q') \in Rel'$
by(*blast dest*: *weakSimE*)
from $\langle P \Longrightarrow\alpha \prec P'\rangle$ **have** $P \parallel R \Longrightarrow\alpha \prec P' \parallel R$ **by**(*rule weakCongPar1*)
moreover from $\langle(R, !Q) \in bangRel\ Rel\rangle$ *C2* **have** $(R, !Q) \in bangRel\ Rel'$
by *induct* (*auto intro*: *bangRel.BRPar bangRel.BRBang*)
with $\langle(P', Q') \in Rel'\rangle$ **have** $(P' \parallel R, Q' \parallel !Q) \in bangRel\ Rel'$
by(*rule bangRel.BRPar*)
ultimately show *?case* **by** *blast*
qed

next
case(*cPar2* α Q')
from ‹(Pa, $Q \parallel !Q$) $\in$ *bangRel Rel*›
show *?case*
proof(*induct rule*: *BRParCases*)
case(*BRPar P R*)
from ‹(R, $!Q$) $\in$ *bangRel Rel*› **obtain** R' **where** $R \Longrightarrow\alpha \prec R'$ **and** (R', Q') $\in$ *bangRel Rel'* **using** *cPar2*
by *blast*
from ‹$R \Longrightarrow\alpha \prec R'$› **have** $P \parallel R \Longrightarrow\alpha \prec P \parallel R'$ **by**(*rule weakCongPar2*)
moreover from ‹(P, Q) $\in$ *Rel*› ‹(R', Q') $\in$ *bangRel Rel'*› *C2* **have** ($P \parallel R'$, $Q \parallel Q'$) $\in$ *bangRel Rel'*
by(*blast intro*: *bangRel.BRPar*)
ultimately show *?case* **by** *blast*
qed
next
case(*cComm a* Q' Q'' *Pa*)
from ‹(Pa, $Q \parallel !Q$) $\in$ *bangRel Rel*›
show *?case*
proof(*induct rule*: *BRParCases*)
case(*BRPar P R*)
from ‹(P, Q) $\in$ *Rel*› **have** $P \rightsquigarrow$<*Rel'*> Q **by**(*rule C1*)
with ‹$Q \longmapsto a \prec Q'$› **obtain** P' **where** $P \Longrightarrow a \prec P'$ **and** (P', Q') $\in$ *Rel'*
by(*blast dest*: *weakSimE*)
from ‹(R, $!Q$) $\in$ *bangRel Rel*› **obtain** R' **where** $R \Longrightarrow$(*coAction a*) $\prec R'$ **and** (R', Q'') $\in$ *bangRel Rel'* **using** *cComm*
by *blast*
from ‹$P \Longrightarrow a \prec P'$› ‹$R \Longrightarrow$(*coAction a*) $\prec R'$› ‹$a \neq \tau$› **have** $P \parallel R \Longrightarrow\tau \prec P' \parallel R'$ **by**(*rule weakCongSync*)
moreover from ‹(P', Q') $\in$ *Rel'*› ‹(R', Q'') $\in$ *bangRel Rel'*› **have** ($P' \parallel R'$, $Q' \parallel Q''$) $\in$ *bangRel Rel'*
by(*rule bangRel.BRPar*)
ultimately show *?case* **by** *blast*
qed
next
case(*cBang* α Q' *Pa*)
from ‹(Pa, $!Q$) $\in$ *bangRel Rel*›
show *?case*
proof(*induct rule*: *BRBangCases*)
case(*BRBang P*)
from ‹(P, Q) $\in$ *Rel*› **have** ($!P$, $!Q$) $\in$ *bangRel Rel* **by**(*rule bangRel.BRBang*)
with ‹(P, Q) $\in$ *Rel*› **have** ($P \parallel !P$, $Q \parallel !Q$) $\in$ *bangRel Rel* **by**(*rule bangRel.BRPar*)
then obtain P' **where** $P \parallel !P \Longrightarrow\alpha \prec P'$ **and** (P', Q') $\in$ *bangRel Rel'* **using** *cBang*
by *blast*
from ‹$P \parallel !P \Longrightarrow\alpha \prec P'$› **have** $!P \Longrightarrow\alpha \prec P'$ **by**(*rule weakCongRepl*)
thus *?case* **using** ‹(P', Q') $\in$ *bangRel Rel'*› **by** *blast*
qed

```
    qed
  }

  moreover from ‹(P, Q) ∈ Rel› have (!P, !Q) ∈ bangRel Rel by(rule BRBang)

  ultimately show ?case using ‹!Q ⟼ α ≺ Q'› by blast
qed

end

theory Weak-Cong-Pres
  imports Weak-Cong Weak-Bisim-Pres Weak-Cong-Sim-Pres
begin

lemma actPres:
  fixes P :: ccs
  and   Q :: ccs
  and   α :: act

  assumes P ≅ Q

  shows α.(P) ≅ α.(Q)
using assms
proof(induct rule: weakCongISym2)
  case(cSim P Q)
  from ‹P ≅ Q› have P ≈ Q by(rule weakCongruenceWeakBisimulation)
  thus ?case by(rule actPres)
qed

lemma sumPres:
  fixes P :: ccs
  and   Q :: ccs
  and   R :: ccs

  assumes P ≅ Q

  shows P ⊕ R ≅ Q ⊕ R
using assms
proof(induct rule: weakCongISym2)
  case(cSim P Q)
  from ‹P ≅ Q› have P ⇝<weakBisimulation> Q by(rule weakCongruenceE)
  thus ?case using Weak-Bisim.reflexive
    by(rule-tac sumPres) auto
qed

lemma parPres:
  fixes P :: ccs
  and   Q :: ccs
  and   R :: ccs
```

assumes $P \cong Q$

shows $P \parallel R \cong Q \parallel R$
using *assms*
proof(*induct rule*: *weakCongISym2*)
case(*cSim P Q*)
from ‹$P \cong Q$› **have** P ⇝<*weakBisimulation*> Q **by**(*rule weakCongruenceE*)
moreover from ‹$P \cong Q$› **have** $P \approx Q$ **by**(*rule weakCongruenceWeakBisimulation*)
ultimately show *?case* **using** *Weak-Bisim-Pres.parPres*
by(*rule parPres*)
qed

lemma *resPres*:
fixes P :: *ccs*
and Q :: *ccs*
and x :: *name*

assumes $P \cong Q$

shows $(\!|\nu x|\!)P \cong (\!|\nu x|\!)Q$
using *assms*
proof(*induct rule*: *weakCongISym2*)
case(*cSim P Q*)
from ‹$P \cong Q$› **have** P ⇝<*weakBisimulation*> Q **by**(*rule weakCongruenceE*)
thus *?case* **using** *Weak-Bisim-Pres.resPres*
by(*rule resPres*)
qed

lemma *weakBisimBangRel*: *bangRel weakBisimulation* $\subseteq$ *weakBisimulation*
proof *auto*
fix P Q
assume $(P, Q) \in$ *bangRel weakBisimulation*
thus $P \approx Q$
proof(*induct rule*: *bangRel.induct*)
case(*BRBang P Q*)
from ‹$P \approx Q$› **show** $!P \approx !Q$ **by**(*rule Weak-Bisim-Pres.bangPres*)
next
case(*BRPar R T P Q*)
from ‹$R \approx T$› **have** $R \parallel P \approx T \parallel P$ **by**(*rule Weak-Bisim-Pres.parPres*)
moreover from ‹$P \approx Q$› **have** $P \parallel T \approx Q \parallel T$ **by**(*rule Weak-Bisim-Pres.parPres*)
hence $T \parallel P \approx T \parallel Q$ **by**(*metis bisimWeakBisimulation Weak-Bisim.transitive parComm*)
ultimately show $R \parallel P \approx T \parallel Q$ **by**(*rule Weak-Bisim.transitive*)
qed
qed

lemma *bangPres*:

fixes *P* :: *ccs*
and *Q* :: *ccs*

assumes *P* ≅ *Q*

shows !*P* ≅ !*Q*
using *assms*
proof(*induct rule*: *weakCongISym2*)
case(*cSim P Q*)
let *?X* = {(*P*, *Q*) | *P Q*. *P* ≅ *Q*}
from ‹*P* ≅ *Q*› **have** (*P*, *Q*) ∈ *?X* **by** *auto*
moreover have ⋀*P Q*. (*P*, *Q*) ∈ *?X* ⟹ *P* ⇝<*weakBisimulation*> *Q* **by**(*auto dest*: *weakCongruenceE*)
moreover have *?X* ⊆ *weakBisimulation* **by**(*auto intro*: *weakCongruenceWeakBisimulation*)
ultimately have !*P* ⇝<*bangRel weakBisimulation*> !*Q* **by**(*rule bangPres*)
thus *?case* **using** *weakBisimBangRel* **by**(*rule Weak-Cong-Sim.weakMonotonic*)
qed

end

References

[1] J. Bengtson. *Formalising process calculi*, volume 94. Uppsala Dissertations from the Faculty of Science and Technology, 2010.