The Calculus of Communicating Systems

Jesper Bengtson

May 27, 2015

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 1
2 Formalisation 2

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 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[cqvt]:
  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: cqvts)

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 $\alpha$ rule: act.strong-induct) auto

lemma coActFresh[simp]:
fixes $x :: \text{name}$
and $a :: \text{act}$

shows $x \notin \text{coAction } a = x \notin a$
by (nominal-induct $a$ rule: act.strong-induct) (auto)

lemma alphaRes:
fixes $y :: \text{name}$
and $P :: \text{ccs}$
and $x :: \text{name}$

assumes $y \notin P$

shows $(\nu x)P = (\nu y)(([x,y]) \cdot P)$
using assms

inductive semantics :: ccs $\Rightarrow$ act $\Rightarrow$ ccs $\Rightarrow$ bool (- $\rightarrow$ $\prec$ $\cdot$ [80, 80, 80] 80)
where
Action: $\alpha.(P) \rightarrow \alpha \prec P$
| Sum1: $P \rightarrow \alpha \prec P' \Rightarrow P \parallel Q \rightarrow \alpha \prec P'$
| Sum2: $Q \rightarrow \alpha \prec Q' \Rightarrow P \parallel Q \rightarrow \alpha \prec Q'$
| Par1: $P \rightarrow \alpha \prec P' \Rightarrow P \parallel Q \rightarrow \alpha \prec P' \parallel Q$
| Par2: $Q \rightarrow \alpha \prec Q' \Rightarrow P \parallel Q \rightarrow \alpha \prec P \parallel Q'$
| Comm: $[P \rightarrow \alpha \prec P'; Q \rightarrow \text{coAction } a \prec Q'; a \neq \tau] \Rightarrow P \parallel Q \rightarrow \tau \prec P' \parallel Q'$
| Res: $[P \rightarrow \alpha \prec P'; x \notin \alpha] \Rightarrow (\nu x)P \rightarrow \alpha \prec (\nu x)P'$
| Bang: $P \parallel !P \rightarrow \alpha \prec P' \Rightarrow !P \rightarrow \alpha \prec P'$

equivariance semantics

by (auto simp add: semantics)

lemma semanticsInduct:
$[R \rightarrow \beta \prec R'; \bigwedge Q \prec C. \text{Prop } C (\alpha.(P)) \alpha P; \bigwedge P \alpha P' Q C. [P \rightarrow \alpha \prec P'; \bigwedge Q \prec C. \text{Prop } C P \alpha P'] \Rightarrow \text{Prop } C (ccs.Sum P Q) \alpha P'; \bigwedge Q \alpha Q' P C. [Q \rightarrow \alpha \prec Q'; \bigwedge Q \prec C. \text{Prop } C Q \alpha Q'] \Rightarrow \text{Prop } C (ccs.Sum P Q) \alpha Q'; \bigwedge P \alpha P' Q C. [P \rightarrow \alpha \prec P'; \bigwedge Q \prec C. \text{Prop } C P \alpha P'] \Rightarrow \text{Prop } C (P \parallel Q \alpha (P' \parallel Q))$
\( \forall Q \alpha Q' P \mathcal{C}. [Q \rightsquigarrow \alpha < Q'; \forall C \cdot Prop \mathcal{C} Q \alpha Q'] \Rightarrow Prop \mathcal{C} (P \parallel Q) \alpha (P \parallel Q') \)

\( \forall P a P' Q Q' \mathcal{C}. [P \rightsquigarrow a < P'; \forall C \cdot Prop \mathcal{C} P a P'; Q \rightsquigarrow (\text{coAction} a) < Q'; \forall C \cdot Prop \mathcal{C} Q (\text{coAction} a) Q'; a \neq \tau] \Rightarrow Prop \mathcal{C} (P \parallel Q) (\tau) (P' \parallel Q') \)

\( \forall P a P' x \mathcal{C}. [x \notin C; P \rightsquigarrow \alpha < P'; \forall C \cdot Prop \mathcal{C} P a P'; x \notin \alpha] \Rightarrow Prop \mathcal{C} ((\nu x)P) \alpha ((\nu x)P') \)

\( \forall P \alpha P' C. [P \parallel !P \rightsquigarrow \alpha < P'; \forall C \cdot Prop \mathcal{C} (P \parallel !P) \alpha P'] \Rightarrow Prop \mathcal{C} !P \alpha P' \)

\( \Rightarrow Prop (C::\text{a::fs-name}) R \beta R' \)

by(erule-tac \( z = C \) in semantics.strong-induct) auto

lemma NilTrans \( \{ \text{dest} \} \):
shows \( 0 \rightsquigarrow \alpha < P' \Rightarrow False \)
and \( (([] b])P \rightsquigarrow (\langle c \rangle) < P' \Rightarrow False \)
and \( (([] b])P \rightsquigarrow (\tau) < P' \Rightarrow False \)
and \( (([] b])P \rightsquigarrow (\langle c \rangle) < P' \Rightarrow False \)
and \( (([] b])P \rightsquigarrow (\tau) < P' \Rightarrow False \)
apply(ind-cases \( 0 \rightsquigarrow \alpha < P' \))
apply(ind-cases \( ([] b])P \rightsquigarrow (\langle c \rangle) < P', \text{auto simp add: ccs.inject} \))
apply(ind-cases \( ([] b])P \rightsquigarrow (\tau) < P', \text{auto simp add: ccs.inject} \))
apply(ind-cases \( (\langle b \rangle)P \rightsquigarrow (\langle c \rangle) < P', \text{auto simp add: ccs.inject} \))
apply(ind-cases \( (\langle b \rangle)P \rightsquigarrow (\tau) < P', \text{auto simp add: ccs.inject} \))
done

lemma freshDerivative:
fixes \( P :: \text{ccs} \)
and \( a :: \text{act} \)
and \( P' :: \text{ccs} \)
and \( x :: \text{name} \)

assumes \( P \rightsquigarrow \alpha < P' \)
and \( x \notin P \)

shows \( x \notin \alpha \) and \( x \notin P' \)
using assms
by(nominal-induct rule: semantics.strong-induct)
\( \text{(auto simp add: ccs.fresh abs-fresh)} \)

lemma actCases \( \{ \text{consumes 1, case-names cAct} \} \):
fixes \( \alpha :: \text{act} \)
and \( P :: \text{ccs} \)
and \( \beta :: \text{act} \)
and \( P' :: \text{ccs} \)

assumes \( \alpha.(P) \rightsquigarrow \beta < P' \)
and Prop α P
shows Prop β P'
using assms
by (ind_cases α.(P ↦→ β ↼ 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 ⊕ Q ↦→ α ↼ R
and \( \bigwedge P'. P ↦→ α ↼ P' \implies Prop P' \)
and \( \bigwedge Q'. Q ↦→ α ↼ Q' \implies Prop Q' \)

shows Prop R
using assms
by (ind_cases P ⊕ Q ↦→ α ↼ 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 || Q ↦→ α ↼ R
and \( \bigwedge P'. P ↦→ α ↼ P' \implies Prop α (P' || Q) \)
and \( \bigwedge Q'. Q ↦→ α ↼ Q' \implies Prop α (P || Q') \)
and \( \bigwedge P' Q'. a, [P ↦→ a ↼ P'; Q ↦→ (coAction a) ↼ Q'; a \neq τ; α = τ] \implies Prop (τ (P' || Q')) \)

shows Prop α R
using assms
by (ind_cases P || Q ↦→ α ↼ 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 \( (\nu x) P ↦→ α ↼ P' \)
and \( \bigwedge P'. [P ↦→ α ↼ P'; x \notin α] \implies Prop ((\nu x)P') \)

shows Prop P'
proof -
from \( (\nu x) P ↦→ α ↼ P' \) have x \notin α and x \notin 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 \(\Rightarrow\) ccs \(\Rightarrow\) bool
where
aux1: bangPred \(P\) \((!P)\)
| aux2: bangPred \(P\) \((P \parallel !_P)\)

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

  assumes \(!P \hookrightarrow_{\alpha} P'\)
  and \(rPar1: \\{\alpha P' C. [\[P \hookrightarrow_{\alpha} P'] \Rightarrow Prop C (P \parallel !_P) \alpha (P' \parallel !_P)\}\)
  and \(rPar2: \\{\alpha P' C. [\[P \hookrightarrow_{\alpha} P'; \forall\alpha C. Prop C ( !_P \alpha P' ) \Rightarrow Prop C (P \parallel !_P) \alpha (P' \parallel !_P)\}\}
  and \(rComm: \\{\alpha P' P'' C. [\[P \hookrightarrow_{\alpha} P'; !_P \hookrightarrow_{\alpha} P''; \alpha = \alpha'\] \Rightarrow Prop C (P \parallel !_P) (\alpha P') P'\parallel P'\}\)
  and \(rBang: \\{\alpha P' C. [\[P \parallel !_P \hookrightarrow_{\alpha} P'; \forall\alpha C. Prop C (P \parallel !_P) \alpha P' \Rightarrow Prop C (P \parallel !_P) \alpha P'\}\}

  shows Prop C ( !_P \alpha P' \]
proof -
{ 
  fix \(X\) \(\alpha\) \(P'\)
  assume \(X \hookrightarrow_{\alpha} P'\) and bangPred \(P\) \(X\)
  hence Prop C \(X\) \(\alpha\) \(P'\)
proof(nominal-induct avoiding: \(C\) rule: semantics.strong-induct)
  case(Action \(\alpha\) \(Pa\)
    thus \(?case
      by \(-\) (ind-cases bangPred \(P\) (\(\alpha.(Pa))\))

  next
  case(Sum1 \(Pa\) \(\alpha\) \(P'\) \(Q\)
    thus \(?case
      by \(-\) (ind-cases bangPred \(P\) (\(Pa \oplus Q))\))

  next
  case(Sum2 Q \(\alpha\) \(Q'\) \(Pa\)
    thus \(?case
      by \(-\) (ind-cases bangPred \(P\) (\(Pa \oplus Q))\))

  next
  case(Par1 \(Pa\) \(\alpha\) \(P'\) \(Q\)
    thus \(?case
      apply \-
      by(ind-cases bangPred \(P\) (\(Pa \parallel Q))\), 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 (|--ν x| 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] → α ≺ 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 \( \text{Rel} :: (\text{ccs} \times \text{ccs}) \text{ set} \)
and \( F :: \text{ccs} \Rightarrow \text{bool} \)

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

shows \( F P \)
using \( \text{assms} \)
by \((- \text{ind-cases} (P, Q \parallel !Q) \in \text{bangRel} \ \text{Rel}, \text{auto simp add: ccs.inject})\)

lemma \( \text{bangRelSubset}: \)
fixes \( \text{Rel} :: (\text{ccs} \times \text{ccs}) \text{ set} \)
and \( \text{Rel}' :: (\text{ccs} \times \text{ccs}) \text{ set} \)

assumes \((P, Q) \in \text{bangRel} \ \text{Rel}\)
and \( \forall P Q. (P, Q) \in \text{Rel} \Rightarrow (P, Q) \in \text{Rel}' \)

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

end

definition \( \text{tauChain} :: \text{ccs} \Rightarrow \text{ccs} \Rightarrow \text{bool} \)
where \( P \Rightarrow_{\tau} P' \equiv (P, P') \in \{ (P, P') \mid P P' \Rightarrow_{\tau} \prec P' \} ^* \)

lemma \( \text{tauChainInduct}[\text{consumes 1}, \text{case-names Base Step}]: \)
assumes \( P \Rightarrow_{\tau} P' \)
and \( \text{Prop P} \)
and \( \forall P' P'', [ P \Rightarrow_{\tau} P'; P' \Rightarrow_{\tau} \prec P''; \text{Prop P'} ] \Rightarrow \text{Prop P''} \)

shows \( \text{Prop P'} \)
using \( \text{assms} \)
by\( (\text{auto simp add: tauChain-def elim: rtrancl-induct})\)

lemma \( \text{tauChainRefl}[\text{simp}]: \)
fixes \( P :: \text{ccs} \)

shows \( P \Rightarrow_{\tau} P \)
by\( (\text{auto simp add: tauChain-def})\)

lemma \( \text{tauChainCons}[\text{dest}]: \)
fixes \( P :: \text{ccs} \)
and \( P' :: \text{ccs} \)

8
and \( P'' :: ccs \)

assumes \( P \rightarrow_{\tau} P' \)
and \( P' \rightarrow_{\tau} \prec P'' \)

shows \( P \rightarrow_{\tau} P'' \)

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 \( P' \rightarrow_{\tau} \prec P'' \)
  and \( P \rightarrow_{\tau} P' \)

  shows \( P \rightarrow_{\tau} P'' \)
  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 \( P \rightarrow_{\tau} P' \)
  and \( P' \rightarrow_{\tau} P'' \)

  shows \( P \rightarrow_{\tau} P'' \)
  using \( P' \rightarrow_{\tau} P'' \) \( (P \rightarrow_{\tau} P) \)
  by(induct rule: tauChainInduct) auto

lemma tauChainSum1:
  fixes \( P :: ccs \)
  and \( P' :: ccs \)
  and \( Q :: ccs \)

  assumes \( P \rightarrow_{\tau} P' \)
  and \( P \neq P' \)

  shows \( P \oplus Q \rightarrow_{\tau} P' \)
  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 $Q \Rightarrow \tau Q'$
  and $Q \neq Q'$
  shows $P \oplus Q \Rightarrow \tau Q'$
  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 $P \Rightarrow \tau P'$
  shows $P \parallel Q \Rightarrow \tau P' \parallel Q$
  using assms
  by (induct rule: tauChainInduct) (auto intro: Par1)

lemma tauChainPar2:
  fixes $Q :: ccs$
  and $Q' :: ccs$
  and $P :: ccs$
  assumes $Q \Rightarrow \tau Q'$
  shows $P \parallel Q \Rightarrow \tau P \parallel Q'$
  using assms
  by (induct rule: tauChainInduct) (auto intro: Par2)

lemma tauChainRes:
  fixes $P :: ccs$
  and $P' :: ccs$
  and $x :: name$
assumes $P \xrightarrow{\tau} P'$

shows $(\nu x)P \xrightarrow{\tau} (\nu x)P'$
using assms
by (induct rule: tauChainInduct) (auto dest: Res)

lemma tauChainRepl:
fixes $P :: ccs$

assumes $P \parallel !P \xrightarrow{\tau} P'$
and $P' \neq P \parallel !P$

shows $!P \xrightarrow{\tau} P'$
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 :: $ccs \Rightarrow act \Rightarrow ccs \Rightarrow bool$
  (- $\Rightarrow$ $\llhd$ $\Rightarrow$ [80, 80, 80]
11

lemma weakCongTransI:
fixes $P :: ccs$
and $\alpha :: act$
and $P' :: ccs$

assumes $P \Rightarrow \alpha \prec P'$

obtains $P'' P'''$ where $P \Rightarrow_{\tau} P'' \text{ and } P'' \Rightarrow_{\tau} \alpha \prec P''' \text{ and } P''' \Rightarrow_{\tau} P'$
using assms
by (auto simp add: weakCongTrans-def)

lemma weakCongTransE:
fixes $P :: ccs$
and $\alpha :: act$
and $P' :: ccs$

assumes $P \Rightarrow_{\alpha} \prec P'$

by auto

end
assumes \( P \rightarrow_{\tau} P'' \)
and \( P'' \rightarrow_{\alpha} \prec P''' \)
and \( P''' \rightarrow_{\tau} P' \)

shows \( P \rightarrow_{\alpha} \prec P' \)
using assms
by\((\text{auto simp add: weakCongTrans-def})\)

lemma transitionWeakCongTransition:
fixes \( P :: \text{ccs} \)
and \( \alpha :: \text{act} \)
and \( P' :: \text{ccs} \)

assumes \( P \rightarrow_{\alpha} \prec P' \)

shows \( P \rightarrow_{\alpha} \prec P' \)
using assms
by\((\text{force simp add: weakCongTrans-def})\)

lemma weakCongAction:
fixes \( a :: \text{name} \)
and \( P :: \text{ccs} \)

shows \( \alpha.(P) \rightarrow_{\alpha} \prec P \)
by\((\text{auto simp add: weakCongTrans-def}) \)
\((\text{blast intro: Action tauChainRef})\)

lemma weakCongSum1:
fixes \( P :: \text{ccs} \)
and \( \alpha :: \text{act} \)
and \( P' :: \text{ccs} \)
and \( Q :: \text{ccs} \)

assumes \( P \rightarrow_{\alpha} \prec P' \)

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

lemma weakCongSum2:
fixes \( Q :: \text{ccs} \)
and \( \alpha :: \text{act} \)
and \( Q' :: \text{ccs} \)
and \( P :: \text{ccs} \)
assumes $Q \Rightarrow \alpha \prec Q'$

shows $P \oplus Q \Rightarrow \alpha \prec Q'$
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 $\alpha :: act$
  and $P' :: ccs$
  and $Q :: ccs$

assumes $P \Rightarrow \alpha \prec P'$

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

lemma weakCongPar2:
  fixes $Q :: ccs$
  and $\alpha :: act$
  and $Q' :: ccs$
  and $P :: ccs$

assumes $Q \Rightarrow \alpha \prec Q'$

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

lemma weakCongSync:
  fixes $P :: ccs$
  and $\alpha :: act$
  and $P' :: ccs$
  and $Q :: ccs$

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

shows $P \parallel Q \Rightarrow \tau \prec P' \parallel Q'$
using assms
apply(auto simp add: weakCongTrans-def)
apply(rule-tac $x=P'' \parallel P'' \cdot a \in exI$)
lemma weakCongRes:
  fixes $P :: ccs$
  and $\alpha :: act$
  and $P' :: ccs$
  and $x :: name$
  assumes $P \Longrightarrow \alpha \prec P'$
  and $x \not\in \alpha$
  shows $(\nu x)P \Longrightarrow \alpha \prec (\nu x)P'$
using assms
by(auto simp add: weakCongTrans-def
  (blast dest: tauChainRes Res))

lemma weakCongRepl:
  fixes $P :: ccs$
  and $\alpha :: act$
  and $P' :: ccs$
  assumes $P \parallel !P \Longrightarrow \alpha \prec P'$
  shows $!P \Longrightarrow \alpha \prec P'$
using assms
apply(auto simp add: weakCongTrans-def)
apply(case-tac $P'' = P \parallel !P$)
apply auto
apply(force intro: Bang simp add: tauChain-def)
by(force intro: tauChainRepl)

theory Weak-Semantics
  imports Weak-Cong-Semantics
begin

definition weakTrans :: ccs $\Rightarrow$ act $\Rightarrow$ ccs $\Rightarrow$ bool $\cdot \Longrightarrow \cdot \prec \cdot [80, 80, 80] 80$
 where $P \Longrightarrow \alpha \prec P' \equiv P \Longrightarrow \alpha \prec P' \vee (\alpha = \tau \land P = P')$

lemma weakEmptyTrans[simp]:
  fixes $P :: ccs$
shows $P \Rightarrow \tau \prec P$
by (auto simp add: weakTrans-def)

lemma weakTransCases [consumes 1, case-names Base Step]:
fixes $P :: ccs$
and $\alpha :: act$
and $P' :: ccs$

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

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

lemma weakCongTransitionWeakTransition:
fixes $P :: ccs$
and $\alpha :: act$
and $P' :: ccs$

assumes $P \Rightarrow \alpha \prec P'$

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

lemma transitionWeakTransition:
fixes $P :: ccs$
and $\alpha :: act$
and $P' :: ccs$

assumes $P \rightarrow\leftarrow \alpha \prec P'$

shows $P \Rightarrow \alpha \prec P'$
using assms
by (auto dest: transitionWeakCongTransition weakCongTransitionWeakTransition)

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

shows $\alpha.(P) \Rightarrow \alpha \prec P$
by (auto simp add: weakTrans-def intro: weakCongAction)

lemma weakSum1:
fixes $P :: ccs$
and $\alpha :: act$
and \( P' :: \text{ccs} \)
and \( Q :: \text{ccs} \)

assumes \( P \Rightarrow \hat{\alpha} \prec P' \)
and \( P \neq P' \)

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

lemma \textit{weakSum2}:
fixes \( Q :: \text{ccs} \)
and \( \alpha :: \text{act} \)
and \( Q' :: \text{ccs} \)
and \( P :: \text{ccs} \)

assumes \( Q \Rightarrow \hat{\alpha} \prec Q' \)
and \( Q \neq Q' \)

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

lemma \textit{weakPar1}:
fixes \( P :: \text{ccs} \)
and \( \alpha :: \text{act} \)
and \( P' :: \text{ccs} \)
and \( Q :: \text{ccs} \)

assumes \( P \Rightarrow \hat{\alpha} \prec P' \)

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

lemma \textit{weakPar2}:
fixes \( Q :: \text{ccs} \)
and \( \alpha :: \text{act} \)
and \( Q' :: \text{ccs} \)
and \( P :: \text{ccs} \)

assumes \( Q \Rightarrow \hat{\alpha} \prec Q' \)

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

lemma \textit{weakSync}:
fixes \( P :: \text{ccs} \)

\[ \]
and \( \alpha :: \text{act} \)
and \( P' :: \text{ccs} \)
and \( Q :: \text{ccs} \)

assumes \( P \Rightarrow \hat{\alpha} \prec P' \)
and \( Q \Rightarrow (\text{caAction} \ \alpha) \prec Q' \)
and \( \alpha \neq \tau \)

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

lemma \text{weakRes}:
fixes \( P :: \text{ccs} \)
and \( \alpha :: \text{act} \)
and \( P' :: \text{ccs} \)
and \( x :: \text{name} \)

assumes \( P \Rightarrow \hat{\alpha} \prec P' \)
and \( x \notin \alpha \)

shows \( (\nu x)P \Rightarrow \hat{\alpha} \prec (\nu x)P' \)
using \text{assms}
by(auto simp add: weakTrans-def intro: weakCongRes)

lemma \text{weakRepl}:
fixes \( P :: \text{ccs} \)
and \( \alpha :: \text{act} \)
and \( P' :: \text{ccs} \)

assumes \( P \parallel !P \Rightarrow \hat{\alpha} \prec P' \)
and \( P' \neq P \parallel !P \)

shows \( !P \Rightarrow \alpha \prec P' \)
using \text{assms}
by(auto simp add: weakTrans-def intro: weakCongRepl)

end

theory \text{Strong-Sim}
imports \text{Agent}
begin

definition \text{simulation} :: \text{ccs} \Rightarrow (\text{ccs} \times \text{ccs}) \Rightarrow \text{ccs} \Rightarrow \text{bool} \ (- \Rightarrow \cdot \cdot \cdot - \cdot \cdot [80, 80, 80])

where
\( P \Rightarrow [\text{Rel}] Q \equiv \forall a \ Q'. \ (a \Rightarrow a \prec Q' \Rightarrow (\exists P'. \ P \Rightarrow a \prec P' \land (P', Q') \in \text{Rel}) \)

lemma \text{simI}[\text{case-names Sim}]:

17
fixes $P :: ccs$
and $Rel :: (ccs \times ccs) \text{ set}$
and $Q :: ccs$

assumes \( \forall \alpha \ Q'. \; Q \rightarrow \alpha \prec Q' \Rightarrow \exists \; P'. \; P \rightarrow \alpha \prec P' \land (P', Q') \in Rel \)

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

lemma simE:
fixes $P :: ccs$
and $Rel :: (ccs \times ccs) \text{ set}$
and $Q :: ccs$
and $\alpha :: act$
and $Q' :: ccs$

assumes $P \rightarrow [Rel] \; Q$
and $Q \rightarrow \alpha \prec Q'$

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

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

assumes $Id \subseteq Rel$

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

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

assumes $P \rightarrow [Rel] \; Q$
and $Q \rightarrow [Rel'] \; R$
and $Rel \circ Rel' \subseteq Rel''$

shows $P \rightarrow [Rel''] \; R$
using assms
by (force simp add: simulation-def)
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 −→ a ⊮ Q′ −→ (∃ P′. P −→ a < 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 −→τ Q′ and (P, Q) ∈ Rel and Sim: ∨ R S. (R, S) ∈ Rel −→ R −→ <Rel> S
  obtains P′ where P −→τ P′ and (P′, Q′) ∈ Rel
using \( Q \rightarrow\tau, Q \backslash \langle P, Q \rangle \in \text{Rel} \)

proof (induct arbitrary: thesis rule: tauChainInduct)

  case Base
  from \( \langle P, Q \rangle \in \text{Rel} \) show \( ?\) case
    by (force intro: Base)

next
  case (Step \( Q'' \) \( Q' \))
  from \( \langle P, Q \rangle \in \text{Rel} \) obtain \( P'' \) where \( P \rightarrow\tau P'' \) and \( \langle P'', Q'' \rangle \in \text{Rel} \)
  by (blast intro: Step)
  from \( \langle P'', Q'' \rangle \in \text{Rel} \) have \( P'' \rightarrow\tau \prec \langle \text{Rel} \rangle Q'' \) by (rule Sim)
  then obtain \( P' \) where \( P'' \rightarrow\tau \prec \langle P', Q' \rangle \in \text{Rel} \) using \( \langle Q'' \rightarrow\tau \prec \langle \text{Rel} \rangle \rangle \)
  by (rule weakSimE)
  with \( \langle P \rightarrow\tau, P'' \rangle \) show thesis
  by (force simp add: weakTrans-def weakCongTrans-def intro: Step)

qed

lemma simE2:
fixes \( P :: \text{ccs} \)
and \( \text{Rel} :: (\text{ccs} \times \text{ccs}) \) set
and \( Q :: \text{ccs} \)
and \( \alpha :: \text{act} \)
and \( Q' :: \text{ccs} \)

assumes \( \langle P, Q \rangle \in \text{Rel} \)
and \( Q \rightarrow\tau \prec Q' \)
and \( \text{Sim}: \bigwedge \langle R, S \rangle. (R, S) \in \text{Rel} \rightarrow R \rightarrow\tau \prec \langle \text{Rel} \rangle S \)

obtains \( P' \) where \( P \rightarrow\tau \prec \langle P', Q' \rangle \in \text{Rel} \)

proof
  assume Goal: \( \bigwedge P'. \langle P \rightarrow\tau \prec \langle P', Q' \rangle \in \text{Rel} \rangle \rightarrow \text{thesis} \)
  moreover from \( \langle Q \rightarrow\tau \prec Q' \rangle \) have \( \bigexists P'. \langle P \rightarrow\tau \prec \langle P', Q' \rangle \rangle \)
  proof (induct rule: weakTransCases)
  case Base
  from \( \langle P, Q \rangle \in \text{Rel} \) show \( ?\) case by force

next
  case Step
  from \( \langle Q \rightarrow\tau \prec Q' \rangle \) obtain \( Q'' \)
  where \( \text{QChain}: Q \rightarrow\tau Q''' \) and \( \langle P''', Q''' \rangle \)
  proof (rule weakCongTransE)
    from \( \langle P''', Q''' \rangle \in \text{Rel} \) Sim obtain \( P'''' \) where \( 
    \text{PChain}: P \rightarrow\tau P'''' \)
  and \( \langle P''', Q''' \rangle \) in \( \text{Rel} \)
    by (rule simTauChain)
    from \( \langle P''', Q''' \rangle \in \text{Rel} \) have \( P'''' \rightarrow\tau \prec \langle \text{Rel} \rangle Q''' \) by (rule Sim)
  then obtain \( P'' \) where \( \langle P''', Q''' \rangle \rightarrow\tau \prec \langle P'', Q'' \rangle \) in \( \text{Rel} \)
        using \( \langle Q'' \rightarrow\tau \prec \langle \text{Rel} \rangle \rangle \)
        from \( \langle Q'' \rangle \) Sim obtain \( P' \) where \( \langle P'' \rangle \rightarrow\tau \prec \langle P', Q' \rangle \) in \( \text{Rel} \)
        by (rule simTauChain)
from $P''$Trans $P''$Chain Step show \(?thesis

proof (induct rule: \textit{weakTransCases})
  case Base
  from $P$Chain $\langle P'' \rightarrow_{\tau} P' \rangle$ have $P \rightarrow_{\tau} P'$
  proof (induct rule: \textit{tauChainInduct})
    case Base
    from $\langle P \rightarrow_{\tau} P' \rangle$ show \(?case
    proof (induct rule: \textit{tauChainInduct})
      case Base
      show \(?case by simp
    next
      case (Step $P'$ $P''$)
      thus \(?case by (fastforce simp add: \textit{weakTrans-def} \textit{weakCongTrans-def})
    qed
  next
  case (Step $P'''$ $P''$)
  thus \(?case by (fastforce simp add: \textit{weakTrans-def} \textit{weakCongTrans-def})
  qed
  with $\langle (P', Q') \in \text{Rel} \rangle$ show \(?case by blast
  next
  case Step
  thus \(?case using $\langle (P', Q') \in \text{Rel} \rangle$ $P$Chain
  by (rule-tac $x=P'$ in exI) (force simp add: \textit{weakTrans-def} \textit{weakCongTrans-def})
  qed
  qed
ultimately show \(?thesis
  by blast
qed

lemma \texttt{reflexive}:
  fixes $P$ :: \texttt{ccs}
  and Rel :: $\langle \texttt{ccs} \times \texttt{ccs} \rangle$ set
  assumes \texttt{Id} $\subseteq$ Rel
  shows $P \rightarrow_{\hat{\text{<\text{Rel}>}}} P$
  using \texttt{assms}
  by (auto simp add: \textit{weakSimulation-def} intro: \textit{transitionWeakCongTransition weakCongTransitionWeakTransition})

lemma \texttt{transitive}:
  fixes $P$ :: \texttt{ccs}
  and Rel :: $\langle \texttt{ccs} \times \texttt{ccs} \rangle$ set
  and $Q$ :: \texttt{ccs}
  and Rel' :: $\langle \texttt{ccs} \times \texttt{ccs} \rangle$ set
  and $R$ :: \texttt{ccs}
  and Rel'' :: $\langle \texttt{ccs} \times \texttt{ccs} \rangle$ set
  assumes $\langle P, Q \rangle \in \text{Rel}$

21
\[
\text{and } Q \rightsquigarrow <\text{Rel}'> R \\
\text{and } \text{Rel} \circ \text{Rel}' \subseteq \text{Rel}'' \\
\text{and } \forall S T. (S, T) \in \text{Rel} \implies S \rightsquigarrow <\text{Rel}> T
\]

shows \( P \rightsquigarrow <\text{Rel}'> R \)

\textbf{proof} (induct rule: \texttt{weakSimI})

\textbf{case} (\texttt{Sim} \( \alpha \) \texttt{R}')

\textbf{thus} ?\textbf{case} using \texttt{assms}

\textbf{apply} (\texttt{drule-tac} \( Q=R \) in \texttt{weakSimE}, \texttt{auto})

\textbf{by} (\texttt{drule-tac} \( Q=Q \) in \texttt{simE2}, \texttt{auto})

\textbf{qed}

\textbf{lemma} \texttt{weakMonotonic}:

\textbf{fixes} \( P :: \texttt{ccs} \)
\textbf{and} \( A :: (\texttt{ccs} \times \texttt{ccs}) \texttt{set} \)
\textbf{and} \( Q :: \texttt{ccs} \)
\textbf{and} \( B :: (\texttt{ccs} \times \texttt{ccs}) \texttt{set} \)

\textbf{assumes} \( P \rightsquigarrow <A> Q \)
\textbf{and} \( A \subseteq B \)

\textbf{shows} \( P \rightsquigarrow <B> Q \)

\textbf{using} \texttt{assms}

\textbf{by} (\texttt{fastforce simp add: weakSimulation-def})

\textbf{lemma} \texttt{simWeakSim}:

\textbf{fixes} \( P :: \texttt{ccs} \)
\textbf{and} \( \text{Rel} :: (\texttt{ccs} \times \texttt{ccs}) \texttt{set} \)
\textbf{and} \( Q :: \texttt{ccs} \)

\textbf{assumes} \( P \rightsquigarrow [\text{Rel}] Q \)

\textbf{shows} \( P \rightsquigarrow <\text{Rel}> Q \)

\textbf{using} \texttt{assms}

\textbf{by} (\texttt{rule-tac} \texttt{weakSimI}, \texttt{auto})

(blast dest: \texttt{simE transitionWeakTransition})

\textbf{end}

\textbf{theory} \texttt{Weak-Cong-Sim}

\textbf{imports} \texttt{Weak-Cong-Semantics Weak-Sim Strong-Sim}

\textbf{begin}

\textbf{definition} \texttt{weakCongSimulation} :: \( \texttt{ccs} \Rightarrow (\texttt{ccs} \times \texttt{ccs}) \texttt{set} \Rightarrow \texttt{ccs} \Rightarrow \texttt{bool} \) (- \( \rightsquigarrow <\rightarrow \)
- [80, 80, 80] 80)

\textbf{where}

\( P \rightsquigarrow <\text{Rel}> Q \equiv \forall a Q'. Q \rightsquigarrow a < Q' \rightsquigarrow (\exists P'. P \rightsquigarrow a < P' \wedge (P', Q') \in \text{Rel}) \)

\textbf{end}

22
lemma weakSimI [case-names Sim]:
fixes P :: ccs
and Rel :: (ccs × ccs) set
and Q :: ccs

assumes \( \forall \alpha \quad Q' \quad Q \rightarrow^{\alpha} Q' \quad \exists P'. \quad P \rightarrow^{\alpha} P' \land (P', Q') \in \text{Rel} \)

shows \( P \rightarrow<\text{Rel}> \quad Q \)
using assms
by (auto simp add: weakCongSimulation-def)

lemma weakSimE:
fixes P :: ccs
and Rel :: (ccs × ccs) set
and Q :: ccs
and \( \alpha \) :: act
and Q' :: ccs

assumes \( P \rightarrow<\text{Rel}> \quad Q \)
and \( Q \rightarrow^{\alpha} Q' \)

obtains \( P' \) where \( P \rightarrow^{\alpha} P' \land (P', Q') \in \text{Rel} \)
using assms
by (auto simp add: weakCongSimulation-def)

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

assumes \( P \rightarrow<\text{Rel}> \quad Q \)

shows \( P \rightarrow<\text{Rel}> \quad Q \)
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 \( P \rightarrow<\text{Rel}> \quad Q \)

shows \( P \rightarrow<\text{Rel}> \quad Q \)
using assms
by (rule-tac Weak-Sim.weakSimI, auto)
(blast dest: weakSimE weakCongTransitionWeakTransition)
Lemma test:
assumes $P \mapsto \tau P'$ shows $P = P' \lor (\exists P''. P \mapsto \tau P'' \land P'' \mapsto \tau P')$
using assms
by (induct rule: tauChainInduct) auto

Lemma tauChainCasesSym [consumes 1, case-names cTauNil cTauStep]
assumes $P \mapsto \tau P'$
and $\text{Prop } P$
and $\bigwedge P'. [P \mapsto \tau P'; P'' \mapsto \tau P'] \Rightarrow \text{Prop } P'$
shows $\text{Prop } P'$
using assms
by (blast dest: test)

Lemma simE2:
fixes $P :: \text{ccs}$
and $\text{Rel} :: (\text{ccs} \times \text{ccs}) \text{ set}$
and $Q :: \text{ccs}$
and $\alpha :: \text{act}$
and $Q' :: \text{ccs}$
assumes $P \mapsto <\text{Rel}> Q$
and $Q \mapsto \alpha < Q'$
and $\text{Sim} : \bigwedge R S. (R, S) \in \text{Rel} \Rightarrow R \sim <\text{Rel}> S$

obtains $P'$ where $P \mapsto \alpha < P'$ and $(P', Q') \in \text{Rel}$
proof
\begin{itemize}
  \item assume $\text{Goal} : \bigwedge P'. [P \mapsto \alpha < P'; (P', Q') \in \text{Rel}] \Rightarrow \text{thesis}$
  \item from $(Q \mapsto \alpha < Q'' ; \text{obtain } Q''' Q''')$
    \begin{itemize}
      \item where $Q\text{Chain} : Q \mapsto \tau Q'''$ and $Q'''\text{Trans} : Q''' \mapsto \alpha < Q''$ and $Q'''\text{Chain} : Q''' \mapsto \tau Q'''
    \end{itemize}
  \item by (rule weakCongTransE)
  \item from $Q\text{Chain} Q'''\text{Trans} \text{ show } \text{thesis}$
  \item proof (induct rule: tauChainCasesSym)
    \begin{itemize}
      \item case cTauNil
        \begin{itemize}
          \item from $(P \mapsto <\text{Rel}> Q ; Q \mapsto \alpha < Q'' ; \text{obtain } P'''$ where $P\text{Trans} : P \mapsto \alpha < P'''$ and $(P', Q') \in \text{Rel}$
          \item by (blast dest: weakSimE)
        \end{itemize}
      \item moreover from $Q'''\text{Chain} (P''', Q') \in \text{Rel} \text{ Sim obtain } P' \text{ where } P'''\text{Chain} : P''' \mapsto \tau P'$ and $(P', Q') \in \text{Rel}$
        \begin{itemize}
          \item by (rule simTauChain)
        \end{itemize}
    \end{itemize}
  \item with $P\text{Trans} P'''\text{Chain} \text{ show } \text{thesis}$
    \begin{itemize}
      \item by (force intro: Goal simp add: weakCongTrans-def weakTrans-def)
    \end{itemize}
\end{itemize}
next
\begin{itemize}
  \item case (cTauStep $Q''''$)
    \begin{itemize}
      \item from $(P \mapsto <\text{Rel}> Q ; Q \mapsto \tau < Q'''' ; \text{obtain } P''''$ where $P\text{Chain} : P \mapsto \tau < P''''$ and $(P'', Q'') \in \text{Rel}$
    \end{itemize}
\end{itemize}
by (drule_tac weakSimE) auto
from \( P^{\prime\prime\prime\prime} \Rightarrow \tau\) \( Q^{\prime\prime\prime\prime} \in \text{Rel} \) obtain \( P^{\prime\prime\prime} \) where \( P^{\prime\prime\prime} \) Chain: \( P^{\prime\prime\prime\prime} \Rightarrow P^{\prime\prime} \) and \( (P^{\prime\prime\prime}, Q^{\prime\prime\prime}) \in \text{Rel} \)
by (rule simTauChain)
from \( (P^{\prime\prime\prime}, Q^{\prime\prime\prime}) \in \text{Rel} \) have \( P^{\prime\prime} \Rightarrow \hat{\alpha} \preceq P^{\prime\prime} \) and \( (P^{\prime\prime}, Q^{\prime\prime}) \in \text{Rel} \)
using \( Q^{\prime\prime} \) Trans by (rule Weak-Sim.weakSimE)
from \( Q^{\prime\prime} \) Chain \( (P^{\prime\prime}, Q^{\prime\prime}) \in \text{Rel} \) obtain \( P^{\prime} \) where \( P^{\prime\prime} \) Chain: \( P^{\prime\prime} \Rightarrow \tau\)
\( P^{\prime}\) and \( (P^{\prime}, Q^{\prime}) \in \text{Rel} \)
by (rule simTauChain)
from \( P^{\prime\prime} \) Chain \( P^{\prime\prime}\) Trans \( P^{\prime}\) Chain
have \( P \Rightarrow \alpha < P^{\prime}\)
apply (auto simp add: weakCongTrans-def weakTrans-def)
apply (rule-tac \( x=P^{\prime\prime}\) aa in \( \exists I\))
apply auto
defers
apply blast
by (auto simp add: tauChain-def)

with \( (P^{\prime}, Q^{\prime}) \in \text{Rel} \) show \?thesis
by (force intro: Goal simp add: weakCongTrans-def weakTrans-def)
qed

lemma reflexive:
fixes \( P \) :: ccs
and \( \text{Rel} \) :: (ccs × ccs) set
assumes \( \text{Id} \subseteq \text{Rel} \)
shows \( P \Rightarrow<\text{Rel}> P \)
using assms
by (auto simp add: weakCongSimulation-def intro: transitionWeakCongTransition)

lemma transitive:
fixes \( P \) :: ccs
and \( \text{Rel} \) :: (ccs × ccs) set
and \( Q \) :: ccs
and \( \text{Rel}^\prime \) :: (ccs × ccs) set
and \( R \) :: ccs
and \( \text{Rel}^\prime\prime \) :: (ccs × ccs) set
assumes \( P \Rightarrow<\text{Rel}> Q \)
and \( Q \Rightarrow<\text{Rel}'> R \)
and \( \text{Rel} \circ \text{Rel}^\prime \subseteq \text{Rel}'' \)
and \( \forall S T. (S, T) \in \text{Rel} \implies S \Rightarrow<\text{Rel}> T \)
sows \( P \Rightarrow<\text{Rel}''\) \( R \)
proof (induct rule: weakSimI)
case (Sim α R')
thus ?case using assms
  apply (drule-tac Q=\alpha R in weakSimE, auto)
  by (drule-tac Q=\alpha 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: weakCongSimulation-def)
end

theory Strong-Sim-SC
  imports Strong-Sim
begin

lemma resNilLeft:
  fixes x :: name

  shows (\nu x \nu x) 0 ↷[Rel] 0
  by (auto simp add: simulation-def)

lemma resNilRight:
  fixes x :: name

  shows 0 ↷[Rel] (\nu x \nu x) 0
  by (auto simp add: simulation-def elim: resCases)

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

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

lemma scopeExtSumLeft:
  fixes x :: name
  and P :: ccs
  and Q :: ccs
assumes $x ↳ P$
and $\forall y. y ↳ R \implies (\langle \nu y \rangle R, R) \in \text{Rel}$
and $\text{Id} \subseteq \text{Rel}$

shows $\langle \nu x \rangle (P \oplus Q) \leadsto [\text{Rel}] P \oplus \langle \nu x \rangle Q$
using asms
apply (auto simp add: simulation-def)
by (elim sumCases resCases) (blast intro: Res Sum1 Sum2 C1 dest: freshDerivative)+

lemma scopeExtSumRight:
fixes $x :: \text{name}$
and $P :: \text{ccs}$
and $Q :: \text{ccs}$

assumes $x ↳ P$
and $\forall y. y ↳ R \implies (R, \langle \nu y \rangle R) \in \text{Rel}$
and $\text{Id} \subseteq \text{Rel}$

shows $P \oplus \langle \nu x \rangle Q \leadsto [\text{Rel}] \langle \nu x \rangle (P \oplus Q)$
using asms
apply (auto simp add: simulation-def)
by (elim sumCases resCases) (blast intro: Res Sum1 Sum2 C1 dest: freshDerivative)+

lemma scopeExtLeft:
fixes $x :: \text{name}$
and $P :: \text{ccs}$
and $Q :: \text{ccs}$

assumes $x ↳ P$
and $\forall y. R T \implies (\langle \nu y \rangle (R \parallel T), R \parallel \langle \nu y \rangle T) \in \text{Rel}$

shows $\langle \nu x \rangle (P \parallel Q) \leadsto [\text{Rel}] \langle \nu x \rangle (P \parallel Q)$
using asms
by (fastforce elim: parCases resCases intro: Res C1 Par1 Par2 Comm dest: freshDerivative simp add: simulation-def)

lemma scopeExtRight:
fixes $x :: \text{name}$
and $P :: \text{ccs}$
and $Q :: \text{ccs}$

assumes $x ↳ P$
and $\forall y. R T \implies (R \parallel \langle \nu y \rangle T, \langle \nu y \rangle (R \parallel T)) \in \text{Rel}$

shows $P \parallel \langle \nu x \rangle Q \leadsto [\text{Rel}] \langle \nu x \rangle (P \parallel Q)$
using asms
by (fastforce elim: parCases resCases intro: Res C1 Par1 Par2 Comm dest: freshDerivative)
lemma sumComm:
  fixes P :: ccs
  and Q :: ccs
  assumes Id ⊆ Rel
  shows P ⊕ Q ⊳ [Rel] Q ⊕ 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 ⊆ Rel
  shows (P ⊕ Q) ⊕ R ⊳ [Rel] P ⊕ (Q ⊕ 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 ⊆ Rel
  shows P ⊕ (Q ⊕ R) ⊳ [Rel] (P ⊕ Q) ⊕ R
  using assms
  by (intro simI, elim sumCases) (blast intro: Sum1 Sum2)+

lemma sumIdLeft:
  fixes P :: ccs
  and Rel :: (ccs × ccs) set
  assumes Id ⊆ Rel
  shows P ⊕ 0 ⊳ [Rel] P
  using assms
  by (auto simp add: simulation-def intro: Sum1)

lemma sumIdRight:
  fixes P :: ccs
  and Rel :: (ccs × ccs) set
  assumes Id ⊆ Rel
shows $P \rightsquigarrow_{[\text{Rel}]} P \oplus 0$
using assms
by (fastforce simp add: simulation-def elim: sumCases)

lemma parComm:
fixes $P :: \text{ccs}$
and $Q :: \text{ccs}$

assumes $C1: \forall R. \forall T. (R \parallel T, T \parallel R) \in \text{Rel}$

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

lemma parAssocLeft:
fixes $P :: \text{ccs}$
and $Q :: \text{ccs}$
and $R :: \text{ccs}$

assumes $C1: \forall S. \forall T. \forall U. ((S \parallel T) \parallel U, S \parallel (T \parallel U)) \in \text{Rel}$

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

lemma parAssocRight:
fixes $P :: \text{ccs}$
and $Q :: \text{ccs}$
and $R :: \text{ccs}$

assumes $C1: \forall S. \forall T. \forall U. ((S \parallel (T \parallel U), (S \parallel T) \parallel U)) \in \text{Rel}$

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

lemma parIdLeft:
fixes $P :: \text{ccs}$
and $\text{Rel} :: (\text{ccs} \times \text{ccs}) \text{ set}$

assumes $\forall Q. (Q \parallel 0, Q) \in \text{Rel}$

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

lemma parIdRight:
fixes $P :: \text{ccs}$
and $\text{Rel} :: (\text{ccs} \times \text{ccs}) \text{ set}$

assumes $\forall Q. (Q, Q \parallel 0) \in \text{Rel}$


shows \( P \rightsquigarrow [\text{Ref}] P \parallel 0 \)
using \text{assms}
by (fastforce simp add: simulation-def elim: parCases)

declare \text{fresh-atm\{simp\}}

\textbf{lemma resActLeft:}
fixes \( x :: \text{name} \)
and \( \alpha :: \text{act} \)
and \( P :: \text{ccs} \)

assumes \( x \not\in\alpha \)
and \( \text{Id} \subseteq \text{Rel} \)

shows \( (\nu x)(\alpha.(P)) \rightsquigarrow [\text{Ref}] (\alpha.(\nu x)P) \)
using \text{assms}
by (fastforce simp add: simulation-def elim: actCases intro: Res Action)

\textbf{lemma resActRight:}
fixes \( x :: \text{name} \)
and \( \alpha :: \text{act} \)
and \( P :: \text{ccs} \)

assumes \( x \not\in\alpha \)
and \( \text{Id} \subseteq \text{Rel} \)

shows \( \alpha.(\nu x)P \rightsquigarrow [\text{Ref}] (\nu x)(\alpha(P)) \)
using \text{assms}
by (fastforce simp add: simulation-def elim: resCases actCases intro: Action)

\textbf{lemma resComm:}
fixes \( x :: \text{name} \)
and \( y :: \text{name} \)
and \( P :: \text{ccs} \)

assumes \( \forall Q. (\nu x)((\nu y)Q), (\nu y)((\nu x)Q)) \in \text{Rel} \)

shows \( (\nu x)(\nu y)P \rightsquigarrow [\text{Ref}] (\nu y)(\nu x)P \)
using \text{assms}
by (fastforce simp add: simulation-def elim: resCases intro: Res)

\textbf{inductive-cases bangCases\{simplified ccs.distinct act.distinct\}}: \( !P \rightsquigarrow \alpha < P' \)

\textbf{lemma bangUnfoldLeft:}
fixes \( P :: \text{ccs} \)

assumes \( \text{Id} \subseteq \text{Rel} \)
shows $P \parallel !P \rightsquigarrow [\text{Rel}] !P$
using assms
by (fastforce simp add: simulation-def ccs.inject elim: bangCases)

lemma bangUnfoldRight:
  fixes $P :: \text{ccs}$
  assumes $\text{Id} \subseteq \text{Rel}$
  shows $!P \rightsquigarrow [\text{Rel}] P \parallel !P$
using assms
by (fastforce simp add: simulation-def ccs.inject intro: Bang)
end

theory Strong-Bisim
  imports Strong-Sim
begin

lemma monotonic:
  fixes $P :: \text{ccs}$
  and $A :: \text{(ccs \times ccs) set}$
  and $Q :: \text{ccs}$
  and $B :: \text{(ccs \times ccs) set}$
  assumes $P \rightsquigarrow [A] Q$
  and $A \subseteq B$
  shows $P \rightsquigarrow [B] Q$
using assms
by (fastforce simp add: simulation-def)

lemma monoCoinduct: $\bigwedge x\ y\ xa\ xb\ P\ Q .
  x \leq y \implies
  (Q \rightsquigarrow \{\{xb, xa\}. x\ xb\ xa\}] P) \implies
  (Q \rightsquigarrow \{\{xb, xa\}. y\ xb\ xa\}] P)$
apply auto
apply (rule monotonic)
by (auto dest: le-funE)

coinductive-set bisim :: (ccs \times ccs) set
where
  $[P \rightsquigarrow \text{bisim}] Q; (Q, P) \in \text{bisim} \implies (P, Q) \in \text{bisim}$
monos monoCoinduct

abbreviation
  bisimJudge (\sim [70, 70]) where $P \sim Q \equiv (P, Q) \in \text{bisim}$

lemma bisimCoinductAux [consumes 1]:

31
fixes $P :: ccs$
and $Q :: ccs$
and $X :: (ccs \times ccs) \text{ set}$

assumes $(P, Q) \in X$
and $\forall P, Q. \ (P, Q) \in X \implies P \sim [(X \cup bisim)] Q \land (Q, P) \in X$

shows $P \sim Q$

proof

have $X \cup \text{bisim} = \{(P, Q). \ (P, Q) \in X \lor (P, Q) \in \text{bisim}\} \text{ by auto}$
with assms show ?thesis
  by coinduct simp
qed

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

assumes $(P, Q) \in X$
and $\forall R, S. \ (R, S) \in X \implies R \sim [(X \cup bisim)] S$
and $\forall R, S. \ (R, S) \in X \implies (S, R) \in X$

shows $P \sim Q$

proof

have $X \cup \text{bisim} = \{(P, Q). \ (P, Q) \in X \lor (P, Q) \in \text{bisim}\} \text{ by auto}$
with assms show ?thesis
  by coinduct simp
qed

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

assumes $(P, Q) \in X$
and $\forall R, S. \ (R, S) \in X \implies R \sim [X] S \land (S, R) \in X$

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 $X :: (ccs \times ccs) \text{ set}$

assumes $(P, Q) \in X$
and $\forall P, Q. \ (P, Q) \in X \implies P \sim [X] Q$
\[ \land P \land Q. (P, Q) \in X \Rightarrow (Q, P) \in X \]

shows \( P \sim Q \)

proof –

have \( X \cup \text{bisim} = \{(P, Q). (P, Q) \in X \lor (P, Q) \in \text{bisim}\} \) by auto

with assms show \( \text{thesis} \)

by(coinduct rule: bisimCoinduct) (blast intro: monotonic)+

qed

lemma bisimE:

fixes \( P :: \text{ccs} \)

and \( Q :: \text{ccs} \)

assumes \( P \sim Q \)

shows \( P \Rightarrow\text{bisim} Q \)

and \( Q \sim P \)

using assms

by(auto simp add: intro: bisim_cases)

lemma bisimI:

fixes \( P :: \text{ccs} \)

and \( Q :: \text{ccs} \)

assumes \( P \Rightarrow\text{bisim} Q \)

and \( Q \sim P \)

shows \( P \sim Q \)

using assms

by(auto intro: bisim_intros)

lemma reflexive:

fixes \( P :: \text{ccs} \)

shows \( P \sim P \)

proof –

have \( (P, P) \in \text{Id} \) by blast

thus \( \text{thesis} \)

by(coinduct rule: bisimCoinduct) (auto intro: reflexive)

qed

lemma symmetric:

fixes \( P :: \text{ccs} \)

and \( Q :: \text{ccs} \)

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

  shows $P \sim R$

proof
  from assms have $(P, R) \in \text{bisim} O \text{bisim}$ 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 $(P, Q) \in X$
  and $rSim: \bigwedge R S. (R, S) \in X \implies R \rightsquigarrow [(\text{bisim} O X O \text{bisim})] S$
  and $rSym: \bigwedge R S. (R, S) \in X \implies (S, R) \in X$

  shows $P \sim Q$

proof
  from $(P, Q) \in X$ have $(P, Q) \in \text{bisim} O X O \text{bisim}$
    by (auto intro: reflexive)
  thus ?thesis
    proof (coinduct rule: bisimWeakCoinduct)
      case (cSim $P Q$)
      from $(P, Q) \in \text{bisim} O X O \text{bisim}$
      obtain $R S$ where $P \sim R$ and $(R, S) \in X$ and $S \sim Q$
        by auto
      moreover from $(R, S) \in X$ have $R \rightsquigarrow [(\text{bisim} O X O \text{bisim})] S$
        by (rule rSim)
      moreover have $\text{bisim} O (\text{bisim} O X O \text{bisim}) \subseteq \text{bisim} O X O \text{bisim}$
        by (auto intro: transitive)
      ultimately have $P \rightsquigarrow [(\text{bisim} O X O \text{bisim})] S$
        by (rule Strong-Sim.transitive)
      moreover from $(S \sim Q)$ have $S \rightsquigarrow [(\text{bisim})] Q$
        by (rule bisimE)
      moreover have $(\text{bisim} O X O \text{bisim}) O \text{bisim} \subseteq \text{bisim} O X O \text{bisim}$
        by (auto intro: transitive)
      ultimately show ?case
    by (rule Strong-Sim.transitive)
  next
    case (cSym $P Q$)
    thus ?case
      by (auto dest: symmetric rSym)
theory Strong-Sim-Pres
imports Strong-Sim
begin

lemma actPres:
fixes $P :: ccs$
and $Q :: ccs$
and $Rel :: (ccs \times ccs) set$
and $a :: name$
and $Rel' :: (ccs \times ccs) set$

assumes $(P, Q) \in Rel$

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

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

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

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

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

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

shows $P \parallel R \sim [Rel''] Q \parallel T$
proof (induct rule: simI)
case(Sim a QT)
from `Q T `a QT`
show ?case
proof(induct rule: parCases)
case(cPar1 Q')
from `P `[Rel] Q: `Q `a Q' obtain P' where P `a P' and (P', Q') `Rel
by(rule simE)
from `P `a P' have P `R `a P' `R `by(rule Par1)
moreover from `(P', Q') `Rel `(R, T) `Rel` have (P' `R, Q' `T) `Rel'' `by(rule C1)
ultimately show ?case `by blast
next
case(cPar2 T')
from `R `[Rel] T: `T `a T' obtain R' where R `a R' and (R', T') `Rel'
by(rule simE)
from `R `a R' have P `R `a P `R `by(rule Par2)
moreover from `(P, Q) `Rel `(R', T') `Rel` have (P `R', Q `T') `Rel'' `by(rule C1)
ultimately show ?case `by blast
next
case(cComm Q' T' a)
from `P `[Rel] Q: `Q `a Q' obtain P' where P `a P' and (P', Q') `Rel
by(rule simE)
from `R `[Rel] T: `T `coAction a T' obtain R' where R `coAction a R' and (R', T') `Rel'
by(rule simE)
from `P `a P' `R `coAction a R' `a `R' `R' `by(rule Comm)
moreover from `(P', Q') `Rel `(R', T') `Rel` have (P' `R', Q' `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 `[Rel] Q
and `(P, Q) `Rel
and C1: `(S T U, (S, T) `Rel} `(S `U, T `U) `Rel`

shows P `R `[Rel] Q `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 ⇝[Rel] Q
  and \( \forall R S y. (R, S) \in Rel \implies ([\nu y] R, [\nu y] S) \in Rel' \)

  shows \( [\nu x] P \sim[Rel'] [\nu x] Q \)
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: \( \forall R S. (R, S) \in Rel \implies R \sim[Rel] S \)

  shows !P ⇝[bangRel Rel] !Q
proof (induct rule: simI)
  case (Sim α Q')
  { fix Pa α Q'
    assume !Q \sim α < Q' and \((Pa, !Q) \in bangRel Rel\)
    hence \( \exists P'. Pa \sim α < P' \land (P', Q') \in bangRel Rel\)
    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 ⇝[Rel] Q by (rule C1)
        with !Q \sim α < Q' obtain P' where P \sim α < P' and (P', Q') \in Rel
        by (blast dest: simE)
        from \(P \sim α < P'\) have P \parallel α < 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)

    qed
  }
\[\text{case} (\text{BRPar} P R)
\quad \text{from } (R, !Q) \in \text{bangRel Rel} \text{ obtain } R' \text{ where } R \xrightarrow{\alpha} R' \text{ and } (R', Q') \in \text{bangRel Rel} \text{ using } \text{cPar2}
\quad \text{by } \text{blast}
\quad \text{from } R \xrightarrow{\alpha} R' \text{ have } P \parallel R \xrightarrow{\alpha} P \parallel R' \text{ by (rule Par2)}
\quad \text{moreover from } (P, Q) \in \text{Rel} \quad (R', Q') \in \text{bangRel Rel} \quad \text{have } (P \parallel R', Q') \in \text{bangRel Rel} \quad \text{by (rule bangRel.BRPar)}
\quad \text{ultimately show } ?\text{case by blast}
\quad \text{qed}
\]

next

\[\text{case} (\text{cComm} a Q' Q'' Pa)
\quad \text{from } (Pa, Q \parallel !Q) \in \text{bangRel Rel}
\quad \text{show } ?\text{case}
\quad \text{proof (induct rule: BRParCases)}
\quad \text{case} (\text{BRPar} P R)
\quad \text{from } (P, Q) \in \text{Rel} \quad \text{have } P \leadsto [\text{Rel}] Q \text{ by (rule C1)}
\quad \text{with } Q \xrightarrow{\alpha} a \prec Q' \text{ obtain } P' \text{ where } P \xrightarrow{\alpha} a \prec P' \text{ and } (P', Q') \in \text{Rel}
\quad \text{by (blast dest: simE)}
\quad \text{from } (R, !Q) \in \text{bangRel Rel} \quad \text{obtain } R' \text{ where } R \xrightarrow{(\text{coAction} a) \prec R'}
\quad \text{and } (R', Q'') \in \text{bangRel Rel} \text{ using } \text{cComm}
\quad \text{by } \text{blast}
\quad \text{from } P \xrightarrow{\alpha} a \prec P' \quad R \xrightarrow{\alpha} (\text{coAction} a) \prec R' \quad (a \neq \tau) \quad \text{have } P \parallel R \xrightarrow{\tau}
\quad \text{< } P' \parallel R' \text{ by (rule Comm)}
\quad \text{moreover from } (P', Q') \in \text{Rel} \quad (R', Q'') \in \text{bangRel Rel} \quad \text{have } (P' \parallel R', Q'') \in \text{bangRel Rel} \quad \text{by (rule bangRel.BRPar)}
\quad \text{ultimately show } ?\text{case by blast}
\quad \text{qed}
\]

next

\[\text{case} (\text{cBang} \alpha Q' Pa)
\quad \text{from } (Pa, !Q) \in \text{bangRel Rel}
\quad \text{show } ?\text{case}
\quad \text{proof (induct rule: BRBangCases)}
\quad \text{case} (\text{BRBang} P)
\quad \text{from } (P, Q) \in \text{Rel} \quad \text{have } (!P, !Q) \in \text{bangRel Rel} \quad \text{by (rule bangRel.BRBang)}
\quad \text{with } (P, Q) \in \text{Rel} \quad \text{have } (P \parallel !P, Q \parallel !Q) \in \text{bangRel Rel} \quad \text{by (rule bangRel.BRPar)}
\quad \text{then obtain } P' \text{ where } P \parallel !P \xrightarrow{\alpha} P' \text{ and } (P', Q') \in \text{bangRel Rel}
\quad \text{using } \text{cBang}
\quad \text{by } \text{blast}
\quad \text{from } P \parallel !P \xrightarrow{\alpha} P' \text{ have } !P \xrightarrow{\alpha} P' \text{ by (rule Bang)}
\quad \text{thus } ?\text{case using } (P', Q') \in \text{bangRel Rel} \text{ by blast}
\quad \text{qed}
\quad \text{qed}
\]

moreover from \((P, Q) \in \text{Rel} \quad \text{have } (!P, !Q) \in \text{bangRel Rel} \quad \text{by (rule BRBang)}
\]

ultimately show ?\text{case using } !Q \xrightarrow{\alpha} Q' \text{ by blast}
\quad \text{qed}
theory Strong-Bisim-Pres
imports Strong-Bisim Strong-Sim-Pres
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: bisimCoinduct) (auto dest: bisimE intro: actPres)
qed

lemma sumPres:
  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 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 ∼ 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: bisimCoinduct, auto) (blast intro: parPres dest: bisimE)+
qed

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

assumes P ∼ Q

shows (\(\nu x\))P ∼ (\(\nu x\))Q

proof –
let ?X = \{(\(\nu x\))P, (\(\nu x\))Q | x P Q. P ∼ Q\}
from assms have \{(\(\nu x\))P, (\(\nu x\))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
theory Struct-Cong
  imports Agent
begin

inductive structCong :: ccs ⇒ ccs ⇒ bool (≡)
where
  Reflex: P ≡ P
  Symmetry: P ≡ Q =⇒ Q ≡ P
  Transitivity: [P ≡ Q; Q ≡ R] =⇒ P ≡ R
  Parallel Commutativity: P ∥ Q ≡ Q ∥ P
  Parallel Association: (P ∥ Q) ∥ R ≡ P ∥ (Q ∥ R)
  Parallel Identity: P ∥ 0 ≡ P
  Sum Commutativity: P ⊕ Q ≡ Q ⊕ P
  Sum Association: (P ⊕ Q) ⊕ R ≡ P ⊕ (Q ⊕ R)
  Sum Identity: P ⊕ 0 ≡ P
  Residual Nil: νx 0 ≡ 0
  Scope Extension for Parallel: x♯P =⇒ (νx)P ∥ Q ≡ (νx)P ∥ (νx)Q
  Scope Extension for Sum: x♯P =⇒ (νx)P ⊕ Q ≡ (νx)P ⊕ (νx)Q
  Scope Action: x♯α =⇒ (νx)(α.(P)) =≡ α.(νx)P
  Scope Commutativity Aux: x ≠ y =⇒ (νx)((νy)P) ≡ (νy)((νx)P)
  Bang Unfold: !P ≡ !P

equivariance structCong
nominal-inductive structCong
by(auto simp add: abs-fresh)

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

  shows (νx)((νy)P) ≡ (νy)((νx)P)
by(cases x=y) (auto intro: Reflex ScopeCommAux)

end

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

lemma resNil:
  fixes x :: name
shows \( (\nu x)0 \sim 0 \)
proof –
  have \((\nu x)0, 0) \in \{(\nu x)0, 0\}\) 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 \(x \# P\)
  shows \((\nu x)(P \parallel Q) \sim P \parallel (\nu x)Q\)
proof –
  let \(?X = \{(\nu x)(P \parallel Q), P \parallel (\nu x)Q \mid x P Q, x \# P\} \cup \{(P \parallel (\nu x)Q, (\nu x)(P \parallel Q)) \mid x P Q, x \# P\}\)
  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 \(((P \oplus Q) \oplus R, P \oplus (Q \oplus R)) \in \{(P \oplus Q) \oplus R, P \oplus (Q \oplus R)\}\) by simp
  thus \(?thesis\)
  by (coinduct rule: bisimCoinduct) (auto intro: sumAssocLeft sumAssocRight reflexive)
qed
lemma sumId:
fixes P :: ccs

shows P ⊕ 0 ∼ P
proof –
  have (P ⊕ 0, P) ∈ \{(P ⊕ 0, P), (P, P ⊕ 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 ∥ Q ∼ Q ∥ P
proof –
  have (P ∥ Q, Q ∥ P) ∈ \{(P ∥ Q, Q ∥ P) | P Q. True\} ∪ \{(Q ∥ P, P ∥ Q) | P Q. 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 ∥ Q) ∥ R ∼ P ∥ (Q ∥ R)
proof –
  have ((P ∥ Q) ∥ R, P ∥ (Q ∥ R)) ∈ \{((P ∥ Q) ∥ R, P ∥ (Q ∥ R)) | P Q R. True\} ∪ \{(P ∥ (Q ∥ R), (P ∥ Q) ∥ R) | P Q R. True\} by auto
  thus ?thesis by (coinduct rule: bisimCoinduct) (force intro: parAssocLeft parAssocRight)
qed

lemma parId:
fixes P :: ccs

shows P ∥ 0 ∼ P
proof –
  have (P ∥ 0, P) ∈ \{(P ∥ 0, P) | P. True\} ∪ \{(P, P ∥ 0) | P. True\} by simp
thus ?thesis by (coinduct rule: bisimCoinduct) (auto intro: parIdLeft parIdRight)
qed

lemma scopeFresh:
fixes x :: name
and P :: ccs
assumes $x \nparallel P$

shows $(\nu x)P \sim P$

proof
- have $(\nu x)P \sim (\nu x)P \parallel 0$ by (rule parId [THEN symmetric])
- moreover have $(\nu x)P \parallel 0 \sim 0 \parallel (\nu x)P$ by (rule parComm)
- moreover have $0 \parallel (\nu x)P \sim (\nu x)(0 \parallel P)$ by (rule scopeExt [THEN symmetric])
  auto
- moreover have $(\nu x)(0 \parallel P) \sim (\nu x)(P \parallel 0)$ by (rule resPres [OF parComm])
- moreover have $P \parallel (\nu x)0 \sim (\nu x)0 \parallel P$ by (rule parComm)
- moreover have $(\nu x)0 \parallel P \sim 0 \parallel P$ by (rule parPres [OF resNil])
- moreover have $0 \parallel P \sim P \parallel 0$ by (rule parComm)
- moreover have $P \parallel 0 \sim P$ by (rule parId)
  ultimately show $\theta$thesis by (metis transitive)

qed

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

assumes $x \nparallel P$

shows $(\nu x)(P \oplus Q) \sim P \oplus (\nu x)Q$

proof
- have $(\nu x)(P \oplus Q), P \oplus (\nu x)Q) \in 
{\{((\nu x)(P \oplus Q), (P \oplus (\nu x)Q), (\nu x)(P \parallel Q))\}}$
  by (simp)
  thus $\theta$thesis using $(x \nparallel P)$
  by (coinduct rule: bisimCoinduct)
    (auto intro: scopeExtSumLeft scopeExtSumRight reflexive scopeFresh scopeFresh [THEN symmetric])

qed

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

assumes $x \nparallel \alpha$

shows $(\nu x)(\alpha.(P)) \sim \alpha.(\nu x)P$

proof
- have $(\nu x)(\alpha.(P)), \alpha.(\nu x)P) \in 
{\{(\nu x)(\alpha.(P)), \alpha.(\nu x)P), \alpha.(\nu x)P\}}$
  by (simp)
  thus $\theta$thesis using $(x \nparallel \alpha)$
  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) | x y P.
        True} by auto
        thus \thesis
        by(coinduct rule: bisimCoinduct) (auto intro: resComm)
    qed

lemma bangUnfold:
    fixes P

    shows !P \sim P || !P
    proof
        have (\nu x)(\nu y)P, (\nu y)(\nu x)P) \in \{((\nu x)(\nu y)P), (\nu y)(\nu x)P) | x y P.
        True} 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 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: \forall x y x a z b P Q.
    x \leq y \Rightarrow
    (Q \sim <\{xb, xa\}, x xb xa} > P) \rightarrow
    (Q \sim <\{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

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 \( \bigwedge P, Q. (P, Q) ∈ X \implies P ⇝<\union weakBisimulation> Q \land (Q, P) ∈ X \)

to 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 \( \bigwedge R, S. (R, S) ∈ X \implies R ⇝<\union weakBisimulation> S \)
and \( \bigwedge R, S. (R, S) ∈ X \implies (S, R) ∈ X \)

to 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 \((P, Q) \in X\)
and \(\bigwedge P Q. (P, Q) \in X \implies P \xrightarrow{\cdot} <X> Q \land (Q, P) \in X\)

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

lemma weakBisimulationCoinduct[consumes 1, case-names cSim cSym]:
fixes \(P :: ccs\)
and \(Q :: ccs\)
and \(X :: (ccs \times ccs)\) set
assumes \((P, Q) \in X\)
and \(\bigwedge P Q. (P, Q) \in X \implies P \xrightarrow{\cdot} <X> Q\)
and \(\bigwedge P Q. (P, Q) \in X \implies (Q, P) \in X\)

shows \(P \approx Q\)
proof
  have \(X \cup \text{weakBisim} = \{(P, Q). (P, Q) \in X \lor (P, Q) \in \text{weakBisim}\}\) 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 \(P \xrightarrow{\cdot} <\text{weakBisimulation}> Q\)
and \(Q \approx P\)
using assms
by (auto simp add: intro: weakBisimulation_cases)

lemma weakBisimulationI:
fixes \(P :: ccs\)
and \(Q :: ccs\)
assumes \(P \xrightarrow{\cdot} <\text{weakBisimulation}> Q\)
and \(Q \approx P\)

shows \(P \approx Q\)
using assms
by (auto intro: weakBisimulation_intros)

lemma reflexive:
fixes \(P :: ccs\)

47
shows $P \approx P$

proof -

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

qed

lemma symmetric:
  fixes $P :: \text{ccs}$
  and $Q :: \text{ccs}$
  assumes $P \approx Q$
  shows $Q \approx P$
  using assms
  by (rule weakBisimulationE)

lemma transitive:
  fixes $P :: \text{ccs}$
  and $Q :: \text{ccs}$
  and $R :: \text{ccs}$
  assumes $P \approx Q$
  and $Q \approx R$
  shows $P \approx R$
  proof -
    from assms have $(P, R) \in \text{weakBisimulation O weakBisimulation}$ by auto
    thus $?\text{thesis}$
    proof (coinduct rule: weakBisimulationCoinduct)
      case $(cSim P R)$
      from $(P, R) \in \text{weakBisimulation O weakBisimulation}$
      obtain $Q$ where $P \approx Q$ and $Q \approx R$ by auto
      note $(P \approx Q)$
      moreover from $(Q \approx R)$ have $Q \sim^\ast <\text{weakBisimulation}> R$ by (rule weak-BisimulationE)
      moreover have weakBisimulation O weakBisimulation $\subseteq (\text{weakBisimulation O weakBisimulation}) \cup \text{weakBisimulation}$
      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 $P :: \text{ccs}$
and \( Q :: ccs \)

assumes \( P \sim Q \)

shows \( P \approx Q \)
using \( \text{assms} \)
by (coinduct rule: \text{weakBisimWeakCoinduct[where } X = \text{bisim]}\)
\begin{itemize}
  \item \( \text{auto dest: bisimE simWeakSim} \)
\end{itemize}

lemma \text{structCongWeakBisimulation}:  
\begin{itemize}
  \item \( \text{fixes } P :: ccs \)
  \item \( \text{and } Q :: ccs \)
\end{itemize}

assumes \( P \equiv_s Q \)

shows \( P \approx Q \)
using \( \text{assms} \)
by (auto intro: \text{bisimWeakBisimulation bisimStructCong})

lemma \text{strongAppend}:  
\begin{itemize}
  \item \( \text{fixes } P :: ccs \)
  \item \( \text{and } Q :: ccs \)
  \item \( \text{and } R :: ccs \)
  \item \( \text{and } \text{Rel} :: (ccs \times ccs) \text{ set} \)
  \item \( \text{and } \text{Rel}' :: (ccs \times ccs) \text{ set} \)
  \item \( \text{and } \text{Rel}'' :: (ccs \times ccs) \text{ set} \)
\end{itemize}

assumes \( PSimQ: P \rightsquigarrow^\ast \langle \text{Rel} \rangle Q \)
and \( QSimR: Q \rightsquigarrow^\ast \langle \text{Rel}' \rangle R \)
and \( \text{Trans: Rel O Rel}' \subseteq \text{Rel}'' \)

shows \( P \rightsquigarrow^\ast \langle \text{Rel}'' \rangle R \)
using \( \text{assms} \)
by (simp add: \text{weakSimulation-def simulation-def}) blast

lemma \text{weakBisimWeakUpto[case-names cSim cSym, consumes 1]}:  
\begin{itemize}
  \item \( \text{assumes } p: (P, Q) \in X \)
  \item \( \text{and } rSim: \bigwedge P Q. (P, Q) \in X \implies P \rightsquigarrow^\ast \langle (\text{weakBisimulation O O bisim}) \rangle Q \)
  \item \( \text{and } rSym: \bigwedge P Q. (P, Q) \in X \implies (Q, P) \in X \)
\end{itemize}

shows \( P \approx Q \)

proof –  
\begin{itemize}
  \item let \( ?X = \text{weakBisimulation O O weakBisimulation} \)
  \item let \( ?Y = \text{weakBisimulation O O bisim} \)
  \item from \( (P, Q) \in X \), have \( (P, Q) \in ?X \) by (blast intro: \text{Strong-Bisim.reflexive reflexive})
  \item thus \( ?\text{thesis} \)
  \item proof (coinduct rule: \text{weakBisimWeakCoinduct})
\end{itemize}
case(cSim P Q)
{
  fix P P' Q' Q
  assume P ≈ P' and (P', Q') ∈ X and Q' ~ Q
  moreover from ⟨(P', Q') ∈ X⟩ have ⟨(P', Q')⟩ ∈ Y by (blast intro: reflexive Strong-Bisim.reflexive)
  moreover from ⟨Q' ~ Q⟩ have Q' ~< ∃?Y O weakBisimulation > Q by (rule weakBisimulationE)
  moreover have ?Y O weakBisimulation ≤ ?X by (blast dest: bisimWeak-Bisimulation transitive)
  moreover { fi
    assume (P', Q') ∈ Y
    then obtain P' Q' where P ≈ P' and (P', Q') ∈ X and Q' ~ Q by auto
    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' ~< ∃?Y > 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

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)

qed
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 ≡ Q ≡ P ⇞<weakBisimulation> Q ∧ Q ⇞<weakBisimulation> P

lemma weakCongruenceE:
  fixes P :: ccs
  and Q :: ccs
  assumes P ≡ Q
  shows P ⇞<weakBisimulation> Q
  and Q ⇞<weakBisimulation> P
  using assms
  by (auto simp add: weakCongruence-def)

lemma weakCongruenceI:
  fixes P :: ccs
  and Q :: ccs
  assumes P ⇞<weakBisimulation> Q
  and Q ⇞<weakBisimulation> P
  shows P ≡ 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 \( P Q . \) Prop P Q ⇒ Prop Q P
  and \( P Q . \) Prop P Q ⇒ (F P) ⇞<weakBisimulation> (F Q)
  shows F P ≡ F Q
  using assms
  by (auto simp add: weakCongruence-def)
lemma weakCongISym2[consumes 1, case-names cSim]:
  fixes $P :: ccs$
  and $Q :: ccs$
  assumes $P \cong Q$
  and $\forall P Q. P \cong Q \implies (F P) \sim<weakBisimulation>(F Q)$
  shows $F P \cong F Q$
  using assms
  by(auto simp add: weakCongruence-def)

lemma reflexive:
  fixes $P :: ccs$
  shows $P \cong P$
  by(auto intro: weakCongruenceI Weak-Bisim.reflexive Weak-Cong-Sim.reflexive)

lemma symmetric:
  fixes $P :: ccs$
  and $Q :: ccs$
  assumes $P \cong Q$
  shows $Q \cong P$
  using assms
  by(auto simp add: weakCongruence-def)

lemma transitive:
  fixes $P :: ccs$
  and $Q :: ccs$
  and $R :: ccs$
  assumes $P \cong Q$
  and $Q \cong R$
  shows $P \cong R$
  proof -
    let $\Prop = \lambda P R. \exists Q. P \cong Q \land Q \cong R$
    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 $P \cong Q$ and $Q \cong R$
      by auto
      from $P \cong Q$ have $P \sim<weakBisimulation> Q$ by(rule weakCongruenceE)
moreover from \( Q \leadsto R \) have \( Q \leadsto<\text{weakBisimulation}> R \) by (rule weakCongruenceE)

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

lemma bisimWeakCongruence:
fixes \( P : \text{ccs} \)
and \( Q : \text{ccs} \)
assumes \( P \sim Q \)
shows \( P \sim= Q \)
using assms
proof (induct rule: weakCongISym)
case \( \text{cSym} \ P \ Q \)
thus ?case by (rule bisimE)
next
case \( \text{cSim} \ P \ Q \)
from \( P \sim Q \) have \( P \leadsto[bisim] Q \) by (rule bisimE)
hence \( P \leadsto[\text{weakBisimulation}] Q \) using bisimWeakBisimulation
by (rule-tac monotonic) auto
thus ?case by (rule simWeakSim)
qed

lemma structCongWeakCongruence:
fixes \( P : \text{ccs} \)
and \( Q : \text{ccs} \)
assumes \( P \equiv s Q \)
shows \( P \sim Q \)
using assms
by (auto intro: bisimWeakCongruence bisimStructCong)

lemma weakCongruenceWeakBisimulation:
fixes \( P : \text{ccs} \)
and \( Q : \text{ccs} \)
assumes \( P \equiv= Q \)
shows \( P \approx Q \)
proof -
let \( \exists X = \{(P, Q) \mid P \equiv Q \} \)
from assms have \((P, Q) \in \exists X\) by auto
thus \( \text{thesis} \)

proof (coinduct rule: weakBisimulationCoinduct)
  case (cSim \( P \) \( Q \))
  from \((P, Q) \in \?X\) have \( P \cong Q \) by auto
  hence \( P \bisim Q \) by (rule Weak-Cong.weakCongruenceE)
  hence \( P \bisim Q \) by (force intro: Weak-Cong-Sim.weakMonotonic)
  thus \( ?\text{case} \) by (rule weakCongSimWeakSim)
next
  case (cSym \( P \) \( Q \))
  from \((P, Q) \in \?X\) show \( ?\text{case} \) by blast dest: symmetric
qed

qed

end

theory Weak-Sim-Pres
  imports Weak-Sim
begin

lemma actPres:
  fixes \( P :: \text{ccs} \)
  and \( Q :: \text{ccs} \)
  and \( \text{Rel} :: (\text{ccs} \times \text{ccs}) \set \)
  and \( a :: \text{name} \)
  and \( \text{Rel'} :: (\text{ccs} \times \text{ccs}) \set \)

  assumes \((P, Q) \in \text{Rel}\)

  shows \( \alpha.(P) \bisim \lt \text{Rel} \gt \alpha.(Q) \)
  using assms
  by (fastforce simp add: weakSimulation-def elim: actCases intro: weakAction)

lemma sumPres:
  fixes \( P :: \text{ccs} \)
  and \( Q :: \text{ccs} \)
  and \( \text{Rel} :: (\text{ccs} \times \text{ccs}) \set \)

  assumes \( P \bisim Q \)
  and \( \text{Rel} \subseteq \text{Rel'} \)
  and \( \text{Id} \subseteq \text{Rel'} \)
  and \( C1: \forall S T U. (S, T) \in \text{Rel} \rightarrow (S \oplus U, T) \in \text{Rel'} \)

  shows \( P \oplus R \bisim \lt \text{Rel'} \gt Q \oplus R \)
  proof (induct rule: weakSimI)
    case (Sim \( P \) \( QR \))
    from \((Q \oplus R) \rightarrow \alpha \lt \text{Rel'} \gt Q \oplus R\) show \( ?\text{case} \)
  proof (induct rule: sumCases)
    case (cSum \( P \) \( Q' \))
from \( P \rightarrow^\ast <\text{Rel}> Q \) \( \rightarrow^\ast \alpha < Q' \)

obtain \( P' \) where \( P \rightarrow^\ast \alpha < P' \) and \( (P', Q') \in \text{Rel} \)

by (\text{blast dest: weakSimE})

thus ?case

proof (induct rule: weakTransCases)
  case Base
  have \( P \oplus R \rightarrow^\ast \tau < P \oplus R \) by simp
  moreover from \( (P, Q') \in \text{Rel} \) have \( (P \oplus R, Q') \in \text{Rel}' \) by (rule C1)
  ultimately show ?case by blast
  qed

next
  case Step
  from \( P \rightarrow^\ast \alpha < P' \) have \( P \oplus R \rightarrow^\ast \alpha < P' \) by (rule weakCongSum1)
  hence \( P \oplus R \rightarrow^\ast \alpha < P' \) by (simp add: weakTrans-def)
  thus ?case using \( (P', Q') \in \text{Rel} \) \( \text{Rel} \subseteq \text{Rel}' \) by blast
  qed

next
  case (cSum2 \( R' \))
  from \( R \rightarrow^\ast \alpha < R' \) have \( R \rightarrow^\ast \alpha < R' \) by (rule transitionWeakCongTransition)
  hence \( P \oplus R \rightarrow^\ast \alpha < R' \) by (rule weakCongSum2)
  hence \( P \oplus R \rightarrow^\ast \alpha < R' \) by (simp add: weakTrans-def)
  thus ?case using \( \text{Id} \subseteq \text{Rel}' \) by blast
  qed

qed

lemma parPresAux:
  fixes \( P \) :: \( \text{ccs} \) and \( Q \) :: \( \text{ccs} \) and \( R \) :: \( \text{ccs} \) and \( T \) :: \( \text{ccs} \) and \( \text{Rel} \) :: \( (\text{ccs} \times \text{ccs}) \text{ set} \) and \( \text{Rel}' \) :: \( (\text{ccs} \times \text{ccs}) \text{ set} \) and \( \text{Rel}'' \) :: \( (\text{ccs} \times \text{ccs}) \text{ set} \)

assumes \( P \rightarrow^\ast <\text{Rel}> Q \) and \( (P, Q) \in \text{Rel} \) and \( R \rightarrow^\ast <\text{Rel}> T \) and \( (R, T) \in \text{Rel}' \) and \( \text{C1}: \bigwedge (P', Q', R', T'). \lfloor (P', Q') \in \text{Rel}; (R', T') \in \text{Rel}' \rfloor \rightarrow (P' \parallel R', Q' \parallel T') \in \text{Rel}'' \)

shows \( P \parallel R \rightarrow^\ast <\text{Rel}''> Q \parallel T \)

proof (induct rule: weakSimI)
  case (Sim \( \alpha QT \))
  from \( Q \parallel T \rightarrow^\ast \alpha < QT \) show ?case
    proof (induct rule: parCases)
      case (cPar1 \( Q' \))
      from \( P \rightarrow^\ast <\text{Rel}> Q \) \( \rightarrow^\ast \alpha < Q' \) obtain \( P' \) where \( P \rightarrow^\ast \alpha < P' \) and \( (P', Q') \in \text{Rel} \)
by (rule weakSimE)
from \( P \Rightarrow \alpha \prec P' \) have \( P \parallel R \Rightarrow \alpha \prec P' \parallel R \) by (rule weakPar1)
moreover from \( \langle P', Q' \rangle \in \text{Rel} \parallel \langle R, T \rangle \in \text{Rel}' \) have \( P' \parallel R, Q' \parallel T \) \( \in \text{Rel}'' \) by (rule C1)
ultimately show \( \text{case by blast} \)
next
case \( \text{cPar2} \ T' \)
from \( R \Rightarrow <\text{Rel'>} \ T \) \( T \Rightarrow \alpha \prec T' \) obtain \( R' \) where \( R \Rightarrow \alpha \prec R' \) and \( \langle R', T' \rangle \in \text{Rel}' \)
by (rule weakSimE)
from \( R \Rightarrow \alpha \prec R' \) have \( P \parallel R \Rightarrow \alpha \prec P' \parallel R' \) by (rule weakPar2)
moreover from \( \langle P, Q \rangle \in \text{Rel} \parallel \langle R', T' \rangle \in \text{Rel}' \) have \( P \parallel R', Q \parallel T' \) \( \in \text{Rel}'' \) by (rule C1)
ultimately show \( \text{case by blast} \)

\[ \begin{align*}
&\text{next} \\
&\text{case \( \text{cComm} \ Q' \ T' \alpha \) }
\end{align*} \]
from \( \langle P \Rightarrow <\text{Rel}> \ Q \rangle \langle Q \Rightarrow <\text{Rel}> \ Q' \rangle \) obtain \( P' \) where \( P \Rightarrow \alpha \prec P' \) and \( \langle P', Q' \rangle \in \text{Rel} \)
by (rule weakSimE)
from \( \langle P \Rightarrow <\text{Rel}> \ Q \rangle \langle Q \Rightarrow <\text{Rel}> \ Q' \rangle \) obtain \( R' \) where \( R \Rightarrow \alpha \prec R' \) and \( \langle R', T' \rangle \in \text{Rel}' \)
by (rule weakSimE)
from \( \langle P \Rightarrow \alpha \prec P' \rangle \langle R \Rightarrow \alpha \prec P' \parallel R' \rangle \) have \( P \parallel R \Rightarrow \tau \prec P' \parallel R' \)
by (auto intro: weakCongSync simp add: weakTrans-def)

\[ \begin{align*}
&\text{hence} \ P \parallel R \Rightarrow \tau \prec P' \parallel R' \text{by (simp add: weakTrans-def)}
\end{align*} \]
moreover from \( \langle P', Q' \rangle \in \text{Rel} \parallel \langle R', T' \rangle \in \text{Rel}' \) have \( P' \parallel R', Q' \parallel T' \) \( \in \text{Rel}''' \) by (rule C1)
ultimately show \( \text{case by blast} \)

\[ \text{qedqed} \]

\[ \text{lemma \ parPres:} \]
\[ \text{fixes \( P \) :: ccs} \]
\[ \text{and \( Q \) :: ccs} \]
\[ \text{and \( R \) :: ccs} \]
\[ \text{and \( \text{Rel} \) :: \( \text{ccs} \times \text{ccs} \) set} \]
\[ \text{and \( \text{Rel}' \) :: \( \text{ccs} \times \text{ccs} \) set} \]
\[ \text{assumes \( P \Rightarrow <\text{Rel}> \ Q \)} \]
\[ \text{and \( \langle P, Q \rangle \in \text{Rel} \)} \]
\[ \text{and \( C1: \forall S T U. \langle S, T \rangle \in \text{Rel} \Rightarrow (S \parallel U, T \parallel U) \in \text{Rel}' \)} \]
\[ \text{shows \( P \parallel R \Rightarrow <\text{Rel}> \ Q \parallel R} \]
\[ \text{using \ assms} \]
\[ \text{by (rule-tac parPresAux[where Rel'=Id and Rel''=Rel']) (auto intro: reflexive)} \]

\[ \text{lemma \ resPres:} \]
\[ \text{fixes \( P \) :: ccs} \]
\[ \text{and \( \text{Rel} \) :: \( \text{ccs} \times \text{ccs} \) set} \]
and \( Q :: ccs \)
and \( x :: \text{name} \)

assumes \( P \leadsto^* <\text{Rel}> Q \)
and \( \bigwedge RSy. (R, S) \in \text{Rel} \implies (\nu y)R, (\nu y)S) \in \text{Rel}' \)

shows \( (\nu x)P \leadsto^* <\text{Rel}'> (\nu x)Q \)

using assms
by \( \text{fastforce simp add: weakSimulation-def elim: resCases intro: weakRes} \)

lemma \( \text{bangPres} \):

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

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

shows \( !P \leadsto^* <\text{Rel}'> !Q \)

proof \( \text{(induct rule: weakSimI)} \)

\begin{cases} \text{case(Sim } \alpha \text{ } Q') \\
\text{fix } Pa \ \alpha \ Q' \\
\text{assume } !Q \leadsto^* \alpha < Q' \text{ and } (Pa, !Q) \in \text{bangRel Rel} \\
\text{hence } \exists P'. Pa \implies \alpha < P' \land (P', Q') \in \text{Rel}' \\
\text{proof(nominal-induct arbitrary: Pa rule: bangInduct)} \\
\text{case(cPar1 } \alpha \text{ } Q') \\
\text{from } \langle (Pa, Q \parallel !Q) \in \text{bangRel Reb} \rangle \\
\text{show } ?\text{case} \\
\text{proof(induct rule: BRParCases)} \\
\text{case(BRPar } P \text{ } R) \\
\text{from } \langle (P, Q) \in \text{Rel} \rangle \text{ have } P \leadsto^* <\text{Rel}> Q \text{ by } \text{rule } C1 \\
\text{with } \langle Q \leadsto^* \alpha < Q' \rangle \text{ obtain } P' \text{ where } P \implies \alpha < P' \land (P', Q') \in \text{Rel} \text{ by } \text{blast dest: weakSimE} \\
\text{from } \langle P \implies \alpha < P' \rangle \text{ have } P \parallel R \implies \alpha < P' \parallel R \text{ by } \text{rule } \text{weakPar1} \\
\text{moreover from } \langle (P', Q') \in \text{Rel} ; (R, !Q) \in \text{bangRel Reb} \rangle \text{ C2 have } (P' \parallel R, Q' \parallel !Q) \in \text{Rel}' \\
\text{by } \text{blast intro: Par} \\
\text{ultimately show } ?\text{case by } \text{blast} \\
\end{cases} \text{qed} \\
\text{next} \\
\text{case(cPar2 } \alpha \text{ } Q') \\
\text{from } \langle (Pa, Q \parallel !Q) \in \text{bangRel Reb} \rangle \\
\text{show } ?\text{case} \\
\text{proof(induct rule: BRParCases)}
\[ \text{case}(\text{BRPar } P \ R) \]
\[ \text{from } (R, !Q) \in \text{bangRel Rel} \ \text{obtain } R' \text{ where } R \Rightarrow \alpha \prec R' \text{ and } (R', Q') \in \text{Rel' using } c\text{Par2} \]
\[ \ \text{by blast} \]
\[ \text{from } R \Rightarrow \alpha \prec R' \text{ have } P \parallel R \Rightarrow \alpha \prec P \parallel R' \text{ by (rule weakPar2)} \]
\[ \text{moreover from } (P, Q) \in \text{Rel} \ (R', Q') \in \text{Rel'} \text{ have } (P \parallel R', Q \parallel Q') \in \text{Rel'} \text{ by (rule Par)} \]
\[ \ultimately \text{show } ?\text{case by blast} \]
\text{qed} \\
\text{next} \\
\text{case}(c\text{Comm } a \ Q' \ Q'' \ Pa) \\
\text{from } (Pa, Q \parallel !Q) \in \text{bangRel Rel} \]
\text{show } ?\text{case} \\
\text{proof (induct rule: BRParCases)} \\
\text{case}(\text{BRPar } P \ R) \\
\text{from } (P, Q) \in \text{Rel} \text{ have } P \Rightarrow \alpha < \text{Rel} > Q \text{ by (rule C1)} \\
\text{with } \alpha \rightarrow a < Q' \text{ obtain } P' \text{ where } P \Rightarrow a \prec P' \text{ and } (P', Q') \in \text{Rel} \text{ by (blast dest: weakSimE)} \\
\text{from } (R, !Q) \in \text{bangRel Rel} \text{ obtain } R' \text{ where } R \Rightarrow \text{coAction } a \prec R' \\
\text{and } (R', Q') \in \text{Rel'} \text{ using } c\text{Comm} \]
\[ \ \text{by blast} \]
\[ \text{from } P \Rightarrow \alpha < P' \parallel R \Rightarrow \text{coAction } a \prec R' \ (a \neq \tau) \text{ have } P \parallel R \Rightarrow \tau \prec P' \parallel R' \]
\[ \ \text{by (auto intro: weakCongSync simp add: weakTrans-def)} \]
\[ \text{moreover from } (P', Q') \in \text{Rel} \ (R', Q'') \in \text{Rel'} \text{ have } (P' \parallel R', Q' \parallel Q'') \in \text{Rel'} \text{ by (rule Par)} \]
\[ \ultimately \text{show } ?\text{case by blast} \]
\text{qed} \\
\text{next} \\
\text{case}(c\text{Bang } \alpha \ Q' \ Pa) \\
\text{from } (Pa, !Q) \in \text{bangRel Rel} \]
\text{show } ?\text{case} \\
\text{proof (induct rule: BRBangCases)} \\
\text{case}(\text{BRBang } P) \\
\text{from } (P, Q) \in \text{Rel} \text{ have } (!P, !Q) \in \text{bangRel Rel} \text{ by (rule bangRel.BRBang)} \\
\text{with } (P, Q) \in \text{Rel} \text{ have } (P \parallel !P, Q \parallel !Q) \in \text{bangRel Rel} \text{ by (rule bangRel.BRBang)} \\
\text{then obtain } P' \text{ where } P \parallel !P \Rightarrow \alpha < P' \text{ and } (P', Q') \in \text{Rel'} \text{ using } c\text{Bang} \]
\[ \ \text{by blast} \]
\[ \text{from } !P \Rightarrow \alpha < P' \]
\text{show } ?\text{case} \\
\text{proof (induct rule: weakTransCases)} \\
\text{case Base} \\
\text{have } !P \Rightarrow \tau < !P \text{ by simp} \\
\text{moreover from } (P', Q') \in \text{Rel'} \ (P \parallel !P = P') \text{ have } (!P, Q') \in \text{Rel'} \text{ by (blast intro: C3)} \]
\[ \ultimately \text{show } ?\text{case by blast} \]
\text{next} \\

58
case Step
from $\langle P || !P \Longrightarrow \alpha \prec P' \rangle$ have $!P \Longrightarrow \alpha \prec P'$ by (rule weakCongRepl)
hence $!P \Longrightarrow \alpha \prec P'$ by (simp add: weakTrans-def)
with $\langle P', Q' \rangle \in \text{Rel}'$ show ?case by blast
qed
qed
qed

moreover from $\langle (P, Q) \in \text{Rel} \rangle$ have $(!P, !Q) \in \text{bangRel} \text{Rel}$ by (rule BRBang)
ultimately show ?case using $!Q \Longrightarrow \alpha \prec Q'$ by blast
qed

end

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

lemma actPres:
  fixes $P :: \text{ccs}$
  and $Q :: \text{ccs}$
  and $\alpha :: \text{act}$

  assumes $P \approx Q$

  shows $\alpha.(P) \approx \alpha.(Q)$

proof -
  let $\mathcal{X} = \{ (\alpha.(P), \alpha.(Q)) \mid P \ Q. \ P \approx Q \}$
  from assms have $(\alpha.(P), \alpha.(Q)) \in \mathcal{X}$ by auto
  thus ?thesis
    by (coinduct rule: weakBisimulationCoinduct) (auto dest: weakBisimulationE
        intro: actPres)
qed

lemma parPres:
  fixes $P :: \text{ccs}$
  and $Q :: \text{ccs}$
  and $R :: \text{ccs}$

  assumes $P \approx Q$

  shows $P || R \approx Q || R$

proof -
  let $\mathcal{X} = \{ (P || R \ Q \ || R) \mid P \ Q \ R. \ P \approx Q \}$
  from assms have $(P || R \ Q \ || R) \in \mathcal{X}$ by blast
  thus ?thesis
    by (coinduct rule: weakBisimulationCoinduct, auto)
lemma resPres:
  fixes \( P :: \text{ccs} \) and \( Q :: \text{ccs} \) and \( x :: \text{name} \)
  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 \( ?\text{thesis} \)
  by (coinduct rule: weakBisimulationCoinduct) (auto intro: resPres dest: weakBisimulationE)
qed

lemma bangPres:
  fixes \( P :: \text{ccs} \) and \( Q :: \text{ccs} \)
  assumes \( P \approx Q \)
  shows \(!P \approx !Q \)

proof –
  let \( ?X = \text{bangRel weakBisimulation} \)
  let \( ?Y = \text{weakBisimulation O ?X O bisim} \)
  { 
    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-Trans, 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 \( ?\text{thesis} \)
  proof (coinduct rule: weakBisimulationWeakUpto)
    case (cSim \( P \ Q \) )
    from \( (P, Q) \in \text{bangRel weakBisimulation} \) show \( ?\text{case} \)
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 reflexive bisimWeakBisimulation)
  }
  ultimately show ?case by (rule bangPres)
next
case (BRPar R T P Q)
  from ⟨R ≈ T⟩ have ⟨R ⇝ ⟨!R⟩, T⟩ ∈ ?Y by (rule weakBisimulationE)
  moreover note ⟨R ≈ T⟩ ⟨P ⇝ ⟨!P⟩, 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 \( P \rightarrow <\text{Rel}> Q \)
and \( \text{Rel} \subseteq \text{Rel}' \)
and \( \text{Id} \subseteq \text{Rel}' \)

shows \( P \parallel R \rightarrow <\text{Rel}'> Q \parallel R \)
using assms
by (force simp add: weakCongSimulation-def elim: sumCases intro: weakCongSum1 weakCongSum2 transitionWeakCongTransition)

**lemma** \( \text{parPres} \):

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

assumes \( P \rightarrow <\text{Rel}> Q \)
and \( (P, Q) \in \text{Rel} \)
and \( C1: \coprod S T U. (S, T) \in \text{Rel} \implies (S \parallel U, T \parallel U) \in \text{Rel}' \)

shows \( P \parallel R \rightarrow <\text{Rel}'> Q \parallel R \)

**proof** (induct rule: weakSimI)

**case** \( \langle Q \parallel R \rightarrow<\alpha \prec QR \rangle \)

**show** ?case

**proof** (induct rule: \( \text{parCases} \))

**case** \( \langle \text{cPar1 } Q' \rangle \)

from \( \langle P \rightarrow<\text{Rel}> Q \rangle \langle Q \rightarrow<\alpha \prec QR' \rangle \) obtain \( P' \) where \( P \rightarrow \alpha \prec P' \) and \( (P', Q') \in \text{Rel} \)

by (rule weakSimE)

from \( \langle P \rightarrow \alpha \prec P' \parallel R \rangle \) have \( P \parallel R \rightarrow \alpha \prec P' \parallel R \) by (rule weakCongPar1)

moreover from \( \langle P', Q' \rangle \in \text{Rel} \) have \( (P' \parallel R', Q' \parallel R) \in \text{Rel}' \) by (rule \( C1 \))

ultimately show ?case by blast

next

**case** \( \langle \text{cPar2 } R' \rangle \)

from \( \langle R \rightarrow \alpha \prec R' \rangle \) have \( R \rightarrow \alpha \prec R' \) by (rule transitionWeakCongTransition)

hence \( P \parallel R \rightarrow \alpha \prec P \parallel R' \) by (rule weakCongPar2)

moreover from \( \langle P, Q \rangle \in \text{Rel} \) have \( (P \parallel R', Q \parallel R') \in \text{Rel}' \) by (rule \( C1 \))

ultimately show ?case by blast

next

**case** \( \langle \text{cComm } Q' R' \rangle \)

from \( \langle P \rightarrow<\text{Rel}> Q \rangle \langle Q \rightarrow<\alpha \prec Q' \rangle \) obtain \( P' \) where \( P \rightarrow \alpha \prec P' \) and \( (P', Q') \in \text{Rel} \)

by (rule weakSimE)

from \( \langle R \rightarrow (\text{coAction } \alpha) \prec R' \rangle \) have \( R \rightarrow (\text{coAction } \alpha) \prec R' \) by (rule transitionWeakCongTransition)

with \( P \rightarrow \alpha \prec P' \parallel R \rightarrow \tau \prec P' \parallel R' \) using \( \alpha \neq \tau \)

by (rule weakCongSync)

moreover from \( \langle P', Q' \rangle \in \text{Rel} \) have \( (P' \parallel R', Q' \parallel R') \in \text{Rel}' \) by (rule \( C1 \))

62
ultimately show ?case by blast
qed
qed

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

  assumes P ⇝<Rel> Q
  and ∏ R S y. (R, S) ∈ Rel ⇒ (νy)R, (νy)S ∈ Rel'

  shows (νx)P ⇝<Rel'> (νx)Q
using assms
by (fastforce simp add: weakCongSimulation-def elim: resCases intro: weakCongRes)

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

  assumes (P, Q) ∈ Rel
  and C1: ∏ R S. (R, S) ∈ Rel ⇒ R ⇝<Rel'> S
  and C2: Rel ⊆ Rel'

  shows !P ⇝<bangRel Rel'> !Q
proof (induct rule: weakSimI)
  case (Sim α Q')
  {
    fix Pa α Q'
    assume !Q' → α < Q' and (Pa, !Q) ∈ bangRel Rel
    hence ∃ P', Pa → α < P' ∧ (P', Q') ∈ bangRel Rel'
    proof (nominal-induct arbitrary: Pa rule: bangInduct)
      case (cPar1 α Q')
      from ⟨(Pa, Q || !Q) ∈ bangRel Rel⟩
      show ?case
      proof (induct rule: BRParCases)
        case (BRPar P R)
        from ⟨(P, Q) ∈ Rel ⟩ have P ⇝<Rel'> Q by (rule C1)
        with (Q → α < Q') obtain P' where P → α < P' and (P', Q') ∈ Rel'
        by (blast dest: weakSimE)
        from P → α < P' have P || R → α < P' || R by (rule weakCongPar1)
        moreover from ⟨(R, !Q) ∈ bangRel Rel ⟩ C2 have (R, !Q) ∈ bangRel Rel'
        by induct (auto intro: bangRel.BRPar bangRel.BRBang)
        with ⟨(P', Q') ∈ Rel' ⟩ have (P' || R, Q' || !Q) ∈ bangRel Rel'
        by (rule bangRel.BRPar)
  }
ultimately show \( \text{case by blast} \)

\text{qed}

\text{next}

\text{case} (cPar2 \ P \ Q')

\text{from} \ (P, Q || !Q) \in \text{bangRel Rel}

\text{show} \ ?\text{case}

\text{proof (induct rule: BRParCases)}

\text{case} (BRPar P R)

\text{from} \ (R, !Q) \in \text{bangRel Rel}; \text{obtain} \ R' \text{ where } R \Longrightarrow \alpha < R' \text{ and } (R', Q') \in \text{bangRel Rel'} \text{ using} \ cPar2

\text{by blast}

\text{from} \ (R \Longrightarrow \alpha < R') \text{ have } P \parallel R \Longrightarrow \alpha < P \parallel R' \text{ by (rule weakCongPar2)}

\text{moreover from} \ (P, Q) \in \text{Rel} \ (R', Q') \in \text{bangRel Rel'} \text{ have } (P \parallel R', Q || Q') \in \text{bangRel Rel'}

\text{by (blast intro: bangRel.BRPar)}

\text{ultimately show} \ ?\text{case by blast}

\text{qed}

\text{next}

\text{case} (cComm \ a \ Q' Q'' Pa)

\text{from} \ (P, Q || !Q) \in \text{bangRel Rel}

\text{show} \ ?\text{case}

\text{proof (induct rule: BRParCases)}

\text{case} (BRPar P R)

\text{from} \ (P, Q) \in \text{Rel}; \text{have} \ P \parallel <\text{Rel}'> \ Q \text{ by (rule C1)}

\text{with} \ (Q \mapsto \alpha < Q') \text{ obtain} \ P' \text{ where } (P \mapsto \alpha < P' \text{ and } (P', Q') \in \text{Rel'}

\text{by (blast dest: weakSimE)}

\text{from} \ (R, !Q) \in \text{bangRel Rel}; \text{obtain} \ R' \text{ where } R \Longrightarrow (\text{coAction} \ a) < R'

\text{and } (R', Q'') \in \text{bangRel Rel'} \text{ using} \ cComm

\text{by blast}

\text{from} \ (P \mapsto \alpha < P') \mapsto (\text{coAction} \ a) < R' \langle \alpha \neq \tau \rangle \text{ have } P \parallel R \Longrightarrow \tau < P' \parallel R' \text{ by (rule weakCongSync)}

\text{moreover from} \ (P', Q') \in \text{Rel'} \ (R', Q'') \in \text{bangRel Rel'} \text{ have } (P' \parallel R', Q' \parallel Q'') \in \text{bangRel Rel'}

\text{by (rule bangRel.BRPar)}

\text{ultimately show} \ ?\text{case by blast}

\text{qed}

\text{next}

\text{case} (cBang \ P \ Q')

\text{from} \ (P, !Q) \in \text{bangRel Rel}

\text{show} \ ?\text{case}

\text{proof (induct rule: BRBangCases)}

\text{case} (BRBang P)

\text{from} \ (P, Q) \in \text{Rel}; \text{have} \ (!P, !Q) \in \text{bangRel Rel} \text{ by (rule bangRel.BRBang)}

\text{with} \ (P, Q) \in \text{Rel}; \text{have} \ (P \parallel !P, Q \parallel !Q) \in \text{bangRel Rel} \text{ by (rule bangRel.BRBang)}

\text{then obtain} \ P' \text{ where } P \parallel !P \Longrightarrow \alpha < P' \text{ and } (P', Q') \in \text{bangRel Rel'}

\text{using} \ cBang

\text{by blast}

\text{from} \ (P \parallel !P \Longrightarrow \alpha < P') \text{ have } !P \Longrightarrow \alpha < P' \text{ by (rule weakCongRepl)}

\text{64}
thus \begin{case}
\langle P', Q' \rangle \in \text{bangRel} \ \text{Rel}' \end{case}
\text{by blast}\n\\
\text{qed}\n\\
\text{qed}\n\\
\}

moreover from \begin{case}
\langle P, Q \rangle \in \text{Rel}\end{case}
\text{have} \begin{case}
\langle P, Q \rangle \in \text{bangRel} \ \text{Rel}' \end{case}
\text{by (rule BRBang)}\n\\
ultimately show \begin{case}
\langle Q \mapsto \alpha \prec Q' \rangle \end{case}
\text{by blast}\n\\
\text{qed}\n\\
end

theory \text{Weak-Cong-Pres}\\
\text{imports} \text{Weak-Cong} \text{ Weak-Bisim-Pres} \text{ Weak-Cong-Sim-Pres}\\
begin

\text{lemma actPres:}\n\text{fixes} \ P :: \text{ccs}\\
\text{and} \ Q :: \text{ccs}\\
\text{and} \ \alpha :: \text{act}\\
\text{assumes} \ P \approx Q\\
\text{shows} \ \alpha.\langle P \rangle \approx \alpha.\langle Q \rangle\\
\text{using assms}\n\text{proof (induct rule: weakCongISym2)}\\
\text{case (cSim} \ P \ Q)\\
\text{from} \ \langle P \approx Q \rangle \ \text{have} \ P \approx Q \ \text{by (rule weakCongruenceWeakBisimulation)}\\
\text{thus} \ \begin{case}\ \alpha.\langle Q \rangle \mapsto \alpha.\langle Q' \rangle \end{case} \text{by (rule actPres)}\n\\
\text{qed}\n\\
\text{lemma sumPres:}\n\text{fixes} \ P :: \text{ccs}\\
\text{and} \ Q :: \text{ccs}\\
\text{and} \ R :: \text{ccs}\\
\text{assumes} \ P \approx Q\\
\text{shows} \ P \oplus R \approx Q \oplus R\\
\text{using assms}\n\text{proof (induct rule: weakCongISym2)}\\
\text{case (cSim} \ P \ Q)\\
\text{from} \ \langle P \approx Q \rangle \ \text{have} \ P \oplus Q \prec Q \oplus R \ \text{by (rule weakCongruenceE)}\\
\text{thus} \ \begin{case}\ \text{Weak-Bisim.reflexive} \end{case} \text{by (rule-tac sumPres) auto}\n\\
\text{qed}\n\\
\text{lemma parPres:}\n\text{fixes} \ P :: \text{ccs}
and $Q :: \text{ccs}$
and $R :: \text{ccs}$

assumes $P \equiv Q$

shows $P \parallel R \equiv Q \parallel R$
using assms
proof (induct rule: weakCongISym2)
  case (cSim $P$ $Q$)
  from $\langle P \equiv Q \rangle$ have $P \bisim Q$ by (rule weakCongruenceE)
  moreover from $\langle P \equiv Q \rangle$ have $P \approx Q$ by (rule weakCongruenceWeakBisimulation)
  ultimately show ?case using Weak-Bisim-Pres.parPres
    by (rule parPres)
qed

lemma resPres:
fixes $P :: \text{ccs}$
and $Q :: \text{ccs}$
and $x :: \text{name}$

assumes $P \equiv Q$

shows $(\nu x)P \equiv (\nu x)Q$
using assms
proof (induct rule: weakCongISym2)
  case (cSim $P$ $Q$)
  from $\langle P \equiv Q \rangle$ have $P \bisim Q$ 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 $\langle P \approx Q \rangle$ show $!P \approx !Q$ by (rule Weak-Bisim-Pres.bangPres)
next
  case (BRPar $R$ $T$ $P$ $Q$)
  from $\langle R \equiv T \rangle$ have $R \parallel P \equiv T \parallel T$ by (rule Weak-Bisim-Pres.parPres)
  moreover from $\langle P \approx Q \rangle$ have $P \parallel T \equiv Q \parallel T$ by (rule Weak-Bisim-Pres.parPres)
  hence $T \parallel P \equiv T \parallel Q$ by (metis bisimWeakBisimulation Weak-Bisim.transitive parComm)
  ultimately show $R \parallel P \equiv T \parallel Q$ by (rule Weak-Bisim.transitive)
qed

66
lemma \textit{bangPres}:
\begin{itemize}
\item fixes $P :: ccs$
\item and $Q :: ccs$
\end{itemize}

assumes $P \simeq Q$

shows $!P \simeq !Q$

using \textit{assms}

proof
\begin{itemize}
\item case ($cSim P Q$)
\item let $?X = \{(P, Q) \mid P \sim Q, P \simeq Q\}$
\item from $(P \simeq Q)$ have $(P, Q) \in ?X$ by auto
\item moreover have $\bigwedge P Q. (P, Q) \in ?X \Rightarrow P \rightsquigarrow Q$ by (auto dest: weakCongruenceE)
\item moreover have $?X \subseteq weakBisimulation$ by (auto intro: weakCongruenceWeakBisimulation)
\item ultimately have $!P \rightsquigarrow !Q$ by (rule bangPres)
\item thus $?case$ using $weakBisimBangRel$ by (rule Weak-Cong-Sim.weakMonotonic)
\end{itemize}

qed

end

References