Analysing and Comparing Encodability Criteria for Process Calculi (Technical Report)

Kirstin Peters*
TU Dresden, Germany

Rob van Glabbeek
NICTA†, Sydney, Australia
Computer Science and Engineering, UNSW, Sydney, Australia

August 05, 2015

Abstract

Encodings or the proof of their absence are the main way to compare process calculi. To analyse the quality of encodings and to rule out trivial or meaningless encodings, they are augmented with quality criteria. There exists a bunch of different criteria and different variants of criteria in order to reason in different settings. This leads to incomparable results. Moreover it is not always clear whether the criteria used to obtain a result in a particular setting do indeed fit to this setting. We show how to formally reason about and compare encodability criteria by mapping them on requirements on a relation between source and target terms that is induced by the encoding function. In particular we analyse the common criteria *full abstraction*, *operational correspondence*, *divergence reflection*, *success sensitiveness*, and *respect of barbs*; e.g. we analyse the exact nature of the simulation relation (coupled simulation versus bisimulation) that is induced by different variants of operational correspondence. This way we reduce the problem of analysing or comparing encodability criteria to the better understood problem of comparing relations on processes.

In the following we present the Isabelle implementation of the underlying theory as well as all proofs of the results presented in the paper *Analysing and Comparing Encodability Criteria* as submitted to EXPRESS/SOS'15.

*Supported by funding of the Excellence Initiative by the German Federal and State Governments (Institutional Strategy, measure 'support the best').

†NICTA is funded by the Australian Government through the Department of Communications and the Australian Research Council through the ICT Centre of Excellence Program.

Contents

theory *Relations*
  imports *Main HOL−Library.LaTeXsugar HOL−Library.OptionalSugar*
begin

1 Relations

1.1 Basic Conditions

We recall the standard definitions for reflexivity, symmetry, transitivity, preoders, equivalence, and inverse relations.

abbreviation *preorder Rel* ≡ *preorder-on UNIV Rel*
abbreviation *equivalence Rel* ≡ *equiv UNIV Rel*

A symmetric preorder is an equivalence.

lemma *symm-preorder-is-equivalence*:
  fixes *Rel* :: $('a \times 'a)$ *set*
  assumes *preorder Rel*
    and *sym Rel*
  shows *equivalence Rel*
    ⟨*proof*⟩

The symmetric closure of a relation is the union of this relation and its inverse.

definition *symcl* :: $('a \times 'a)$ *set* ⇒ $('a \times 'a)$ *set* **where**
  symcl Rel = *Rel* ∪ Rel^{-1}

For all (a, b) in R, the symmetric closure of R contains (a, b) as well as (b, a).

lemma *elem-of-symcl*:
  fixes *Rel* :: $('a \times 'a)$ *set*
   and *a b* :: $'a$
  assumes *elem*: $(a, b) \in$ *Rel*
  shows $(a, b) \in$ *symcl Rel*
   and $(b, a) \in$ *symcl Rel*
   ⟨*proof*⟩

The symmetric closure of a relation is symmetric.

lemma *sym-symcl*:
  fixes *Rel* :: $('a \times 'a)$ *set*
  shows *sym* (*symcl Rel*)
   ⟨*proof*⟩

The reflexive and symmetric closure of a relation is equal to its symmetric and reflexive closure.

lemma *refl-symm-closure-is-symm-refl-closure*:
  fixes *Rel* :: $('a \times 'a)$ *set*
  shows *symcl* ($Rel^{=}$) = (*symcl Rel*)$^{=}$
   ⟨*proof*⟩

The symmetric closure of a reflexive relation is reflexive.

lemma *refl-symcl-of-refl-rel*:
  fixes *Rel* :: $('a \times 'a)$ *set*
   and *A* :: $'a$ *set*
  assumes *refl-on A Rel*
  shows *refl-on A* (*symcl Rel*)
    ⟨*proof*⟩

Accordingly, the reflexive, symmetric, and transitive closure of a relation is equal to its symmetric, reflexive, and transitive closure.

lemma *refl-symm-trans-closure-is-symm-refl-trans-closure*:

fixes $Rel :: ('a \times 'a)$ *set*
shows $(symcl\ (Rel^{=}))^{+} = (symcl\ Rel)^{*}$
⟨*proof*⟩

The reflexive closure of a symmetric relation is symmetric.

lemma *sym-reflcl-of-symm-rel*:
fixes $Rel :: ('a \times 'a)$ *set*
assumes *sym Rel*
shows *sym* $(Rel^{=})$
⟨*proof*⟩

The reflexive closure of a reflexive relation is the relation itself.

lemma *reflcl-of-refl-rel*:
fixes $Rel :: ('a \times 'a)$ *set*
assumes *refl Rel*
shows $Rel^{=} = Rel$
⟨*proof*⟩

The symmetric closure of a symmetric relation is the relation itself.

lemma *symm-closure-of-symm-rel*:
fixes $Rel :: ('a \times 'a)$ *set*
assumes *sym Rel*
shows *symcl Rel* = *Rel*
⟨*proof*⟩

The reflexive and transitive closure of a preorder Rel is Rel.

lemma *rtrancl-of-preorder*:
fixes $Rel :: ('a \times 'a)$ *set*
assumes *preorder Rel*
shows $Rel^{*} = Rel$
⟨*proof*⟩

The reflexive and transitive closure of a relation is a subset of its reflexive, symmetric, and transtive closure.

lemma *refl-trans-closure-subset-of-refl-symm-trans-closure*:
fixes $Rel :: ('a \times 'a)$ *set*
shows $Rel^{*} \subseteq (symcl\ (Rel^{=}))^{+}$
⟨*proof*⟩

If a preorder Rel satisfies the following two conditions, then its symmetric closure is transitive: (1) If (a, b) and (c, b) in Rel but not (a, c) in Rel, then (b, a) in Rel or (b, c) in Rel. (2) If (a, b) and (a, c) in Rel but not (b, c) in Rel, then (b, a) in Rel or (c, a) in Rel.

lemma *symm-closure-of-preorder-is-trans*:
fixes $Rel :: ('a \times 'a)$ *set*
assumes *condA*: $\forall a\ b\ c.\ (a, b) \in Rel \land (c, b) \in Rel \land (a, c) \notin Rel$
$\longrightarrow (b, a) \in Rel \lor (b, c) \in Rel$
and *condB*: $\forall a\ b\ c.\ (a, b) \in Rel \land (a, c) \in Rel \land (b, c) \notin Rel$
$\longrightarrow (b, a) \in Rel \lor (c, a) \in Rel$
and *reflR*: *refl Rel*
and *tranR*: *trans Rel*
shows *trans* (*symcl Rel*)
⟨*proof*⟩

1.2 Preservation, Reflection, and Respection of Predicates

A relation R preserves some predicate P if P(a) implies P(b) for all (a, b) in R.

abbreviation *rel-preserves-pred* :: $('a \times 'a)\ set \Rightarrow ('a \Rightarrow bool) \Rightarrow bool$ **where**
rel-preserves-pred Rel Pred $\equiv \forall a\ b.\ (a, b) \in Rel \land Pred\ a \longrightarrow Pred\ b$

abbreviation *rel-preserves-binary-pred* :: $('a \times 'a)$ *set* $\Rightarrow$ $('a \Rightarrow 'b \Rightarrow$ *bool*$)$ $\Rightarrow$ *bool* **where**
rel-preserves-binary-pred Rel Pred $\equiv \forall\, a\ b\ x.\ (a,\ b) \in Rel \wedge Pred\ a\ x \longrightarrow Pred\ b\ x$

A relation R reflects some predicate P if P(b) implies P(a) for all (a, b) in R.

abbreviation *rel-reflects-pred* :: $('a \times 'a)$ *set* $\Rightarrow$ $('a \Rightarrow$ *bool*$)$ $\Rightarrow$ *bool* **where**
rel-reflects-pred Rel Pred $\equiv \forall\, a\ b.\ (a,\ b) \in Rel \wedge Pred\ b \longrightarrow Pred\ a$

abbreviation *rel-reflects-binary-pred* :: $('a \times 'a)$ *set* $\Rightarrow$ $('a \Rightarrow 'b \Rightarrow$ *bool*$)$ $\Rightarrow$ *bool* **where**
rel-reflects-binary-pred Rel Pred $\equiv \forall\, a\ b\ x.\ (a,\ b) \in Rel \wedge Pred\ b\ x \longrightarrow Pred\ a\ x$

A relation respects a predicate if it preserves and reflects it.

abbreviation *rel-respects-pred* :: $('a \times 'a)$ *set* $\Rightarrow$ $('a \Rightarrow$ *bool*$)$ $\Rightarrow$ *bool* **where**
rel-respects-pred Rel Pred $\equiv$ *rel-preserves-pred Rel Pred* $\wedge$ *rel-reflects-pred Rel Pred*

abbreviation *rel-respects-binary-pred* :: $('a \times 'a)$ *set* $\Rightarrow$ $('a \Rightarrow 'b \Rightarrow$ *bool*$)$ $\Rightarrow$ *bool* **where**
rel-respects-binary-pred Rel Pred $\equiv$
rel-preserves-binary-pred Rel Pred $\wedge$ *rel-reflects-binary-pred Rel Pred*

For symmetric relations preservation, reflection, and respection of predicates means the same.

lemma *symm-relation-impl-preservation-equals-reflection*:
fixes *Rel* :: $('a \times 'a)$ *set*
and *Pred* :: $'a \Rightarrow$ *bool*
assumes *symm*: *sym Rel*
shows *rel-preserves-pred Rel Pred* = *rel-reflects-pred Rel Pred*
and *rel-preserves-pred Rel Pred* = *rel-respects-pred Rel Pred*
and *rel-reflects-pred Rel Pred* = *rel-respects-pred Rel Pred*
⟨*proof*⟩

lemma *symm-relation-impl-preservation-equals-reflection-of-binary-predicates*:
fixes *Rel* :: $('a \times 'a)$ *set*
and *Pred* :: $'a \Rightarrow 'b \Rightarrow$ *bool*
assumes *symm*: *sym Rel*
shows *rel-preserves-binary-pred Rel Pred* = *rel-reflects-binary-pred Rel Pred*
and *rel-preserves-binary-pred Rel Pred* = *rel-respects-binary-pred Rel Pred*
and *rel-reflects-binary-pred Rel Pred* = *rel-respects-binary-pred Rel Pred*
⟨*proof*⟩

If a relation preserves a predicate then so does its reflexive or/and transitive closure.

lemma *preservation-and-closures*:
fixes *Rel* :: $('a \times 'a)$ *set*
and *Pred* :: $'a \Rightarrow$ *bool*
assumes *preservation*: *rel-preserves-pred Rel Pred*
shows *rel-preserves-pred* $(Rel^=)$ *Pred*
and *rel-preserves-pred* (Rel^+) *Pred*
and *rel-preserves-pred* (Rel^*) *Pred*
⟨*proof*⟩

lemma *preservation-of-binary-predicates-and-closures*:
fixes *Rel* :: $('a \times 'a)$ *set*
and *Pred* :: $'a \Rightarrow 'b \Rightarrow$ *bool*
assumes *preservation*: *rel-preserves-binary-pred Rel Pred*
shows *rel-preserves-binary-pred* $(Rel^=)$ *Pred*
and *rel-preserves-binary-pred* (Rel^+) *Pred*
and *rel-preserves-binary-pred* (Rel^*) *Pred*
⟨*proof*⟩

If a relation reflects a predicate then so does its reflexive or/and transitive closure.

lemma *reflection-and-closures*:

fixes *Rel* :: $('a \times 'a)$ *set*
  and *Pred* :: $'a \Rightarrow$ *bool*
assumes *reflection*: *rel-reflects-pred Rel Pred*
shows *rel-reflects-pred* $(Rel^=)$ *Pred*
  and *rel-reflects-pred* (Rel^+) *Pred*
  and *rel-reflects-pred* (Rel^*) *Pred*
⟨*proof*⟩

lemma *reflection-of-binary-predicates-and-closures*:
fixes *Rel* :: $('a \times 'a)$ *set*
  and *Pred* :: $'a \Rightarrow 'b \Rightarrow$ *bool*
assumes *reflection*: *rel-reflects-binary-pred Rel Pred*
shows *rel-reflects-binary-pred* $(Rel^=)$ *Pred*
  and *rel-reflects-binary-pred* (Rel^+) *Pred*
  and *rel-reflects-binary-pred* (Rel^*) *Pred*
⟨*proof*⟩

If a relation respects a predicate then so does its reflexive, symmetric, or/and transitive closure.

lemma *respection-and-closures*:
fixes *Rel* :: $('a \times 'a)$ *set*
  and *Pred* :: $'a \Rightarrow$ *bool*
assumes *respection*: *rel-respects-pred Rel Pred*
shows *rel-respects-pred* $(Rel^=)$ *Pred*
  and *rel-respects-pred* (*symcl Rel*) *Pred*
  and *rel-respects-pred* (Rel^+) *Pred*
  and *rel-respects-pred* (*symcl* $(Rel^=)$) *Pred*
  and *rel-respects-pred* (Rel^*) *Pred*
  and *rel-respects-pred* $((symcl\ (Rel^=))^+)$ *Pred*
⟨*proof*⟩

lemma *respection-of-binary-predicates-and-closures*:
fixes *Rel* :: $('a \times 'a)$ *set*
  and *Pred* :: $'a \Rightarrow 'b \Rightarrow$ *bool*
assumes *respection*: *rel-respects-binary-pred Rel Pred*
shows *rel-respects-binary-pred* $(Rel^=)$ *Pred*
  and *rel-respects-binary-pred* (*symcl Rel*) *Pred*
  and *rel-respects-binary-pred* (Rel^+) *Pred*
  and *rel-respects-binary-pred* (*symcl* $(Rel^=)$) *Pred*
  and *rel-respects-binary-pred* (Rel^*) *Pred*
  and *rel-respects-binary-pred* $((symcl\ (Rel^=))^+)$ *Pred*
⟨*proof*⟩

end
theory *ProcessCalculi*
imports *Relations*
begin

2 Process Calculi

A process calculus is given by a set of process terms (syntax) and a relation on terms (semantics). We consider reduction as well as labelled variants of the semantics.

2.1 Reduction Semantics

A set of process terms and a relation on pairs of terms (called reduction semantics) define a process calculus.

record *'proc processCalculus* =
  Reductions :: $'proc \Rightarrow 'proc \Rightarrow$ *bool*

A pair of the reduction relation is called a (reduction) step.

abbreviation *step* :: $'proc \Rightarrow 'proc\ processCalculus \Rightarrow 'proc \Rightarrow bool$
  $(\text{-} \longmapsto\text{-}\ \text{-}\ [70,\ 70,\ 70]\ 80)$
 where
 $P \longmapsto Cal\ Q \equiv Reductions\ Cal\ P\ Q$

We use * to indicate the reflexive and transitive closure of the reduction relation.

primrec *nSteps*
 $:: 'proc \Rightarrow 'proc\ processCalculus \Rightarrow nat \Rightarrow 'proc \Rightarrow bool$
  $(\text{-} \longmapsto\text{-}^{\text{-}}\ \text{-}\ [70,\ 70,\ 70,\ 70]\ 80)$
 where
 $P \longmapsto Cal^{0}\ Q \quad = (P = Q)\ |$
 $P \longmapsto Cal^{Suc\ n}\ Q = (\exists\, P'.\ P \longmapsto Cal^{n}\ P' \land P' \longmapsto Cal\ Q)$

definition *steps*
 $:: 'proc \Rightarrow 'proc\ processCalculus \Rightarrow 'proc \Rightarrow bool$
  $(\text{-} \longmapsto\text{-}{*}\ \text{-}\ [70,\ 70,\ 70]\ 80)$
 where
 $P \longmapsto Cal{*}\ Q \equiv \exists\, n.\ P \longmapsto Cal^{n}\ Q$

A process is divergent, if it can perform an infinite sequence of steps.

definition *divergent*
 $:: 'proc \Rightarrow 'proc\ processCalculus \Rightarrow bool$
  $(\text{-} \longmapsto\text{-}\omega\ [70,\ 70]\ 80)$
 where
 $P \longmapsto (Cal)\omega \equiv \forall\, P'.\ P \longmapsto Cal{*}\ P' \longrightarrow (\exists\, P''.\ P' \longmapsto Cal\ P'')$

Each term can perform an (empty) sequence of steps to itself.

lemma *steps-refl*:
 fixes $Cal :: 'proc\ processCalculus$
  and $P \quad :: 'proc$
 shows $P \longmapsto Cal{*}\ P$
⟨*proof*⟩

A single step is a sequence of steps of length one.

lemma *step-to-steps*:
 fixes $Cal\ :: 'proc\ processCalculus$
  and $P\ P' :: 'proc$
 assumes *step*: $P \longmapsto Cal\ P'$
 shows $P \longmapsto Cal{*}\ P'$
⟨*proof*⟩

If there is a sequence of steps from P to Q and from Q to R, then there is also a sequence of steps from P to R.

lemma *nSteps-add*:
 fixes $Cal\quad :: 'proc\ processCalculus$
  and $n1\ n2 :: nat$
 shows $\forall\, P\ Q\ R.\ P \longmapsto Cal^{n1}\ Q \land Q \longmapsto Cal^{n2}\ R \longrightarrow P \longmapsto Cal^{(n1\ +\ n2)}\ R$
⟨*proof*⟩

lemma *steps-add*:
 fixes $Cal\quad :: 'proc\ processCalculus$
  and $P\ Q\ R :: 'proc$
 assumes *A1*: $P \longmapsto Cal{*}\ Q$
   and *A2*: $Q \longmapsto Cal{*}\ R$
 shows $P \longmapsto Cal{*}\ R$
⟨*proof*⟩

2.1.1 Observables or Barbs

We assume a predicate that tests terms for some kind of observables. At this point we do not limit or restrict the kind of observables used for a calculus nor the method to check them.

record ('*proc*, '*barbs*) *calculusWithBarbs* =
  Calculus :: '*proc processCalculus*
  HasBarb :: '*proc* ⇒ '*barbs* ⇒ *bool* (-↓- [*70*, *70*] *80*)

abbreviation *hasBarb*
  :: '*proc* ⇒ ('*proc*, '*barbs*) *calculusWithBarbs* ⇒ '*barbs* ⇒ *bool*
    (-↓<->- [*70*, *70*, *70*] *80*)
  where
  P↓<*CWB*>*a* ≡ *HasBarb CWB P a*

A term reaches a barb if it can evolve to a term that has this barb.

abbreviation *reachesBarb*
  :: '*proc* ⇒ ('*proc*, '*barbs*) *calculusWithBarbs* ⇒ '*barbs* ⇒ *bool*
    (-⇓<->- [*70*, *70*, *70*] *80*)
  where
  P⇓<*CWB*>*a* ≡ ∃ *P'*. *P* ⟼(*Calculus CWB*)∗ *P'* ∧ *P'*↓<*CWB*>*a*

A relation R preserves barbs if whenever (P, Q) in R and P has a barb then also Q has this barb.

abbreviation *rel-preserves-barb-set*
    :: ('*proc* × '*proc*) *set* ⇒ ('*proc*, '*barbs*) *calculusWithBarbs* ⇒ '*barbs set* ⇒ *bool*
  where
  rel-preserves-barb-set Rel CWB Barbs ≡
   rel-preserves-binary-pred Rel (λ*P a*. *a* ∈ *Barbs* ∧ *P*↓<*CWB*>*a*)

abbreviation *rel-preserves-barbs*
    :: ('*proc* × '*proc*) *set* ⇒ ('*proc*, '*barbs*) *calculusWithBarbs* ⇒ *bool*
  where
  rel-preserves-barbs Rel CWB ≡ *rel-preserves-binary-pred Rel* (*HasBarb CWB*)

lemma *preservation-of-barbs-and-set-of-barbs*:
  fixes *Rel* :: ('*proc* × '*proc*) *set*
    and *CWB* :: ('*proc*, '*barbs*) *calculusWithBarbs*
  shows *rel-preserves-barbs Rel CWB* = (∀ *Barbs*. *rel-preserves-barb-set Rel CWB Barbs*)
    ⟨*proof*⟩

A relation R reflects barbs if whenever (P, Q) in R and Q has a barb then also P has this barb.

abbreviation *rel-reflects-barb-set*
    :: ('*proc* × '*proc*) *set* ⇒ ('*proc*, '*barbs*) *calculusWithBarbs* ⇒ '*barbs set* ⇒ *bool*
  where
  rel-reflects-barb-set Rel CWB Barbs ≡
   rel-reflects-binary-pred Rel (λ*P a*. *a* ∈ *Barbs* ∧ *P*↓<*CWB*>*a*)

abbreviation *rel-reflects-barbs*
    :: ('*proc* × '*proc*) *set* ⇒ ('*proc*, '*barbs*) *calculusWithBarbs* ⇒ *bool*
  where
  rel-reflects-barbs Rel CWB ≡ *rel-reflects-binary-pred Rel* (*HasBarb CWB*)

lemma *reflection-of-barbs-and-set-of-barbs*:
  fixes *Rel* :: ('*proc* × '*proc*) *set*
    and *CWB* :: ('*proc*, '*barbs*) *calculusWithBarbs*
  shows *rel-reflects-barbs Rel CWB* = (∀ *Barbs*. *rel-reflects-barb-set Rel CWB Barbs*)
    ⟨*proof*⟩

A relation respects barbs if it preserves and reflects barbs.

abbreviation *rel-respects-barb-set*

:: $('proc \times 'proc)$ *set* $\Rightarrow$ $('proc, 'barbs)$ *calculusWithBarbs* $\Rightarrow$ *'barbs set* $\Rightarrow$ *bool*
where
rel-respects-barb-set Rel CWB Barbs $\equiv$
rel-preserves-barb-set Rel CWB Barbs $\wedge$ *rel-reflects-barb-set Rel CWB Barbs*

abbreviation *rel-respects-barbs*
:: $('proc \times 'proc)$ *set* $\Rightarrow$ $('proc, 'barbs)$ *calculusWithBarbs* $\Rightarrow$ *bool*
where
rel-respects-barbs Rel CWB $\equiv$ *rel-preserves-barbs Rel CWB* $\wedge$ *rel-reflects-barbs Rel CWB*

lemma *respection-of-barbs-and-set-of-barbs*:
fixes *Rel* :: $('proc \times 'proc)$ *set*
and *CWB* :: $('proc, 'barbs)$ *calculusWithBarbs*
shows *rel-respects-barbs Rel CWB* = ($\forall$ *Barbs*. *rel-respects-barb-set Rel CWB Barbs*)
⟨*proof*⟩

If a relation preserves barbs then so does its reflexive or/and transitive closure.

lemma *preservation-of-barbs-and-closures*:
fixes *Rel* :: $('proc \times 'proc)$ *set*
and *CWB* :: $('proc, 'barbs)$ *calculusWithBarbs*
assumes *preservation*: *rel-preserves-barbs Rel CWB*
shows *rel-preserves-barbs* $(Rel^=)$ *CWB*
and *rel-preserves-barbs* (Rel^+) *CWB*
and *rel-preserves-barbs* (Rel^*) *CWB*
⟨*proof*⟩

If a relation reflects barbs then so does its reflexive or/and transitive closure.

lemma *reflection-of-barbs-and-closures*:
fixes *Rel* :: $('proc \times 'proc)$ *set*
and *CWB* :: $('proc, 'barbs)$ *calculusWithBarbs*
assumes *reflection*: *rel-reflects-barbs Rel CWB*
shows *rel-reflects-barbs* $(Rel^=)$ *CWB*
and *rel-reflects-barbs* (Rel^+) *CWB*
and *rel-reflects-barbs* (Rel^*) *CWB*
⟨*proof*⟩

If a relation respects barbs then so does its reflexive, symmetric, or/and transitive closure.

lemma *respection-of-barbs-and-closures*:
fixes *Rel* :: $('proc \times 'proc)$ *set*
and *CWB* :: $('proc, 'barbs)$ *calculusWithBarbs*
assumes *respection*: *rel-respects-barbs Rel CWB*
shows *rel-respects-barbs* $(Rel^=)$ *CWB*
and *rel-respects-barbs* (*symcl Rel*) *CWB*
and *rel-respects-barbs* (Rel^+) *CWB*
and *rel-respects-barbs* (*symcl* $(Rel^=)$) *CWB*
and *rel-respects-barbs* (Rel^*) *CWB*
and *rel-respects-barbs* $((symcl\ (Rel^=))^+)$ *CWB*
⟨*proof*⟩

A relation R weakly preserves barbs if it preserves reachability of barbs, i.e., if (P, Q) in R and P reaches a barb then also Q has to reach this barb.

abbreviation *rel-weakly-preserves-barb-set*
:: $('proc \times 'proc)$ *set* $\Rightarrow$ $('proc, 'barbs)$ *calculusWithBarbs* $\Rightarrow$ *'barbs set* $\Rightarrow$ *bool*
where
rel-weakly-preserves-barb-set Rel CWB Barbs $\equiv$
rel-preserves-binary-pred Rel $(\lambda P\ a.\ a \in Barbs \wedge P\Downarrow$<CWB>$a)$

abbreviation *rel-weakly-preserves-barbs*
:: $('proc \times 'proc)$ *set* $\Rightarrow$ $('proc, 'barbs)$ *calculusWithBarbs* $\Rightarrow$ *bool*
where

rel-weakly-preserves-barbs Rel CWB $\equiv$ *rel-preserves-binary-pred Rel* $(\lambda P\ a.\ P{\Downarrow}{<}CWB{>}a)$

lemma *weak-preservation-of-barbs-and-set-of-barbs*:
fixes *Rel* :: (*'proc* $\times$ *'proc*) *set*
and *CWB* :: (*'proc*, *'barbs*) *calculusWithBarbs*
shows *rel-weakly-preserves-barbs Rel CWB*
= ($\forall$ *Barbs*. *rel-weakly-preserves-barb-set Rel CWB Barbs*)
⟨*proof*⟩

A relation R weakly reflects barbs if it reflects reachability of barbs, i.e., if (P, Q) in R and Q reaches a barb then also P has to reach this barb.

abbreviation *rel-weakly-reflects-barb-set*
:: (*'proc* $\times$ *'proc*) *set* $\Rightarrow$ (*'proc*, *'barbs*) *calculusWithBarbs* $\Rightarrow$ *'barbs set* $\Rightarrow$ *bool*
where
rel-weakly-reflects-barb-set Rel CWB Barbs $\equiv$
rel-reflects-binary-pred Rel $(\lambda P\ a.\ a \in Barbs \wedge P{\Downarrow}{<}CWB{>}a)$

abbreviation *rel-weakly-reflects-barbs*
:: (*'proc* $\times$ *'proc*) *set* $\Rightarrow$ (*'proc*, *'barbs*) *calculusWithBarbs* $\Rightarrow$ *bool*
where
rel-weakly-reflects-barbs Rel CWB $\equiv$ *rel-reflects-binary-pred Rel* $(\lambda P\ a.\ P{\Downarrow}{<}CWB{>}a)$

lemma *weak-reflection-of-barbs-and-set-of-barbs*:
fixes *Rel* :: (*'proc* $\times$ *'proc*) *set*
and *CWB* :: (*'proc*, *'barbs*) *calculusWithBarbs*
shows *rel-weakly-reflects-barbs Rel CWB* = ($\forall$ *Barbs*. *rel-weakly-reflects-barb-set Rel CWB Barbs*)
⟨*proof*⟩

A relation weakly respects barbs if it weakly preserves and weakly reflects barbs.

abbreviation *rel-weakly-respects-barb-set*
:: (*'proc* $\times$ *'proc*) *set* $\Rightarrow$ (*'proc*, *'barbs*) *calculusWithBarbs* $\Rightarrow$ *'barbs set* $\Rightarrow$ *bool*
where
rel-weakly-respects-barb-set Rel CWB Barbs $\equiv$
rel-weakly-preserves-barb-set Rel CWB Barbs $\wedge$ *rel-weakly-reflects-barb-set Rel CWB Barbs*

abbreviation *rel-weakly-respects-barbs*
:: (*'proc* $\times$ *'proc*) *set* $\Rightarrow$ (*'proc*, *'barbs*) *calculusWithBarbs* $\Rightarrow$ *bool*
where
rel-weakly-respects-barbs Rel CWB $\equiv$
rel-weakly-preserves-barbs Rel CWB $\wedge$ *rel-weakly-reflects-barbs Rel CWB*

lemma *weak-respection-of-barbs-and-set-of-barbs*:
fixes *Rel* :: (*'proc* $\times$ *'proc*) *set*
and *CWB* :: (*'proc*, *'barbs*) *calculusWithBarbs*
shows *rel-weakly-respects-barbs Rel CWB* = ($\forall$ *Barbs*. *rel-weakly-respects-barb-set Rel CWB Barbs*)
⟨*proof*⟩

If a relation weakly preserves barbs then so does its reflexive or/and transitive closure.

lemma *weak-preservation-of-barbs-and-closures*:
fixes *Rel* :: (*'proc* $\times$ *'proc*) *set*
and *CWB* :: (*'proc*, *'barbs*) *calculusWithBarbs*
assumes *preservation*: *rel-weakly-preserves-barbs Rel CWB*
shows *rel-weakly-preserves-barbs* ($Rel^=$) *CWB*
and *rel-weakly-preserves-barbs* (Rel^+) *CWB*
and *rel-weakly-preserves-barbs* (Rel^*) *CWB*
⟨*proof*⟩

If a relation weakly reflects barbs then so does its reflexive or/and transitive closure.

lemma *weak-reflection-of-barbs-and-closures*:
fixes *Rel* :: (*'proc* $\times$ *'proc*) *set*

and *CWB* :: ('*proc*, '*barbs*) *calculusWithBarbs*
assumes *reflection*: *rel-weakly-reflects-barbs Rel CWB*
shows *rel-weakly-reflects-barbs* ($Rel^{=}$) *CWB*
and *rel-weakly-reflects-barbs* (Rel^{+}) *CWB*
and *rel-weakly-reflects-barbs* (Rel^{*}) *CWB*
⟨*proof*⟩

If a relation weakly respects barbs then so does its reflexive, symmetric, or/and transitive closure.

lemma *weak-respection-of-barbs-and-closures*:
fixes *Rel* :: ('*proc* × '*proc*) *set*
and *CWB* :: ('*proc*, '*barbs*) *calculusWithBarbs*
assumes *respection*: *rel-weakly-respects-barbs Rel CWB*
shows *rel-weakly-respects-barbs* ($Rel^{=}$) *CWB*
and *rel-weakly-respects-barbs* (*symcl Rel*) *CWB*
and *rel-weakly-respects-barbs* (Rel^{+}) *CWB*
and *rel-weakly-respects-barbs* (*symcl* ($Rel^{=}$)) *CWB*
and *rel-weakly-respects-barbs* (Rel^{*}) *CWB*
and *rel-weakly-respects-barbs* ((*symcl* ($Rel^{=}$))$^{+}$) *CWB*
⟨*proof*⟩

end
theory *SimulationRelations*
imports *ProcessCalculi*
begin

3 Simulation Relations

Simulation relations are a special kind of property on relations on processes. They usually require that steps are (strongly or weakly) preserved and/or reflected modulo the relation. We consider different kinds of simulation relations.

3.1 Simulation

A weak reduction simulation is relation R such that if (P, Q) in R and P evolves to some P' then there exists some Q' such that Q evolves to Q' and (P', Q') in R.

abbreviation *weak-reduction-simulation*
:: ('*proc* × '*proc*) *set* ⇒ '*proc processCalculus* ⇒ *bool*
where
weak-reduction-simulation Rel Cal ≡
$\forall P\ Q\ P'.\ (P,\ Q) \in Rel \land P \longmapsto Cal{*}\ P' \longrightarrow (\exists Q'.\ Q \longmapsto Cal{*}\ Q' \land (P',\ Q') \in Rel)$

A weak barbed simulation is weak reduction simulation that weakly preserves barbs.

abbreviation *weak-barbed-simulation*
:: ('*proc* × '*proc*) *set* ⇒ ('*proc*, '*barbs*) *calculusWithBarbs* ⇒ *bool*
where
weak-barbed-simulation Rel CWB ≡
weak-reduction-simulation Rel (*Calculus CWB*) ∧ *rel-weakly-preserves-barbs Rel CWB*

The reflexive and/or transitive closure of a weak simulation is a weak simulation.

lemma *weak-reduction-simulation-and-closures*:
fixes *Rel* :: ('*proc* × '*proc*) *set*
and *Cal* :: '*proc processCalculus*
assumes *simulation*: *weak-reduction-simulation Rel Cal*
shows *weak-reduction-simulation* ($Rel^{=}$) *Cal*
and *weak-reduction-simulation* (Rel^{+}) *Cal*
and *weak-reduction-simulation* (Rel^{*}) *Cal*
⟨*proof*⟩

lemma *weak-barbed-simulation-and-closures*:
  fixes $Rel :: ('proc \times 'proc)\ set$
    and $CWB :: ('proc, 'barbs)\ calculusWithBarbs$
  assumes *simulation*: *weak-barbed-simulation Rel CWB*
  shows *weak-barbed-simulation* $(Rel^=)$ *CWB*
    and *weak-barbed-simulation* (Rel^+) *CWB*
    and *weak-barbed-simulation* (Rel^*) *CWB*
⟨*proof*⟩

In the case of a simulation weak preservation of barbs can be replaced by the weaker condition that whenever (P, Q) in the relation and P has a barb then Q have to be able to reach this barb.

abbreviation *weak-barbed-preservation-cond*
    $:: ('proc \times 'proc)\ set \Rightarrow ('proc, 'barbs)\ calculusWithBarbs \Rightarrow bool$
  where
  $weak\text{-}barbed\text{-}preservation\text{-}cond\ Rel\ CWB \equiv \forall\, P\ Q\ a.\ (P,\ Q) \in Rel \land P{\downarrow}{<}CWB{>}a \longrightarrow Q{\Downarrow}{<}CWB{>}a$

lemma *weak-preservation-of-barbs*:
  fixes $Rel :: ('proc \times 'proc)\ set$
    and $CWB :: ('proc, 'barbs)\ calculusWithBarbs$
  assumes *preservation*: *rel-weakly-preserves-barbs Rel CWB*
  shows *weak-barbed-preservation-cond Rel CWB*
⟨*proof*⟩

lemma *simulation-impl-equality-of-preservation-of-barbs-conditions*:
  fixes $Rel :: ('proc \times 'proc)\ set$
    and $CWB :: ('proc, 'barbs)\ calculusWithBarbs$
  assumes *simulation*: *weak-reduction-simulation Rel* (*Calculus CWB*)
  shows *rel-weakly-preserves-barbs Rel CWB* = *weak-barbed-preservation-cond Rel CWB*
⟨*proof*⟩

A strong reduction simulation is relation R such that for each pair (P, Q) in R and each step of P to some P' there exists some Q' such that there is a step of Q to Q' and (P', Q') in R.

abbreviation $strong\text{-}reduction\text{-}simulation :: ('proc \times 'proc)\ set \Rightarrow 'proc\ processCalculus \Rightarrow bool$
  where
  $strong\text{-}reduction\text{-}simulation\ Rel\ Cal \equiv$
  $\forall\, P\ Q\ P'.\ (P,\ Q) \in Rel \land P \longmapsto Cal\ P' \longrightarrow (\exists\, Q'.\ Q \longmapsto Cal\ Q' \land (P',\ Q') \in Rel)$

A strong barbed simulation is strong reduction simulation that preserves barbs.

abbreviation *strong-barbed-simulation*
    $:: ('proc \times 'proc)\ set \Rightarrow ('proc, 'barbs)\ calculusWithBarbs \Rightarrow bool$
  where
  strong-barbed-simulation Rel CWB $\equiv$
   strong-reduction-simulation Rel (*Calculus CWB*) $\land$ *rel-preserves-barbs Rel CWB*

A strong strong simulation is also a weak simulation.

lemma *strong-impl-weak-reduction-simulation*:
  fixes $Rel :: ('proc \times 'proc)\ set$
    and $Cal :: 'proc\ processCalculus$
  assumes *simulation*: *strong-reduction-simulation Rel Cal*
  shows *weak-reduction-simulation Rel Cal*
⟨*proof*⟩

lemma *strong-barbed-simulation-impl-weak-preservation-of-barbs*:
  fixes $Rel :: ('proc \times 'proc)\ set$
    and $CWB :: ('proc, 'barbs)\ calculusWithBarbs$
  assumes *simulation*: *strong-barbed-simulation Rel CWB*
  shows *rel-weakly-preserves-barbs Rel CWB*
⟨*proof*⟩

lemma *strong-impl-weak-barbed-simulation*:
fixes *Rel* :: $('proc \times 'proc)$ *set*
and *CWB* :: $('proc,\ 'barbs)$ *calculusWithBarbs*
assumes *simulation*: *strong-barbed-simulation Rel CWB*
shows *weak-barbed-simulation Rel CWB*
⟨*proof*⟩

The reflexive and/or transitive closure of a strong simulation is a strong simulation.

lemma *strong-reduction-simulation-and-closures*:
fixes *Rel* :: $('proc \times 'proc)$ *set*
and *Cal* :: *'proc processCalculus*
assumes *simulation*: *strong-reduction-simulation Rel Cal*
shows *strong-reduction-simulation* $(Rel^{=})$ *Cal*
and *strong-reduction-simulation* (Rel^{+}) *Cal*
and *strong-reduction-simulation* (Rel^{*}) *Cal*
⟨*proof*⟩

lemma *strong-barbed-simulation-and-closures*:
fixes *Rel* :: $('proc \times 'proc)$ *set*
and *CWB* :: $('proc,\ 'barbs)$ *calculusWithBarbs*
assumes *simulation*: *strong-barbed-simulation Rel CWB*
shows *strong-barbed-simulation* $(Rel^{=})$ *CWB*
and *strong-barbed-simulation* (Rel^{+}) *CWB*
and *strong-barbed-simulation* (Rel^{*}) *CWB*
⟨*proof*⟩

3.2 Contrasimulation

A weak reduction contrasimulation is relation R such that if (P, Q) in R and P evolves to some P’ then there exists some Q’ such that Q evolves to Q’ and (Q’, P’) in R.

abbreviation *weak-reduction-contrasimulation*
:: $('proc \times 'proc)$ *set* $\Rightarrow$ *'proc processCalculus* $\Rightarrow$ *bool*
where
weak-reduction-contrasimulation Rel Cal $\equiv$
$\forall P\ Q\ P'.\ (P,\ Q) \in Rel \wedge P \longmapsto Cal{*}\ P' \longrightarrow (\exists\, Q'.\ Q \longmapsto Cal{*}\ Q' \wedge (Q',\ P') \in Rel)$

A weak barbed contrasimulation is weak reduction contrasimulation that weakly preserves barbs.

abbreviation *weak-barbed-contrasimulation*
:: $('proc \times 'proc)$ *set* $\Rightarrow$ $('proc,\ 'barbs)$ *calculusWithBarbs* $\Rightarrow$ *bool*
where
weak-barbed-contrasimulation Rel CWB $\equiv$
weak-reduction-contrasimulation Rel (*Calculus CWB*) $\wedge$ *rel-weakly-preserves-barbs Rel CWB*

The reflexive and/or transitive closure of a weak contrasimulation is a weak contrasimulation.

lemma *weak-reduction-contrasimulation-and-closures*:
fixes *Rel* :: $('proc \times 'proc)$ *set*
and *Cal* :: *'proc processCalculus*
assumes *contrasimulation*: *weak-reduction-contrasimulation Rel Cal*
shows *weak-reduction-contrasimulation* $(Rel^{=})$ *Cal*
and *weak-reduction-contrasimulation* (Rel^{+}) *Cal*
and *weak-reduction-contrasimulation* (Rel^{*}) *Cal*
⟨*proof*⟩

lemma *weak-barbed-contrasimulation-and-closures*:
fixes *Rel* :: $('proc \times 'proc)$ *set*
and *CWB* :: $('proc,\ 'barbs)$ *calculusWithBarbs*
assumes *contrasimulation*: *weak-barbed-contrasimulation Rel CWB*
shows *weak-barbed-contrasimulation* $(Rel^{=})$ *CWB*

and *weak-barbed-contrasimulation* (Rel^+) *CWB*
and *weak-barbed-contrasimulation* (Rel^*) *CWB*
⟨*proof*⟩

3.3 Coupled Simulation

A weak reduction coupled simulation is relation R such that if (P, Q) in R and P evolves to some P' then there exists some Q' such that Q evolves to Q' and (P', Q') in R and there exits some Q' such that Q evolves to Q' and (Q', P') in R.

abbreviation *weak-reduction-coupled-simulation*
:: $('proc \times 'proc)$ *set* $\Rightarrow$ *'proc processCalculus* $\Rightarrow$ *bool*
where
weak-reduction-coupled-simulation Rel Cal $\equiv$
$\forall P\ Q\ P'.\ (P,\ Q) \in Rel \wedge P \longmapsto Cal{*}\ P'$
$\longrightarrow (\exists Q'.\ Q \longmapsto Cal{*}\ Q' \wedge (P',\ Q') \in Rel) \wedge (\exists Q'.\ Q \longmapsto Cal{*}\ Q' \wedge (Q',\ P') \in Rel)$

A weak barbed coupled simulation is weak reduction coupled simulation that weakly preserves barbs.

abbreviation *weak-barbed-coupled-simulation*
:: $('proc \times 'proc)$ *set* $\Rightarrow$ $('proc,\ 'barbs)$ *calculusWithBarbs* $\Rightarrow$ *bool*
where
weak-barbed-coupled-simulation Rel CWB $\equiv$
weak-reduction-coupled-simulation Rel (*Calculus CWB*) $\wedge$ *rel-weakly-preserves-barbs Rel CWB*

A weak coupled simulation combines the conditions on a weak simulation and a weak contrasimulation.

lemma *weak-reduction-coupled-simulation-versus-simulation-and-contrasimulation*:
fixes *Rel* :: $('proc \times 'proc)$ *set*
and *Cal* :: *'proc processCalculus*
shows *weak-reduction-coupled-simulation Rel Cal*
= (*weak-reduction-simulation Rel Cal* $\wedge$ *weak-reduction-contrasimulation Rel Cal*)
⟨*proof*⟩

lemma *weak-barbed-coupled-simulation-versus-simulation-and-contrasimulation*:
fixes *Rel* :: $('proc \times 'proc)$ *set*
and *CWB* :: $('proc,\ 'barbs)$ *calculusWithBarbs*
shows *weak-barbed-coupled-simulation Rel CWB*
= (*weak-barbed-simulation Rel CWB* $\wedge$ *weak-barbed-contrasimulation Rel CWB*)
⟨*proof*⟩

The reflexive and/or transitive closure of a weak coupled simulation is a weak coupled simulation.

lemma *weak-reduction-coupled-simulation-and-closures*:
fixes *Rel* :: $('proc \times 'proc)$ *set*
and *Cal* :: *'proc processCalculus*
assumes *coupledSimulation*: *weak-reduction-coupled-simulation Rel Cal*
shows *weak-reduction-coupled-simulation* $(Rel^=)$ *Cal*
and *weak-reduction-coupled-simulation* (Rel^+) *Cal*
and *weak-reduction-coupled-simulation* (Rel^*) *Cal*
⟨*proof*⟩

lemma *weak-barbed-coupled-simulation-and-closures*:
fixes *Rel* :: $('proc \times 'proc)$ *set*
and *CWB* :: $('proc,\ 'barbs)$ *calculusWithBarbs*
assumes *coupledSimulation*: *weak-barbed-coupled-simulation Rel CWB*
shows *weak-barbed-coupled-simulation* $(Rel^=)$ *CWB*
and *weak-barbed-coupled-simulation* (Rel^+) *CWB*
and *weak-barbed-coupled-simulation* (Rel^*) *CWB*
⟨*proof*⟩

3.4 Correspondence Simulation

A weak reduction correspondence simulation is relation R such that (1) if (P, Q) in R and P evolves to some P' then there exists some Q' such that Q evolves to Q' and (P', Q') in R, and (2) if (P, Q) in R and P evolves to some P' then there exists some P" and Q" such that P evolves to P" and Q' evolves to Q" and (P", Q") in Rel.

abbreviation *weak-reduction-correspondence-simulation*
 $:: ('proc \times 'proc)\ set \Rightarrow 'proc\ processCalculus \Rightarrow bool$
 where
 weak-reduction-correspondence-simulation Rel Cal $\equiv$
 $(\forall P\ Q\ P'.\ (P,\ Q) \in Rel \wedge P \longmapsto Cal{*}\ P' \longrightarrow (\exists Q'.\ Q \longmapsto Cal{*}\ Q' \wedge (P',\ Q') \in Rel))$
 $\wedge\ (\forall P\ Q\ Q'.\ (P,\ Q) \in Rel \wedge Q \longmapsto Cal{*}\ Q'$
 $\longrightarrow (\exists P''\ Q''.\ P \longmapsto Cal{*}\ P'' \wedge Q' \longmapsto Cal{*}\ Q'' \wedge (P'',\ Q'') \in Rel))$

A weak barbed correspondence simulation is weak reduction correspondence simulation that weakly respects barbs.

abbreviation *weak-barbed-correspondence-simulation*
 $:: ('proc \times 'proc)\ set \Rightarrow ('proc,\ 'barbs)\ calculusWithBarbs \Rightarrow bool$
 where
 weak-barbed-correspondence-simulation Rel CWB $\equiv$
 weak-reduction-correspondence-simulation Rel (Calculus CWB)
 $\wedge$ *rel-weakly-respects-barbs Rel CWB*

For each weak correspondence simulation R there exists a weak coupled simulation that contains all pairs of R in both directions.

inductive-set *cSim-cs* $:: ('proc \times 'proc)\ set \Rightarrow 'proc\ processCalculus \Rightarrow ('proc \times 'proc)\ set$
 for $Rel :: ('proc \times 'proc)\ set$
 and $Cal :: 'proc\ processCalculus$
 where
 left: $[\![Q \longmapsto Cal{*}\ Q';\ (P',\ Q') \in Rel]\!] \Longrightarrow (P',\ Q) \in cSim\text{-}cs\ Rel\ Cal\ |$
 right: $[\![P \longmapsto Cal{*}\ P';\ (Q,\ P) \in Rel]\!] \Longrightarrow (P',\ Q) \in cSim\text{-}cs\ Rel\ Cal\ |$
 trans: $[\![(P,\ Q) \in cSim\text{-}cs\ Rel\ Cal;\ (Q,\ R) \in cSim\text{-}cs\ Rel\ Cal]\!] \Longrightarrow (P,\ R) \in cSim\text{-}cs\ Rel\ Cal$

lemma *weak-reduction-correspondence-simulation-impl-coupled-simulation*:
 fixes $Rel :: ('proc \times 'proc)\ set$
 and $Cal :: 'proc\ processCalculus$
 assumes *corrSim*: *weak-reduction-correspondence-simulation Rel Cal*
 shows *weak-reduction-coupled-simulation (cSim-cs Rel Cal) Cal*
 and $\forall P\ Q.\ (P,\ Q) \in Rel \longrightarrow (P,\ Q) \in cSim\text{-}cs\ Rel\ Cal \wedge (Q,\ P) \in cSim\text{-}cs\ Rel\ Cal$
⟨*proof*⟩

lemma *weak-barbed-correspondence-simulation-impl-coupled-simulation*:
 fixes $Rel :: ('proc \times 'proc)\ set$
 and $CWB :: ('proc,\ 'barbs)\ calculusWithBarbs$
 assumes *corrSim*: *weak-barbed-correspondence-simulation Rel CWB*
 shows *weak-barbed-coupled-simulation (cSim-cs Rel (Calculus CWB)) CWB*
 and $\forall P\ Q.\ (P,\ Q) \in Rel \longrightarrow (P,\ Q) \in cSim\text{-}cs\ Rel\ (Calculus\ CWB)$
 $\wedge\ (Q,\ P) \in cSim\text{-}cs\ Rel\ (Calculus\ CWB)$
⟨*proof*⟩

lemma *reduction-correspondence-simulation-condition-trans*:
 fixes $Cal\ :: 'proc\ processCalculus$
 and $P\ Q\ R :: 'proc$
 and $Rel\ :: ('proc \times 'proc)\ set$
 assumes *A1*: $\forall Q'.\ Q \longmapsto Cal{*}\ Q' \longrightarrow (\exists P''\ Q''.\ P \longmapsto Cal{*}\ P'' \wedge Q' \longmapsto Cal{*}\ Q'' \wedge (P'',\ Q'') \in Rel)$
 and *A2*: $\forall R'.\ R \longmapsto Cal{*}\ R' \longrightarrow (\exists Q''\ R''.\ Q \longmapsto Cal{*}\ Q'' \wedge R' \longmapsto Cal{*}\ R'' \wedge (Q'',\ R'') \in Rel)$
 and *A3*: *weak-reduction-simulation Rel Cal*
 and *A4*: *trans Rel*
 shows $\forall R'.\ R \longmapsto Cal{*}\ R' \longrightarrow (\exists P''\ R''.\ P \longmapsto Cal{*}\ P'' \wedge R' \longmapsto Cal{*}\ R'' \wedge (P'',\ R'') \in Rel)$

$\langle proof \rangle$

The reflexive and/or transitive closure of a weak correspondence simulation is a weak correspondence simulation.

lemma *weak-reduction-correspondence-simulation-and-closures*:
fixes *Rel* :: $('proc \times 'proc)$ *set*
and *Cal* :: *'proc processCalculus*
assumes *corrSim*: *weak-reduction-correspondence-simulation Rel Cal*
shows *weak-reduction-correspondence-simulation* $(Rel^=)$ *Cal*
and *weak-reduction-correspondence-simulation* (Rel^+) *Cal*
and *weak-reduction-correspondence-simulation* (Rel^*) *Cal*
$\langle proof \rangle$

lemma *weak-barbed-correspondence-simulation-and-closures*:
fixes *Rel* :: $('proc \times 'proc)$ *set*
and *CWB* :: $('proc, 'barbs)$ *calculusWithBarbs*
assumes *corrSim*: *weak-barbed-correspondence-simulation Rel CWB*
shows *weak-barbed-correspondence-simulation* $(Rel^=)$ *CWB*
and *weak-barbed-correspondence-simulation* (Rel^+) *CWB*
and *weak-barbed-correspondence-simulation* (Rel^*) *CWB*
$\langle proof \rangle$

3.5 Bisimulation

A weak reduction bisimulation is relation R such that (1) if (P, Q) in R and P evolves to some P' then there exists some Q' such that Q evolves to Q' and (P', Q') in R, and (2) if (P, Q) in R and Q evolves to some Q' then there exists some P' such that P evolves to P' and (P', Q') in R.

abbreviation *weak-reduction-bisimulation*
:: $('proc \times 'proc)$ *set* $\Rightarrow$ *'proc processCalculus* $\Rightarrow$ *bool*
where
weak-reduction-bisimulation Rel Cal $\equiv$
$(\forall P\ Q\ P'.\ (P, Q) \in Rel \wedge P \longmapsto Cal{*}\ P' \longrightarrow (\exists Q'.\ Q \longmapsto Cal{*}\ Q' \wedge (P', Q') \in Rel))$
$\wedge (\forall P\ Q\ Q'.\ (P, Q) \in Rel \wedge Q \longmapsto Cal{*}\ Q' \longrightarrow (\exists P'.\ P \longmapsto Cal{*}\ P' \wedge (P', Q') \in Rel))$

A weak barbed bisimulation is weak reduction bisimulation that weakly respects barbs.

abbreviation *weak-barbed-bisimulation*
:: $('proc \times 'proc)$ *set* $\Rightarrow$ $('proc, 'barbs)$ *calculusWithBarbs* $\Rightarrow$ *bool*
where
weak-barbed-bisimulation Rel CWB $\equiv$
weak-reduction-bisimulation Rel (*Calculus CWB*) $\wedge$ *rel-weakly-respects-barbs Rel CWB*

A symetric weak simulation is a weak bisimulation.

lemma *symm-weak-reduction-simulation-is-bisimulation*:
fixes *Rel* :: $('proc \times 'proc)$ *set*
and *Cal* :: *'proc processCalculus*
assumes *sym Rel*
and *weak-reduction-simulation Rel Cal*
shows *weak-reduction-bisimulation Rel Cal*
$\langle proof \rangle$

lemma *symm-weak-barbed-simulation-is-bisimulation*:
fixes *Rel* :: $('proc \times 'proc)$ *set*
and *CWB* :: $('proc, 'barbs)$ *calculusWithBarbs*
assumes *sym Rel*
and *weak-barbed-simulation Rel Cal*
shows *weak-barbed-bisimulation Rel Cal*
$\langle proof \rangle$

If a relation as well as its inverse are weak simulations, then this relation is a weak bisimulation.

lemma *weak-reduction-simulations-impl-bisimulation*:
fixes *Rel* :: ($'proc \times 'proc$) *set*
and *Cal* :: $'proc$ *processCalculus*
assumes *sim*: *weak-reduction-simulation Rel Cal*
and *simInv*: *weak-reduction-simulation* (Rel^{-1}) *Cal*
shows *weak-reduction-bisimulation Rel Cal*
⟨*proof*⟩

lemma *weak-reduction-bisimulations-impl-inverse-is-simulation*:
fixes *Rel* :: ($'proc \times 'proc$) *set*
and *Cal* :: $'proc$ *processCalculus*
assumes *bisim*: *weak-reduction-bisimulation Rel Cal*
shows *weak-reduction-simulation* (Rel^{-1}) *Cal*
⟨*proof*⟩

lemma *weak-reduction-simulations-iff-bisimulation*:
fixes *Rel* :: ($'proc \times 'proc$) *set*
and *Cal* :: $'proc$ *processCalculus*
shows (*weak-reduction-simulation Rel Cal* $\wedge$ *weak-reduction-simulation* (Rel^{-1}) *Cal*)
= *weak-reduction-bisimulation Rel Cal*
⟨*proof*⟩

lemma *weak-barbed-simulations-iff-bisimulation*:
fixes *Rel* :: ($'proc \times 'proc$) *set*
and *CWB* :: ($'proc$, $'barbs$) *calculusWithBarbs*
shows (*weak-barbed-simulation Rel CWB* $\wedge$ *weak-barbed-simulation* (Rel^{-1}) *CWB*)
= *weak-barbed-bisimulation Rel CWB*
⟨*proof*⟩

A weak bisimulation is a weak correspondence simulation.

lemma *weak-reduction-bisimulation-is-correspondence-simulation*:
fixes *Rel* :: ($'proc \times 'proc$) *set*
and *Cal* :: $'proc$ *processCalculus*
assumes *bisim*: *weak-reduction-bisimulation Rel Cal*
shows *weak-reduction-correspondence-simulation Rel Cal*
⟨*proof*⟩

lemma *weak-barbed-bisimulation-is-correspondence-simulation*:
fixes *Rel* :: ($'proc \times 'proc$) *set*
and *CWB* :: ($'proc$, $'barbs$) *calculusWithBarbs*
assumes *bisim*: *weak-barbed-bisimulation Rel CWB*
shows *weak-barbed-correspondence-simulation Rel CWB*
⟨*proof*⟩

The reflexive, symmetric, and/or transitive closure of a weak bisimulation is a weak bisimulation.

lemma *weak-reduction-bisimulation-and-closures*:
fixes *Rel* :: ($'proc \times 'proc$) *set*
and *Cal* :: $'proc$ *processCalculus*
assumes *bisim*: *weak-reduction-bisimulation Rel Cal*
shows *weak-reduction-bisimulation* ($Rel^{=}$) *Cal*
and *weak-reduction-bisimulation* (*symcl Rel*) *Cal*
and *weak-reduction-bisimulation* (Rel^{+}) *Cal*
and *weak-reduction-bisimulation* (*symcl* ($Rel^{=}$)) *Cal*
and *weak-reduction-bisimulation* (Rel^{*}) *Cal*
and *weak-reduction-bisimulation* ((*symcl* ($Rel^{=}$))$^{+}$) *Cal*
⟨*proof*⟩

lemma *weak-barbed-bisimulation-and-closures*:
fixes *Rel* :: ($'proc \times 'proc$) *set*
and *CWB* :: ($'proc$, $'barbs$) *calculusWithBarbs*

assumes *bisim*: *weak-barbed-bisimulation Rel CWB*
shows *weak-barbed-bisimulation* $(Rel^{=})$ *CWB*
and *weak-barbed-bisimulation* (*symcl Rel*) *CWB*
and *weak-barbed-bisimulation* (Rel^{+}) *CWB*
and *weak-barbed-bisimulation* (*symcl* $(Rel^{=})$) *CWB*
and *weak-barbed-bisimulation* (Rel^{*}) *CWB*
and *weak-barbed-bisimulation* $((symcl\ (Rel^{=}))^{+})$ *CWB*
⟨*proof*⟩

A strong reduction bisimulation is relation R such that (1) if (P, Q) in R and P' is a derivative of P then there exists some Q' such that Q' is a derivative of Q and (P', Q') in R, and (2) if (P, Q) in R and Q' is a derivative of Q then there exists some P' such that P' is a derivative of P and (P', Q') in R.

abbreviation *strong-reduction-bisimulation*
$:: ('proc \times 'proc)\ set \Rightarrow 'proc\ processCalculus \Rightarrow bool$
where
strong-reduction-bisimulation Rel Cal ≡
$(\forall P\ Q\ P'.\ (P,\ Q) \in Rel \wedge P \longmapsto Cal\ P' \longrightarrow (\exists Q'.\ Q \longmapsto Cal\ Q' \wedge (P',\ Q') \in Rel))$
$\wedge (\forall P\ Q\ Q'.\ (P,\ Q) \in Rel \wedge Q \longmapsto Cal\ Q' \longrightarrow (\exists P'.\ P \longmapsto Cal\ P' \wedge (P',\ Q') \in Rel))$

A strong barbed bisimulation is strong reduction bisimulation that respects barbs.

abbreviation *strong-barbed-bisimulation*
$:: ('proc \times 'proc)\ set \Rightarrow ('proc,\ 'barbs)\ calculusWithBarbs \Rightarrow bool$
where
strong-barbed-bisimulation Rel CWB ≡
strong-reduction-bisimulation Rel (*Calculus CWB*) ∧ *rel-respects-barbs Rel CWB*

A symetric strong simulation is a strong bisimulation.

lemma *symm-strong-reduction-simulation-is-bisimulation*:
fixes *Rel* $:: ('proc \times 'proc)\ set$
and *Cal* $:: 'proc\ processCalculus$
assumes *sym Rel*
and *strong-reduction-simulation Rel Cal*
shows *strong-reduction-bisimulation Rel Cal*
⟨*proof*⟩

lemma *symm-strong-barbed-simulation-is-bisimulation*:
fixes *Rel* $:: ('proc \times 'proc)\ set$
and *CWB* $:: ('proc,\ 'barbs)\ calculusWithBarbs$
assumes *sym Rel*
and *strong-barbed-simulation Rel CWB*
shows *strong-barbed-bisimulation Rel CWB*
⟨*proof*⟩

If a relation as well as its inverse are strong simulations, then this relation is a strong bisimulation.

lemma *strong-reduction-simulations-impl-bisimulation*:
fixes *Rel* $:: ('proc \times 'proc)\ set$
and *Cal* $:: 'proc\ processCalculus$
assumes *sim*: *strong-reduction-simulation Rel Cal*
and *simInv*: *strong-reduction-simulation* (Rel^{-1}) *Cal*
shows *strong-reduction-bisimulation Rel Cal*
⟨*proof*⟩

lemma *strong-reduction-bisimulations-impl-inverse-is-simulation*:
fixes *Rel* $:: ('proc \times 'proc)\ set$
and *Cal* $:: 'proc\ processCalculus$
assumes *bisim*: *strong-reduction-bisimulation Rel Cal*
shows *strong-reduction-simulation* (Rel^{-1}) *Cal*
⟨*proof*⟩

lemma *strong-reduction-simulations-iff-bisimulation*:
fixes *Rel* :: ($'proc \times 'proc$) *set*
and *Cal* :: $'proc$ *processCalculus*
shows (*strong-reduction-simulation Rel Cal* $\wedge$ *strong-reduction-simulation* (Rel^{-1}) *Cal*)
= *strong-reduction-bisimulation Rel Cal*
⟨*proof*⟩

lemma *strong-barbed-simulations-iff-bisimulation*:
fixes *Rel* :: ($'proc \times 'proc$) *set*
and *CWB* :: ($'proc$, $'barbs$) *calculusWithBarbs*
shows (*strong-barbed-simulation Rel CWB* $\wedge$ *strong-barbed-simulation* (Rel^{-1}) *CWB*)
= *strong-barbed-bisimulation Rel CWB*
⟨*proof*⟩

A strong bisimulation is a weak bisimulation.

lemma *strong-impl-weak-reduction-bisimulation*:
fixes *Rel* :: ($'proc \times 'proc$) *set*
and *Cal* :: $'proc$ *processCalculus*
assumes *bisim*: *strong-reduction-bisimulation Rel Cal*
shows *weak-reduction-bisimulation Rel Cal*
⟨*proof*⟩

lemma *strong-barbed-bisimulation-impl-weak-respection-of-barbs*:
fixes *Rel* :: ($'proc \times 'proc$) *set*
and *CWB* :: ($'proc$, $'barbs$) *calculusWithBarbs*
assumes *bisim*: *strong-barbed-bisimulation Rel CWB*
shows *rel-weakly-respects-barbs Rel CWB*
⟨*proof*⟩

lemma *strong-impl-weak-barbed-bisimulation*:
fixes *Rel* :: ($'proc \times 'proc$) *set*
and *CWB* :: ($'proc$, $'barbs$) *calculusWithBarbs*
assumes *bisim*: *strong-barbed-bisimulation Rel CWB*
shows *weak-barbed-bisimulation Rel CWB*
⟨*proof*⟩

The reflexive, symmetric, and/or transitive closure of a strong bisimulation is a strong bisimulation.

lemma *strong-reduction-bisimulation-and-closures*:
fixes *Rel* :: ($'proc \times 'proc$) *set*
and *Cal* :: $'proc$ *processCalculus*
assumes *bisim*: *strong-reduction-bisimulation Rel Cal*
shows *strong-reduction-bisimulation* ($Rel^{=}$) *Cal*
and *strong-reduction-bisimulation* (*symcl Rel*) *Cal*
and *strong-reduction-bisimulation* (Rel^{+}) *Cal*
and *strong-reduction-bisimulation* (*symcl* ($Rel^{=}$)) *Cal*
and *strong-reduction-bisimulation* (Rel^{*}) *Cal*
and *strong-reduction-bisimulation* ((*symcl* ($Rel^{=}$))$^{+}$) *Cal*
⟨*proof*⟩

lemma *strong-barbed-bisimulation-and-closures*:
fixes *Rel* :: ($'proc \times 'proc$) *set*
and *CWB* :: ($'proc$, $'barbs$) *calculusWithBarbs*
assumes *bisim*: *strong-barbed-bisimulation Rel CWB*
shows *strong-barbed-bisimulation* ($Rel^{=}$) *CWB*
and *strong-barbed-bisimulation* (*symcl Rel*) *CWB*
and *strong-barbed-bisimulation* (Rel^{+}) *CWB*
and *strong-barbed-bisimulation* (*symcl* ($Rel^{=}$)) *CWB*
and *strong-barbed-bisimulation* (Rel^{*}) *CWB*
and *strong-barbed-bisimulation* ((*symcl* ($Rel^{=}$))$^{+}$) *CWB*
⟨*proof*⟩

3.6 Step Closure of Relations

The step closure of a relation on process terms is the transitive closure of the union of the relation and the inverse of the reduction relation of the respective calculus.

inductive-set *stepsClosure* :: $('a \times 'a)$ *set* $\Rightarrow$ *'a processCalculus* $\Rightarrow$ $('a \times 'a)$ *set*
  for *Rel* :: $('a \times 'a)$ *set*
  and *Cal* :: *'a processCalculus*
 where
 rel: $(P,\ Q) \in Rel \Longrightarrow (P,\ Q) \in$ *stepsClosure Rel Cal* |
 steps: $P \longmapsto Cal{*}\ P' \Longrightarrow (P',\ P) \in$ *stepsClosure Rel Cal* |
 trans: $[\![(P,\ Q) \in$ *stepsClosure Rel Cal*; $(Q,\ R) \in$ *stepsClosure Rel Cal*$]\!]$
   $\Longrightarrow (P,\ R) \in$ *stepsClosure Rel Cal*

abbreviation *stepsClosureInfix* ::
  $'a \Rightarrow ('a \times 'a)\ set \Rightarrow 'a\ processCalculus \Rightarrow 'a \Rightarrow bool$ (- $\mathcal{R}\mapsto$<-,-> - $[75,\ 75,\ 75,\ 75]\ 80$)
 where
 $P\ \mathcal{R}\mapsto<Rel,Cal>\ Q \equiv (P,\ Q) \in$ *stepsClosure Rel Cal*

Applying the steps closure twice does not change the relation.

lemma *steps-closure-of-steps-closure*:
 fixes *Rel* :: $('a \times 'a)$ *set*
  and *Cal* :: *'a processCalculus*
 shows *stepsClosure* (*stepsClosure Rel Cal*) *Cal* = *stepsClosure Rel Cal*
⟨*proof*⟩

The steps closure is a preorder.

lemma *stepsClosure-refl*:
 fixes *Rel* :: $('a \times 'a)$ *set*
  and *Cal* :: *'a processCalculus*
 shows *refl* (*stepsClosure Rel Cal*)
  ⟨*proof*⟩

lemma *refl-trans-closure-of-rel-impl-steps-closure*:
 fixes *Rel* :: $('a \times 'a)$ *set*
  and *Cal* :: *'a processCalculus*
  and *P Q* :: *'a*
 assumes $(P,\ Q) \in Rel^*$
 shows $P\ \mathcal{R}\mapsto<Rel,Cal>\ Q$
  ⟨*proof*⟩

The steps closure of a relation is always a weak reduction simulation.

lemma *steps-closure-is-weak-reduction-simulation*:
 fixes *Rel* :: $('a \times 'a)$ *set*
  and *Cal* :: *'a processCalculus*
 shows *weak-reduction-simulation* (*stepsClosure Rel Cal*) *Cal*
⟨*proof*⟩

If Rel is a weak simulation and its inverse is a weak contrasimulation, then the steps closure of Rel is a contrasimulation.

lemma *inverse-contrasimulation-impl-reverse-pair-in-steps-closure*:
 fixes *Rel* :: $('a \times 'a)$ *set*
  and *Cal* :: *'a processCalculus*
  and *P Q* :: *'a*
 assumes *con*: *weak-reduction-contrasimulation* (Rel^{-1}) *Cal*
   and *pair*: $(P,\ Q) \in Rel$
 shows $Q\ \mathcal{R}\mapsto<Rel,Cal>\ P$
⟨*proof*⟩

lemma *simulation-and-inverse-contrasimulation-impl-steps-closure-is-contrasimulation*:

fixes *Rel* :: $('a \times 'a)$ *set*
and *Cal* :: $'a$ *processCalculus*
assumes *sim*: *weak-reduction-simulation Rel Cal*
and *con*: *weak-reduction-contrasimulation* (Rel^{-1}) *Cal*
shows *weak-reduction-contrasimulation* (*stepsClosure Rel Cal*) *Cal*
⟨*proof*⟩

Accordingly, if Rel is a weak simulation and its inverse is a weak contrasimulation, then the steps closure of Rel is a coupled simulation.

lemma *simulation-and-inverse-contrasimulation-impl-steps-closure-is-coupled-simulation*:
fixes *Rel* :: $('a \times 'a)$ *set*
and *Cal* :: $'a$ *processCalculus*
assumes *sim*: *weak-reduction-simulation Rel Cal*
and *con*: *weak-reduction-contrasimulation* (Rel^{-1}) *Cal*
shows *weak-reduction-coupled-simulation* (*stepsClosure Rel Cal*) *Cal*
⟨*proof*⟩

If the relation that is closed under steps is a (contra)simulation, then we can conlude from a pair in the closure on a pair in the original relation.

lemma *stepsClosure-simulation-impl-refl-trans-closure-of-Rel*:
fixes *Rel* :: $('a \times 'a)$ *set*
and *Cal* :: $'a$ *processCalculus*
and $P\ Q :: 'a$
assumes *A1*: $P\ \mathcal{R}{\mapsto}{<}Rel,Cal{>}\ Q$
and *A2*: *weak-reduction-simulation Rel Cal*
shows $\exists\, Q'.\ Q \longmapsto Cal*\ Q' \wedge (P,\ Q') \in Rel^*$
⟨*proof*⟩

lemma *stepsClosure-contrasimulation-impl-refl-trans-closure-of-Rel*:
fixes *Rel* :: $('a \times 'a)$ *set*
and *Cal* :: $'a$ *processCalculus*
and $P\ Q :: 'a$
assumes *A1*: $P\ \mathcal{R}{\mapsto}{<}Rel,Cal{>}\ Q$
and *A2*: *weak-reduction-contrasimulation Rel Cal*
shows $\exists\, Q'.\ Q \longmapsto Cal*\ Q' \wedge (Q',\ P) \in Rel^*$
⟨*proof*⟩

lemma *stepsClosure-contrasimulation-of-inverse-impl-refl-trans-closure-of-Rel*:
fixes *Rel* :: $('a \times 'a)$ *set*
and *Cal* :: $'a$ *processCalculus*
and $P\ Q :: 'a$
assumes *A1*: $P\ \mathcal{R}{\mapsto}{<}Rel^{-1},Cal{>}\ Q$
and *A2*: *weak-reduction-contrasimulation* (Rel^{-1}) *Cal*
shows $\exists\, Q'.\ Q \longmapsto Cal*\ Q' \wedge (P,\ Q') \in Rel^*$
⟨*proof*⟩

end
theory *Encodings*
imports *ProcessCalculi*
begin

4 Encodings

In the simplest case an encoding from a source into a target language is a mapping from source into target terms. Encodability criteria describe properties on such mappings. To analyse encodability criteria we map them on conditions on relations between source and target terms. More precisely, we consider relations on pairs of the disjoint union of source and target terms. We denote this disjoint union of source and target terms by Proc.

datatype $('procS, 'procT)$ *Proc* =
  SourceTerm $'procS$ |
  TargetTerm $'procT$

definition *STCal*
  :: $'procS$ *processCalculus* $\Rightarrow$ $'procT$ *processCalculus*
    $\Rightarrow$ $(('procS, 'procT)$ *Proc*) *processCalculus*
  where
  STCal Source Target $\equiv$
  $(\!|$*Reductions* = $\lambda P\ P'$.
  $(\exists\, SP\ SP'.\ P = SourceTerm\ SP \wedge P' = SourceTerm\ SP' \wedge Reductions\ Source\ SP\ SP') \vee$
  $(\exists\, TP\ TP'.\ P = TargetTerm\ TP \wedge P' = TargetTerm\ TP' \wedge Reductions\ Target\ TP\ TP')|\!)$

definition *STCalWB*
  :: $('procS, 'barbs)$ *calculusWithBarbs* $\Rightarrow$ $('procT, 'barbs)$ *calculusWithBarbs*
    $\Rightarrow$ $(('procS, 'procT)$ *Proc*, $'barbs)$ *calculusWithBarbs*
  where
  STCalWB Source Target $\equiv$
  $(\!|Calculus = STCal\ (calculusWithBarbs.Calculus\ Source)\ (calculusWithBarbs.Calculus\ Target),$
  $HasBarb\ \ = \lambda P\ a.\ (\exists\, SP.\ P = SourceTerm\ SP \wedge (calculusWithBarbs.HasBarb\ Source)\ SP\ a) \vee$
  $(\exists\, TP.\ P = TargetTerm\ TP \wedge (calculusWithBarbs.HasBarb\ Target)\ TP\ a)|\!)$

An encoding consists of a source language, a target language, and a mapping from source into target terms.

locale *encoding* =
  fixes *Source* :: $'procS$ *processCalculus*
    and *Target* :: $'procT$ *processCalculus*
    and *Enc* :: $'procS \Rightarrow 'procT$
begin

abbreviation *enc* :: $'procS \Rightarrow 'procT$ ($[\![-]\!]$ [65] 70) **where**
  $[\![S]\!] \equiv Enc\ S$

abbreviation *isSource* :: $('procS, 'procT)\ Proc \Rightarrow bool$ (- $\in$ *ProcS* [70] 80) **where**
  $P \in ProcS \equiv (\exists\, S.\ P = SourceTerm\ S)$

abbreviation *isTarget* :: $('procS, 'procT)\ Proc \Rightarrow bool$ (- $\in$ *ProcT* [70] 80) **where**
  $P \in ProcT \equiv (\exists\, T.\ P = TargetTerm\ T)$

abbreviation *getSource*
  :: $'procS \Rightarrow ('procS, 'procT)\ Proc \Rightarrow bool$ (- $\in$S - [70, 70] 80)
  where
  $S \in S\ P \equiv (P = SourceTerm\ S)$

abbreviation *getTarget*
  :: $'procT \Rightarrow ('procS, 'procT)\ Proc \Rightarrow bool$ (- $\in$T - [70, 70] 80)
  where
  $T \in T\ P \equiv (P = TargetTerm\ T)$

A step of a term in Proc is either a source term step or a target term step.

abbreviation *stepST*
  :: $('procS, 'procT)\ Proc \Rightarrow ('procS, 'procT)\ Proc \Rightarrow bool$ (- $\longmapsto$ST - [70, 70] 80)
  where
  $P \longmapsto ST\ P' \equiv$
  $(\exists\, S\ S'.\ S \in S\ P \wedge S' \in S\ P' \wedge S \longmapsto Source\ S') \vee (\exists\, T\ T'.\ T \in T\ P \wedge T' \in T\ P' \wedge T \longmapsto Target\ T')$

lemma *stepST-STCal-step*:
  fixes $P\ P'$ :: $('procS, 'procT)$ *Proc*
  shows $P \longmapsto (STCal\ Source\ Target)\ P' = P \longmapsto ST\ P'$
    $\langle proof \rangle$

lemma *STStep-step*:
  fixes S :: $'procS$
    and T :: $'procT$
    and P' :: $('procS, 'procT)$ *Proc*
  shows *SourceTerm* $S \longmapsto ST\ P' = (\exists S'.\ S' \in S\ P' \wedge S \longmapsto Source\ S')$
    and *TargetTerm* $T \longmapsto ST\ P' = (\exists T'.\ T' \in T\ P' \wedge T \longmapsto Target\ T')$
    ⟨*proof*⟩

lemma *STCal-step*:
  fixes S :: $'procS$
    and T :: $'procT$
    and P' :: $('procS, 'procT)$ *Proc*
  shows *SourceTerm* $S \longmapsto (STCal\ Source\ Target)\ P' = (\exists S'.\ S' \in S\ P' \wedge S \longmapsto Source\ S')$
    and *TargetTerm* $T \longmapsto (STCal\ Source\ Target)\ P' = (\exists T'.\ T' \in T\ P' \wedge T \longmapsto Target\ T')$
    ⟨*proof*⟩

A sequence of steps of a term in Proc is either a sequence of source term steps or a sequence of target term steps.

abbreviation *stepsST*
    :: $('procS, 'procT)$ *Proc* $\Rightarrow$ $('procS, 'procT)$ *Proc* $\Rightarrow$ *bool* $(- \longmapsto ST{*}\ -\ [70, 70]\ 80)$
  where
  $P \longmapsto ST{*}\ P' \equiv$
  $(\exists S\ S'.\ S \in S\ P \wedge S' \in S\ P' \wedge S \longmapsto Source{*}\ S') \vee (\exists T\ T'.\ T \in T\ P \wedge T' \in T\ P' \wedge T \longmapsto Target{*}\ T')$

lemma *STSteps-steps*:
  fixes S :: $'procS$
    and T :: $'procT$
    and P' :: $('procS, 'procT)$ *Proc*
  shows *SourceTerm* $S \longmapsto ST{*}\ P' = (\exists S'.\ S' \in S\ P' \wedge S \longmapsto Source{*}\ S')$
    and *TargetTerm* $T \longmapsto ST{*}\ P' = (\exists T'.\ T' \in T\ P' \wedge T \longmapsto Target{*}\ T')$
    ⟨*proof*⟩

lemma *STCal-steps*:
  fixes S :: $'procS$
    and T :: $'procT$
    and P' :: $('procS, 'procT)$ *Proc*
  shows *SourceTerm* $S \longmapsto (STCal\ Source\ Target){*}\ P' = (\exists S'.\ S' \in S\ P' \wedge S \longmapsto Source{*}\ S')$
    and *TargetTerm* $T \longmapsto (STCal\ Source\ Target){*}\ P' = (\exists T'.\ T' \in T\ P' \wedge T \longmapsto Target{*}\ T')$
⟨*proof*⟩

lemma *stepsST-STCal-steps*:
  fixes $P\ P'$ :: $('procS, 'procT)$ *Proc*
  shows $P \longmapsto (STCal\ Source\ Target){*}\ P' = P \longmapsto ST{*}\ P'$
⟨*proof*⟩

lemma *stepsST-refl*:
  fixes P :: $('procS, 'procT)$ *Proc*
  shows $P \longmapsto ST{*}\ P$
    ⟨*proof*⟩

lemma *stepsST-add*:
  fixes $P\ Q\ R$ :: $('procS, 'procT)$ *Proc*
  assumes *A1*: $P \longmapsto ST{*}\ Q$
      and *A2*: $Q \longmapsto ST{*}\ R$
  shows $P \longmapsto ST{*}\ R$
⟨*proof*⟩

A divergent term of Proc is either a divergent source term or a divergent target term.

abbreviation *divergentST*

$:: ('procS, 'procT)\ Proc \Rightarrow bool\ (\text{-} \longmapsto ST\omega\ [70]\ 80)$
where
$P \longmapsto ST\omega \equiv (\exists\, S.\ S \in S\ P \wedge S \longmapsto (Source)\omega) \vee (\exists\, T.\ T \in T\ P \wedge T \longmapsto (Target)\omega)$

lemma *STCal-divergent*:
fixes $S\ ::\ 'procS$
and $T\ ::\ 'procT$
shows $SourceTerm\ S \longmapsto (STCal\ Source\ Target)\omega = S \longmapsto (Source)\omega$
and $TargetTerm\ T \longmapsto (STCal\ Source\ Target)\omega = T \longmapsto (Target)\omega$
$\langle proof \rangle$

lemma *divergentST-STCal-divergent*:
fixes $P :: ('procS, 'procT)\ Proc$
shows $P \longmapsto (STCal\ Source\ Target)\omega = P \longmapsto ST\omega$
$\langle proof \rangle$

Similar to relations we define what it means for an encoding to preserve, reflect, or respect a predicate. An encoding preserves some predicate P if P(S) implies P(enc S) for all source terms S.

abbreviation *enc-preserves-pred* $:: (('procS, 'procT)\ Proc \Rightarrow bool) \Rightarrow bool$ **where**
$enc\text{-}preserves\text{-}pred\ Pred \equiv \forall\, S.\ Pred\ (SourceTerm\ S) \longrightarrow Pred\ (TargetTerm\ (\llbracket S \rrbracket))$

abbreviation *enc-preserves-binary-pred*
$:: (('procS, 'procT)\ Proc \Rightarrow 'b \Rightarrow bool) \Rightarrow bool$
where
$enc\text{-}preserves\text{-}binary\text{-}pred\ Pred \equiv \forall\, S\ x.\ Pred\ (SourceTerm\ S)\ x \longrightarrow Pred\ (TargetTerm\ (\llbracket S \rrbracket))\ x$

An encoding reflects some predicate P if P(S) implies P(enc S) for all source terms S.

abbreviation *enc-reflects-pred* $:: (('procS, 'procT)\ Proc \Rightarrow bool) \Rightarrow bool$ **where**
$enc\text{-}reflects\text{-}pred\ Pred \equiv \forall\, S.\ Pred\ (TargetTerm\ (\llbracket S \rrbracket)) \longrightarrow Pred\ (SourceTerm\ S)$

abbreviation *enc-reflects-binary-pred*
$:: (('procS, 'procT)\ Proc \Rightarrow 'b \Rightarrow bool) \Rightarrow bool$
where
$enc\text{-}reflects\text{-}binary\text{-}pred\ Pred \equiv \forall\, S\ x.\ Pred\ (TargetTerm\ (\llbracket S \rrbracket))\ x \longrightarrow Pred\ (SourceTerm\ S)\ x$

An encoding respects a predicate if it preserves and reflects it.

abbreviation *enc-respects-pred* $:: (('procS, 'procT)\ Proc \Rightarrow bool) \Rightarrow bool$ **where**
$enc\text{-}respects\text{-}pred\ Pred \equiv enc\text{-}preserves\text{-}pred\ Pred \wedge enc\text{-}reflects\text{-}pred\ Pred$

abbreviation *enc-respects-binary-pred*
$:: (('procS, 'procT)\ Proc \Rightarrow 'b \Rightarrow bool) \Rightarrow bool$
where
$enc\text{-}respects\text{-}binary\text{-}pred\ Pred \equiv$
$enc\text{-}preserves\text{-}binary\text{-}pred\ Pred \wedge enc\text{-}reflects\text{-}binary\text{-}pred\ Pred$

end

To compare source terms and target terms w.r.t. their barbs or observables we assume that each languages defines its own predicate for the existence of barbs.

locale *encoding-wrt-barbs* =
encoding Source Target Enc
for $Source :: 'procS\ processCalculus$
and $Target :: 'procT\ processCalculus$
and $Enc\ \ :: 'procS \Rightarrow 'procT +$
fixes $SWB :: ('procS, 'barbs)\ calculusWithBarbs$
and $TWB :: ('procT, 'barbs)\ calculusWithBarbs$
assumes *calS*: $calculusWithBarbs.Calculus\ SWB = Source$
and *calT*: $calculusWithBarbs.Calculus\ TWB = Target$
begin

lemma *STCalWB-STCal*:
shows $\mathit{Calculus}\ (\mathit{STCalWB}\ \mathit{SWB}\ \mathit{TWB}) = \mathit{STCal}\ \mathit{Source}\ \mathit{Target}$
⟨*proof*⟩

We say a term P of Proc has some barbs a if either P is a source term that has barb a or P is a target term that has the barb b. For simplicity we assume that the sets of barbs is large enough to contain all barbs of the source terms, the target terms, and all barbs they might have in common.

abbreviation *hasBarbST*
$::\ ('procS,\ 'procT)\ Proc \Rightarrow\ 'barbs \Rightarrow bool\ (\text{-}{\downarrow}.\text{-}\ [70,\ 70]\ 80)$
where
$P{\downarrow}.a \equiv (\exists S.\ S \in S\ P \wedge S{\downarrow}<SWB>a) \vee (\exists T.\ T \in T\ P \wedge T{\downarrow}<TWB>a)$

lemma *STCalWB-hasBarbST*:
fixes $P :: ('procS,\ 'procT)\ Proc$
and $a ::\ 'barbs$
shows $P{\downarrow}<STCalWB\ SWB\ TWB>a = P{\downarrow}.a$
⟨*proof*⟩

lemma *preservation-of-barbs-in-barbed-encoding*:
fixes $Rel :: ((('procS,\ 'procT)\ Proc \times ('procS,\ 'procT)\ Proc)\ set$
and $P\ Q :: ('procS,\ 'procT)\ Proc$
and $a ::\ 'barbs$
assumes *preservation*: *rel-preserves-barbs Rel* (*STCalWB SWB TWB*)
and *rel*: $(P,\ Q) \in Rel$
and *barb*: $P{\downarrow}.a$
shows $Q{\downarrow}.a$
⟨*proof*⟩

lemma *reflection-of-barbs-in-barbed-encoding*:
fixes $Rel :: ((('procS,\ 'procT)\ Proc \times ('procS,\ 'procT)\ Proc)\ set$
and $P\ Q :: ('procS,\ 'procT)\ Proc$
and $a ::\ 'barbs$
assumes *reflection*: *rel-reflects-barbs Rel* (*STCalWB SWB TWB*)
and *rel*: $(P,\ Q) \in Rel$
and *barb*: $Q{\downarrow}.a$
shows $P{\downarrow}.a$
⟨*proof*⟩

lemma *respection-of-barbs-in-barbed-encoding*:
fixes $Rel :: ((('procS,\ 'procT)\ Proc \times ('procS,\ 'procT)\ Proc)\ set$
and $P\ Q :: ('procS,\ 'procT)\ Proc$
and $a ::\ 'barbs$
assumes *respection*: *rel-respects-barbs Rel* (*STCalWB SWB TWB*)
and *rel*: $(P,\ Q) \in Rel$
shows $P{\downarrow}.a = Q{\downarrow}.a$
⟨*proof*⟩

A term P of Proc reaches a barb a if either P is a source term that reaches a or P is a target term that reaches a.

abbreviation *reachesBarbST*
$::\ ('procS,\ 'procT)\ Proc \Rightarrow\ 'barbs \Rightarrow bool\ (\text{-}{\Downarrow}.\text{-}\ [70,\ 70]\ 80)$
where
$P{\Downarrow}.a \equiv (\exists S.\ S \in S\ P \wedge S{\Downarrow}<SWB>a) \vee (\exists T.\ T \in T\ P \wedge T{\Downarrow}<TWB>a)$

lemma *STCalWB-reachesBarbST*:
fixes $P :: ('procS,\ 'procT)\ Proc$
and $a ::\ 'barbs$
shows $P{\Downarrow}<STCalWB\ SWB\ TWB>a = P{\Downarrow}.a$
⟨*proof*⟩

lemma *weak-preservation-of-barbs-in-barbed-encoding*:
fixes *Rel* :: (($'procS$, $'procT$) *Proc* × ($'procS$, $'procT$) *Proc*) *set*
and *P Q* :: ($'procS$, $'procT$) *Proc*
and *a* :: $'barbs$
assumes *preservation*: *rel-weakly-preserves-barbs Rel* (*STCalWB SWB TWB*)
and *rel*: $(P, Q) \in Rel$
and *barb*: $P{\Downarrow}.a$
shows $Q{\Downarrow}.a$
⟨*proof*⟩

lemma *weak-reflection-of-barbs-in-barbed-encoding*:
fixes *Rel* :: (($'procS$, $'procT$) *Proc* × ($'procS$, $'procT$) *Proc*) *set*
and *P Q* :: ($'procS$, $'procT$) *Proc*
and *a* :: $'barbs$
assumes *reflection*: *rel-weakly-reflects-barbs Rel* (*STCalWB SWB TWB*)
and *rel*: $(P, Q) \in Rel$
and *barb*: $Q{\Downarrow}.a$
shows $P{\Downarrow}.a$
⟨*proof*⟩

lemma *weak-respection-of-barbs-in-barbed-encoding*:
fixes *Rel* :: (($'procS$, $'procT$) *Proc* × ($'procS$, $'procT$) *Proc*) *set*
and *P Q* :: ($'procS$, $'procT$) *Proc*
and *a* :: $'barbs$
assumes *respection*: *rel-weakly-respects-barbs Rel* (*STCalWB SWB TWB*)
and *rel*: $(P, Q) \in Rel$
shows $P{\Downarrow}.a = Q{\Downarrow}.a$
⟨*proof*⟩

end

end
theory *SourceTargetRelation*
imports *Encodings SimulationRelations*
begin

5 Relation between Source and Target Terms

5.1 Relations Induced by the Encoding Function

We map encodability criteria on conditions of relations between source and target terms. The encoding function itself induces such relations. To analyse the preservation of source term behaviours we use relations that contain the pairs (S, enc S) for all source terms S.

inductive-set (**in** *encoding*) *indRelR*
:: ((($'procS$, $'procT$) *Proc*) × (($'procS$, $'procT$) *Proc*)) *set*
where
encR: (*SourceTerm S*, *TargetTerm* ($[\![S]\!]$)) ∈ *indRelR*

abbreviation (**in** *encoding*) *indRelRinfix* ::
($'procS$, $'procT$) *Proc* ⇒ ($'procS$, $'procT$) *Proc* ⇒ *bool* (- $\mathcal{R}[\![\cdot]\!]R$ - [*75*, *75*] *80*)
where
$P\ \mathcal{R}[\![\cdot]\!]R\ Q \equiv (P, Q) \in indRelR$

inductive-set (**in** *encoding*) *indRelRPO*
:: ((($'procS$, $'procT$) *Proc*) × (($'procS$, $'procT$) *Proc*)) *set*
where
encR: (*SourceTerm S*, *TargetTerm* ($[\![S]\!]$)) ∈ *indRelRPO* |
source: (*SourceTerm S*, *SourceTerm S*) ∈ *indRelRPO* |

target: (*TargetTerm T*, *TargetTerm T*) $\in$ *indRelRPO* |
trans: $[\![(P, Q) \in indRelRPO; (Q, R) \in indRelRPO]\!] \Longrightarrow (P, R) \in indRelRPO$

abbreviation (**in** *encoding*) *indRelRPOinfix* ::
$('procS, 'procT)\ Proc \Rightarrow ('procS, 'procT)\ Proc \Rightarrow bool$ (- $\lesssim[\![\cdot]\!]R$ - [75, 75] 80)
where
$P \lesssim[\![\cdot]\!]R\ Q \equiv (P, Q) \in indRelRPO$

lemma (**in** *encoding*) *indRelRPO-refl*:
shows *refl indRelRPO*
⟨*proof*⟩

lemma (**in** *encoding*) *indRelRPO-is-preorder*:
shows *preorder indRelRPO*
⟨*proof*⟩

lemma (**in** *encoding*) *refl-trans-closure-of-indRelR*:
shows $indRelRPO = indRelR^*$
⟨*proof*⟩

The relation indRelR is the smallest relation that relates all source terms and their literal translations. Thus there exists a relation that relates source terms and their literal translations and satisfies some predicate on its pairs iff the predicate holds for the pairs of indRelR.

lemma (**in** *encoding*) *indRelR-impl-exists-source-target-relation*:
fixes *PredA* :: $(('procS, 'procT)\ Proc \times ('procS, 'procT)\ Proc)\ set \Rightarrow bool$
and *PredB* :: $(('procS, 'procT)\ Proc \times ('procS, 'procT)\ Proc) \Rightarrow bool$
shows $PredA\ indRelR \Longrightarrow \exists Rel. (\forall S. (SourceTerm\ S, TargetTerm\ ([\![S]\!])) \in Rel) \wedge PredA\ Rel$
and $\forall (P, Q) \in indRelR.\ PredB\ (P, Q)$
$\Longrightarrow \exists Rel. (\forall S. (SourceTerm\ S, TargetTerm\ ([\![S]\!])) \in Rel) \wedge (\forall (P, Q) \in Rel.\ PredB\ (P, Q))$
⟨*proof*⟩

lemma (**in** *encoding*) *source-target-relation-impl-indRelR*:
fixes *Rel* :: $(('procS, 'procT)\ Proc \times ('procS, 'procT)\ Proc)\ set$
and *Pred* :: $(('procS, 'procT)\ Proc \times ('procS, 'procT)\ Proc) \Rightarrow bool$
assumes *encRRel*: $\forall S. (SourceTerm\ S, TargetTerm\ ([\![S]\!])) \in Rel$
and *condRel*: $\forall (P, Q) \in Rel.\ Pred\ (P, Q)$
shows $\forall (P, Q) \in indRelR.\ Pred\ (P, Q)$
⟨*proof*⟩

lemma (**in** *encoding*) *indRelR-iff-exists-source-target-relation*:
fixes *Pred* :: $(('procS, 'procT)\ Proc \times ('procS, 'procT)\ Proc) \Rightarrow bool$
shows $(\forall (P, Q) \in indRelR.\ Pred\ (P, Q))$
$= (\exists Rel. (\forall S. (SourceTerm\ S, TargetTerm\ ([\![S]\!])) \in Rel) \wedge (\forall (P, Q) \in Rel.\ Pred\ (P, Q)))$
⟨*proof*⟩

lemma (**in** *encoding*) *indRelR-modulo-pred-impl-indRelRPO-modulo-pred*:
fixes *Pred* :: $(('procS, 'procT)\ Proc \times ('procS, 'procT)\ Proc) \Rightarrow bool$
assumes *reflCond*: $\forall P.\ Pred\ (P, P)$
and *transCond*: $\forall P\ Q\ R.\ Pred\ (P, Q) \wedge Pred\ (Q, R) \longrightarrow Pred\ (P, R)$
shows $(\forall (P, Q) \in indRelR.\ Pred\ (P, Q)) = (\forall (P, Q) \in indRelRPO.\ Pred\ (P, Q))$
⟨*proof*⟩

lemma (**in** *encoding*) *indRelRPO-iff-exists-source-target-relation*:
fixes *Pred* :: $(('procS, 'procT)\ Proc \times ('procS, 'procT)\ Proc) \Rightarrow bool$
shows $(\forall (P, Q) \in indRelRPO.\ Pred\ (P, Q)) = (\exists Rel. (\forall S. (SourceTerm\ S, TargetTerm\ ([\![S]\!])) \in Rel)$
$\wedge (\forall (P, Q) \in Rel.\ Pred\ (P, Q)) \wedge preorder\ Rel)$
⟨*proof*⟩

An encoding preserves, reflects, or respects a predicate iff indRelR preserves, reflects, or respects this predicate.

lemma (**in** *encoding*) *enc-satisfies-pred-impl-indRelR-satisfies-pred*:
fixes *Pred* :: $(('procS, 'procT)\ Proc \times ('procS, 'procT)\ Proc) \Rightarrow bool$
assumes *encCond*: $\forall S.\ Pred\ (SourceTerm\ S,\ TargetTerm\ ([\![S]\!]))$
shows $\forall (P,\ Q) \in indRelR.\ Pred\ (P,\ Q)$
⟨*proof*⟩

lemma (**in** *encoding*) *indRelR-satisfies-pred-impl-enc-satisfies-pred*:
fixes *Pred* :: $(('procS, 'procT)\ Proc \times ('procS, 'procT)\ Proc) \Rightarrow bool$
assumes *relCond*: $\forall (P,\ Q) \in indRelR.\ Pred\ (P,\ Q)$
shows $\forall S.\ Pred\ (SourceTerm\ S,\ TargetTerm\ ([\![S]\!]))$
⟨*proof*⟩

lemma (**in** *encoding*) *enc-satisfies-pred-iff-indRelR-satisfies-pred*:
fixes *Pred* :: $(('procS, 'procT)\ Proc \times ('procS, 'procT)\ Proc) \Rightarrow bool$
shows $(\forall S.\ Pred\ (SourceTerm\ S,\ TargetTerm\ ([\![S]\!]))) = (\forall (P,\ Q) \in indRelR.\ Pred\ (P,\ Q))$
⟨*proof*⟩

lemma (**in** *encoding*) *enc-satisfies-binary-pred-iff-indRelR-satisfies-binary-pred*:
fixes *Pred* :: $(('procS, 'procT)\ Proc \times ('procS, 'procT)\ Proc) \Rightarrow 'b \Rightarrow bool$
shows $(\forall S\ a.\ Pred\ (SourceTerm\ S,\ TargetTerm\ ([\![S]\!]))\ a) = (\forall (P,\ Q) \in indRelR.\ \forall a.\ Pred\ (P,\ Q)\ a)$
⟨*proof*⟩

lemma (**in** *encoding*) *enc-preserves-pred-iff-indRelR-preserves-pred*:
fixes *Pred* :: $('procS, 'procT)\ Proc \Rightarrow bool$
shows *enc-preserves-pred Pred* = *rel-preserves-pred indRelR Pred*
⟨*proof*⟩

lemma (**in** *encoding*) *enc-preserves-binary-pred-iff-indRelR-preserves-binary-pred*:
fixes *Pred* :: $('procS, 'procT)\ Proc \Rightarrow 'b \Rightarrow bool$
shows *enc-preserves-binary-pred Pred* = *rel-preserves-binary-pred indRelR Pred*
⟨*proof*⟩

lemma (**in** *encoding*) *enc-preserves-pred-iff-indRelRPO-preserves-pred*:
fixes *Pred* :: $('procS, 'procT)\ Proc \Rightarrow bool$
shows *enc-preserves-pred Pred* = *rel-preserves-pred indRelRPO Pred*
⟨*proof*⟩

lemma (**in** *encoding*) *enc-reflects-pred-iff-indRelR-reflects-pred*:
fixes *Pred* :: $('procS, 'procT)\ Proc \Rightarrow bool$
shows *enc-reflects-pred Pred* = *rel-reflects-pred indRelR Pred*
⟨*proof*⟩

lemma (**in** *encoding*) *enc-reflects-binary-pred-iff-indRelR-reflects-binary-pred*:
fixes *Pred* :: $('procS, 'procT)\ Proc \Rightarrow 'b \Rightarrow bool$
shows *enc-reflects-binary-pred Pred* = *rel-reflects-binary-pred indRelR Pred*
⟨*proof*⟩

lemma (**in** *encoding*) *enc-reflects-pred-iff-indRelRPO-reflects-pred*:
fixes *Pred* :: $('procS, 'procT)\ Proc \Rightarrow bool$
shows *enc-reflects-pred Pred* = *rel-reflects-pred indRelRPO Pred*
⟨*proof*⟩

lemma (**in** *encoding*) *enc-respects-pred-iff-indRelR-respects-pred*:
fixes *Pred* :: $('procS, 'procT)\ Proc \Rightarrow bool$
shows *enc-respects-pred Pred* = *rel-respects-pred indRelR Pred*
⟨*proof*⟩

lemma (**in** *encoding*) *enc-respects-binary-pred-iff-indRelR-respects-binary-pred*:
fixes *Pred* :: $('procS, 'procT)\ Proc \Rightarrow 'b \Rightarrow bool$
shows *enc-respects-binary-pred Pred* = *rel-respects-binary-pred indRelR Pred*
⟨*proof*⟩

lemma (**in** *encoding*) *enc-respects-pred-iff-indRelRPO-respects-pred*:
fixes $Pred :: ('procS, 'procT)\ Proc \Rightarrow bool$
shows $enc\text{-}respects\text{-}pred\ Pred = rel\text{-}respects\text{-}pred\ indRelRPO\ Pred$
⟨*proof*⟩

Accordingly an encoding preserves, reflects, or respects a predicate iff there exists a relation that relates source terms with their literal translations and preserves, reflects, or respects this predicate.

lemma (**in** *encoding*) *enc-satisfies-pred-iff-source-target-satisfies-pred*:
fixes $Pred :: (('procS, 'procT)\ Proc \times ('procS, 'procT)\ Proc) \Rightarrow bool$
shows $(\forall S.\ Pred\ (SourceTerm\ S,\ TargetTerm\ (\llbracket S \rrbracket)))$
$= (\exists Rel.\ (\forall S.\ (SourceTerm\ S,\ TargetTerm\ (\llbracket S \rrbracket)) \in Rel) \wedge (\forall (P,\ Q) \in Rel.\ Pred\ (P,\ Q)))$
and $\llbracket \forall P\ Q\ R.\ Pred\ (P,\ Q) \wedge Pred\ (Q,\ R) \longrightarrow Pred\ (P,\ R);\ \forall P.\ Pred\ (P,\ P) \rrbracket \Longrightarrow$
$(\forall S.\ Pred\ (SourceTerm\ S,\ TargetTerm\ (\llbracket S \rrbracket))) = (\exists Rel.\ (\forall S.$
$(SourceTerm\ S,\ TargetTerm\ (\llbracket S \rrbracket)) \in Rel) \wedge (\forall (P,\ Q) \in Rel.\ Pred\ (P,\ Q)) \wedge preorder\ Rel)$
⟨*proof*⟩

lemma (**in** *encoding*) *enc-preserves-pred-iff-source-target-rel-preserves-pred*:
fixes $Pred :: ('procS, 'procT)\ Proc \Rightarrow bool$
shows $enc\text{-}preserves\text{-}pred\ Pred$
$= (\exists Rel.\ (\forall S.\ (SourceTerm\ S,\ TargetTerm\ (\llbracket S \rrbracket)) \in Rel) \wedge rel\text{-}preserves\text{-}pred\ Rel\ Pred)$
and $enc\text{-}preserves\text{-}pred\ Pred = (\exists Rel.\ (\forall S.\ (SourceTerm\ S,\ TargetTerm\ (\llbracket S \rrbracket)) \in Rel)$
$\wedge\ rel\text{-}preserves\text{-}pred\ Rel\ Pred \wedge preorder\ Rel)$
⟨*proof*⟩

lemma (**in** *encoding*) *enc-preserves-binary-pred-iff-source-target-rel-preserves-binary-pred*:
fixes $Pred :: ('procS, 'procT)\ Proc \Rightarrow 'b \Rightarrow bool$
shows $enc\text{-}preserves\text{-}binary\text{-}pred\ Pred = (\exists Rel.\ (\forall S.\ (SourceTerm\ S,\ TargetTerm\ (\llbracket S \rrbracket)) \in Rel)$
$\wedge\ rel\text{-}preserves\text{-}binary\text{-}pred\ Rel\ Pred)$
⟨*proof*⟩

lemma (**in** *encoding*) *enc-reflects-pred-iff-source-target-rel-reflects-pred*:
fixes $Pred :: ('procS, 'procT)\ Proc \Rightarrow bool$
shows $enc\text{-}reflects\text{-}pred\ Pred$
$= (\exists Rel.\ (\forall S.\ (SourceTerm\ S,\ TargetTerm\ (\llbracket S \rrbracket)) \in Rel) \wedge rel\text{-}reflects\text{-}pred\ Rel\ Pred)$
and $enc\text{-}reflects\text{-}pred\ Pred = (\exists Rel.\ (\forall S.\ (SourceTerm\ S,\ TargetTerm\ (\llbracket S \rrbracket)) \in Rel)$
$\wedge\ rel\text{-}reflects\text{-}pred\ Rel\ Pred \wedge preorder\ Rel)$
⟨*proof*⟩

lemma (**in** *encoding*) *enc-reflects-binary-pred-iff-source-target-rel-reflects-binary-pred*:
fixes $Pred :: ('procS, 'procT)\ Proc \Rightarrow 'b \Rightarrow bool$
shows $enc\text{-}reflects\text{-}binary\text{-}pred\ Pred = (\exists Rel.\ (\forall S.\ (SourceTerm\ S,\ TargetTerm\ (\llbracket S \rrbracket)) \in Rel)$
$\wedge\ rel\text{-}reflects\text{-}binary\text{-}pred\ Rel\ Pred)$
⟨*proof*⟩

lemma (**in** *encoding*) *enc-respects-pred-iff-source-target-rel-respects-pred-encR*:
fixes $Pred :: ('procS, 'procT)\ Proc \Rightarrow bool$
shows $enc\text{-}respects\text{-}pred\ Pred$
$= (\exists Rel.\ (\forall S.\ (SourceTerm\ S,\ TargetTerm\ (\llbracket S \rrbracket)) \in Rel) \wedge rel\text{-}respects\text{-}pred\ Rel\ Pred)$
and $enc\text{-}respects\text{-}pred\ Pred = (\exists Rel.\ (\forall S.\ (SourceTerm\ S,\ TargetTerm\ (\llbracket S \rrbracket)) \in Rel)$
$\wedge\ rel\text{-}respects\text{-}pred\ Rel\ Pred \wedge preorder\ Rel)$
⟨*proof*⟩

lemma (**in** *encoding*) *enc-respects-binary-pred-iff-source-target-rel-respects-binary-pred-encR*:
fixes $Pred :: ('procS, 'procT)\ Proc \Rightarrow 'b \Rightarrow bool$
shows $enc\text{-}respects\text{-}binary\text{-}pred\ Pred = (\exists Rel.\ (\forall S.\ (SourceTerm\ S,\ TargetTerm\ (\llbracket S \rrbracket)) \in Rel)$
$\wedge\ rel\text{-}respects\text{-}binary\text{-}pred\ Rel\ Pred)$
⟨*proof*⟩

To analyse the reflection of source term behaviours we use relations that contain the pairs (enc S, S) for all source terms S.

inductive-set (**in** *encoding*) *indRelL*
:: ((('*procS*, '*procT*) *Proc*) × (('*procS*, '*procT*) *Proc*)) *set*
where
encL: (*TargetTerm* ($[\![S]\!]$), *SourceTerm S*) ∈ *indRelL*

abbreviation (**in** *encoding*) *indRelLinfix* ::
('*procS*, '*procT*) *Proc* ⇒ ('*procS*, '*procT*) *Proc* ⇒ *bool* (- $\mathcal{R}[\![\cdot]\!]L$ - [*75*, *75*] *80*)
where
P $\mathcal{R}[\![\cdot]\!]L$ *Q* ≡ (*P*, *Q*) ∈ *indRelL*

inductive-set (**in** *encoding*) *indRelLPO*
:: ((('*procS*, '*procT*) *Proc*) × (('*procS*, '*procT*) *Proc*)) *set*
where
encL: (*TargetTerm* ($[\![S]\!]$), *SourceTerm S*) ∈ *indRelLPO* |
source: (*SourceTerm S*, *SourceTerm S*) ∈ *indRelLPO* |
target: (*TargetTerm T*, *TargetTerm T*) ∈ *indRelLPO* |
trans: $[\![(P, Q) \in indRelLPO; (Q, R) \in indRelLPO]\!] \Longrightarrow (P, R) \in indRelLPO$

abbreviation (**in** *encoding*) *indRelLPOinfix* ::
('*procS*, '*procT*) *Proc* ⇒ ('*procS*, '*procT*) *Proc* ⇒ *bool* (- $\lesssim[\![\cdot]\!]L$ - [*75*, *75*] *80*)
where
P $\lesssim[\![\cdot]\!]L$ *Q* ≡ (*P*, *Q*) ∈ *indRelLPO*

lemma (**in** *encoding*) *indRelLPO-refl*:
shows *refl indRelLPO*
⟨*proof*⟩

lemma (**in** *encoding*) *indRelLPO-is-preorder*:
shows *preorder indRelLPO*
⟨*proof*⟩

lemma (**in** *encoding*) *refl-trans-closure-of-indRelL*:
shows *indRelLPO* = $indRelL^*$
⟨*proof*⟩

The relations indRelR and indRelL are dual. indRelR preserves some predicate iff indRelL reflects it. indRelR reflects some predicate iff indRelL reflects it. indRelR respects some predicate iff indRelL does.

lemma (**in** *encoding*) *indRelR-preserves-pred-iff-indRelL-reflects-pred*:
fixes *Pred* :: ('*procS*, '*procT*) *Proc* ⇒ *bool*
shows *rel-preserves-pred indRelR Pred* = *rel-reflects-pred indRelL Pred*
⟨*proof*⟩

lemma (**in** *encoding*) *indRelR-preserves-binary-pred-iff-indRelL-reflects-binary-pred*:
fixes *Pred* :: ('*procS*, '*procT*) *Proc* ⇒ '*b* ⇒ *bool*
shows *rel-preserves-binary-pred indRelR Pred* = *rel-reflects-binary-pred indRelL Pred*
⟨*proof*⟩

lemma (**in** *encoding*) *indRelR-reflects-pred-iff-indRelL-preserves-pred*:
fixes *Pred* :: ('*procS*, '*procT*) *Proc* ⇒ *bool*
shows *rel-reflects-pred indRelR Pred* = *rel-preserves-pred indRelL Pred*
⟨*proof*⟩

lemma (**in** *encoding*) *indRelR-reflects-binary-pred-iff-indRelL-preserves-binary-pred*:
fixes *Pred* :: ('*procS*, '*procT*) *Proc* ⇒ '*b* ⇒ *bool*
shows *rel-reflects-binary-pred indRelR Pred* = *rel-preserves-binary-pred indRelL Pred*
⟨*proof*⟩

lemma (**in** *encoding*) *indRelR-respects-pred-iff-indRelL-respects-pred*:
fixes *Pred* :: ('*procS*, '*procT*) *Proc* ⇒ *bool*

shows *rel-respects-pred indRelR Pred* = *rel-respects-pred indRelL Pred*
⟨*proof*⟩

lemma (**in** *encoding*) *indRelR-respects-binary-pred-iff-indRelL-respects-binary-pred*:
fixes *Pred* :: $('procS, 'procT)$ *Proc* ⇒ $'b$ ⇒ *bool*
shows *rel-respects-binary-pred indRelR Pred* = *rel-respects-binary-pred indRelL Pred*
⟨*proof*⟩

lemma (**in** *encoding*) *indRelR-cond-preservation-iff-indRelL-cond-reflection*:
fixes *Pred* :: $('procS, 'procT)$ *Proc* ⇒ *bool*
shows (∃ *Rel*. (∀ *S*. (*SourceTerm S*, *TargetTerm* (⟦*S*⟧)) ∈ *Rel*) ∧ *rel-preserves-pred Rel Pred*)
= (∃ *Rel*. (∀ *S*. (*TargetTerm* (⟦*S*⟧), *SourceTerm S*) ∈ *Rel*) ∧ *rel-reflects-pred Rel Pred*)
⟨*proof*⟩

lemma (**in** *encoding*) *indRelR-cond-binary-preservation-iff-indRelL-cond-binary-reflection*:
fixes *Pred* :: $('procS, 'procT)$ *Proc* ⇒ $'b$ ⇒ *bool*
shows (∃ *Rel*. (∀ *S*. (*SourceTerm S*, *TargetTerm* (⟦*S*⟧)) ∈ *Rel*) ∧ *rel-preserves-binary-pred Rel Pred*)
= (∃ *Rel*. (∀ *S*. (*TargetTerm* (⟦*S*⟧), *SourceTerm S*) ∈ *Rel*)
∧ *rel-reflects-binary-pred Rel Pred*)
⟨*proof*⟩

lemma (**in** *encoding*) *indRelR-cond-reflection-iff-indRelL-cond-preservation*:
fixes *Pred* :: $('procS, 'procT)$ *Proc* ⇒ *bool*
shows (∃ *Rel*. (∀ *S*. (*SourceTerm S*, *TargetTerm* (⟦*S*⟧)) ∈ *Rel*) ∧ *rel-reflects-pred Rel Pred*)
= (∃ *Rel*. (∀ *S*. (*TargetTerm* (⟦*S*⟧), *SourceTerm S*) ∈ *Rel*) ∧ *rel-preserves-pred Rel Pred*)
⟨*proof*⟩

lemma (**in** *encoding*) *indRelR-cond-binary-reflection-iff-indRelL-cond-binary-preservation*:
fixes *Pred* :: $('procS, 'procT)$ *Proc* ⇒ $'b$ ⇒ *bool*
shows (∃ *Rel*. (∀ *S*. (*SourceTerm S*, *TargetTerm* (⟦*S*⟧)) ∈ *Rel*) ∧ *rel-reflects-binary-pred Rel Pred*)
= (∃ *Rel*. (∀ *S*. (*TargetTerm* (⟦*S*⟧), *SourceTerm S*) ∈ *Rel*)
∧ *rel-preserves-binary-pred Rel Pred*)
⟨*proof*⟩

lemma (**in** *encoding*) *indRelR-cond-respection-iff-indRelL-cond-respection*:
fixes *Pred* :: $('procS, 'procT)$ *Proc* ⇒ *bool*
shows (∃ *Rel*. (∀ *S*. (*SourceTerm S*, *TargetTerm* (⟦*S*⟧)) ∈ *Rel*) ∧ *rel-respects-pred Rel Pred*)
= (∃ *Rel*. (∀ *S*. (*TargetTerm* (⟦*S*⟧), *SourceTerm S*) ∈ *Rel*) ∧ *rel-respects-pred Rel Pred*)
⟨*proof*⟩

lemma (**in** *encoding*) *indRelR-cond-binary-respection-iff-indRelL-cond-binary-respection*:
fixes *Pred* :: $('procS, 'procT)$ *Proc* ⇒ $'b$ ⇒ *bool*
shows (∃ *Rel*. (∀ *S*. (*SourceTerm S*, *TargetTerm* (⟦*S*⟧)) ∈ *Rel*) ∧ *rel-respects-binary-pred Rel Pred*)
= (∃ *Rel*. (∀ *S*. (*TargetTerm* (⟦*S*⟧), *SourceTerm S*) ∈ *Rel*)
∧ *rel-respects-binary-pred Rel Pred*)
⟨*proof*⟩

An encoding preserves, reflects, or respects a predicate iff indRelL reflects, preserves, or respects this predicate.

lemma (**in** *encoding*) *enc-preserves-pred-iff-indRelL-reflects-pred*:
fixes *Pred* :: $('procS, 'procT)$ *Proc* ⇒ *bool*
shows *enc-preserves-pred Pred* = *rel-reflects-pred indRelL Pred*
⟨*proof*⟩

lemma (**in** *encoding*) *enc-reflects-pred-iff-indRelL-preserves-pred*:
fixes *Pred* :: $('procS, 'procT)$ *Proc* ⇒ *bool*
shows *enc-reflects-pred Pred* = *rel-preserves-pred indRelL Pred*
⟨*proof*⟩

lemma (**in** *encoding*) *enc-respects-pred-iff-indRelL-respects-pred*:

fixes *Pred* :: ('*procS*, '*procT*) *Proc* ⇒ *bool*
shows *enc-respects-pred Pred* = *rel-respects-pred indRelL Pred*
⟨*proof*⟩

An encoding preserves, reflects, or respects a predicate iff there exists a relation, namely indRelL, that relates literal translations with their source terms and reflects, preserves, or respects this predicate.

lemma (**in** *encoding*) *enc-preserves-pred-iff-source-target-rel-reflects-pred*:
fixes *Pred* :: ('*procS*, '*procT*) *Proc* ⇒ *bool*
shows *enc-preserves-pred Pred*
= (∃ *Rel*. (∀ *S*. (*TargetTerm* (⟦*S*⟧), *SourceTerm S*) ∈ *Rel*) ∧ *rel-reflects-pred Rel Pred*)
⟨*proof*⟩

lemma (**in** *encoding*) *enc-reflects-pred-iff-source-target-rel-preserves-pred*:
fixes *Pred* :: ('*procS*, '*procT*) *Proc* ⇒ *bool*
shows *enc-reflects-pred Pred*
= (∃ *Rel*. (∀ *S*. (*TargetTerm* (⟦*S*⟧), *SourceTerm S*) ∈ *Rel*) ∧ *rel-preserves-pred Rel Pred*)
⟨*proof*⟩

lemma (**in** *encoding*) *enc-respects-pred-iff-source-target-rel-respects-pred-encL*:
fixes *Pred* :: ('*procS*, '*procT*) *Proc* ⇒ *bool*
shows *enc-respects-pred Pred*
= (∃ *Rel*. (∀ *S*. (*TargetTerm* (⟦*S*⟧), *SourceTerm S*) ∈ *Rel*) ∧ *rel-respects-pred Rel Pred*)
⟨*proof*⟩

To analyse the respection of source term behaviours we use relations that contain both kind of pairs: (S, enc S) as well as (enc S, S) for all source terms S.

inductive-set (**in** *encoding*) *indRel*
:: ((('*procS*, '*procT*) *Proc*) × (('*procS*, '*procT*) *Proc*)) *set*
where
encR: (*SourceTerm S*, *TargetTerm* (⟦*S*⟧)) ∈ *indRel* |
encL: (*TargetTerm* (⟦*S*⟧), *SourceTerm S*) ∈ *indRel*

abbreviation (**in** *encoding*) *indRelInfix* ::
('*procS*, '*procT*) *Proc* ⇒ ('*procS*, '*procT*) *Proc* ⇒ *bool* (- $\mathcal{R}$⟦·⟧ - [*75*, *75*] *80*)
where
P $\mathcal{R}$⟦·⟧ *Q* ≡ (*P*, *Q*) ∈ *indRel*

lemma (**in** *encoding*) *indRel-symm*:
shows *sym indRel*
⟨*proof*⟩

inductive-set (**in** *encoding*) *indRelEQ*
:: ((('*procS*, '*procT*) *Proc*) × (('*procS*, '*procT*) *Proc*)) *set*
where
encR: (*SourceTerm S*, *TargetTerm* (⟦*S*⟧)) ∈ *indRelEQ* |
encL: (*TargetTerm* (⟦*S*⟧), *SourceTerm S*) ∈ *indRelEQ* |
target: (*TargetTerm T*, *TargetTerm T*) ∈ *indRelEQ* |
trans: ⟦(*P*, *Q*) ∈ *indRelEQ*; (*Q*, *R*) ∈ *indRelEQ*⟧ ⟹ (*P*, *R*) ∈ *indRelEQ*

abbreviation (**in** *encoding*) *indRelEQinfix* ::
('*procS*, '*procT*) *Proc* ⇒ ('*procS*, '*procT*) *Proc* ⇒ *bool* (- ∼⟦·⟧ - [*75*, *75*] *80*)
where
P ∼⟦·⟧ *Q* ≡ (*P*, *Q*) ∈ *indRelEQ*

lemma (**in** *encoding*) *indRelEQ-refl*:
shows *refl indRelEQ*
⟨*proof*⟩

lemma (**in** *encoding*) *indRelEQ-is-preorder*:
shows *preorder indRelEQ*

⟨*proof*⟩

lemma (**in** *encoding*) *indRelEQ-symm*:
shows *sym indRelEQ*
⟨*proof*⟩

lemma (**in** *encoding*) *indRelEQ-is-equivalence*:
shows *equivalence indRelEQ*
⟨*proof*⟩

lemma (**in** *encoding*) *refl-trans-closure-of-indRel*:
shows $indRelEQ = indRel^*$
⟨*proof*⟩

lemma (**in** *encoding*) *refl-symm-trans-closure-of-indRel*:
shows $indRelEQ = (symcl\ (indRel^=))^+$
⟨*proof*⟩

lemma (**in** *encoding*) *symm-closure-of-indRelR*:
shows $indRel = symcl\ indRelR$
and $indRelEQ = (symcl\ (indRelR^=))^+$
⟨*proof*⟩

lemma (**in** *encoding*) *symm-closure-of-indRelL*:
shows $indRel = symcl\ indRelL$
and $indRelEQ = (symcl\ (indRelL^=))^+$
⟨*proof*⟩

The relation indRel is a combination of indRelL and indRelR. indRel respects a predicate iff indRelR (or indRelL) respects it.

lemma (**in** *encoding*) *indRel-respects-pred-iff-indRelR-respects-pred*:
fixes $Pred :: ('procS, 'procT)\ Proc \Rightarrow bool$
shows *rel-respects-pred indRel Pred* = *rel-respects-pred indRelR Pred*
⟨*proof*⟩

lemma (**in** *encoding*) *indRel-respects-binary-pred-iff-indRelR-respects-binary-pred*:
fixes $Pred :: ('procS, 'procT)\ Proc \Rightarrow 'b \Rightarrow bool$
shows *rel-respects-binary-pred indRel Pred* = *rel-respects-binary-pred indRelR Pred*
⟨*proof*⟩

lemma (**in** *encoding*) *indRel-cond-respection-iff-indRelR-cond-respection*:
fixes $Pred :: ('procS, 'procT)\ Proc \Rightarrow bool$
shows $(\exists Rel.$
$(\forall S.\ (SourceTerm\ S,\ TargetTerm\ ([\![S]\!])) \in Rel \wedge (TargetTerm\ ([\![S]\!]),\ SourceTerm\ S) \in Rel)$
$\wedge$ *rel-respects-pred Rel Pred*)
$= (\exists Rel.\ (\forall S.\ (SourceTerm\ S,\ TargetTerm\ ([\![S]\!])) \in Rel) \wedge$ *rel-respects-pred Rel Pred*)
⟨*proof*⟩

lemma (**in** *encoding*) *indRel-cond-binary-respection-iff-indRelR-cond-binary-respection*:
fixes $Pred :: ('procS, 'procT)\ Proc \Rightarrow 'b \Rightarrow bool$
shows $(\exists Rel.$
$(\forall S.\ (SourceTerm\ S,\ TargetTerm\ ([\![S]\!])) \in Rel \wedge (TargetTerm\ ([\![S]\!]),\ SourceTerm\ S) \in Rel)$
$\wedge$ *rel-respects-binary-pred Rel Pred*)
$= (\exists Rel.\ (\forall S.\ (SourceTerm\ S,\ TargetTerm\ ([\![S]\!])) \in Rel)$
$\wedge$ *rel-respects-binary-pred Rel Pred*)
⟨*proof*⟩

An encoding respects a predicate iff indRel respects this predicate.

lemma (**in** *encoding*) *enc-respects-pred-iff-indRel-respects-pred*:
fixes $Pred :: ('procS, 'procT)\ Proc \Rightarrow bool$

shows *enc-respects-pred Pred* = *rel-respects-pred indRel Pred*
⟨*proof*⟩

An encoding respects a predicate iff there exists a relation, namely indRel, that relates source terms and their literal translations in both directions and respects this predicate.

lemma (**in** *encoding*) *enc-respects-pred-iff-source-target-rel-respects-pred-encRL*:
fixes *Pred* :: ($'procS$, $'procT$) *Proc* ⇒ *bool*
shows *enc-respects-pred Pred*
= (∃ *Rel*.
(∀ *S*. (*SourceTerm S*, *TargetTerm* (⟦*S*⟧)) ∈ *Rel* ∧ (*TargetTerm* (⟦*S*⟧), *SourceTerm S*) ∈ *Rel*)
∧ *rel-respects-pred Rel Pred*)
⟨*proof*⟩

5.2 Relations Induced by the Encoding and a Relation on Target Terms

Some encodability like e.g. operational correspondence are defined w.r.t. a relation on target terms. To analyse such criteria we include the respective target term relation in the considered relation on the disjoint union of source and target terms.

inductive-set (**in** *encoding*) *indRelRT*
:: ($'procT$ × $'procT$) *set* ⇒ ((($'procS$, $'procT$) *Proc*) × (($'procS$, $'procT$) *Proc*)) *set*
for *TRel* :: ($'procT$ × $'procT$) *set*
where
encR: (*SourceTerm S*, *TargetTerm* (⟦*S*⟧)) ∈ *indRelRT TRel* |
target: (*T1*, *T2*) ∈ *TRel* ⟹ (*TargetTerm T1*, *TargetTerm T2*) ∈ *indRelRT TRel*

abbreviation (**in** *encoding*) *indRelRTinfix*
:: ($'procS$, $'procT$) *Proc* ⇒ ($'procT$ × $'procT$) *set* ⇒ ($'procS$, $'procT$) *Proc* ⇒ *bool*
(- $\mathcal{R}$⟦·⟧*RT*<-> - [*75*, *75*, *75*] *80*)
where
P $\mathcal{R}$⟦·⟧*RT*<*TRel*> *Q* ≡ (*P*, *Q*) ∈ *indRelRT TRel*

inductive-set (**in** *encoding*) *indRelRTPO*
:: ($'procT$ × $'procT$) *set* ⇒ ((($'procS$, $'procT$) *Proc*) × (($'procS$, $'procT$) *Proc*)) *set*
for *TRel* :: ($'procT$ × $'procT$) *set*
where
encR: (*SourceTerm S*, *TargetTerm* (⟦*S*⟧)) ∈ *indRelRTPO TRel* |
source: (*SourceTerm S*, *SourceTerm S*) ∈ *indRelRTPO TRel* |
target: (*T1*, *T2*) ∈ *TRel* ⟹ (*TargetTerm T1*, *TargetTerm T2*) ∈ *indRelRTPO TRel* |
trans: ⟦(*P*, *Q*) ∈ *indRelRTPO TRel*; (*Q*, *R*) ∈ *indRelRTPO TRel*⟧ ⟹ (*P*, *R*) ∈ *indRelRTPO TRel*

abbreviation (**in** *encoding*) *indRelRTPOinfix*
:: ($'procS$, $'procT$) *Proc* ⇒ ($'procT$ × $'procT$) *set* ⇒ ($'procS$, $'procT$) *Proc* ⇒ *bool*
(- ≲⟦·⟧*RT*<-> - [*75*, *75*, *75*] *80*)
where
P ≲⟦·⟧*RT*<*TRel*> *Q* ≡ (*P*, *Q*) ∈ *indRelRTPO TRel*

lemma (**in** *encoding*) *indRelRTPO-refl*:
fixes *TRel* :: ($'procT$ × $'procT$) *set*
assumes *refl*: *refl TRel*
shows *refl* (*indRelRTPO TRel*)
⟨*proof*⟩

lemma (**in** *encoding*) *refl-trans-closure-of-indRelRT*:
fixes *TRel* :: ($'procT$ × $'procT$) *set*
assumes *refl*: *refl TRel*
shows *indRelRTPO TRel* = (*indRelRT TRel*)*
⟨*proof*⟩

lemma (**in** *encoding*) *indRelRTPO-is-preorder*:

fixes $TRel :: ('procT \times 'procT)\ set$
assumes *reflT*: $refl\ TRel$
shows $preorder\ (indRelRTPO\ TRel)$
⟨*proof*⟩

lemma (**in** *encoding*) *transitive-closure-of-TRel-to-indRelRTPO*:
fixes $TRel\ :: ('procT \times 'procT)\ set$
and $TP\ TQ :: 'procT$
shows $(TP, TQ) \in TRel^+ \Longrightarrow TargetTerm\ TP \lesssim[\![\cdot]\!]RT{<}TRel{>}\ TargetTerm\ TQ$
⟨*proof*⟩

The relation indRelRT is the smallest relation that relates all source terms and their literal translations and contains TRel. Thus there exists a relation that relates source terms and their literal translations and satisfies some predicate on its pairs iff the predicate holds for the pairs of indRelR.

lemma (**in** *encoding*) *indRelR-modulo-pred-impl-indRelRT-modulo-pred*:
fixes $Pred :: (('procS, 'procT)\ Proc \times ('procS, 'procT)\ Proc) \Rightarrow bool$
shows $(\forall (P, Q) \in indRelR.\ Pred\ (P, Q)) = (\forall\ TRel.\ (\forall (TP, TQ) \in TRel.$
$Pred\ (TargetTerm\ TP, TargetTerm\ TQ)) \longleftrightarrow (\forall (P, Q) \in indRelRT\ TRel.\ Pred\ (P, Q)))$
⟨*proof*⟩

lemma (**in** *encoding*) *indRelRT-iff-exists-source-target-relation*:
fixes $Pred :: (('procS, 'procT)\ Proc \times ('procS, 'procT)\ Proc) \Rightarrow bool$
shows $(\forall\ TRel.\ (\forall (TP, TQ) \in TRel.\ Pred\ (TargetTerm\ TP, TargetTerm\ TQ))$
$\longleftrightarrow (\forall (P, Q) \in indRelRT\ TRel.\ Pred\ (P, Q)))$
$= (\exists\ Rel.\ (\forall\ S.\ (SourceTerm\ S, TargetTerm\ ([\![S]\!])) \in Rel) \wedge (\forall (P, Q) \in Rel.\ Pred\ (P, Q)))$
⟨*proof*⟩

lemma (**in** *encoding*) *indRelRT-modulo-pred-impl-indRelRTPO-modulo-pred*:
fixes $TRel :: ('procT \times 'procT)\ set$
and $Pred :: (('procS, 'procT)\ Proc \times ('procS, 'procT)\ Proc) \Rightarrow bool$
assumes *reflCond*: $\forall\ P.\ Pred\ (P, P)$
and *transCond*: $\forall\ P\ Q\ R.\ Pred\ (P, Q) \wedge Pred\ (Q, R) \longrightarrow Pred\ (P, R)$
shows $(\forall (P, Q) \in indRelRT\ TRel.\ Pred\ (P, Q)) = (\forall (P, Q) \in indRelRTPO\ TRel.\ Pred\ (P, Q))$
⟨*proof*⟩

lemma (**in** *encoding*) *indRelR-modulo-pred-impl-indRelRTPO-modulo-pred*:
fixes $Pred :: (('procS, 'procT)\ Proc \times ('procS, 'procT)\ Proc) \Rightarrow bool$
assumes $\forall\ P.\ Pred\ (P, P)$
and $\forall\ P\ Q\ R.\ Pred\ (P, Q) \wedge Pred\ (Q, R) \longrightarrow Pred\ (P, R)$
shows $(\forall (P, Q) \in indRelR.\ Pred\ (P, Q))$
$= (\forall\ TRel.\ (\forall (TP, TQ) \in TRel.\ Pred\ (TargetTerm\ TP, TargetTerm\ TQ))$
$\longleftrightarrow (\forall (P, Q) \in indRelRTPO\ TRel.\ Pred\ (P, Q)))$
⟨*proof*⟩

The relation indRelLT includes TRel and relates literal translations and their source terms.

inductive-set (**in** *encoding*) *indRelLT*
$:: ('procT \times 'procT)\ set \Rightarrow ((('procS, 'procT)\ Proc) \times (('procS, 'procT)\ Proc))\ set$
for $TRel :: ('procT \times 'procT)\ set$
where
encL: $(TargetTerm\ ([\![S]\!]), SourceTerm\ S) \in indRelLT\ TRel\ |$
target: $(T1, T2) \in TRel \Longrightarrow (TargetTerm\ T1, TargetTerm\ T2) \in indRelLT\ TRel$

abbreviation (**in** *encoding*) *indRelLTinfix*
$:: ('procS, 'procT)\ Proc \Rightarrow ('procT \times 'procT)\ set \Rightarrow ('procS, 'procT)\ Proc \Rightarrow bool$
$(\text{-}\ \mathcal{R}[\![\cdot]\!]LT{<}\text{-}{>}\ \text{-}\ [75, 75, 75]\ 80)$
where
$P\ \mathcal{R}[\![\cdot]\!]LT{<}TRel{>}\ Q \equiv (P, Q) \in indRelLT\ TRel$

inductive-set (**in** *encoding*) *indRelLTPO*
$:: ('procT \times 'procT)\ set \Rightarrow ((('procS, 'procT)\ Proc) \times (('procS, 'procT)\ Proc))\ set$

for *TRel* :: (*'procT* × *'procT*) *set*
where
encL: (*TargetTerm* (⟦*S*⟧), *SourceTerm S*) ∈ *indRelLTPO TRel* |
source: (*SourceTerm S*, *SourceTerm S*) ∈ *indRelLTPO TRel* |
target: (*T1*, *T2*) ∈ *TRel* ⟹ (*TargetTerm T1*, *TargetTerm T2*) ∈ *indRelLTPO TRel* |
trans: ⟦(*P*, *Q*) ∈ *indRelLTPO TRel*; (*Q*, *R*) ∈ *indRelLTPO TRel*⟧ ⟹ (*P*, *R*) ∈ *indRelLTPO TRel*

abbreviation (**in** *encoding*) *indRelLTPOinfix*
:: (*'procS*, *'procT*) *Proc* ⇒ (*'procT* × *'procT*) *set* ⇒ (*'procS*, *'procT*) *Proc* ⇒ *bool*
(- ≲⟦·⟧*LT*<-> - [*75*, *75*, *75*] *80*)
where
P ≲⟦·⟧*LT*<*TRel*> *Q* ≡ (*P*, *Q*) ∈ *indRelLTPO TRel*

lemma (**in** *encoding*) *indRelLTPO-refl*:
fixes *TRel* :: (*'procT* × *'procT*) *set*
assumes *refl*: *refl TRel*
shows *refl* (*indRelLTPO TRel*)
⟨*proof*⟩

lemma (**in** *encoding*) *refl-trans-closure-of-indRelLT*:
fixes *TRel* :: (*'procT* × *'procT*) *set*
assumes *refl*: *refl TRel*
shows *indRelLTPO TRel* = (*indRelLT TRel*)*
⟨*proof*⟩

inductive-set (**in** *encoding*) *indRelT*
:: (*'procT* × *'procT*) *set* ⇒ (((*'procS*, *'procT*) *Proc*) × ((*'procS*, *'procT*) *Proc*)) *set*
for *TRel* :: (*'procT* × *'procT*) *set*
where
encR: (*SourceTerm S*, *TargetTerm* (⟦*S*⟧)) ∈ *indRelT TRel* |
encL: (*TargetTerm* (⟦*S*⟧), *SourceTerm S*) ∈ *indRelT TRel* |
target: (*T1*, *T2*) ∈ *TRel* ⟹ (*TargetTerm T1*, *TargetTerm T2*) ∈ *indRelT TRel*

abbreviation (**in** *encoding*) *indRelTinfix*
:: (*'procS*, *'procT*) *Proc* ⇒ (*'procT* × *'procT*) *set* ⇒ (*'procS*, *'procT*) *Proc* ⇒ *bool*
(- $\mathcal{R}$⟦·⟧*T*<-> - [*75*, *75*, *75*] *80*)
where
P $\mathcal{R}$⟦·⟧*T*<*TRel*> *Q* ≡ (*P*, *Q*) ∈ *indRelT TRel*

lemma (**in** *encoding*) *indRelT-symm*:
fixes *TRel* :: (*'procT* × *'procT*) *set*
assumes *symm*: *sym TRel*
shows *sym* (*indRelT TRel*)
⟨*proof*⟩

inductive-set (**in** *encoding*) *indRelTEQ*
:: (*'procT* × *'procT*) *set* ⇒ (((*'procS*, *'procT*) *Proc*) × ((*'procS*, *'procT*) *Proc*)) *set*
for *TRel* :: (*'procT* × *'procT*) *set*
where
encR: (*SourceTerm S*, *TargetTerm* (⟦*S*⟧)) ∈ *indRelTEQ TRel* |
encL: (*TargetTerm* (⟦*S*⟧), *SourceTerm S*) ∈ *indRelTEQ TRel* |
target: (*T1*, *T2*) ∈ *TRel* ⟹ (*TargetTerm T1*, *TargetTerm T2*) ∈ *indRelTEQ TRel* |
trans: ⟦(*P*, *Q*) ∈ *indRelTEQ TRel*; (*Q*, *R*) ∈ *indRelTEQ TRel*⟧ ⟹ (*P*, *R*) ∈ *indRelTEQ TRel*

abbreviation (**in** *encoding*) *indRelTEQinfix*
:: (*'procS*, *'procT*) *Proc* ⇒ (*'procT* × *'procT*) *set* ⇒ (*'procS*, *'procT*) *Proc* ⇒ *bool*
(- ∼⟦·⟧*T*<-> - [*75*, *75*, *75*] *80*)
where
P ∼⟦·⟧*T*<*TRel*> *Q* ≡ (*P*, *Q*) ∈ *indRelTEQ TRel*

lemma (**in** *encoding*) *indRelTEQ-refl*:

fixes *TRel* :: (*'procT* × *'procT*) *set*
assumes *refl*: *refl TRel*
shows *refl* (*indRelTEQ TRel*)
⟨*proof*⟩

lemma (**in** *encoding*) *indRelTEQ-symm*:
fixes *TRel* :: (*'procT* × *'procT*) *set*
assumes *symm*: *sym TRel*
shows *sym* (*indRelTEQ TRel*)
⟨*proof*⟩

lemma (**in** *encoding*) *refl-trans-closure-of-indRelT*:
fixes *TRel* :: (*'procT* × *'procT*) *set*
assumes *refl*: *refl TRel*
shows *indRelTEQ TRel* = $(indRelT\ TRel)^*$
⟨*proof*⟩

lemma (**in** *encoding*) *refl-symm-trans-closure-of-indRelT*:
fixes *TRel* :: (*'procT* × *'procT*) *set*
assumes *refl*: *refl TRel*
and *symm*: *sym TRel*
shows *indRelTEQ TRel* = $(symcl\ ((indRelT\ TRel)^=))^+$
⟨*proof*⟩

lemma (**in** *encoding*) *symm-closure-of-indRelRT*:
fixes *TRel* :: (*'procT* × *'procT*) *set*
assumes *refl*: *refl TRel*
and *symm*: *sym TRel*
shows *indRelT TRel* = *symcl* (*indRelRT TRel*)
and *indRelTEQ TRel* = $(symcl\ ((indRelRT\ TRel)^=))^+$
⟨*proof*⟩

lemma (**in** *encoding*) *symm-closure-of-indRelLT*:
fixes *TRel* :: (*'procT* × *'procT*) *set*
assumes *refl*: *refl TRel*
and *symm*: *sym TRel*
shows *indRelT TRel* = *symcl* (*indRelLT TRel*)
and *indRelTEQ TRel* = $(symcl\ ((indRelLT\ TRel)^=))^+$
⟨*proof*⟩

If the relations indRelRT, indRelLT, or indRelT contain a pair of target terms, then this pair is also related by the considered target term relation.

lemma (**in** *encoding*) *indRelRT-to-TRel*:
fixes *TRel* :: (*'procT* × *'procT*) *set*
and *TP TQ* :: *'procT*
assumes *rel*: *TargetTerm TP* $\mathcal{R}[\![\cdot]\!]RT{<}TRel{>}$ *TargetTerm TQ*
shows (*TP*, *TQ*) ∈ *TRel*
⟨*proof*⟩

lemma (**in** *encoding*) *indRelLT-to-TRel*:
fixes *TRel* :: (*'procT* × *'procT*) *set*
and *TP TQ* :: *'procT*
assumes *rel*: *TargetTerm TP* $\mathcal{R}[\![\cdot]\!]LT{<}TRel{>}$ *TargetTerm TQ*
shows (*TP*, *TQ*) ∈ *TRel*
⟨*proof*⟩

lemma (**in** *encoding*) *indRelT-to-TRel*:
fixes *TRel* :: (*'procT* × *'procT*) *set*
and *TP TQ* :: *'procT*
assumes *rel*: *TargetTerm TP* $\mathcal{R}[\![\cdot]\!]T{<}TRel{>}$ *TargetTerm TQ*

shows $(TP,\ TQ) \in TRel$
⟨*proof*⟩

If the preorders indRelRTPO, indRelLTPO, or the equivalence indRelTEQ contain a pair of terms, then the pair of target terms that is related to these two terms is also related by the reflexive and transitive closure of the considered target term relation.

lemma (**in** *encoding*) *indRelRTPO-to-TRel*:
fixes $TRel :: ('procT \times 'procT)\ set$
and $P\ Q\ :: ('procS,\ 'procT)\ Proc$
assumes *rel*: $P \lesssim[\![\cdot]\!]RT{<}TRel{>}\ Q$
shows $\forall\, SP\ SQ.\ SP \in S\ P \wedge SQ \in S\ Q \longrightarrow SP = SQ$
and $\forall\, SP\ TQ.\ SP \in S\ P \wedge TQ \in T\ Q$
$\longrightarrow ([\![SP]\!],\ TQ) \in (TRel \cup \{(T1,\ T2).\ \exists\, S.\ T1 = [\![S]\!] \wedge T2 = [\![S]\!]\})^+$
and $\forall\, TP\ SQ.\ TP \in T\ P \wedge SQ \in S\ Q \longrightarrow False$
and $\forall\, TP\ TQ.\ TP \in T\ P \wedge TQ \in T\ Q \longrightarrow (TP,\ TQ) \in TRel^+$
⟨*proof*⟩

lemma (**in** *encoding*) *indRelLTPO-to-TRel*:
fixes $TRel :: ('procT \times 'procT)\ set$
and $P\ Q\ :: ('procS,\ 'procT)\ Proc$
assumes *rel*: $P \lesssim[\![\cdot]\!]LT{<}TRel{>}\ Q$
shows $\forall\, SP\ SQ.\ SP \in S\ P \wedge SQ \in S\ Q \longrightarrow SP = SQ$
and $\forall\, SP\ TQ.\ SP \in S\ P \wedge TQ \in T\ Q \longrightarrow False$
and $\forall\, TP\ SQ.\ TP \in T\ P \wedge SQ \in S\ Q$
$\longrightarrow (TP,\ [\![SQ]\!]) \in (TRel \cup \{(T1,\ T2).\ \exists\, S.\ T1 = [\![S]\!] \wedge T2 = [\![S]\!]\})^+$
and $\forall\, TP\ TQ.\ TP \in T\ P \wedge TQ \in T\ Q \longrightarrow (TP,\ TQ) \in TRel^+$
⟨*proof*⟩

lemma (**in** *encoding*) *indRelTEQ-to-TRel*:
fixes $TRel :: ('procT \times 'procT)\ set$
and $P\ Q\ :: ('procS,\ 'procT)\ Proc$
assumes *rel*: $P \sim[\![\cdot]\!]T{<}TRel{>}\ Q$
shows $\forall\, SP\ SQ.\ SP \in S\ P \wedge SQ \in S\ Q$
$\longrightarrow ([\![SP]\!],\ [\![SQ]\!]) \in (TRel \cup \{(T1,\ T2).\ \exists\, S.\ T1 = [\![S]\!] \wedge T2 = [\![S]\!]\})^+$
and $\forall\, SP\ TQ.\ SP \in S\ P \wedge TQ \in T\ Q$
$\longrightarrow ([\![SP]\!],\ TQ) \in (TRel \cup \{(T1,\ T2).\ \exists\, S.\ T1 = [\![S]\!] \wedge T2 = [\![S]\!]\})^+$
and $\forall\, TP\ SQ.\ TP \in T\ P \wedge SQ \in S\ Q$
$\longrightarrow (TP,\ [\![SQ]\!]) \in (TRel \cup \{(T1,\ T2).\ \exists\, S.\ T1 = [\![S]\!] \wedge T2 = [\![S]\!]\})^+$
and $\forall\, TP\ TQ.\ TP \in T\ P \wedge TQ \in T\ Q$
$\longrightarrow (TP,\ TQ) \in (TRel \cup \{(T1,\ T2).\ \exists\, S.\ T1 = [\![S]\!] \wedge T2 = [\![S]\!]\})^+$
⟨*proof*⟩

lemma (**in** *encoding*) *trans-closure-of-TRel-refl-cond*:
fixes $TRel\ :: ('procT \times 'procT)\ set$
and $TP\ TQ :: 'procT$
assumes $(TP,\ TQ) \in (TRel \cup \{(T1,\ T2).\ \exists\, S.\ T1 = [\![S]\!] \wedge T2 = [\![S]\!]\})^+$
shows $(TP,\ TQ) \in TRel^*$
⟨*proof*⟩

Note that if indRelRTPO relates a source term S to a target term T, then the translation of S is equal to T or indRelRTPO also relates the translation of S to T.

lemma (**in** *encoding*) *indRelRTPO-relates-source-target*:
fixes $TRel :: ('procT \times 'procT)\ set$
and $S\ \ :: 'procS$
and $T\ \ :: 'procT$
assumes *pair*: $SourceTerm\ S \lesssim[\![\cdot]\!]RT{<}TRel{>}\ TargetTerm\ T$
shows $(TargetTerm\ ([\![S]\!]),\ TargetTerm\ T) \in (indRelRTPO\ TRel)^=$
⟨*proof*⟩

If indRelRTPO, indRelLTPO, or indRelTPO preserves barbs then so does the corresponding target term relation.

lemma (**in** *encoding-wrt-barbs*) *rel-with-target-impl-TRel-preserves-barbs*:
fixes *TRel* :: ('*procT* × '*procT*) *set*
and *Rel* :: (('*procS*, '*procT*) *Proc* × ('*procS*, '*procT*) *Proc*) *set*
assumes *preservation*: *rel-preserves-barbs Rel* (*STCalWB SWB TWB*)
and *targetInRel*: $\forall$ *T1 T2*. (*T1*, *T2*) $\in$ *TRel* $\longrightarrow$ (*TargetTerm T1*, *TargetTerm T2*) $\in$ *Rel*
shows *rel-preserves-barbs TRel TWB*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelRTPO-impl-TRel-preserves-barbs*:
fixes *TRel* :: ('*procT* × '*procT*) *set*
assumes *preservation*: *rel-preserves-barbs* (*indRelRTPO TRel*) (*STCalWB SWB TWB*)
shows *rel-preserves-barbs TRel TWB*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelLTPO-impl-TRel-preserves-barbs*:
fixes *TRel* :: ('*procT* × '*procT*) *set*
assumes *preservation*: *rel-preserves-barbs* (*indRelLTPO TRel*) (*STCalWB SWB TWB*)
shows *rel-preserves-barbs TRel TWB*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelTEQ-impl-TRel-preserves-barbs*:
fixes *TRel* :: ('*procT* × '*procT*) *set*
assumes *preservation*: *rel-preserves-barbs* (*indRelTEQ TRel*) (*STCalWB SWB TWB*)
shows *rel-preserves-barbs TRel TWB*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *rel-with-target-impl-TRel-weakly-preserves-barbs*:
fixes *TRel* :: ('*procT* × '*procT*) *set*
and *Rel* :: (('*procS*, '*procT*) *Proc* × ('*procS*, '*procT*) *Proc*) *set*
assumes *preservation*: *rel-weakly-preserves-barbs Rel* (*STCalWB SWB TWB*)
and *targetInRel*: $\forall$ *T1 T2*. (*T1*, *T2*) $\in$ *TRel* $\longrightarrow$ (*TargetTerm T1*, *TargetTerm T2*) $\in$ *Rel*
shows *rel-weakly-preserves-barbs TRel TWB*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelRTPO-impl-TRel-weakly-preserves-barbs*:
fixes *TRel* :: ('*procT* × '*procT*) *set*
assumes *preservation*: *rel-weakly-preserves-barbs* (*indRelRTPO TRel*) (*STCalWB SWB TWB*)
shows *rel-weakly-preserves-barbs TRel TWB*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelLTPO-impl-TRel-weakly-preserves-barbs*:
fixes *TRel* :: ('*procT* × '*procT*) *set*
assumes *preservation*: *rel-weakly-preserves-barbs* (*indRelLTPO TRel*) (*STCalWB SWB TWB*)
shows *rel-weakly-preserves-barbs TRel TWB*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelTEQ-impl-TRel-weakly-preserves-barbs*:
fixes *TRel* :: ('*procT* × '*procT*) *set*
assumes *preservation*: *rel-weakly-preserves-barbs* (*indRelTEQ TRel*) (*STCalWB SWB TWB*)
shows *rel-weakly-preserves-barbs TRel TWB*
⟨*proof*⟩

If indRelRTPO, indRelLTPO, or indRelTPO reflects barbs then so does the corresponding target term relation.

lemma (**in** *encoding-wrt-barbs*) *rel-with-target-impl-TRel-reflects-barbs*:
fixes *TRel* :: ('*procT* × '*procT*) *set*
and *Rel* :: (('*procS*, '*procT*) *Proc* × ('*procS*, '*procT*) *Proc*) *set*
assumes *reflection*: *rel-reflects-barbs Rel* (*STCalWB SWB TWB*)
and *targetInRel*: $\forall$ *T1 T2*. (*T1*, *T2*) $\in$ *TRel* $\longrightarrow$ (*TargetTerm T1*, *TargetTerm T2*) $\in$ *Rel*

shows *rel-reflects-barbs TRel TWB*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelRTPO-impl-TRel-reflects-barbs*:
fixes *TRel* :: (*'procT* × *'procT*) *set*
assumes *reflection*: *rel-reflects-barbs* (*indRelRTPO TRel*) (*STCalWB SWB TWB*)
shows *rel-reflects-barbs TRel TWB*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelLTPO-impl-TRel-reflects-barbs*:
fixes *TRel* :: (*'procT* × *'procT*) *set*
assumes *reflection*: *rel-reflects-barbs* (*indRelLTPO TRel*) (*STCalWB SWB TWB*)
shows *rel-reflects-barbs TRel TWB*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelTEQ-impl-TRel-reflects-barbs*:
fixes *TRel* :: (*'procT* × *'procT*) *set*
assumes *reflection*: *rel-reflects-barbs* (*indRelTEQ TRel*) (*STCalWB SWB TWB*)
shows *rel-reflects-barbs TRel TWB*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *rel-with-target-impl-TRel-weakly-reflects-barbs*:
fixes *TRel* :: (*'procT* × *'procT*) *set*
and *Rel* :: ((*'procS*, *'procT*) *Proc* × (*'procS*, *'procT*) *Proc*) *set*
assumes *reflection*: *rel-weakly-reflects-barbs Rel* (*STCalWB SWB TWB*)
and *targetInRel*: ∀ *T1 T2*. (*T1*, *T2*) ∈ *TRel* ⟶ (*TargetTerm T1*, *TargetTerm T2*) ∈ *Rel*
shows *rel-weakly-reflects-barbs TRel TWB*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelRTPO-impl-TRel-weakly-reflects-barbs*:
fixes *TRel* :: (*'procT* × *'procT*) *set*
assumes *reflection*: *rel-weakly-reflects-barbs* (*indRelRTPO TRel*) (*STCalWB SWB TWB*)
shows *rel-weakly-reflects-barbs TRel TWB*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelLTPO-impl-TRel-weakly-reflects-barbs*:
fixes *TRel* :: (*'procT* × *'procT*) *set*
assumes *reflection*: *rel-weakly-reflects-barbs* (*indRelLTPO TRel*) (*STCalWB SWB TWB*)
shows *rel-weakly-reflects-barbs TRel TWB*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelTEQ-impl-TRel-weakly-reflects-barbs*:
fixes *TRel* :: (*'procT* × *'procT*) *set*
assumes *reflection*: *rel-weakly-reflects-barbs* (*indRelTEQ TRel*) (*STCalWB SWB TWB*)
shows *rel-weakly-reflects-barbs TRel TWB*
⟨*proof*⟩

If indRelRTPO, indRelLTPO, or indRelTPO respects barbs then so does the corresponding target term relation.

lemma (**in** *encoding-wrt-barbs*) *indRelRTPO-impl-TRel-respects-barbs*:
fixes *TRel* :: (*'procT* × *'procT*) *set*
assumes *respection*: *rel-respects-barbs* (*indRelRTPO TRel*) (*STCalWB SWB TWB*)
shows *rel-respects-barbs TRel TWB*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelLTPO-impl-TRel-respects-barbs*:
fixes *TRel* :: (*'procT* × *'procT*) *set*
assumes *respection*: *rel-respects-barbs* (*indRelLTPO TRel*) (*STCalWB SWB TWB*)
shows *rel-respects-barbs TRel TWB*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelTEQ-impl-TRel-respects-barbs*:
fixes $TRel :: ('procT \times 'procT)\ set$
assumes *respection*: *rel-respects-barbs* (*indRelTEQ TRel*) (*STCalWB SWB TWB*)
shows *rel-respects-barbs TRel TWB*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelRTPO-impl-TRel-weakly-respects-barbs*:
fixes $TRel :: ('procT \times 'procT)\ set$
assumes *respection*: *rel-weakly-respects-barbs* (*indRelRTPO TRel*) (*STCalWB SWB TWB*)
shows *rel-weakly-respects-barbs TRel TWB*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelLTPO-impl-TRel-weakly-respects-barbs*:
fixes $TRel :: ('procT \times 'procT)\ set$
assumes *respection*: *rel-weakly-respects-barbs* (*indRelLTPO TRel*) (*STCalWB SWB TWB*)
shows *rel-weakly-respects-barbs TRel TWB*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelTEQ-impl-TRel-weakly-respects-barbs*:
fixes $TRel :: ('procT \times 'procT)\ set$
assumes *respection*: *rel-weakly-respects-barbs* (*indRelTEQ TRel*) (*STCalWB SWB TWB*)
shows *rel-weakly-respects-barbs TRel TWB*
⟨*proof*⟩

If indRelRTPO, indRelLTPO, or indRelTEQ is a simulation then so is the corresponding target term relation.

lemma (**in** *encoding*) *rel-with-target-impl-transC-TRel-is-weak-reduction-simulation*:
fixes $TRel :: ('procT \times 'procT)\ set$
and $Rel :: (('procS, 'procT)\ Proc \times ('procS, 'procT)\ Proc)\ set$
assumes *sim*: *weak-reduction-simulation Rel* (*STCal Source Target*)
and *target*: $\forall T1\ T2.\ (T1, T2) \in TRel \longrightarrow (TargetTerm\ T1, TargetTerm\ T2) \in Rel$
and *trel*: $\forall T1\ T2.\ (TargetTerm\ T1, TargetTerm\ T2) \in Rel \longrightarrow (T1, T2) \in TRel^+$
shows *weak-reduction-simulation* $(TRel^+)$ *Target*
⟨*proof*⟩

lemma (**in** *encoding*) *indRelRTPO-impl-TRel-is-weak-reduction-simulation*:
fixes $TRel :: ('procT \times 'procT)\ set$
assumes *sim*: *weak-reduction-simulation* (*indRelRTPO TRel*) (*STCal Source Target*)
shows *weak-reduction-simulation* $(TRel^+)$ *Target*
⟨*proof*⟩

lemma (**in** *encoding*) *indRelLTPO-impl-TRel-is-weak-reduction-simulation*:
fixes $TRel :: ('procT \times 'procT)\ set$
assumes *sim*: *weak-reduction-simulation* (*indRelLTPO TRel*) (*STCal Source Target*)
shows *weak-reduction-simulation* $(TRel^+)$ *Target*
⟨*proof*⟩

lemma (**in** *encoding*) *rel-with-target-impl-transC-TRel-is-weak-reduction-simulation-rev*:
fixes $TRel :: ('procT \times 'procT)\ set$
and $Rel :: (('procS, 'procT)\ Proc \times ('procS, 'procT)\ Proc)\ set$
assumes *sim*: *weak-reduction-simulation* (Rel^{-1}) (*STCal Source Target*)
and *target*: $\forall T1\ T2.\ (T1, T2) \in TRel \longrightarrow (TargetTerm\ T1, TargetTerm\ T2) \in Rel$
and *trel*: $\forall T1\ T2.\ (TargetTerm\ T1, TargetTerm\ T2) \in Rel \longrightarrow (T1, T2) \in TRel^+$
shows *weak-reduction-simulation* $((TRel^+)^{-1})$ *Target*
⟨*proof*⟩

lemma (**in** *encoding*) *indRelRTPO-impl-TRel-is-weak-reduction-simulation-rev*:
fixes $TRel :: ('procT \times 'procT)\ set$
assumes *sim*: *weak-reduction-simulation* $((indRelRTPO\ TRel)^{-1})$ (*STCal Source Target*)

shows *weak-reduction-simulation* $((TRel^+)^{-1})$ *Target*
⟨*proof*⟩

lemma (**in** *encoding*) *indRelLTPO-impl-TRel-is-weak-reduction-simulation-rev*:
fixes *TRel* :: $('procT \times 'procT)$ *set*
assumes *sim*: *weak-reduction-simulation* $((indRelLTPO\ TRel)^{-1})$ (*STCal Source Target*)
shows *weak-reduction-simulation* $((TRel^+)^{-1})$ *Target*
⟨*proof*⟩

lemma (**in** *encoding*) *rel-with-target-impl-reflC-transC-TRel-is-weak-reduction-simulation*:
fixes *TRel* :: $('procT \times 'procT)$ *set*
and *Rel* :: $(('procS, 'procT)\ Proc \times ('procS, 'procT)\ Proc)$ *set*
assumes *sim*: *weak-reduction-simulation Rel* (*STCal Source Target*)
and *target*: $\forall T1\ T2.\ (T1, T2) \in TRel \longrightarrow (TargetTerm\ T1, TargetTerm\ T2) \in Rel$
and *trel*: $\forall T1\ T2.\ (TargetTerm\ T1, TargetTerm\ T2) \in Rel \longrightarrow (T1, T2) \in TRel^*$
shows *weak-reduction-simulation* $(TRel^*)$ *Target*
⟨*proof*⟩

lemma (**in** *encoding*) *indRelTEQ-impl-TRel-is-weak-reduction-simulation*:
fixes *TRel* :: $('procT \times 'procT)$ *set*
assumes *sim*: *weak-reduction-simulation* (*indRelTEQ TRel*) (*STCal Source Target*)
shows *weak-reduction-simulation* $(TRel^*)$ *Target*
⟨*proof*⟩

lemma (**in** *encoding*) *rel-with-target-impl-transC-TRel-is-strong-reduction-simulation*:
fixes *TRel* :: $('procT \times 'procT)$ *set*
and *Rel* :: $(('procS, 'procT)\ Proc \times ('procS, 'procT)\ Proc)$ *set*
assumes *sim*: *strong-reduction-simulation Rel* (*STCal Source Target*)
and *target*: $\forall T1\ T2.\ (T1, T2) \in TRel \longrightarrow (TargetTerm\ T1, TargetTerm\ T2) \in Rel$
and *trel*: $\forall T1\ T2.\ (TargetTerm\ T1, TargetTerm\ T2) \in Rel \longrightarrow (T1, T2) \in TRel^+$
shows *strong-reduction-simulation* $(TRel^+)$ *Target*
⟨*proof*⟩

lemma (**in** *encoding*) *indRelRTPO-impl-TRel-is-strong-reduction-simulation*:
fixes *TRel* :: $('procT \times 'procT)$ *set*
assumes *sim*: *strong-reduction-simulation* (*indRelRTPO TRel*) (*STCal Source Target*)
shows *strong-reduction-simulation* $(TRel^+)$ *Target*
⟨*proof*⟩

lemma (**in** *encoding*) *indRelLTPO-impl-TRel-is-strong-reduction-simulation*:
fixes *TRel* :: $('procT \times 'procT)$ *set*
assumes *sim*: *strong-reduction-simulation* (*indRelLTPO TRel*) (*STCal Source Target*)
shows *strong-reduction-simulation* $(TRel^+)$ *Target*
⟨*proof*⟩

lemma (**in** *encoding*) *rel-with-target-impl-transC-TRel-is-strong-reduction-simulation-rev*:
fixes *TRel* :: $('procT \times 'procT)$ *set*
and *Rel* :: $(('procS, 'procT)\ Proc \times ('procS, 'procT)\ Proc)$ *set*
assumes *sim*: *strong-reduction-simulation* (Rel^{-1}) (*STCal Source Target*)
and *target*: $\forall T1\ T2.\ (T1, T2) \in TRel \longrightarrow (TargetTerm\ T1, TargetTerm\ T2) \in Rel$
and *trel*: $\forall T1\ T2.\ (TargetTerm\ T1, TargetTerm\ T2) \in Rel \longrightarrow (T1, T2) \in TRel^+$
shows *strong-reduction-simulation* $((TRel^+)^{-1})$ *Target*
⟨*proof*⟩

lemma (**in** *encoding*) *indRelRTPO-impl-TRel-is-strong-reduction-simulation-rev*:
fixes *TRel* :: $('procT \times 'procT)$ *set*
assumes *sim*: *strong-reduction-simulation* $((indRelRTPO\ TRel)^{-1})$ (*STCal Source Target*)
shows *strong-reduction-simulation* $((TRel^+)^{-1})$ *Target*
⟨*proof*⟩

lemma (**in** *encoding*) *indRelLTPO-impl-TRel-is-strong-reduction-simulation-rev*:

fixes $TRel$ :: ('procT × 'procT) set
assumes *sim*: *strong-reduction-simulation* $((indRelLTPO\ TRel)^{-1})$ (*STCal Source Target*)
shows *strong-reduction-simulation* $((TRel^{+})^{-1})$ *Target*
⟨*proof*⟩

lemma (**in** *encoding*) *rel-with-target-impl-reflC-transC-TRel-is-strong-reduction-simulation*:
fixes *TRel* :: ('procT × 'procT) set
and *Rel* :: (('procS, 'procT) Proc × ('procS, 'procT) Proc) set
assumes *sim*: *strong-reduction-simulation Rel* (*STCal Source Target*)
and *target*: $\forall$ *T1 T2*. (*T1*, *T2*) ∈ *TRel* ⟶ (*TargetTerm T1*, *TargetTerm T2*) ∈ *Rel*
and *trel*: $\forall$ *T1 T2*. (*TargetTerm T1*, *TargetTerm T2*) ∈ *Rel*
⟶ (*T1*, *T2*) ∈ $TRel^{*}$
shows *strong-reduction-simulation* $(TRel^{*})$ *Target*
⟨*proof*⟩

lemma (**in** *encoding*) *indRelTEQ-impl-TRel-is-strong-reduction-simulation*:
fixes *TRel* :: ('procT × 'procT) set
assumes *sim*: *strong-reduction-simulation* (*indRelTEQ TRel*) (*STCal Source Target*)
shows *strong-reduction-simulation* $(TRel^{*})$ *Target*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelRTPO-impl-TRel-is-weak-barbed-simulation*:
fixes *TRel* :: ('procT × 'procT) set
assumes *sim*: *weak-barbed-simulation* (*indRelRTPO TRel*) (*STCalWB SWB TWB*)
shows *weak-barbed-simulation* $(TRel^{+})$ *TWB*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelLTPO-impl-TRel-is-weak-barbed-simulation*:
fixes *TRel* :: ('procT × 'procT) set
assumes *sim*: *weak-barbed-simulation* (*indRelLTPO TRel*) (*STCalWB SWB TWB*)
shows *weak-barbed-simulation* $(TRel^{+})$ *TWB*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelTEQ-impl-TRel-is-weak-barbed-simulation*:
fixes *TRel* :: ('procT × 'procT) set
assumes *sim*: *weak-barbed-simulation* (*indRelTEQ TRel*) (*STCalWB SWB TWB*)
shows *weak-barbed-simulation* $(TRel^{*})$ *TWB*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelRTPO-impl-TRel-is-strong-barbed-simulation*:
fixes *TRel* :: ('procT × 'procT) set
assumes *sim*: *strong-barbed-simulation* (*indRelRTPO TRel*) (*STCalWB SWB TWB*)
shows *strong-barbed-simulation* $(TRel^{+})$ *TWB*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelLTPO-impl-TRel-is-strong-barbed-simulation*:
fixes *TRel* :: ('procT × 'procT) set
assumes *sim*: *strong-barbed-simulation* (*indRelLTPO TRel*) (*STCalWB SWB TWB*)
shows *strong-barbed-simulation* $(TRel^{+})$ *TWB*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelTEQ-impl-TRel-is-strong-barbed-simulation*:
fixes *TRel* :: ('procT × 'procT) set
assumes *sim*: *strong-barbed-simulation* (*indRelTEQ TRel*) (*STCalWB SWB TWB*)
shows *strong-barbed-simulation* $(TRel^{*})$ *TWB*
⟨*proof*⟩

If indRelRTPO, indRelLTPO, or indRelTEQ is a contrasimulation then so is the corresponding target term relation.

lemma (**in** *encoding*) *rel-with-target-impl-transC-TRel-is-weak-reduction-contrasimulation*:

fixes *TRel* :: (*'procT* × *'procT*) *set*
and *Rel* :: ((*'procS*, *'procT*) *Proc* × (*'procS*, *'procT*) *Proc*) *set*
assumes *conSim*: *weak-reduction-contrasimulation Rel* (*STCal Source Target*)
and *target*: $\forall$ *T1 T2*. (*T1*, *T2*) $\in$ *TRel* $\longrightarrow$ (*TargetTerm T1*, *TargetTerm T2*) $\in$ *Rel*
and *trel*: $\forall$ *T1 T2*. (*TargetTerm T1*, *TargetTerm T2*) $\in$ *Rel* $\longrightarrow$ (*T1*, *T2*) $\in$ $TRel^+$
shows *weak-reduction-contrasimulation* ($TRel^+$) *Target*
⟨*proof*⟩

lemma (**in** *encoding*) *indRelRTPO-impl-TRel-is-weak-reduction-contrasimulation*:
fixes *TRel* :: (*'procT* × *'procT*) *set*
assumes *conSim*: *weak-reduction-contrasimulation* (*indRelRTPO TRel*) (*STCal Source Target*)
shows *weak-reduction-contrasimulation* ($TRel^+$) *Target*
⟨*proof*⟩

lemma (**in** *encoding*) *indRelLTPO-impl-TRel-is-weak-reduction-contrasimulation*:
fixes *TRel* :: (*'procT* × *'procT*) *set*
assumes *conSim*: *weak-reduction-contrasimulation* (*indRelLTPO TRel*) (*STCal Source Target*)
shows *weak-reduction-contrasimulation* ($TRel^+$) *Target*
⟨*proof*⟩

lemma (**in** *encoding*) *rel-with-target-impl-reflC-transC-TRel-is-weak-reduction-contrasimulation*:
fixes *TRel* :: (*'procT* × *'procT*) *set*
and *Rel* :: ((*'procS*, *'procT*) *Proc* × (*'procS*, *'procT*) *Proc*) *set*
assumes *conSim*: *weak-reduction-contrasimulation Rel* (*STCal Source Target*)
and *target*: $\forall$ *T1 T2*. (*T1*, *T2*) $\in$ *TRel* $\longrightarrow$ (*TargetTerm T1*, *TargetTerm T2*) $\in$ *Rel*
and *trel*: $\forall$ *T1 T2*. (*TargetTerm T1*, *TargetTerm T2*) $\in$ *Rel* $\longrightarrow$ (*T1*, *T2*) $\in$ $TRel^*$
shows *weak-reduction-contrasimulation* ($TRel^*$) *Target*
⟨*proof*⟩

lemma (**in** *encoding*) *indRelTEQ-impl-TRel-is-weak-reduction-contrasimulation*:
fixes *TRel* :: (*'procT* × *'procT*) *set*
assumes *conSim*: *weak-reduction-contrasimulation* (*indRelTEQ TRel*) (*STCal Source Target*)
shows *weak-reduction-contrasimulation* ($TRel^*$) *Target*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelRTPO-impl-TRel-is-weak-barbed-contrasimulation*:
fixes *TRel* :: (*'procT* × *'procT*) *set*
assumes *conSim*: *weak-barbed-contrasimulation* (*indRelRTPO TRel*) (*STCalWB SWB TWB*)
shows *weak-barbed-contrasimulation* ($TRel^+$) *TWB*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelLTPO-impl-TRel-is-weak-barbed-contrasimulation*:
fixes *TRel* :: (*'procT* × *'procT*) *set*
assumes *conSim*: *weak-barbed-contrasimulation* (*indRelLTPO TRel*) (*STCalWB SWB TWB*)
shows *weak-barbed-contrasimulation* ($TRel^+$) *TWB*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelTEQ-impl-TRel-is-weak-barbed-contrasimulation*:
fixes *TRel* :: (*'procT* × *'procT*) *set*
assumes *conSim*: *weak-barbed-contrasimulation* (*indRelTEQ TRel*) (*STCalWB SWB TWB*)
shows *weak-barbed-contrasimulation* ($TRel^*$) *TWB*
⟨*proof*⟩

If indRelRTPO, indRelLTPO, or indRelTEQ is a coupled simulation then so is the corresponding target term relation.

lemma (**in** *encoding*) *indRelRTPO-impl-TRel-is-weak-reduction-coupled-simulation*:
fixes *TRel* :: (*'procT* × *'procT*) *set*
assumes *couSim*: *weak-reduction-coupled-simulation* (*indRelRTPO TRel*) (*STCal Source Target*)
shows *weak-reduction-coupled-simulation* ($TRel^+$) *Target*
⟨*proof*⟩

lemma (**in** *encoding*) *indRelLTPO-impl-TRel-is-weak-reduction-coupled-simulation*:
fixes *TRel* :: ('*procT* × '*procT*) *set*
assumes *couSim*: *weak-reduction-coupled-simulation* (*indRelLTPO TRel*) (*STCal Source Target*)
shows *weak-reduction-coupled-simulation* ($TRel^+$) *Target*
⟨*proof*⟩

lemma (**in** *encoding*) *indRelTEQ-impl-TRel-is-weak-reduction-coupled-simulation*:
fixes *TRel* :: ('*procT* × '*procT*) *set*
assumes *couSim*: *weak-reduction-coupled-simulation* (*indRelTEQ TRel*) (*STCal Source Target*)
shows *weak-reduction-coupled-simulation* ($TRel^*$) *Target*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelRTPO-impl-TRel-is-weak-barbed-coupled-simulation*:
fixes *TRel* :: ('*procT* × '*procT*) *set*
assumes *couSim*: *weak-barbed-coupled-simulation* (*indRelRTPO TRel*) (*STCalWB SWB TWB*)
shows *weak-barbed-coupled-simulation* ($TRel^+$) *TWB*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelLTPO-impl-TRel-is-weak-barbed-coupled-simulation*:
fixes *TRel* :: ('*procT* × '*procT*) *set*
assumes *couSim*: *weak-barbed-coupled-simulation* (*indRelLTPO TRel*) (*STCalWB SWB TWB*)
shows *weak-barbed-coupled-simulation* ($TRel^+$) *TWB*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelTEQ-impl-TRel-is-weak-barbed-coupled-simulation*:
fixes *TRel* :: ('*procT* × '*procT*) *set*
assumes *couSim*: *weak-barbed-coupled-simulation* (*indRelTEQ TRel*) (*STCalWB SWB TWB*)
shows *weak-barbed-coupled-simulation* ($TRel^*$) *TWB*
⟨*proof*⟩

If indRelRTPO, indRelLTPO, or indRelTEQ is a correspondence simulation then so is the corresponding target term relation.

lemma (**in** *encoding*) *rel-with-target-impl-transC-TRel-is-weak-reduction-correspondence-simulation*:
fixes *TRel* :: ('*procT* × '*procT*) *set*
and *Rel* :: (('*procS*, '*procT*) *Proc* × ('*procS*, '*procT*) *Proc*) *set*
assumes *corSim*: *weak-reduction-correspondence-simulation Rel* (*STCal Source Target*)
and *target*: $\forall$ *T1 T2*. (*T1*, *T2*) $\in$ *TRel* $\longrightarrow$ (*TargetTerm T1*, *TargetTerm T2*) $\in$ *Rel*
and *trel*: $\forall$ *T1 T2*. (*TargetTerm T1*, *TargetTerm T2*) $\in$ *Rel* $\longrightarrow$ (*T1*, *T2*) $\in$ $TRel^+$
shows *weak-reduction-correspondence-simulation* ($TRel^+$) *Target*
⟨*proof*⟩

lemma (**in** *encoding*) *indRelRTPO-impl-TRel-is-weak-reduction-correspondence-simulation*:
fixes *TRel* :: ('*procT* × '*procT*) *set*
assumes *cSim*: *weak-reduction-correspondence-simulation* (*indRelRTPO TRel*) (*STCal Source Target*)
shows *weak-reduction-correspondence-simulation* ($TRel^+$) *Target*
⟨*proof*⟩

lemma (**in** *encoding*) *indRelLTPO-impl-TRel-is-weak-reduction-correspondence-simulation*:
fixes *TRel* :: ('*procT* × '*procT*) *set*
assumes *cSim*: *weak-reduction-correspondence-simulation* (*indRelLTPO TRel*) (*STCal Source Target*)
shows *weak-reduction-correspondence-simulation* ($TRel^+$) *Target*
⟨*proof*⟩

lemma (**in** *encoding*)
rel-with-target-impl-reflC-transC-TRel-is-weak-reduction-correspondence-simulation:
fixes *TRel* :: ('*procT* × '*procT*) *set*
and *Rel* :: (('*procS*, '*procT*) *Proc* × ('*procS*, '*procT*) *Proc*) *set*
assumes *corSim*: *weak-reduction-correspondence-simulation Rel* (*STCal Source Target*)
and *target*: $\forall$ *T1 T2*. (*T1*, *T2*) $\in$ *TRel* $\longrightarrow$ (*TargetTerm T1*, *TargetTerm T2*) $\in$ *Rel*

and *trel*: $\forall$ *T1 T2*. (*TargetTerm T1*, *TargetTerm T2*) $\in$ *Rel* $\longrightarrow$ (*T1*, *T2*) $\in$ *TRel**
shows *weak-reduction-correspondence-simulation* (*TRel**) *Target*
⟨*proof*⟩

lemma (**in** *encoding*) *indRelTEQ-impl-TRel-is-weak-reduction-correspondence-simulation*:
fixes *TRel* :: (*'procT* $\times$ *'procT*) *set*
assumes *corSim*: *weak-reduction-correspondence-simulation* (*indRelTEQ TRel*) (*STCal Source Target*)
shows *weak-reduction-correspondence-simulation* (*TRel**) *Target*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelRTPO-impl-TRel-is-weak-barbed-correspondence-simulation*:
fixes *TRel* :: (*'procT* $\times$ *'procT*) *set*
assumes *corSim*: *weak-barbed-correspondence-simulation* (*indRelRTPO TRel*) (*STCalWB SWB TWB*)
shows *weak-barbed-correspondence-simulation* (*TRel*$^+$) *TWB*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelLTPO-impl-TRel-is-weak-barbed-correspondence-simulation*:
fixes *TRel* :: (*'procT* $\times$ *'procT*) *set*
assumes *corSim*: *weak-barbed-correspondence-simulation* (*indRelLTPO TRel*) (*STCalWB SWB TWB*)
shows *weak-barbed-correspondence-simulation* (*TRel*$^+$) *TWB*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelTEQ-impl-TRel-is-weak-barbed-correspondence-simulation*:
fixes *TRel* :: (*'procT* $\times$ *'procT*) *set*
assumes *corSim*: *weak-barbed-correspondence-simulation* (*indRelTEQ TRel*) (*STCalWB SWB TWB*)
shows *weak-barbed-correspondence-simulation* (*TRel**) *TWB*
⟨*proof*⟩

If indRelRTPO, indRelLTPO, or indRelTEQ is a bisimulation then so is the corresponding target term relation.

lemma (**in** *encoding*) *rel-with-target-impl-transC-TRel-is-weak-reduction-bisimulation*:
fixes *TRel* :: (*'procT* $\times$ *'procT*) *set*
and *Rel* :: ((*'procS*, *'procT*) *Proc* $\times$ (*'procS*, *'procT*) *Proc*) *set*
assumes *bisim*: *weak-reduction-bisimulation Rel* (*STCal Source Target*)
and *target*: $\forall$ *T1 T2*. (*T1*, *T2*) $\in$ *TRel* $\longrightarrow$ (*TargetTerm T1*, *TargetTerm T2*) $\in$ *Rel*
and *trel*: $\forall$ *T1 T2*. (*TargetTerm T1*, *TargetTerm T2*) $\in$ *Rel* $\longrightarrow$ (*T1*, *T2*) $\in$ *TRel*$^+$
shows *weak-reduction-bisimulation* (*TRel*$^+$) *Target*
⟨*proof*⟩

lemma (**in** *encoding*) *indRelRTPO-impl-TRel-is-weak-reduction-bisimulation*:
fixes *TRel* :: (*'procT* $\times$ *'procT*) *set*
assumes *bisim*: *weak-reduction-bisimulation* (*indRelRTPO TRel*) (*STCal Source Target*)
shows *weak-reduction-bisimulation* (*TRel*$^+$) *Target*
⟨*proof*⟩

lemma (**in** *encoding*) *indRelLTPO-impl-TRel-is-weak-reduction-bisimulation*:
fixes *TRel* :: (*'procT* $\times$ *'procT*) *set*
assumes *bisim*: *weak-reduction-bisimulation* (*indRelLTPO TRel*) (*STCal Source Target*)
shows *weak-reduction-bisimulation* (*TRel*$^+$) *Target*
⟨*proof*⟩

lemma (**in** *encoding*) *rel-with-target-impl-reflC-transC-TRel-is-weak-reduction-bisimulation*:
fixes *TRel* :: (*'procT* $\times$ *'procT*) *set*
and *Rel* :: ((*'procS*, *'procT*) *Proc* $\times$ (*'procS*, *'procT*) *Proc*) *set*
assumes *bisim*: *weak-reduction-bisimulation Rel* (*STCal Source Target*)
and *target*: $\forall$ *T1 T2*. (*T1*, *T2*) $\in$ *TRel* $\longrightarrow$ (*TargetTerm T1*, *TargetTerm T2*) $\in$ *Rel*
and *trel*: $\forall$ *T1 T2*. (*TargetTerm T1*, *TargetTerm T2*) $\in$ *Rel* $\longrightarrow$ (*T1*, *T2*) $\in$ *TRel**
shows *weak-reduction-bisimulation* (*TRel**) *Target*
⟨*proof*⟩

lemma (**in** *encoding*) *indRelTEQ-impl-TRel-is-weak-reduction-bisimulation*:
  fixes *TRel* :: ('*procT* × '*procT*) *set*
  assumes *bisim*: *weak-reduction-bisimulation* (*indRelTEQ TRel*) (*STCal Source Target*)
  shows *weak-reduction-bisimulation* ($TRel^*$) *Target*
    ⟨*proof*⟩

lemma (**in** *encoding*) *rel-with-target-impl-transC-TRel-is-strong-reduction-bisimulation*:
  fixes *TRel* :: ('*procT* × '*procT*) *set*
   and *Rel* :: (('*procS*, '*procT*) *Proc* × ('*procS*, '*procT*) *Proc*) *set*
  assumes *bisim*: *strong-reduction-bisimulation Rel* (*STCal Source Target*)
    and *target*: ∀ *T1 T2*. (*T1*, *T2*) ∈ *TRel* ⟶ (*TargetTerm T1*, *TargetTerm T2*) ∈ *Rel*
    and *trel*: ∀ *T1 T2*. (*TargetTerm T1*, *TargetTerm T2*) ∈ *Rel* ⟶ (*T1*, *T2*) ∈ $TRel^+$
  shows *strong-reduction-bisimulation* ($TRel^+$) *Target*
⟨*proof*⟩

lemma (**in** *encoding*) *indRelRTPO-impl-TRel-is-strong-reduction-bisimulation*:
  fixes *TRel* :: ('*procT* × '*procT*) *set*
  assumes *bisim*: *strong-reduction-bisimulation* (*indRelRTPO TRel*) (*STCal Source Target*)
  shows *strong-reduction-bisimulation* ($TRel^+$) *Target*
    ⟨*proof*⟩

lemma (**in** *encoding*) *indRelLTPO-impl-TRel-is-strong-reduction-bisimulation*:
  fixes *TRel* :: ('*procT* × '*procT*) *set*
  assumes *bisim*: *strong-reduction-bisimulation* (*indRelLTPO TRel*) (*STCal Source Target*)
  shows *strong-reduction-bisimulation* ($TRel^+$) *Target*
    ⟨*proof*⟩

lemma (**in** *encoding*) *rel-with-target-impl-reflC-transC-TRel-is-strong-reduction-bisimulation*:
  fixes *TRel* :: ('*procT* × '*procT*) *set*
   and *Rel* :: (('*procS*, '*procT*) *Proc* × ('*procS*, '*procT*) *Proc*) *set*
  assumes *bisim*: *strong-reduction-bisimulation Rel* (*STCal Source Target*)
    and *target*: ∀ *T1 T2*. (*T1*, *T2*) ∈ *TRel* ⟶ (*TargetTerm T1*, *TargetTerm T2*) ∈ *Rel*
    and *trel*: ∀ *T1 T2*. (*TargetTerm T1*, *TargetTerm T2*) ∈ *Rel* ⟶ (*T1*, *T2*) ∈ $TRel^*$
  shows *strong-reduction-bisimulation* ($TRel^*$) *Target*
⟨*proof*⟩

lemma (**in** *encoding*) *indRelTEQ-impl-TRel-is-strong-reduction-bisimulation*:
  fixes *TRel* :: ('*procT* × '*procT*) *set*
  assumes *bisim*: *strong-reduction-bisimulation* (*indRelTEQ TRel*) (*STCal Source Target*)
  shows *strong-reduction-bisimulation* ($TRel^*$) *Target*
    ⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelRTPO-impl-TRel-is-weak-barbed-bisimulation*:
  fixes *TRel* :: ('*procT* × '*procT*) *set*
  assumes *bisim*: *weak-barbed-bisimulation* (*indRelRTPO TRel*) (*STCalWB SWB TWB*)
  shows *weak-barbed-bisimulation* ($TRel^+$) *TWB*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelLTPO-impl-TRel-is-weak-barbed-bisimulation*:
  fixes *TRel* :: ('*procT* × '*procT*) *set*
  assumes *bisim*: *weak-barbed-bisimulation* (*indRelLTPO TRel*) (*STCalWB SWB TWB*)
  shows *weak-barbed-bisimulation* ($TRel^+$) *TWB*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelTEQ-impl-TRel-is-weak-barbed-bisimulation*:
  fixes *TRel* :: ('*procT* × '*procT*) *set*
  assumes *bisim*: *weak-barbed-bisimulation* (*indRelTEQ TRel*) (*STCalWB SWB TWB*)
  shows *weak-barbed-bisimulation* ($TRel^*$) *TWB*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelRTPO-impl-TRel-is-strong-barbed-bisimulation*:

fixes *TRel* :: (*'procT* × *'procT*) *set*
assumes *bisim*: *strong-barbed-bisimulation* (*indRelRTPO TRel*) (*STCalWB SWB TWB*)
shows *strong-barbed-bisimulation* ($TRel^+$) *TWB*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelLTPO-impl-TRel-is-strong-barbed-bisimulation*:
fixes *TRel* :: (*'procT* × *'procT*) *set*
assumes *bisim*: *strong-barbed-bisimulation* (*indRelLTPO TRel*) (*STCalWB SWB TWB*)
shows *strong-barbed-bisimulation* ($TRel^+$) *TWB*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelTEQ-impl-TRel-is-strong-barbed-bisimulation*:
fixes *TRel* :: (*'procT* × *'procT*) *set*
assumes *bisim*: *strong-barbed-bisimulation* (*indRelTEQ TRel*) (*STCalWB SWB TWB*)
shows *strong-barbed-bisimulation* ($TRel^*$) *TWB*
⟨*proof*⟩

5.3 Relations Induced by the Encoding and Relations on Source Terms and Target Terms

Some encodability like e.g. full abstraction are defined w.r.t. a relation on source terms and a relation on target terms. To analyse such criteria we include these two relations in the considered relation on the disjoint union of source and target terms.

inductive-set (**in** *encoding*) *indRelRST*
:: (*'procS* × *'procS*) *set* ⇒ (*'procT* × *'procT*) *set*
⇒ (((*'procS*, *'procT*) *Proc*) × ((*'procS*, *'procT*) *Proc*)) *set*
for *SRel* :: (*'procS* × *'procS*) *set*
and *TRel* :: (*'procT* × *'procT*) *set*
where
encR: (*SourceTerm S*, *TargetTerm* ($[\![S]\!]$)) ∈ *indRelRST SRel TRel* |
source: (*S1*, *S2*) ∈ *SRel* ⟹ (*SourceTerm S1*, *SourceTerm S2*) ∈ *indRelRST SRel TRel* |
target: (*T1*, *T2*) ∈ *TRel* ⟹ (*TargetTerm T1*, *TargetTerm T2*) ∈ *indRelRST SRel TRel*

abbreviation (**in** *encoding*) *indRelRSTinfix*
:: (*'procS*, *'procT*) *Proc* ⇒ (*'procS* × *'procS*) *set* ⇒ (*'procT* × *'procT*) *set*
⇒ (*'procS*, *'procT*) *Proc* ⇒ *bool* ($- \mathcal{R}[\![\cdot]\!]R<-,-> -$ [*75*, *75*, *75*, *75*] *80*)
where
$P\ \mathcal{R}[\![\cdot]\!]R<SRel,TRel>\ Q \equiv (P,\ Q) \in indRelRST\ SRel\ TRel$

inductive-set (**in** *encoding*) *indRelRSTPO*
:: (*'procS* × *'procS*) *set* ⇒ (*'procT* × *'procT*) *set*
⇒ (((*'procS*, *'procT*) *Proc*) × ((*'procS*, *'procT*) *Proc*)) *set*
for *SRel* :: (*'procS* × *'procS*) *set*
and *TRel* :: (*'procT* × *'procT*) *set*
where
encR: (*SourceTerm S*, *TargetTerm* ($[\![S]\!]$)) ∈ *indRelRSTPO SRel TRel* |
source: (*S1*, *S2*) ∈ *SRel* ⟹ (*SourceTerm S1*, *SourceTerm S2*) ∈ *indRelRSTPO SRel TRel* |
target: (*T1*, *T2*) ∈ *TRel* ⟹ (*TargetTerm T1*, *TargetTerm T2*) ∈ *indRelRSTPO SRel TRel* |
trans: ⟦(*P*, *Q*) ∈ *indRelRSTPO SRel TRel*; (*Q*, *R*) ∈ *indRelRSTPO SRel TRel*⟧
⟹ (*P*, *R*) ∈ *indRelRSTPO SRel TRel*

abbreviation (**in** *encoding*) *indRelRSTPOinfix* ::
(*'procS*, *'procT*) *Proc* ⇒ (*'procS* × *'procS*) *set* ⇒ (*'procT* × *'procT*) *set*
⇒ (*'procS*, *'procT*) *Proc* ⇒ *bool* ($- \lesssim[\![\cdot]\!]R<-,-> -$ [*75*, *75*, *75*, *75*] *80*)
where
$P \lesssim[\![\cdot]\!]R<SRel,TRel>\ Q \equiv (P,\ Q) \in indRelRSTPO\ SRel\ TRel$

lemma (**in** *encoding*) *indRelRSTPO-refl*:
fixes *SRel* :: (*'procS* × *'procS*) *set*
and *TRel* :: (*'procT* × *'procT*) *set*

assumes *reflS*: *refl SRel*
and *reflT*: *refl TRel*
shows *refl* (*indRelRSTPO SRel TRel*)
⟨*proof*⟩

lemma (**in** *encoding*) *indRelRSTPO-trans*:
fixes *SRel* :: (*'procS* × *'procS*) *set*
and *TRel* :: (*'procT* × *'procT*) *set*
shows *trans* (*indRelRSTPO SRel TRel*)
⟨*proof*⟩

lemma (**in** *encoding*) *refl-trans-closure-of-indRelRST*:
fixes *SRel* :: (*'procS* × *'procS*) *set*
and *TRel* :: (*'procT* × *'procT*) *set*
assumes *reflS*: *refl SRel*
and *reflT*: *refl TRel*
shows *indRelRSTPO SRel TRel* = (*indRelRST SRel TRel*)*
⟨*proof*⟩

inductive-set (**in** *encoding*) *indRelLST*
:: (*'procS* × *'procS*) *set* ⇒ (*'procT* × *'procT*) *set*
⇒ (((*'procS*, *'procT*) *Proc*) × ((*'procS*, *'procT*) *Proc*)) *set*
for *SRel* :: (*'procS* × *'procS*) *set*
and *TRel* :: (*'procT* × *'procT*) *set*
where
encL: (*TargetTerm* (⟦*S*⟧), *SourceTerm S*) ∈ *indRelLST SRel TRel* |
source: (*S1*, *S2*) ∈ *SRel* ⟹ (*SourceTerm S1*, *SourceTerm S2*) ∈ *indRelLST SRel TRel* |
target: (*T1*, *T2*) ∈ *TRel* ⟹ (*TargetTerm T1*, *TargetTerm T2*) ∈ *indRelLST SRel TRel*

abbreviation (**in** *encoding*) *indRelLSTinfix*
:: (*'procS*, *'procT*) *Proc* ⇒ (*'procS* × *'procS*) *set* ⇒ (*'procT* × *'procT*) *set*
⇒ (*'procS*, *'procT*) *Proc* ⇒ *bool* (- $\mathcal{R}$⟦·⟧*L*<-,-> - [*75*, *75*, *75*, *75*] *80*)
where
P $\mathcal{R}$⟦·⟧*L*<*SRel*,*TRel*> *Q* ≡ (*P*, *Q*) ∈ *indRelLST SRel TRel*

inductive-set (**in** *encoding*) *indRelLSTPO*
:: (*'procS* × *'procS*) *set* ⇒ (*'procT* × *'procT*) *set*
⇒ (((*'procS*, *'procT*) *Proc*) × ((*'procS*, *'procT*) *Proc*)) *set*
for *SRel* :: (*'procS* × *'procS*) *set*
and *TRel* :: (*'procT* × *'procT*) *set*
where
encL: (*TargetTerm* (⟦*S*⟧), *SourceTerm S*) ∈ *indRelLSTPO SRel TRel* |
source: (*S1*, *S2*) ∈ *SRel* ⟹ (*SourceTerm S1*, *SourceTerm S2*) ∈ *indRelLSTPO SRel TRel* |
target: (*T1*, *T2*) ∈ *TRel* ⟹ (*TargetTerm T1*, *TargetTerm T2*) ∈ *indRelLSTPO SRel TRel* |
trans: ⟦(*P*, *Q*) ∈ *indRelLSTPO SRel TRel*; (*Q*, *R*) ∈ *indRelLSTPO SRel TRel*⟧
⟹ (*P*, *R*) ∈ *indRelLSTPO SRel TRel*

abbreviation (**in** *encoding*) *indRelLSTPOinfix*
:: (*'procS*, *'procT*) *Proc* ⇒ (*'procS* × *'procS*) *set* ⇒ (*'procT* × *'procT*) *set*
⇒ (*'procS*, *'procT*) *Proc* ⇒ *bool* (- ≲⟦·⟧*L*<-,-> - [*75*, *75*, *75*, *75*] *80*)
where
P ≲⟦·⟧*L*<*SRel*,*TRel*> *Q* ≡ (*P*, *Q*) ∈ *indRelLSTPO SRel TRel*

lemma (**in** *encoding*) *indRelLSTPO-refl*:
fixes *SRel* :: (*'procS* × *'procS*) *set*
and *TRel* :: (*'procT* × *'procT*) *set*
assumes *reflS*: *refl SRel*
and *reflT*: *refl TRel*
shows *refl* (*indRelLSTPO SRel TRel*)
⟨*proof*⟩

lemma (**in** *encoding*) *indRelLSTPO-trans*:
fixes *SRel* :: ('*procS* × '*procS*) *set*
and *TRel* :: ('*procT* × '*procT*) *set*
shows *trans* (*indRelLSTPO SRel TRel*)
⟨*proof*⟩

lemma (**in** *encoding*) *refl-trans-closure-of-indRelLST*:
fixes *SRel* :: ('*procS* × '*procS*) *set*
and *TRel* :: ('*procT* × '*procT*) *set*
assumes *reflS*: *refl SRel*
and *reflT*: *refl TRel*
shows *indRelLSTPO SRel TRel* = (*indRelLST SRel TRel*)*
⟨*proof*⟩

inductive-set (**in** *encoding*) *indRelST*
:: ('*procS* × '*procS*) *set* ⇒ ('*procT* × '*procT*) *set*
⇒ ((('*procS*, '*procT*) *Proc*) × (('*procS*, '*procT*) *Proc*)) *set*
for *SRel* :: ('*procS* × '*procS*) *set*
and *TRel* :: ('*procT* × '*procT*) *set*
where
encR: (*SourceTerm S*, *TargetTerm* (⟦*S*⟧)) ∈ *indRelST SRel TRel* |
encL: (*TargetTerm* (⟦*S*⟧), *SourceTerm S*) ∈ *indRelST SRel TRel* |
source: (*S1*, *S2*) ∈ *SRel* ⟹ (*SourceTerm S1*, *SourceTerm S2*) ∈ *indRelST SRel TRel* |
target: (*T1*, *T2*) ∈ *TRel* ⟹ (*TargetTerm T1*, *TargetTerm T2*) ∈ *indRelST SRel TRel*

abbreviation (**in** *encoding*) *indRelSTinfix*
:: ('*procS*, '*procT*) *Proc* ⇒ ('*procS* × '*procS*) *set* ⇒ ('*procT* × '*procT*) *set*
⇒ ('*procS*, '*procT*) *Proc* ⇒ *bool* (- 𝓡⟦·⟧<-,-> - [*75*, *75*, *75*, *75*] *80*)
where
P 𝓡⟦·⟧<*SRel*,*TRel*> *Q* ≡ (*P*, *Q*) ∈ *indRelST SRel TRel*

lemma (**in** *encoding*) *indRelST-symm*:
fixes *SRel* :: ('*procS* × '*procS*) *set*
and *TRel* :: ('*procT* × '*procT*) *set*
assumes *symmS*: *sym SRel*
and *symmT*: *sym TRel*
shows *sym* (*indRelST SRel TRel*)
⟨*proof*⟩

inductive-set (**in** *encoding*) *indRelSTEQ*
:: ('*procS* × '*procS*) *set* ⇒ ('*procT* × '*procT*) *set*
⇒ ((('*procS*, '*procT*) *Proc*) × (('*procS*, '*procT*) *Proc*)) *set*
for *SRel* :: ('*procS* × '*procS*) *set*
and *TRel* :: ('*procT* × '*procT*) *set*
where
encR: (*SourceTerm S*, *TargetTerm* (⟦*S*⟧)) ∈ *indRelSTEQ SRel TRel* |
encL: (*TargetTerm* (⟦*S*⟧), *SourceTerm S*) ∈ *indRelSTEQ SRel TRel* |
source: (*S1*, *S2*) ∈ *SRel* ⟹ (*SourceTerm S1*, *SourceTerm S2*) ∈ *indRelSTEQ SRel TRel* |
target: (*T1*, *T2*) ∈ *TRel* ⟹ (*TargetTerm T1*, *TargetTerm T2*) ∈ *indRelSTEQ SRel TRel* |
trans: ⟦(*P*, *Q*) ∈ *indRelSTEQ SRel TRel*; (*Q*, *R*) ∈ *indRelSTEQ SRel TRel*⟧
⟹ (*P*, *R*) ∈ *indRelSTEQ SRel TRel*

abbreviation (**in** *encoding*) *indRelSTEQinfix*
:: ('*procS*, '*procT*) *Proc* ⇒ ('*procS* × '*procS*) *set* ⇒ ('*procT* × '*procT*) *set*
⇒ ('*procS*, '*procT*) *Proc* ⇒ *bool* (- ∼⟦·⟧<-,-> - [*75*, *75*, *75*, *75*] *80*)
where
P ∼⟦·⟧<*SRel*,*TRel*> *Q* ≡ (*P*, *Q*) ∈ *indRelSTEQ SRel TRel*

lemma (**in** *encoding*) *indRelSTEQ-refl*:
fixes *SRel* :: ('*procS* × '*procS*) *set*
and *TRel* :: ('*procT* × '*procT*) *set*

assumes *reflT*: *refl TRel*
shows *refl* (*indRelSTEQ SRel TRel*)
⟨*proof*⟩

lemma (**in** *encoding*) *indRelSTEQ-symm*:
fixes *SRel* :: ('*procS* × '*procS*) *set*
and *TRel* :: ('*procT* × '*procT*) *set*
assumes *symmS*: *sym SRel*
and *symmT*: *sym TRel*
shows *sym* (*indRelSTEQ SRel TRel*)
⟨*proof*⟩

lemma (**in** *encoding*) *indRelSTEQ-trans*:
fixes *SRel* :: ('*procS* × '*procS*) *set*
and *TRel* :: ('*procT* × '*procT*) *set*
shows *trans* (*indRelSTEQ SRel TRel*)
⟨*proof*⟩

lemma (**in** *encoding*) *refl-trans-closure-of-indRelST*:
fixes *SRel* :: ('*procS* × '*procS*) *set*
and *TRel* :: ('*procT* × '*procT*) *set*
assumes *reflT*: *refl TRel*
shows *indRelSTEQ SRel TRel* = (*indRelST SRel TRel*)*
⟨*proof*⟩

lemma (**in** *encoding*) *refl-symm-trans-closure-of-indRelST*:
fixes *SRel* :: ('*procS* × '*procS*) *set*
and *TRel* :: ('*procT* × '*procT*) *set*
assumes *reflT*: *refl TRel*
and *symmS*: *sym SRel*
and *symmT*: *sym TRel*
shows *indRelSTEQ SRel TRel* = (*symcl* ((*indRelST SRel TRel*)$^=$))$^+$
⟨*proof*⟩

lemma (**in** *encoding*) *symm-closure-of-indRelRST*:
fixes *SRel* :: ('*procS* × '*procS*) *set*
and *TRel* :: ('*procT* × '*procT*) *set*
assumes *reflT*: *refl TRel*
and *symmS*: *sym SRel*
and *symmT*: *sym TRel*
shows *indRelST SRel TRel* = *symcl* (*indRelRST SRel TRel*)
and *indRelSTEQ SRel TRel* = (*symcl* ((*indRelRST SRel TRel*)$^=$))$^+$
⟨*proof*⟩

lemma (**in** *encoding*) *symm-closure-of-indRelLST*:
fixes *SRel* :: ('*procS* × '*procS*) *set*
and *TRel* :: ('*procT* × '*procT*) *set*
assumes *reflT*: *refl TRel*
and *symmS*: *sym SRel*
and *symmT*: *sym TRel*
shows *indRelST SRel TRel* = *symcl* (*indRelLST SRel TRel*)
and *indRelSTEQ SRel TRel* = (*symcl* ((*indRelLST SRel TRel*)$^=$))$^+$
⟨*proof*⟩

lemma (**in** *encoding*) *symm-trans-closure-of-indRelRSTPO*:
fixes *SRel* :: ('*procS* × '*procS*) *set*
and *TRel* :: ('*procT* × '*procT*) *set*
assumes *symmS*: *sym SRel*
and *symmT*: *sym TRel*
shows *indRelSTEQ SRel TRel* = (*symcl* (*indRelRSTPO SRel TRel*))$^+$
⟨*proof*⟩

lemma (**in** *encoding*) *symm-trans-closure-of-indRelLSTPO*:
fixes *SRel* :: (*'procS* × *'procS*) *set*
and *TRel* :: (*'procT* × *'procT*) *set*
assumes *symmS*: *sym SRel*
and *symmT*: *sym TRel*
shows *indRelSTEQ SRel TRel* = (*symcl* (*indRelLSTPO SRel TRel*))$^+$
⟨*proof*⟩

If the relations indRelRST, indRelLST, or indRelST contain a pair of target terms, then this pair is also related by the considered target term relation. Similarly a pair of source terms is related by the considered source term relation.

lemma (**in** *encoding*) *indRelRST-to-SRel*:
fixes *SRel* :: (*'procS* × *'procS*) *set*
and *TRel* :: (*'procT* × *'procT*) *set*
and *SP SQ* :: *'procS*
assumes *rel*: *SourceTerm SP* $\mathcal{R}[\![\cdot]\!]R$<*SRel*,*TRel*> *SourceTerm SQ*
shows (*SP*, *SQ*) ∈ *SRel*
⟨*proof*⟩

lemma (**in** *encoding*) *indRelRST-to-TRel*:
fixes *SRel* :: (*'procS* × *'procS*) *set*
and *TRel* :: (*'procT* × *'procT*) *set*
and *TP TQ* :: *'procT*
assumes *rel*: *TargetTerm TP* $\mathcal{R}[\![\cdot]\!]R$<*SRel*,*TRel*> *TargetTerm TQ*
shows (*TP*, *TQ*) ∈ *TRel*
⟨*proof*⟩

lemma (**in** *encoding*) *indRelLST-to-SRel*:
fixes *SRel* :: (*'procS* × *'procS*) *set*
and *TRel* :: (*'procT* × *'procT*) *set*
and *SP SQ* :: *'procS*
assumes *rel*: *SourceTerm SP* $\mathcal{R}[\![\cdot]\!]L$<*SRel*,*TRel*> *SourceTerm SQ*
shows (*SP*, *SQ*) ∈ *SRel*
⟨*proof*⟩

lemma (**in** *encoding*) *indRelLST-to-TRel*:
fixes *SRel* :: (*'procS* × *'procS*) *set*
and *TRel* :: (*'procT* × *'procT*) *set*
and *TP TQ* :: *'procT*
assumes *rel*: *TargetTerm TP* $\mathcal{R}[\![\cdot]\!]L$<*SRel*,*TRel*> *TargetTerm TQ*
shows (*TP*, *TQ*) ∈ *TRel*
⟨*proof*⟩

lemma (**in** *encoding*) *indRelST-to-SRel*:
fixes *SRel* :: (*'procS* × *'procS*) *set*
and *TRel* :: (*'procT* × *'procT*) *set*
and *SP SQ* :: *'procS*
assumes *rel*: *SourceTerm SP* $\mathcal{R}[\![\cdot]\!]$<*SRel*,*TRel*> *SourceTerm SQ*
shows (*SP*, *SQ*) ∈ *SRel*
⟨*proof*⟩

lemma (**in** *encoding*) *indRelST-to-TRel*:
fixes *SRel* :: (*'procS* × *'procS*) *set*
and *TRel* :: (*'procT* × *'procT*) *set*
and *TP TQ* :: *'procT*
assumes *rel*: *TargetTerm TP* $\mathcal{R}[\![\cdot]\!]$<*SRel*,*TRel*> *TargetTerm TQ*
shows (*TP*, *TQ*) ∈ *TRel*
⟨*proof*⟩

If the relations indRelRSTPO or indRelLSTPO contain a pair of target terms, then this pair is also related by the transitive closure of the considered target term relation. Similarly a pair of source terms is related by the transitive closure of the source term relation. A pair of a source and a target term results from the combination of pairs in the source relation, the target relation, and the encoding function. Note that, because of the symmetry, no similar condition holds for indRelSTEQ.

lemma (**in** *encoding*) *indRelRSTPO-to-SRel-and-TRel*:
  fixes *SRel* :: (*'procS* × *'procS*) *set*
    and *TRel* :: (*'procT* × *'procT*) *set*
    and *P Q* :: (*'procS*, *'procT*) *Proc*
  assumes $P \lesssim[\![\cdot]\!]R{<}SRel,TRel{>}\ Q$
  shows $\forall\, SP\ SQ.\ SP \in S\ P \wedge SQ \in S\ Q \longrightarrow (SP,\ SQ) \in SRel^{+}$
    and $\forall\, SP\ TQ.\ SP \in S\ P \wedge TQ \in T\ Q \longrightarrow (\exists\, S.\ (SP,\ S) \in SRel^{*} \wedge ([\![S]\!],\ TQ) \in TRel^{*})$
    and $\forall\, TP\ SQ.\ TP \in T\ P \wedge SQ \in S\ Q \longrightarrow False$
    and $\forall\, TP\ TQ.\ TP \in T\ P \wedge TQ \in T\ Q \longrightarrow (TP,\ TQ) \in TRel^{+}$
      ⟨*proof*⟩

lemma (**in** *encoding*) *indRelLSTPO-to-SRel-and-TRel*:
  fixes *SRel* :: (*'procS* × *'procS*) *set*
    and *TRel* :: (*'procT* × *'procT*) *set*
    and *P Q* :: (*'procS*, *'procT*) *Proc*
  assumes $P \lesssim[\![\cdot]\!]L{<}SRel,TRel{>}\ Q$
  shows $\forall\, SP\ SQ.\ SP \in S\ P \wedge SQ \in S\ Q \longrightarrow (SP,\ SQ) \in SRel^{+}$
    and $\forall\, SP\ TQ.\ SP \in S\ P \wedge TQ \in T\ Q \longrightarrow False$
    and $\forall\, TP\ SQ.\ TP \in T\ P \wedge SQ \in S\ Q \longrightarrow (\exists\, S.\ (TP,\ [\![S]\!]) \in TRel^{*} \wedge (S,\ SQ) \in SRel^{*})$
    and $\forall\, TP\ TQ.\ TP \in T\ P \wedge TQ \in T\ Q \longrightarrow (TP,\ TQ) \in TRel^{+}$
      ⟨*proof*⟩

If indRelRSTPO, indRelLSTPO, or indRelSTPO preserves barbs then so do the corresponding source term and target term relations.

lemma (**in** *encoding-wrt-barbs*) *rel-with-source-impl-SRel-preserves-barbs*:
  fixes *SRel* :: (*'procS* × *'procS*) *set*
    and *Rel* :: ((*'procS*, *'procT*) *Proc* × (*'procS*, *'procT*) *Proc*) *set*
  assumes *preservation*: *rel-preserves-barbs Rel* (*STCalWB SWB TWB*)
      and *sourceInRel*: $\forall\, S1\ S2.\ (S1,\ S2) \in SRel \longrightarrow (SourceTerm\ S1,\ SourceTerm\ S2) \in Rel$
  shows *rel-preserves-barbs SRel SWB*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelRSTPO-impl-SRel-and-TRel-preserve-barbs*:
  fixes *SRel* :: (*'procS* × *'procS*) *set*
    and *TRel* :: (*'procT* × *'procT*) *set*
  assumes *preservation*: *rel-preserves-barbs* (*indRelRSTPO SRel TRel*) (*STCalWB SWB TWB*)
  shows *rel-preserves-barbs SRel SWB*
    and *rel-preserves-barbs TRel TWB*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelLSTPO-impl-SRel-and-TRel-preserve-barbs*:
  fixes *SRel* :: (*'procS* × *'procS*) *set*
    and *TRel* :: (*'procT* × *'procT*) *set*
  assumes *preservation*: *rel-preserves-barbs* (*indRelLSTPO SRel TRel*) (*STCalWB SWB TWB*)
  shows *rel-preserves-barbs SRel SWB*
    and *rel-preserves-barbs TRel TWB*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelSTEQ-impl-SRel-and-TRel-preserve-barbs*:
  fixes *SRel* :: (*'procS* × *'procS*) *set*
    and *TRel* :: (*'procT* × *'procT*) *set*
  assumes *preservation*: *rel-preserves-barbs* (*indRelSTEQ SRel TRel*) (*STCalWB SWB TWB*)
  shows *rel-preserves-barbs SRel SWB*
    and *rel-preserves-barbs TRel TWB*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *rel-with-source-impl-SRel-weakly-preserves-barbs*:
fixes *SRel* :: (*'procS* × *'procS*) *set*
and *Rel* :: ((*'procS*, *'procT*) *Proc* × (*'procS*, *'procT*) *Proc*) *set*
assumes *preservation*: *rel-weakly-preserves-barbs Rel* (*STCalWB SWB TWB*)
and *sourceInRel*: ∀ *S1 S2*. (*S1*, *S2*) ∈ *SRel* ⟶ (*SourceTerm S1*, *SourceTerm S2*) ∈ *Rel*
shows *rel-weakly-preserves-barbs SRel SWB*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelRSTPO-impl-SRel-and-TRel-weakly-preserve-barbs*:
fixes *SRel* :: (*'procS* × *'procS*) *set*
and *TRel* :: (*'procT* × *'procT*) *set*
assumes *preservation*: *rel-weakly-preserves-barbs* (*indRelRSTPO SRel TRel*) (*STCalWB SWB TWB*)
shows *rel-weakly-preserves-barbs SRel SWB*
and *rel-weakly-preserves-barbs TRel TWB*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelLSTPO-impl-SRel-and-TRel-weakly-preserve-barbs*:
fixes *SRel* :: (*'procS* × *'procS*) *set*
and *TRel* :: (*'procT* × *'procT*) *set*
assumes *preservation*: *rel-weakly-preserves-barbs* (*indRelLSTPO SRel TRel*) (*STCalWB SWB TWB*)
shows *rel-weakly-preserves-barbs SRel SWB*
and *rel-weakly-preserves-barbs TRel TWB*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelSTEQ-impl-SRel-and-TRel-weakly-preserve-barbs*:
fixes *SRel* :: (*'procS* × *'procS*) *set*
and *TRel* :: (*'procT* × *'procT*) *set*
assumes *preservation*: *rel-weakly-preserves-barbs* (*indRelSTEQ SRel TRel*) (*STCalWB SWB TWB*)
shows *rel-weakly-preserves-barbs SRel SWB*
and *rel-weakly-preserves-barbs TRel TWB*
⟨*proof*⟩

If indRelRSTPO, indRelLSTPO, or indRelSTPO reflects barbs then so do the corresponding source term and target term relations.

lemma (**in** *encoding-wrt-barbs*) *rel-with-source-impl-SRel-reflects-barbs*:
fixes *SRel* :: (*'procS* × *'procS*) *set*
and *Rel* :: ((*'procS*, *'procT*) *Proc* × (*'procS*, *'procT*) *Proc*) *set*
assumes *reflection*: *rel-reflects-barbs Rel* (*STCalWB SWB TWB*)
and *sourceInRel*: ∀ *S1 S2*. (*S1*, *S2*) ∈ *SRel* ⟶ (*SourceTerm S1*, *SourceTerm S2*) ∈ *Rel*
shows *rel-reflects-barbs SRel SWB*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelRSTPO-impl-SRel-and-TRel-reflect-barbs*:
fixes *SRel* :: (*'procS* × *'procS*) *set*
and *TRel* :: (*'procT* × *'procT*) *set*
assumes *reflection*: *rel-reflects-barbs* (*indRelRSTPO SRel TRel*) (*STCalWB SWB TWB*)
shows *rel-reflects-barbs SRel SWB*
and *rel-reflects-barbs TRel TWB*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelLSTPO-impl-SRel-and-TRel-reflect-barbs*:
fixes *SRel* :: (*'procS* × *'procS*) *set*
and *TRel* :: (*'procT* × *'procT*) *set*
assumes *reflection*: *rel-reflects-barbs* (*indRelLSTPO SRel TRel*) (*STCalWB SWB TWB*)
shows *rel-reflects-barbs SRel SWB*
and *rel-reflects-barbs TRel TWB*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelSTEQ-impl-SRel-and-TRel-reflect-barbs*:
fixes *SRel* :: (*'procS* × *'procS*) *set*

and *TRel* :: ('*procT* × '*procT*) *set*
assumes *reflection*: *rel-reflects-barbs* (*indRelSTEQ SRel TRel*) (*STCalWB SWB TWB*)
shows *rel-reflects-barbs SRel SWB*
and *rel-reflects-barbs TRel TWB*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *rel-with-source-impl-SRel-weakly-reflects-barbs*:
fixes *SRel* :: ('*procS* × '*procS*) *set*
and *Rel* :: (('*procS*, '*procT*) *Proc* × ('*procS*, '*procT*) *Proc*) *set*
assumes *reflection*: *rel-weakly-reflects-barbs Rel* (*STCalWB SWB TWB*)
and *sourceInRel*: $\forall$ *S1 S2*. (*S1*, *S2*) $\in$ *SRel* $\longrightarrow$ (*SourceTerm S1*, *SourceTerm S2*) $\in$ *Rel*
shows *rel-weakly-reflects-barbs SRel SWB*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelRSTPO-impl-SRel-and-TRel-weakly-reflect-barbs*:
fixes *SRel* :: ('*procS* × '*procS*) *set*
and *TRel* :: ('*procT* × '*procT*) *set*
assumes *reflection*: *rel-weakly-reflects-barbs* (*indRelRSTPO SRel TRel*) (*STCalWB SWB TWB*)
shows *rel-weakly-reflects-barbs SRel SWB*
and *rel-weakly-reflects-barbs TRel TWB*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelLSTPO-impl-SRel-and-TRel-weakly-reflect-barbs*:
fixes *SRel* :: ('*procS* × '*procS*) *set*
and *TRel* :: ('*procT* × '*procT*) *set*
assumes *reflection*: *rel-weakly-reflects-barbs* (*indRelLSTPO SRel TRel*) (*STCalWB SWB TWB*)
shows *rel-weakly-reflects-barbs SRel SWB*
and *rel-weakly-reflects-barbs TRel TWB*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelSTEQ-impl-SRel-and-TRel-weakly-reflect-barbs*:
fixes *SRel* :: ('*procS* × '*procS*) *set*
and *TRel* :: ('*procT* × '*procT*) *set*
assumes *reflection*: *rel-weakly-reflects-barbs* (*indRelSTEQ SRel TRel*) (*STCalWB SWB TWB*)
shows *rel-weakly-reflects-barbs SRel SWB*
and *rel-weakly-reflects-barbs TRel TWB*
⟨*proof*⟩

If indRelRSTPO, indRelLSTPO, or indRelSTPO respects barbs then so do the corresponding source term and target term relations.

lemma (**in** *encoding-wrt-barbs*) *indRelRSTPO-impl-SRel-and-TRel-respect-barbs*:
fixes *SRel* :: ('*procS* × '*procS*) *set*
and *TRel* :: ('*procT* × '*procT*) *set*
assumes *respection*: *rel-respects-barbs* (*indRelRSTPO SRel TRel*) (*STCalWB SWB TWB*)
shows *rel-respects-barbs SRel SWB*
and *rel-respects-barbs TRel TWB*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelLSTPO-impl-SRel-and-TRel-respect-barbs*:
fixes *SRel* :: ('*procS* × '*procS*) *set*
and *TRel* :: ('*procT* × '*procT*) *set*
assumes *respection*: *rel-respects-barbs* (*indRelLSTPO SRel TRel*) (*STCalWB SWB TWB*)
shows *rel-respects-barbs SRel SWB*
and *rel-respects-barbs TRel TWB*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelSTEQ-impl-SRel-and-TRel-respect-barbs*:
fixes *SRel* :: ('*procS* × '*procS*) *set*
and *TRel* :: ('*procT* × '*procT*) *set*
assumes *respection*: *rel-respects-barbs* (*indRelSTEQ SRel TRel*) (*STCalWB SWB TWB*)

shows *rel-respects-barbs SRel SWB*
and *rel-respects-barbs TRel TWB*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelRSTPO-impl-SRel-and-TRel-weakly-respect-barbs*:
fixes *SRel* :: (*'procS* × *'procS*) *set*
and *TRel* :: (*'procT* × *'procT*) *set*
assumes *respection*: *rel-weakly-respects-barbs* (*indRelRSTPO SRel TRel*) (*STCalWB SWB TWB*)
shows *rel-weakly-respects-barbs SRel SWB*
and *rel-weakly-respects-barbs TRel TWB*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelLSTPO-impl-SRel-and-TRel-weakly-respect-barbs*:
fixes *SRel* :: (*'procS* × *'procS*) *set*
and *TRel* :: (*'procT* × *'procT*) *set*
assumes *respection*: *rel-weakly-respects-barbs* (*indRelLSTPO SRel TRel*) (*STCalWB SWB TWB*)
shows *rel-weakly-respects-barbs SRel SWB*
and *rel-weakly-respects-barbs TRel TWB*
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *indRelSTEQ-impl-SRel-and-TRel-weakly-respect-barbs*:
fixes *SRel* :: (*'procS* × *'procS*) *set*
and *TRel* :: (*'procT* × *'procT*) *set*
assumes *respection*: *rel-weakly-respects-barbs* (*indRelSTEQ SRel TRel*) (*STCalWB SWB TWB*)
shows *rel-weakly-respects-barbs SRel SWB*
and *rel-weakly-respects-barbs TRel TWB*
⟨*proof*⟩

If TRel is reflexive then ind relRTPO is a subrelation of indRelTEQ. If SRel is reflexive then indRelRTPO is a subrelation of indRelRTPO. Moreover, indRelRSTPO is a subrelation of indRelSTEQ.

lemma (**in** *encoding*) *indRelRTPO-to-indRelTEQ*:
fixes *TRel* :: (*'procT* × *'procT*) *set*
and *P Q* :: (*'procS*, *'procT*) *Proc*
assumes *rel*: $P \lesssim[\![\cdot]\!]RT<TRel>\ Q$
and *reflT*: *refl TRel*
shows $P \sim[\![\cdot]\!]T<TRel>\ Q$
⟨*proof*⟩

lemma (**in** *encoding*) *indRelRTPO-to-indRelRSTPO*:
fixes *SRel* :: (*'procS* × *'procS*) *set*
and *TRel* :: (*'procT* × *'procT*) *set*
and *P Q* :: (*'procS*, *'procT*) *Proc*
assumes *rel*: $P \lesssim[\![\cdot]\!]RT<TRel>\ Q$
and *reflS*: *refl SRel*
shows $P \lesssim[\![\cdot]\!]R<SRel,TRel>\ Q$
⟨*proof*⟩

lemma (**in** *encoding*) *indRelRSTPO-to-indRelSTEQ*:
fixes *SRel* :: (*'procS* × *'procS*) *set*
and *TRel* :: (*'procT* × *'procT*) *set*
and *P Q* :: (*'procS*, *'procT*) *Proc*
assumes *rel*: $P \lesssim[\![\cdot]\!]R<SRel,TRel>\ Q$
shows $P \sim[\![\cdot]\!]<SRel,TRel>\ Q$
⟨*proof*⟩

If indRelRTPO is a bisimulation and SRel is a reflexive bisimulation then also indRelRSTPO is a bisimulation.

lemma (**in** *encoding*) *indRelRTPO-weak-reduction-bisimulation-impl-indRelRSTPO-bisimulation*:
fixes *SRel* :: (*'procS* × *'procS*) *set*
and *TRel* :: (*'procT* × *'procT*) *set*

assumes *bisimT*: *weak-reduction-bisimulation* (*indRelRTPO TRel*) (*STCal Source Target*)
and *bisimS*: *weak-reduction-bisimulation SRel Source*
and *reflS*: *refl SRel*
shows *weak-reduction-bisimulation* (*indRelRSTPO SRel TRel*) (*STCal Source Target*)
⟨*proof*⟩

end
theory *SuccessSensitiveness*
imports *SourceTargetRelation*
begin

6 Success Sensitiveness and Barbs

To compare the abstract behavior of two terms, often some notion of success or successful termination is used. Daniele Gorla assumes a constant process (similar to the empty process) that represents successful termination in order to compare the behavior of source terms with their literal translations. Then an encoding is success sensitive if, for all source terms S, S reaches success iff the translation of S reaches success. Successful termination can be considered as some special kind of barb. Accordingly we generalize successful termination to the respection of an arbitrary subset of barbs. An encoding respects a set of barbs if, for every source term S and all considered barbs a, S reaches a iff the translation of S reaches a.

abbreviation (**in** *encoding-wrt-barbs*) *enc-weakly-preserves-barb-set* :: *'barbs set* ⇒ *bool* **where**
enc-weakly-preserves-barb-set Barbs ≡ *enc-preserves-binary-pred* ($\lambda P\ a.\ a \in Barbs \wedge P\Downarrow.a$)

abbreviation (**in** *encoding-wrt-barbs*) *enc-weakly-preserves-barbs* :: *bool* **where**
enc-weakly-preserves-barbs ≡ *enc-preserves-binary-pred* ($\lambda P\ a.\ P\Downarrow.a$)

lemma (**in** *encoding-wrt-barbs*) *enc-weakly-preserves-barbs-and-barb-set*:
shows *enc-weakly-preserves-barbs* = (∀ *Barbs*. *enc-weakly-preserves-barb-set Barbs*)
⟨*proof*⟩

abbreviation (**in** *encoding-wrt-barbs*) *enc-weakly-reflects-barb-set* :: *'barbs set* ⇒ *bool* **where**
enc-weakly-reflects-barb-set Barbs ≡ *enc-reflects-binary-pred* ($\lambda P\ a.\ a \in Barbs \wedge P\Downarrow.a$)

abbreviation (**in** *encoding-wrt-barbs*) *enc-weakly-reflects-barbs* :: *bool* **where**
enc-weakly-reflects-barbs ≡ *enc-reflects-binary-pred* ($\lambda P\ a.\ P\Downarrow.a$)

lemma (**in** *encoding-wrt-barbs*) *enc-weakly-reflects-barbs-and-barb-set*:
shows *enc-weakly-reflects-barbs* = (∀ *Barbs*. *enc-weakly-reflects-barb-set Barbs*)
⟨*proof*⟩

abbreviation (**in** *encoding-wrt-barbs*) *enc-weakly-respects-barb-set* :: *'barbs set* ⇒ *bool* **where**
enc-weakly-respects-barb-set Barbs ≡
enc-weakly-preserves-barb-set Barbs ∧ *enc-weakly-reflects-barb-set Barbs*

abbreviation (**in** *encoding-wrt-barbs*) *enc-weakly-respects-barbs* :: *bool* **where**
enc-weakly-respects-barbs ≡ *enc-weakly-preserves-barbs* ∧ *enc-weakly-reflects-barbs*

lemma (**in** *encoding-wrt-barbs*) *enc-weakly-respects-barbs-and-barb-set*:
shows *enc-weakly-respects-barbs* = (∀ *Barbs*. *enc-weakly-respects-barb-set Barbs*)
⟨*proof*⟩

An encoding strongly respects some set of barbs if, for every source term S and all considered barbs a, S has a iff the translation of S has a.

abbreviation (**in** *encoding-wrt-barbs*) *enc-preserves-barb-set* :: *'barbs set* ⇒ *bool* **where**
enc-preserves-barb-set Barbs ≡ *enc-preserves-binary-pred* ($\lambda P\ a.\ a \in Barbs \wedge P\downarrow.a$)

abbreviation (**in** *encoding-wrt-barbs*) *enc-preserves-barbs* :: *bool* **where**

enc-preserves-barbs $\equiv$ *enc-preserves-binary-pred* $(\lambda P\ a.\ P{\downarrow}.a)$

lemma (**in** *encoding-wrt-barbs*) *enc-preserves-barbs-and-barb-set*:
shows *enc-preserves-barbs* $=$ $(\forall\, Barbs.\ $*enc-preserves-barb-set Barbs*$)$
⟨*proof*⟩

abbreviation (**in** *encoding-wrt-barbs*) *enc-reflects-barb-set* :: *'barbs set* $\Rightarrow$ *bool* **where**
enc-reflects-barb-set Barbs $\equiv$ *enc-reflects-binary-pred* $(\lambda P\ a.\ a \in Barbs \wedge P{\downarrow}.a)$

abbreviation (**in** *encoding-wrt-barbs*) *enc-reflects-barbs* :: *bool* **where**
enc-reflects-barbs $\equiv$ *enc-reflects-binary-pred* $(\lambda P\ a.\ P{\downarrow}.a)$

lemma (**in** *encoding-wrt-barbs*) *enc-reflects-barbs-and-barb-set*:
shows *enc-reflects-barbs* $=$ $(\forall\, Barbs.\ $*enc-reflects-barb-set Barbs*$)$
⟨*proof*⟩

abbreviation (**in** *encoding-wrt-barbs*) *enc-respects-barb-set* :: *'barbs set* $\Rightarrow$ *bool* **where**
enc-respects-barb-set Barbs $\equiv$ *enc-preserves-barb-set Barbs* $\wedge$ *enc-reflects-barb-set Barbs*

abbreviation (**in** *encoding-wrt-barbs*) *enc-respects-barbs* :: *bool* **where**
enc-respects-barbs $\equiv$ *enc-preserves-barbs* $\wedge$ *enc-reflects-barbs*

lemma (**in** *encoding-wrt-barbs*) *enc-respects-barbs-and-barb-set*:
shows *enc-respects-barbs* $=$ $(\forall\, Barbs.\ $*enc-respects-barb-set Barbs*$)$
⟨*proof*⟩

An encoding (weakly) preserves barbs iff (1) there exists a relation, like indRelR, that relates source terms and their literal translations and preserves (reachability/)existence of barbs, or (2) there exists a relation, like indRelL, that relates literal translations and their source terms and reflects (reachability/)existence of barbs.

lemma (**in** *encoding-wrt-barbs*) *enc-weakly-preserves-barb-set-iff-source-target-rel*:
fixes *Barbs* :: *'barbs set*
and *TRel* :: (*'procT* $\times$ *'procT*) *set*
shows *enc-weakly-preserves-barb-set Barbs*
$= (\exists\, Rel.\ (\forall\, S.\ (SourceTerm\ S,\ TargetTerm\ ([\![S]\!])) \in Rel)$
$\wedge$ *rel-weakly-preserves-barb-set Rel* (*STCalWB SWB TWB*) *Barbs*)
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *enc-weakly-preserves-barbs-iff-source-target-rel*:
shows *enc-weakly-preserves-barbs*
$= (\exists\, Rel.\ (\forall\, S.\ (SourceTerm\ S,\ TargetTerm\ ([\![S]\!])) \in Rel)$
$\wedge$ *rel-weakly-preserves-barbs Rel* (*STCalWB SWB TWB*))
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *enc-preserves-barb-set-iff-source-target-rel*:
fixes *Barbs* :: *'barbs set*
shows *enc-preserves-barb-set Barbs*
$= (\exists\, Rel.\ (\forall\, S.\ (SourceTerm\ S,\ TargetTerm\ ([\![S]\!])) \in Rel)$
$\wedge$ *rel-preserves-barb-set Rel* (*STCalWB SWB TWB*) *Barbs*)
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *enc-preserves-barbs-iff-source-target-rel*:
shows *enc-preserves-barbs*
$= (\exists\, Rel.\ (\forall\, S.\ (SourceTerm\ S,\ TargetTerm\ ([\![S]\!])) \in Rel)$
$\wedge$ *rel-preserves-barbs Rel* (*STCalWB SWB TWB*))
⟨*proof*⟩

An encoding (weakly) reflects barbs iff (1) there exists a relation, like indRelR, that relates source terms and their literal translations and reflects (reachability/)existence of barbs, or (2) there exists a relation, like indRelL, that relates literal translations and their source terms and preserves (reachability/)existence of barbs.

lemma (**in** *encoding-wrt-barbs*) *enc-weakly-reflects-barb-set-iff-source-target-rel*:
fixes *Barbs* :: *'barbs set*
shows *enc-weakly-reflects-barb-set Barbs*
$= (\exists\, Rel.\ (\forall\, S.\ (SourceTerm\ S,\ TargetTerm\ (\llbracket S \rrbracket)) \in Rel)$
$\wedge$ *rel-weakly-reflects-barb-set Rel* (*STCalWB SWB TWB*) *Barbs*)
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *enc-weakly-reflects-barbs-iff-source-target-rel*:
shows *enc-weakly-reflects-barbs*
$= (\exists\, Rel.\ (\forall\, S.\ (SourceTerm\ S,\ TargetTerm\ (\llbracket S \rrbracket)) \in Rel)$
$\wedge$ *rel-weakly-reflects-barbs Rel* (*STCalWB SWB TWB*))
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *enc-reflects-barb-set-iff-source-target-rel*:
fixes *Barbs* :: *'barbs set*
shows *enc-reflects-barb-set Barbs*
$= (\exists\, Rel.\ (\forall\, S.\ (SourceTerm\ S,\ TargetTerm\ (\llbracket S \rrbracket)) \in Rel)$
$\wedge$ *rel-reflects-barb-set Rel* (*STCalWB SWB TWB*) *Barbs*)
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *enc-reflects-barbs-iff-source-target-rel*:
shows *enc-reflects-barbs*
$= (\exists\, Rel.\ (\forall\, S.\ (SourceTerm\ S,\ TargetTerm\ (\llbracket S \rrbracket)) \in Rel)$
$\wedge$ *rel-reflects-barbs Rel* (*STCalWB SWB TWB*))
⟨*proof*⟩

An encoding (weakly) respects barbs iff (1) there exists a relation, like indRelR, that relates source terms and their literal translations and respects (reachability/)existence of barbs, or (2) there exists a relation, like indRelL, that relates literal translations and their source terms and respects (reachability/)existence of barbs, or (3) there exists a relation, like indRel, that relates source terms and their literal translations in both directions and respects (reachability/)existence of barbs.

lemma (**in** *encoding-wrt-barbs*) *enc-weakly-respects-barb-set-iff-source-target-rel*:
fixes *Barbs* :: *'barbs set*
shows *enc-weakly-respects-barb-set Barbs*
$= (\exists\, Rel.\ (\forall\, S.\ (SourceTerm\ S,\ TargetTerm\ (\llbracket S \rrbracket)) \in Rel)$
$\wedge$ *rel-weakly-respects-barb-set Rel* (*STCalWB SWB TWB*) *Barbs*)
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *enc-weakly-respects-barbs-iff-source-target-rel*:
shows *enc-weakly-respects-barbs*
$= (\exists\, Rel.\ (\forall\, S.\ (SourceTerm\ S,\ TargetTerm\ (\llbracket S \rrbracket)) \in Rel)$
$\wedge$ *rel-weakly-respects-barbs Rel* (*STCalWB SWB TWB*))
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *enc-respects-barb-set-iff-source-target-rel*:
fixes *Barbs* :: *'barbs set*
shows *enc-respects-barb-set Barbs*
$= (\exists\, Rel.\ (\forall\, S.\ (SourceTerm\ S,\ TargetTerm\ (\llbracket S \rrbracket)) \in Rel)$
$\wedge$ *rel-respects-barb-set Rel* (*STCalWB SWB TWB*) *Barbs*)
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *enc-respects-barbs-iff-source-target-rel*:
shows *enc-respects-barbs*
$= (\exists\, Rel.\ (\forall\, S.\ (SourceTerm\ S,\ TargetTerm\ (\llbracket S \rrbracket)) \in Rel)$
$\wedge$ *rel-respects-barbs Rel* (*STCalWB SWB TWB*))
⟨*proof*⟩

Accordingly an encoding is success sensitive iff there exists such a relation between source and target terms that weakly respects the barb success.

lemma (**in** *encoding-wrt-barbs*) *success-sensitive-cond*:
fixes *success* :: *'barbs*
shows *enc-weakly-respects-barb-set* $\{success\} = (\forall S.\ S{\Downarrow}{<}SWB{>}success \longleftrightarrow [\![S]\!]{\Downarrow}{<}TWB{>}success)$
$\langle proof \rangle$

lemma (**in** *encoding-wrt-barbs*) *success-sensitive-iff-source-target-rel-weakly-respects-success*:
fixes *success* :: *'barbs*
shows *enc-weakly-respects-barb-set* $\{success\}$
$= (\exists Rel.\ (\forall S.\ (SourceTerm\ S,\ TargetTerm\ ([\![S]\!])) \in Rel)$
$\wedge$ *rel-weakly-respects-barb-set Rel* $(STCalWB\ SWB\ TWB)\ \{success\})$
$\langle proof \rangle$

lemma (**in** *encoding-wrt-barbs*) *success-sensitive-iff-source-target-rel-respects-success*:
fixes *success* :: *'barbs*
shows *enc-respects-barb-set* $\{success\}$
$= (\exists Rel.\ (\forall S.\ (SourceTerm\ S,\ TargetTerm\ ([\![S]\!])) \in Rel)$
$\wedge$ *rel-respects-barb-set Rel* $(STCalWB\ SWB\ TWB)\ \{success\})$
$\langle proof \rangle$

end
theory *DivergenceReflection*
imports *SourceTargetRelation*
begin

7 Divergence Reflection

Divergence reflection forbids for encodings that introduce loops of internal actions. Thus they determine the practicability of encodings in particular with respect to implementations. An encoding reflects divergence if each loop in a target term result from the translation of a divergent source term.

abbreviation (**in** *encoding*) *enc-preserves-divergence* :: *bool* **where**
enc-preserves-divergence $\equiv$ *enc-preserves-pred* $(\lambda P.\ P \longmapsto ST\omega)$

lemma (**in** *encoding*) *divergence-preservation-cond*:
shows *enc-preserves-divergence* $= (\forall S.\ S \longmapsto (Source)\omega \longrightarrow [\![S]\!] \longmapsto (Target)\omega)$
$\langle proof \rangle$

abbreviation (**in** *encoding*) *enc-reflects-divergence* :: *bool* **where**
enc-reflects-divergence $\equiv$ *enc-reflects-pred* $(\lambda P.\ P \longmapsto ST\omega)$

lemma (**in** *encoding*) *divergence-reflection-cond*:
shows *enc-reflects-divergence* $= (\forall S.\ [\![S]\!] \longmapsto (Target)\omega \longrightarrow S \longmapsto (Source)\omega)$
$\langle proof \rangle$

abbreviation *rel-preserves-divergence*
:: *('proc × 'proc) set ⇒ 'proc processCalculus ⇒ bool*
where
rel-preserves-divergence Rel Cal $\equiv$ *rel-preserves-pred Rel* $(\lambda P.\ P \longmapsto (Cal)\omega)$

abbreviation *rel-reflects-divergence*
:: *('proc × 'proc) set ⇒ 'proc processCalculus ⇒ bool*
where
rel-reflects-divergence Rel Cal $\equiv$ *rel-reflects-pred Rel* $(\lambda P.\ P \longmapsto (Cal)\omega)$

Apart from divergence reflection we consider divergence respection. An encoding respects divergence if each divergent source term is translated into a divergent target term and each divergent target term result from the translation of a divergent source term.

abbreviation (**in** *encoding*) *enc-respects-divergence* :: *bool* **where**
enc-respects-divergence $\equiv$ *enc-respects-pred* $(\lambda P.\ P \longmapsto ST\omega)$

lemma (**in** *encoding*) *divergence-respection-cond*:
shows $enc\text{-}respects\text{-}divergence = (\forall S.\ [\![S]\!] \longmapsto(Target)\omega \longleftrightarrow S \longmapsto(Source)\omega)$
⟨*proof*⟩

abbreviation *rel-respects-divergence*
$:: ('proc \times 'proc)\ set \Rightarrow 'proc\ processCalculus \Rightarrow bool$
where
$rel\text{-}respects\text{-}divergence\ Rel\ Cal \equiv rel\text{-}respects\text{-}pred\ Rel\ (\lambda P.\ P \longmapsto(Cal)\omega)$

An encoding preserves divergence iff (1) there exists a relation that relates source terms and their literal translations and preserves divergence, or (2) there exists a relation that relates literal translations and their source terms and reflects divergence.

lemma (**in** *encoding*) *divergence-preservation-iff-source-target-rel-preserves-divergence*:
shows *enc-preserves-divergence*
$= (\exists Rel.\ (\forall S.\ (SourceTerm\ S,\ TargetTerm\ ([\![S]\!])) \in Rel)$
$\wedge\ rel\text{-}preserves\text{-}divergence\ Rel\ (STCal\ Source\ Target))$
⟨*proof*⟩

lemma (**in** *encoding*) *divergence-preservation-iff-source-target-rel-reflects-divergence*:
shows *enc-preserves-divergence*
$= (\exists Rel.\ (\forall S.\ (TargetTerm\ ([\![S]\!]),\ SourceTerm\ S) \in Rel)$
$\wedge\ rel\text{-}reflects\text{-}divergence\ Rel\ (STCal\ Source\ Target))$
⟨*proof*⟩

An encoding reflects divergence iff (1) there exists a relation that relates source terms and their literal translations and reflects divergence, or (2) there exists a relation that relates literal translations and their source terms and preserves divergence.

lemma (**in** *encoding*) *divergence-reflection-iff-source-target-rel-reflects-divergence*:
shows *enc-reflects-divergence*
$= (\exists Rel.\ (\forall S.\ (SourceTerm\ S,\ TargetTerm\ ([\![S]\!])) \in Rel)$
$\wedge\ rel\text{-}reflects\text{-}divergence\ Rel\ (STCal\ Source\ Target))$
⟨*proof*⟩

lemma (**in** *encoding*) *divergence-reflection-iff-source-target-rel-preserves-divergence*:
shows *enc-reflects-divergence*
$= (\exists Rel.\ (\forall S.\ (TargetTerm\ ([\![S]\!]),\ SourceTerm\ S) \in Rel)$
$\wedge\ rel\text{-}preserves\text{-}divergence\ Rel\ (STCal\ Source\ Target))$
⟨*proof*⟩

An encoding respects divergence iff there exists a relation that relates source terms and their literal translations in both directions and respects divergence.

lemma (**in** *encoding*) *divergence-respection-iff-source-target-rel-respects-divergence*:
shows $enc\text{-}respects\text{-}divergence = (\exists Rel.\ (\forall S.\ (SourceTerm\ S,\ TargetTerm\ ([\![S]\!])) \in Rel)$
$\wedge\ rel\text{-}respects\text{-}divergence\ Rel\ (STCal\ Source\ Target))$
and $enc\text{-}respects\text{-}divergence = (\exists Rel.$
$(\forall S.\ (SourceTerm\ S,\ TargetTerm\ ([\![S]\!])) \in Rel \wedge (TargetTerm\ ([\![S]\!]),\ SourceTerm\ S) \in Rel)$
$\wedge\ rel\text{-}respects\text{-}divergence\ Rel\ (STCal\ Source\ Target))$
⟨*proof*⟩

end
theory *OperationalCorrespondence*
imports *SourceTargetRelation*
begin

8 Operational Correspondence

We consider different variants of operational correspondence. This criterion consists of a completeness and a soundness condition and is often defined with respect to a relation TRel on target terms.

Operational completeness modulo TRel ensures that an encoding preserves source term behaviour modulo TRel by requiring that each sequence of source term steps can be mimicked by its translation such that the respective derivatives are related by TRel.

abbreviation (**in** *encoding*) *operational-complete* :: ($'procT \times 'procT$) *set* $\Rightarrow$ *bool* **where**
operational-complete TRel $\equiv$
$\forall S\ S'.\ S \longmapsto Source{*}\ S' \longrightarrow (\exists\, T.\ [\![S]\!] \longmapsto Target{*}\ T \wedge ([\![S']\!],\ T) \in TRel)$

We call an encoding strongly operational complete modulo TRel if each source term step has to be mimicked by single target term step of its translation.

abbreviation (**in** *encoding*) *strongly-operational-complete* :: ($'procT \times 'procT$) *set* $\Rightarrow$ *bool* **where**
strongly-operational-complete TRel $\equiv$
$\forall S\ S'.\ S \longmapsto Source\ S' \longrightarrow (\exists\, T.\ [\![S]\!] \longmapsto Target\ T \wedge ([\![S']\!],\ T) \in TRel)$

Operational soundness ensures that the encoding does not introduce new behaviour. An encoding is weakly operational sound modulo TRel if each sequence of target term steps is part of the translation of a sequence of source term steps such that the derivatives are related by TRel. It allows for intermediate states on the translation of source term step that are not the result of translating a source term.

abbreviation (**in** *encoding*) *weakly-operational-sound* :: ($'procT \times 'procT$) *set* $\Rightarrow$ *bool* **where**
weakly-operational-sound TRel $\equiv$
$\forall S\ T.\ [\![S]\!] \longmapsto Target{*}\ T \longrightarrow (\exists\, S'\ T'.\ S \longmapsto Source{*}\ S' \wedge T \longmapsto Target{*}\ T' \wedge ([\![S']\!],\ T') \in TRel)$

And encoding is operational sound modulo TRel if each sequence of target term steps is the translation of a sequence of source term steps such that the derivatives are related by TRel. This criterion does not allow for intermediate states, i.e., does not allow to a reach target term from an encoded source term that is not related by TRel to the translation of a source term.

abbreviation (**in** *encoding*) *operational-sound* :: ($'procT \times 'procT$) *set* $\Rightarrow$ *bool* **where**
operational-sound TRel $\equiv \forall S\ T.\ [\![S]\!] \longmapsto Target{*}\ T \longrightarrow (\exists\, S'.\ S \longmapsto Source{*}\ S' \wedge ([\![S']\!],\ T) \in TRel)$

Strong operational soundness modulo TRel is a stricter variant of operational soundness, where a single target term step has to be mapped on a single source term step.

abbreviation (**in** *encoding*) *strongly-operational-sound* :: ($'procT \times 'procT$) *set* $\Rightarrow$ *bool* **where**
strongly-operational-sound TRel $\equiv$
$\forall S\ T.\ [\![S]\!] \longmapsto Target\ T \longrightarrow (\exists\, S'.\ S \longmapsto Source\ S' \wedge ([\![S']\!],\ T) \in TRel)$

An encoding is weakly operational corresponding modulo TRel if it is operational complete and weakly operational sound modulo TRel.

abbreviation (**in** *encoding*) *weakly-operational-corresponding*
:: ($'procT \times 'procT$) *set* $\Rightarrow$ *bool*
where
weakly-operational-corresponding TRel $\equiv$
operational-complete TRel $\wedge$ *weakly-operational-sound TRel*

Operational correspondence modulo is the combination of operational completeness and operational soundness modulo TRel.

abbreviation (**in** *encoding*) *operational-corresponding* :: ($'procT \times 'procT$) *set* $\Rightarrow$ *bool* **where**
operational-corresponding TRel $\equiv$ *operational-complete TRel* $\wedge$ *operational-sound TRel*

An encoding is strongly operational corresponding modulo TRel if it is strongly operational complete and strongly operational sound modulo TRel.

abbreviation (**in** *encoding*) *strongly-operational-corresponding*
:: ($'procT \times 'procT$) *set* $\Rightarrow$ *bool*
where
strongly-operational-corresponding TRel $\equiv$
strongly-operational-complete TRel $\wedge$ *strongly-operational-sound TRel*

8.1 Trivial Operational Correspondence Results

Every encoding is (weakly) operational corresponding modulo the all relation on target terms.

lemma (**in** *encoding*) *operational-correspondence-modulo-all-relation*:
shows *operational-complete* $\{(T1,\ T2).\ True\}$
and *weakly-operational-sound* $\{(T1,\ T2).\ True\}$
and *operational-sound* $\{(T1,\ T2).\ True\}$
⟨*proof*⟩

lemma *all-relation-is-weak-reduction-bisimulation*:
fixes *Cal* :: *'a processCalculus*
shows *weak-reduction-bisimulation* $\{(a,\ b).\ True\}$ *Cal*
⟨*proof*⟩

lemma (**in** *encoding*) *operational-correspondence-modulo-some-target-relation*:
shows $\exists$ *TRel. weakly-operational-corresponding TRel*
and $\exists$ *TRel. operational-corresponding TRel*
and $\exists$ *TRel. weakly-operational-corresponding TRel* $\wedge$ *weak-reduction-bisimulation TRel Target*
and $\exists$ *TRel. operational-corresponding TRel* $\wedge$ *weak-reduction-bisimulation TRel Target*
⟨*proof*⟩

Strong operational correspondence requires that source can perform a step iff their translations can perform a step.

lemma (**in** *encoding*) *strong-operational-correspondence-modulo-some-target-relation*:
shows ($\exists$ *TRel. strongly-operational-corresponding TRel*)
$= (\forall S.\ (\exists S'.\ S \longmapsto Source\ S') \longleftrightarrow (\exists T.\ [\![S]\!] \longmapsto Target\ T))$
and ($\exists$ *TRel. strongly-operational-corresponding TRel*
$\wedge$ *weak-reduction-bisimulation TRel Target*)
$= (\forall S.\ (\exists S'.\ S \longmapsto Source\ S') \longleftrightarrow (\exists T.\ [\![S]\!] \longmapsto Target\ T))$
⟨*proof*⟩

8.2 (Strong) Operational Completeness vs (Strong) Simulation

An encoding is operational complete modulo a weak simulation on target terms TRel iff there is a relation, like indRelRTPO, that relates at least all source terms to their literal translations, includes TRel, and is a weak simulation.

lemma (**in** *encoding*) *weak-reduction-simulation-impl-OCom*:
fixes *Rel* :: $(('procS,\ 'procT)\ Proc \times ('procS,\ 'procT)\ Proc)\ set$
and *TRel* :: $('procT \times 'procT)\ set$
assumes *A1*: $\forall S.\ (SourceTerm\ S,\ TargetTerm\ ([\![S]\!])) \in Rel$
and *A2*: $\forall S\ T.\ (SourceTerm\ S,\ TargetTerm\ T) \in Rel \longrightarrow ([\![S]\!],\ T) \in TRel^*$
and *A3*: *weak-reduction-simulation Rel* (*STCal Source Target*)
shows *operational-complete* $(TRel^*)$
⟨*proof*⟩

lemma (**in** *encoding*) *OCom-iff-indRelRTPO-is-weak-reduction-simulation*:
fixes *TRel* :: $('procT \times 'procT)\ set$
shows (*operational-complete* $(TRel^*)$
$\wedge$ *weak-reduction-simulation* $(TRel^+)$ *Target*)
= *weak-reduction-simulation* (*indRelRTPO TRel*) (*STCal Source Target*)
⟨*proof*⟩

lemma (**in** *encoding*) *OCom-iff-weak-reduction-simulation*:
fixes *TRel* :: $('procT \times 'procT)\ set$
shows (*operational-complete* $(TRel^*)$
$\wedge$ *weak-reduction-simulation* $(TRel^+)$ *Target*)
$= (\exists Rel.\ (\forall S.\ (SourceTerm\ S,\ TargetTerm\ ([\![S]\!])) \in Rel)$
$\wedge (\forall T1\ T2.\ (T1,\ T2) \in TRel \longrightarrow (TargetTerm\ T1,\ TargetTerm\ T2) \in Rel)$
$\wedge (\forall T1\ T2.\ (TargetTerm\ T1,\ TargetTerm\ T2) \in Rel \longrightarrow (T1,\ T2) \in TRel^+)$

$\wedge$ ($\forall$ *S T*. (*SourceTerm S*, *TargetTerm T*) $\in$ *Rel* $\longrightarrow$ ($[\![S]\!]$, *T*) $\in$ *TRel**)
$\wedge$ *weak-reduction-simulation Rel* (*STCal Source Target*))
⟨*proof*⟩

An encoding is strong operational complete modulo a strong simulation on target terms TRel iff there is a relation, like indRelRTPO, that relates at least all source terms to their literal translations, includes TRel, and is a strong simulation.

lemma (**in** *encoding*) *strong-reduction-simulation-impl-SOCom*:
fixes *Rel* :: ((*'procS*, *'procT*) *Proc* × (*'procS*, *'procT*) *Proc*) *set*
and *TRel* :: (*'procT* × *'procT*) *set*
assumes *A1*: $\forall$ *S*. (*SourceTerm S*, *TargetTerm* ($[\![S]\!]$)) $\in$ *Rel*
and *A2*: $\forall$ *S T*. (*SourceTerm S*, *TargetTerm T*) $\in$ *Rel* $\longrightarrow$ ($[\![S]\!]$, *T*) $\in$ *TRel**
and *A3*: *strong-reduction-simulation Rel* (*STCal Source Target*)
shows *strongly-operational-complete* (*TRel**)
⟨*proof*⟩

lemma (**in** *encoding*) *SOCom-iff-indRelRTPO-is-strong-reduction-simulation*:
fixes *TRel* :: (*'procT* × *'procT*) *set*
shows (*strongly-operational-complete* (*TRel**)
$\wedge$ *strong-reduction-simulation* (*TRel*$^+$) *Target*)
= *strong-reduction-simulation* (*indRelRTPO TRel*) (*STCal Source Target*)
⟨*proof*⟩

lemma (**in** *encoding*) *SOCom-iff-strong-reduction-simulation*:
fixes *TRel* :: (*'procT* × *'procT*) *set*
shows (*strongly-operational-complete* (*TRel**)
$\wedge$ *strong-reduction-simulation* (*TRel*$^+$) *Target*)
= ($\exists$ *Rel*. ($\forall$ *S*. (*SourceTerm S*, *TargetTerm* ($[\![S]\!]$)) $\in$ *Rel*)
$\wedge$ ($\forall$ *T1 T2*. (*T1*, *T2*) $\in$ *TRel* $\longrightarrow$ (*TargetTerm T1*, *TargetTerm T2*) $\in$ *Rel*)
$\wedge$ ($\forall$ *T1 T2*. (*TargetTerm T1*, *TargetTerm T2*) $\in$ *Rel* $\longrightarrow$ (*T1*, *T2*) $\in$ *TRel*$^+$)
$\wedge$ ($\forall$ *S T*. (*SourceTerm S*, *TargetTerm T*) $\in$ *Rel* $\longrightarrow$ ($[\![S]\!]$, *T*) $\in$ *TRel**)
$\wedge$ *strong-reduction-simulation Rel* (*STCal Source Target*))
⟨*proof*⟩

lemma (**in** *encoding*) *target-relation-from-source-target-relation*:
fixes *Rel* :: ((*'procS*, *'procT*) *Proc* × (*'procS*, *'procT*) *Proc*) *set*
assumes *stre*: $\forall$ *S T*. (*SourceTerm S*, *TargetTerm T*) $\in$ *Rel*
$\longrightarrow$ (*TargetTerm* ($[\![S]\!]$), *TargetTerm T*) $\in$ *Rel*$^=$
shows $\exists$ *TRel*. ($\forall$ *T1 T2*. (*T1*, *T2*) $\in$ *TRel* $\longrightarrow$ (*TargetTerm T1*, *TargetTerm T2*) $\in$ *Rel*)
$\wedge$ ($\forall$ *T1 T2*. (*TargetTerm T1*, *TargetTerm T2*) $\in$ *Rel* $\longrightarrow$ (*T1*, *T2*) $\in$ *TRel*$^+$)
$\wedge$ ($\forall$ *S T*. (*SourceTerm S*, *TargetTerm T*) $\in$ *Rel* $\longrightarrow$ ($[\![S]\!]$, *T*) $\in$ *TRel**)
⟨*proof*⟩

lemma (**in** *encoding*) *SOCom-modulo-TRel-iff-strong-reduction-simulation*:
shows ($\exists$ *TRel*. *strongly-operational-complete* (*TRel**)
$\wedge$ *strong-reduction-simulation* (*TRel*$^+$) *Target*)
= ($\exists$ *Rel*. ($\forall$ *S*. (*SourceTerm S*, *TargetTerm* ($[\![S]\!]$)) $\in$ *Rel*)
$\wedge$ ($\forall$ *S T*. (*SourceTerm S*, *TargetTerm T*) $\in$ *Rel* $\longrightarrow$ (*TargetTerm* ($[\![S]\!]$), *TargetTerm T*) $\in$ *Rel*$^=$)
$\wedge$ *strong-reduction-simulation Rel* (*STCal Source Target*))
⟨*proof*⟩

8.3 Weak Operational Soundness vs Contrasimulation

If the inverse of a relation that includes TRel and relates source terms and their literal translations is a contrasimulation, then the encoding is weakly operational sound.

lemma (**in** *encoding*) *weak-reduction-contrasimulation-impl-WOSou*:
fixes *Rel* :: ((*'procS*, *'procT*) *Proc* × (*'procS*, *'procT*) *Proc*) *set*
and *TRel* :: (*'procT* × *'procT*) *set*
assumes *A1*: $\forall$ *S*. (*SourceTerm S*, *TargetTerm* ($[\![S]\!]$)) $\in$ *Rel*

and $A2$: $\forall S\ T.\ (SourceTerm\ S,\ TargetTerm\ T) \in Rel \longrightarrow ([\![S]\!],\ T) \in TRel^*$
and $A3$: *weak-reduction-contrasimulation* (Rel^{-1}) $(STCal\ Source\ Target)$
shows *weakly-operational-sound* $(TRel^*)$
$\langle proof \rangle$

8.4 (Strong) Operational Soundness vs (Strong) Simulation

An encoding is operational sound modulo a relation TRel whose inverse is a weak reduction simulation on target terms iff there is a relation, like indRelRTPO, that relates at least all source terms to their literal translations, includes TRel, and whose inverse is a weak simulation.

lemma (**in** *encoding*) *weak-reduction-simulation-impl-OSou*:
fixes $Rel :: (('procS,\ 'procT)\ Proc \times ('procS,\ 'procT)\ Proc)\ set$
and $TRel :: ('procT \times 'procT)\ set$
assumes $A1$: $\forall S.\ (SourceTerm\ S,\ TargetTerm\ ([\![S]\!])) \in Rel$
and $A2$: $\forall S\ T.\ (SourceTerm\ S,\ TargetTerm\ T) \in Rel \longrightarrow ([\![S]\!],\ T) \in TRel^*$
and $A3$: *weak-reduction-simulation* (Rel^{-1}) $(STCal\ Source\ Target)$
shows *operational-sound* $(TRel^*)$
$\langle proof \rangle$

lemma (**in** *encoding*) *OSou-iff-inverse-of-indRelRTPO-is-weak-reduction-simulation*:
fixes $TRel :: ('procT \times 'procT)\ set$
shows (*operational-sound* $(TRel^*)$
$\wedge$ *weak-reduction-simulation* $((TRel^+)^{-1})$ $Target$)
$=$ *weak-reduction-simulation* $((indRelRTPO\ TRel)^{-1})$ $(STCal\ Source\ Target)$
$\langle proof \rangle$

lemma (**in** *encoding*) *OSou-iff-weak-reduction-simulation*:
fixes $TRel :: ('procT \times 'procT)\ set$
shows (*operational-sound* $(TRel^*)$
$\wedge$ *weak-reduction-simulation* $((TRel^+)^{-1})$ $Target$)
$= (\exists Rel.\ (\forall S.\ (SourceTerm\ S,\ TargetTerm\ ([\![S]\!])) \in Rel)$
$\wedge\ (\forall T1\ T2.\ (T1,\ T2) \in TRel \longrightarrow (TargetTerm\ T1,\ TargetTerm\ T2) \in Rel)$
$\wedge\ (\forall T1\ T2.\ (TargetTerm\ T1,\ TargetTerm\ T2) \in Rel \longrightarrow (T1,\ T2) \in TRel^+)$
$\wedge\ (\forall S\ T.\ (SourceTerm\ S,\ TargetTerm\ T) \in Rel \longrightarrow ([\![S]\!],\ T) \in TRel^*)$
$\wedge$ *weak-reduction-simulation* (Rel^{-1}) $(STCal\ Source\ Target))$
$\langle proof \rangle$

An encoding is strongly operational sound modulo a relation TRel whose inverse is a strong reduction simulation on target terms iff there is a relation, like indRelRTPO, that relates at least all source terms to their literal translations, includes TRel, and whose inverse is a strong simulation.

lemma (**in** *encoding*) *strong-reduction-simulation-impl-SOSou*:
fixes $Rel :: (('procS,\ 'procT)\ Proc \times ('procS,\ 'procT)\ Proc)\ set$
and $TRel :: ('procT \times 'procT)\ set$
assumes $A1$: $\forall S.\ (SourceTerm\ S,\ TargetTerm\ ([\![S]\!])) \in Rel$
and $A2$: $\forall S\ T.\ (SourceTerm\ S,\ TargetTerm\ T) \in Rel \longrightarrow ([\![S]\!],\ T) \in TRel^*$
and $A3$: *strong-reduction-simulation* (Rel^{-1}) $(STCal\ Source\ Target)$
shows *strongly-operational-sound* $(TRel^*)$
$\langle proof \rangle$

lemma (**in** *encoding*) *SOSou-iff-inverse-of-indRelRTPO-is-strong-reduction-simulation*:
fixes $TRel :: ('procT \times 'procT)\ set$
shows (*strongly-operational-sound* $(TRel^*)$
$\wedge$ *strong-reduction-simulation* $((TRel^+)^{-1})$ $Target$)
$=$ *strong-reduction-simulation* $((indRelRTPO\ TRel)^{-1})$ $(STCal\ Source\ Target)$
$\langle proof \rangle$

lemma (**in** *encoding*) *SOSou-iff-strong-reduction-simulation*:
fixes $TRel :: ('procT \times 'procT)\ set$
shows (*strongly-operational-sound* $(TRel^*)$ $\wedge$ *strong-reduction-simulation* $((TRel^+)^{-1})$ $Target$)

$= (\exists\, Rel.\ (\forall\, S.\ (SourceTerm\ S,\ TargetTerm\ (⟦S⟧)) \in Rel)$
$\wedge\ (\forall\, T1\ T2.\ (T1,\ T2) \in TRel \longrightarrow (TargetTerm\ T1,\ TargetTerm\ T2) \in Rel)$
$\wedge\ (\forall\, T1\ T2.\ (TargetTerm\ T1,\ TargetTerm\ T2) \in Rel \longrightarrow (T1,\ T2) \in TRel^+)$
$\wedge\ (\forall\, S\ T.\ (SourceTerm\ S,\ TargetTerm\ T) \in Rel \longrightarrow (⟦S⟧,\ T) \in TRel^*)$
$\wedge$ *strong-reduction-simulation* (Rel^{-1}) $(STCal\ Source\ Target))$
⟨*proof*⟩

lemma (**in** *encoding*) *SOSou-modulo-TRel-iff-strong-reduction-simulation*:
shows $(\exists\, TRel.$ *strongly-operational-sound* $(TRel^*)$
$\wedge$ *strong-reduction-simulation* $((TRel^+)^{-1})$ *Target*)
$= (\exists\, Rel.\ (\forall\, S.\ (SourceTerm\ S,\ TargetTerm\ (⟦S⟧)) \in Rel)$
$\wedge\ (\forall\, S\ T.\ (SourceTerm\ S,\ TargetTerm\ T) \in Rel \longrightarrow (TargetTerm\ (⟦S⟧),\ TargetTerm\ T) \in Rel^=)$
$\wedge$ *strong-reduction-simulation* (Rel^{-1}) $(STCal\ Source\ Target))$
⟨*proof*⟩

8.5 Weak Operational Correspondence vs Correspondence Similarity

If there exists a relation that relates at least all source terms and their literal translations, includes TRel, and is a correspondence simulation then the encoding is weakly operational corresponding w.r.t. TRel.

lemma (**in** *encoding*) *weak-reduction-correspondence-simulation-impl-WOC*:
fixes Rel :: $(('procS,\ 'procT)\ Proc \times ('procS,\ 'procT)\ Proc)\ set$
and $TRel$ :: $('procT \times 'procT)\ set$
assumes *enc*: $\forall\, S.\ (SourceTerm\ S,\ TargetTerm\ (⟦S⟧)) \in Rel$
and *tRel*: $(\forall\, S\ T.\ (SourceTerm\ S,\ TargetTerm\ T) \in Rel \longrightarrow (⟦S⟧,\ T) \in TRel^*)$
and *cs*: *weak-reduction-correspondence-simulation Rel* $(STCal\ Source\ Target)$
shows *weakly-operational-corresponding* $(TRel^*)$
⟨*proof*⟩

An encoding is weakly operational corresponding w.r.t. a correspondence simulation on target terms TRel iff there exists a relation, like indRelRTPO, that relates at least all source terms and their literal translations, includes TRel, and is a correspondence simulation.

lemma (**in** *encoding*) *WOC-iff-indRelRTPO-is-reduction-correspondence-simulation*:
fixes $TRel$ :: $('procT \times 'procT)\ set$
shows (*weakly-operational-corresponding* $(TRel^*)$
$\wedge$ *weak-reduction-correspondence-simulation* $(TRel^+)$ *Target*)
$=$ *weak-reduction-correspondence-simulation* (*indRelRTPO TRel*) $(STCal\ Source\ Target)$
⟨*proof*⟩

lemma (**in** *encoding*) *WOC-iff-reduction-correspondence-simulation*:
fixes $TRel$ :: $('procT \times 'procT)\ set$
shows (*weakly-operational-corresponding* $(TRel^*)$
$\wedge$ *weak-reduction-correspondence-simulation* $(TRel^+)$ *Target*)
$= (\exists\, Rel.\ (\forall\, S.\ (SourceTerm\ S,\ TargetTerm\ (⟦S⟧)) \in Rel)$
$\wedge\ (\forall\, T1\ T2.\ (T1,\ T2) \in TRel \longrightarrow (TargetTerm\ T1,\ TargetTerm\ T2) \in Rel)$
$\wedge\ (\forall\, T1\ T2.\ (TargetTerm\ T1,\ TargetTerm\ T2) \in Rel \longrightarrow (T1,\ T2) \in TRel^+)$
$\wedge\ (\forall\, S\ T.\ (SourceTerm\ S,\ TargetTerm\ T) \in Rel \longrightarrow (⟦S⟧,\ T) \in TRel^*)$
$\wedge$ *weak-reduction-correspondence-simulation Rel* $(STCal\ Source\ Target))$
⟨*proof*⟩

lemma *rel-includes-TRel-modulo-preorder*:
fixes Rel :: $(('procS,\ 'procT)\ Proc \times ('procS,\ 'procT)\ Proc)\ set$
and $TRel$ :: $('procT \times 'procT)\ set$
assumes *transT*: *trans TRel*
shows $((\forall\, T1\ T2.\ (T1,\ T2) \in TRel \longrightarrow (TargetTerm\ T1,\ TargetTerm\ T2) \in Rel)$
$\wedge\ (\forall\, T1\ T2.\ (TargetTerm\ T1,\ TargetTerm\ T2) \in Rel \longrightarrow (T1,\ T2) \in TRel^+))$
$= (TRel = \{(T1,\ T2).\ (TargetTerm\ T1,\ TargetTerm\ T2) \in Rel\})$
⟨*proof*⟩

lemma (**in** *encoding*) *WOC-wrt-preorder-iff-reduction-correspondence-simulation*:
fixes $TRel :: ('procT \times 'procT)\ set$
shows $(weakly\text{-}operational\text{-}corresponding\ TRel \wedge preorder\ TRel$
$\wedge\ weak\text{-}reduction\text{-}correspondence\text{-}simulation\ TRel\ Target)$
$= (\exists Rel.\ (\forall S.\ (SourceTerm\ S,\ TargetTerm\ (\llbracket S \rrbracket)) \in Rel)$
$\wedge\ TRel = \{(T1,\ T2).\ (TargetTerm\ T1,\ TargetTerm\ T2) \in Rel\}$
$\wedge\ (\forall S\ T.\ (SourceTerm\ S,\ TargetTerm\ T) \in Rel \longrightarrow (\llbracket S \rrbracket,\ T) \in TRel)$
$\wedge\ preorder\ Rel$
$\wedge\ weak\text{-}reduction\text{-}correspondence\text{-}simulation\ Rel\ (STCal\ Source\ Target))$
$\langle proof \rangle$

8.6 (Strong) Operational Correspondence vs (Strong) Bisimilarity

An encoding is operational corresponding w.r.t a weak bisimulation on target terms TRel iff there exists a relation, like indRelRTPO, that relates at least all source terms and their literal translations, includes TRel, and is a weak bisimulation. Thus this variant of operational correspondence ensures that source terms and their translations are weak bisimilar.

lemma (**in** *encoding*) *OC-iff-indRelRTPO-is-weak-reduction-bisimulation*:
fixes $TRel :: ('procT \times 'procT)\ set$
shows $(operational\text{-}corresponding\ (TRel^*)$
$\wedge\ weak\text{-}reduction\text{-}bisimulation\ (TRel^+)\ Target)$
$= weak\text{-}reduction\text{-}bisimulation\ (indRelRTPO\ TRel)\ (STCal\ Source\ Target)$
$\langle proof \rangle$

lemma (**in** *encoding*) *OC-iff-weak-reduction-bisimulation*:
fixes $TRel :: ('procT \times 'procT)\ set$
shows $(operational\text{-}corresponding\ (TRel^*) \wedge weak\text{-}reduction\text{-}bisimulation\ (TRel^+)\ Target)$
$= (\exists Rel.\ (\forall S.\ (SourceTerm\ S,\ TargetTerm\ (\llbracket S \rrbracket)) \in Rel)$
$\wedge\ (\forall T1\ T2.\ (T1,\ T2) \in TRel \longrightarrow (TargetTerm\ T1,\ TargetTerm\ T2) \in Rel)$
$\wedge\ (\forall T1\ T2.\ (TargetTerm\ T1,\ TargetTerm\ T2) \in Rel \longrightarrow (T1,\ T2) \in TRel^+)$
$\wedge\ (\forall S\ T.\ (SourceTerm\ S,\ TargetTerm\ T) \in Rel \longrightarrow (\llbracket S \rrbracket,\ T) \in TRel^*)$
$\wedge\ weak\text{-}reduction\text{-}bisimulation\ Rel\ (STCal\ Source\ Target))$
$\langle proof \rangle$

lemma (**in** *encoding*) *OC-wrt-preorder-iff-weak-reduction-bisimulation*:
fixes $TRel :: ('procT \times 'procT)\ set$
shows $(operational\text{-}corresponding\ TRel \wedge preorder\ TRel$
$\wedge\ weak\text{-}reduction\text{-}bisimulation\ TRel\ Target)$
$= (\exists Rel.\ (\forall S.\ (SourceTerm\ S,\ TargetTerm\ (\llbracket S \rrbracket)) \in Rel)$
$\wedge\ TRel = \{(T1,\ T2).\ (TargetTerm\ T1,\ TargetTerm\ T2) \in Rel\}$
$\wedge\ (\forall S\ T.\ (SourceTerm\ S,\ TargetTerm\ T) \in Rel \longrightarrow (\llbracket S \rrbracket,\ T) \in TRel)$
$\wedge\ preorder\ Rel$
$\wedge\ weak\text{-}reduction\text{-}bisimulation\ Rel\ (STCal\ Source\ Target))$
$\langle proof \rangle$

lemma (**in** *encoding*) *OC-wrt-equivalence-iff-indRelTEQ-weak-reduction-bisimulation*:
fixes $TRel :: ('procT \times 'procT)\ set$
assumes eqT: $equivalence\ TRel$
shows $(operational\text{-}corresponding\ TRel \wedge weak\text{-}reduction\text{-}bisimulation\ TRel\ Target) \longleftrightarrow$
$weak\text{-}reduction\text{-}bisimulation\ (indRelTEQ\ TRel)\ (STCal\ Source\ Target)$
$\langle proof \rangle$

lemma (**in** *encoding*) *OC-wrt-equivalence-iff-weak-reduction-bisimulation*:
fixes $TRel :: ('procT \times 'procT)\ set$
assumes eqT: $equivalence\ TRel$
shows $(operational\text{-}corresponding\ TRel \wedge weak\text{-}reduction\text{-}bisimulation\ TRel\ Target) \longleftrightarrow (\exists Rel.$
$(\forall S.\ (SourceTerm\ S,\ TargetTerm\ (\llbracket S \rrbracket)) \in Rel \wedge (TargetTerm\ (\llbracket S \rrbracket),\ SourceTerm\ S) \in Rel)$
$\wedge\ TRel = \{(T1,\ T2).\ (TargetTerm\ T1,\ TargetTerm\ T2) \in Rel\}$
$\wedge\ trans\ Rel \wedge weak\text{-}reduction\text{-}bisimulation\ Rel\ (STCal\ Source\ Target))$
$\langle proof \rangle$

An encoding is strong operational corresponding w.r.t a strong bisimulation on target terms TRel iff there exists a relation, like indRelRTPO, that relates at least all source terms and their literal translations, includes TRel, and is a strong bisimulation. Thus this variant of operational correspondence ensures that source terms and their translations are strong bisimilar.

lemma (**in** *encoding*) *SOC-iff-indRelRTPO-is-strong-reduction-bisimulation*:
  fixes $TRel :: ('procT \times 'procT)\ set$
  shows $(strongly\text{-}operational\text{-}corresponding\ (TRel^*)$
    $\wedge\ strong\text{-}reduction\text{-}bisimulation\ (TRel^+)\ Target)$
    $=\ strong\text{-}reduction\text{-}bisimulation\ (indRelRTPO\ TRel)\ (STCal\ Source\ Target)$
⟨*proof*⟩

lemma (**in** *encoding*) *SOC-iff-strong-reduction-bisimulation*:
  fixes $TRel :: ('procT \times 'procT)\ set$
  shows $(strongly\text{-}operational\text{-}corresponding\ (TRel^*)$
    $\wedge\ strong\text{-}reduction\text{-}bisimulation\ (TRel^+)\ Target)$
    $= (\exists\, Rel.\ (\forall\, S.\ (SourceTerm\ S,\ TargetTerm\ (\llbracket S \rrbracket)) \in Rel)$
      $\wedge\ (\forall\, T1\ T2.\ (T1,\ T2) \in TRel \longrightarrow (TargetTerm\ T1,\ TargetTerm\ T2) \in Rel)$
      $\wedge\ (\forall\, T1\ T2.\ (TargetTerm\ T1,\ TargetTerm\ T2) \in Rel \longrightarrow (T1,\ T2) \in TRel^+)$
      $\wedge\ (\forall\, S\ T.\ (SourceTerm\ S,\ TargetTerm\ T) \in Rel \longrightarrow (\llbracket S \rrbracket,\ T) \in TRel^*)$
      $\wedge\ strong\text{-}reduction\text{-}bisimulation\ Rel\ (STCal\ Source\ Target))$
⟨*proof*⟩

lemma (**in** *encoding*) *SOC-wrt-preorder-iff-strong-reduction-bisimulation*:
  fixes $TRel :: ('procT \times 'procT)\ set$
  shows $(strongly\text{-}operational\text{-}corresponding\ TRel \wedge preorder\ TRel$
    $\wedge\ strong\text{-}reduction\text{-}bisimulation\ TRel\ Target)$
    $= (\exists\, Rel.\ (\forall\, S.\ (SourceTerm\ S,\ TargetTerm\ (\llbracket S \rrbracket)) \in Rel)$
      $\wedge\ TRel = \{(T1,\ T2).\ (TargetTerm\ T1,\ TargetTerm\ T2) \in Rel\}$
      $\wedge\ (\forall\, S\ T.\ (SourceTerm\ S,\ TargetTerm\ T) \in Rel \longrightarrow (\llbracket S \rrbracket,\ T) \in TRel)$
      $\wedge\ preorder\ Rel$
      $\wedge\ strong\text{-}reduction\text{-}bisimulation\ Rel\ (STCal\ Source\ Target))$
⟨*proof*⟩

lemma (**in** *encoding*) *SOC-wrt-TRel-iff-strong-reduction-bisimulation*:
  shows $(\exists\, TRel.\ strongly\text{-}operational\text{-}corresponding\ (TRel^*)$
    $\wedge\ strong\text{-}reduction\text{-}bisimulation\ (TRel^+)\ Target)$
    $= (\exists\, Rel.\ (\forall\, S.\ (SourceTerm\ S,\ TargetTerm\ (\llbracket S \rrbracket)) \in Rel)$
      $\wedge\ (\forall\, S\ T.\ (SourceTerm\ S,\ TargetTerm\ T) \in Rel$
        $\longrightarrow (TargetTerm\ (\llbracket S \rrbracket),\ TargetTerm\ T) \in Rel^=)$
      $\wedge\ strong\text{-}reduction\text{-}bisimulation\ Rel\ (STCal\ Source\ Target))$
⟨*proof*⟩

lemma (**in** *encoding*) *SOC-wrt-equivalence-iff-indRelTEQ-strong-reduction-bisimulation*:
  fixes $TRel :: ('procT \times 'procT)\ set$
  assumes *eqT*: *equivalence TRel*
  shows $(strongly\text{-}operational\text{-}corresponding\ TRel \wedge strong\text{-}reduction\text{-}bisimulation\ TRel\ Target)$
    $\longleftrightarrow strong\text{-}reduction\text{-}bisimulation\ (indRelTEQ\ TRel)\ (STCal\ Source\ Target)$
⟨*proof*⟩

lemma (**in** *encoding*) *SOC-wrt-equivalence-iff-strong-reduction-bisimulation*:
  fixes $TRel :: ('procT \times 'procT)\ set$
  assumes *eqT*: *equivalence TRel*
  shows $(strongly\text{-}operational\text{-}corresponding\ TRel \wedge strong\text{-}reduction\text{-}bisimulation\ TRel\ Target)$
    $\longleftrightarrow (\exists\, Rel.$
    $(\forall\, S.\ (SourceTerm\ S,\ TargetTerm\ (\llbracket S \rrbracket)) \in Rel \wedge (TargetTerm\ (\llbracket S \rrbracket),\ SourceTerm\ S) \in Rel)$
    $\wedge\ TRel = \{(T1,\ T2).\ (TargetTerm\ T1,\ TargetTerm\ T2) \in Rel\}$
    $\wedge\ trans\ Rel \wedge strong\text{-}reduction\text{-}bisimulation\ Rel\ (STCal\ Source\ Target))$
⟨*proof*⟩

end

theory *FullAbstraction*
  imports *SourceTargetRelation*
begin

9 Full Abstraction

An encoding is fully abstract w.r.t. some source term relation SRel and some target term relation TRel if two source terms S1 and S2 form a pair (S1, S2) in SRel iff their literal translations form a pair (enc S1, enc S2) in TRel.

abbreviation (**in** *encoding*) *fully-abstract*
  :: $('procS \times 'procS)\ set \Rightarrow ('procT \times 'procT)\ set \Rightarrow bool$
  where
  $fully\text{-}abstract\ SRel\ TRel \equiv \forall S1\ S2.\ (S1,\ S2) \in SRel \longleftrightarrow ([\![S1]\!],\ [\![S2]\!]) \in TRel$

9.1 Trivial Full Abstraction Results

We start with some trivial full abstraction results. Each injective encoding is fully abstract w.r.t. to the identity relation on the source and the identity relation on the target.

lemma (**in** *encoding*) *inj-enc-is-fully-abstract-wrt-identities*:
  assumes *injectivity*: $\forall S1\ S2.\ [\![S1]\!] = [\![S2]\!] \longrightarrow S1 = S2$
  shows $fully\text{-}abstract\ \{(S1,\ S2).\ S1 = S2\}\ \{(T1,\ T2).\ T1 = T2\}$
    ⟨*proof*⟩

Each encoding is fully abstract w.r.t. the empty relation on the source and the target.

lemma (**in** *encoding*) *fully-abstract-wrt-empty-relation*:
  shows *fully-abstract* {} {}
    ⟨*proof*⟩

Similarly, each encoding is fully abstract w.r.t. the all-relation on the source and the target.

lemma (**in** *encoding*) *fully-abstract-wrt-all-relation*:
  shows $fully\text{-}abstract\ \{(S1,\ S2).\ True\}\ \{(T1,\ T2).\ True\}$
    ⟨*proof*⟩

If the encoding is injective then for each source term relation RelS there exists a target term relation RelT such that the encoding is fully abstract w.r.t. RelS and RelT.

lemma (**in** *encoding*) *fully-abstract-wrt-source-relation*:
  fixes $RelS :: ('procS \times 'procS)\ set$
  assumes *injectivity*: $\forall S1\ S2.\ [\![S1]\!] = [\![S2]\!] \longrightarrow S1 = S2$
  shows $\exists RelT.\ fully\text{-}abstract\ RelS\ RelT$
⟨*proof*⟩

If all source terms that are translated to the same target term are related by a trans source term relation RelS, then there exists a target term relation RelT such that the encoding is fully abstract w.r.t. RelS and RelT.

lemma (**in** *encoding*) *fully-abstract-wrt-trans-source-relation*:
  fixes $RelS :: ('procS \times 'procS)\ set$
  assumes *encRelS*: $\forall S1\ S2.\ [\![S1]\!] = [\![S2]\!] \longrightarrow (S1,\ S2) \in RelS$
    and *transS*: *trans RelS*
  shows $\exists RelT.\ fully\text{-}abstract\ RelS\ RelT$
⟨*proof*⟩

lemma (**in** *encoding*) *fully-abstract-wrt-trans-closure-of-source-relation*:
  fixes $RelS :: ('procS \times 'procS)\ set$
  assumes *encRelS*: $\forall S1\ S2.\ [\![S1]\!] = [\![S2]\!] \longrightarrow (S1,\ S2) \in RelS^+$
  shows $\exists RelT.\ fully\text{-}abstract\ (RelS^+)\ RelT$
    ⟨*proof*⟩

For every encoding and every target term relation RelT there exists a source term relation RelS such that the encoding is fully abstract w.r.t. RelS and RelT.

lemma (**in** *encoding*) *fully-abstract-wrt-target-relation*:
  fixes $RelT :: ('procT \times 'procT)\ set$
  shows $\exists RelS.\ fully\text{-}abstract\ RelS\ RelT$
$\langle proof \rangle$

9.2 Fully Abstract Encodings

Thus, as long as we can choose one of the two relations, full abstraction is trivial. For fixed source and target term relations encodings are not trivially fully abstract. For all encodings and relations SRel and TRel we can construct a relation on the disjunctive union of source and target terms, whose reduction to source terms is SRel and whose reduction to target terms is TRel. But full abstraction ensures that each trans relation that relates source terms and their literal translations in both directions includes SRel iff it includes TRel restricted to translated source terms.

lemma (**in** *encoding*) *full-abstraction-and-trans-relation-contains-SRel-impl-TRel*:
  fixes $Rel :: (('procS, 'procT)\ Proc \times ('procS, 'procT)\ Proc)\ set$
    and $SRel :: ('procS \times 'procS)\ set$
    and $TRel :: ('procT \times 'procT)\ set$
  assumes *fullAbs*: $fully\text{-}abstract\ SRel\ TRel$
      and *encR*: $\forall S.\ (SourceTerm\ S,\ TargetTerm\ (\llbracket S \rrbracket)) \in Rel$
      and *srel*: $SRel = \{(S1,\ S2).\ (SourceTerm\ S1,\ SourceTerm\ S2) \in Rel\}$
      and *trans*: $trans\ (Rel \cup \{(P,\ Q).\ \exists S.\ \llbracket S \rrbracket \in T\ P \wedge S \in S\ Q\})$
  shows $\forall S1\ S2.\ (\llbracket S1 \rrbracket,\ \llbracket S2 \rrbracket) \in TRel \longleftrightarrow (TargetTerm\ (\llbracket S1 \rrbracket),\ TargetTerm\ (\llbracket S2 \rrbracket)) \in Rel$
$\langle proof \rangle$

lemma (**in** *encoding*) *full-abstraction-and-trans-relation-contains-TRel-impl-SRel*:
  fixes $Rel :: (('procS, 'procT)\ Proc \times ('procS, 'procT)\ Proc)\ set$
    and $SRel :: ('procS \times 'procS)\ set$
    and $TRel :: ('procT \times 'procT)\ set$
  assumes *fullAbs*: $fully\text{-}abstract\ SRel\ TRel$
      and *encR*: $\forall S.\ (SourceTerm\ S,\ TargetTerm\ (\llbracket S \rrbracket)) \in Rel$
      and *trel*: $\forall S1\ S2.\ (\llbracket S1 \rrbracket,\ \llbracket S2 \rrbracket) \in TRel \longleftrightarrow (TargetTerm\ (\llbracket S1 \rrbracket),\ TargetTerm\ (\llbracket S2 \rrbracket)) \in Rel$
      and *trans*: $trans\ (Rel \cup \{(P,\ Q).\ \exists S.\ \llbracket S \rrbracket \in T\ P \wedge S \in S\ Q\})$
  shows $SRel = \{(S1,\ S2).\ (SourceTerm\ S1,\ SourceTerm\ S2) \in Rel\}$
$\langle proof \rangle$

lemma (**in** *encoding*) *full-abstraction-impl-trans-relation-contains-SRel-iff-TRel*:
  fixes $Rel :: (('procS, 'procT)\ Proc \times ('procS, 'procT)\ Proc)\ set$
    and $SRel :: ('procS \times 'procS)\ set$
    and $TRel :: ('procT \times 'procT)\ set$
  assumes *fullAbs*: $fully\text{-}abstract\ SRel\ TRel$
      and *encR*: $\forall S.\ (SourceTerm\ S,\ TargetTerm\ (\llbracket S \rrbracket)) \in Rel$
      and *trans*: $trans\ (Rel \cup \{(P,\ Q).\ \exists S.\ \llbracket S \rrbracket \in T\ P \wedge S \in S\ Q\})$
  shows $(\forall S1\ S2.\ (\llbracket S1 \rrbracket,\ \llbracket S2 \rrbracket) \in TRel \longleftrightarrow (TargetTerm\ (\llbracket S1 \rrbracket),\ TargetTerm\ (\llbracket S2 \rrbracket)) \in Rel)$
      $\longleftrightarrow (SRel = \{(S1,\ S2).\ (SourceTerm\ S1,\ SourceTerm\ S2) \in Rel\})$
$\langle proof \rangle$

lemma (**in** *encoding*) *full-abstraction-impl-trans-relation-contains-SRel-iff-TRel-encRL*:
  fixes $Rel :: (('procS, 'procT)\ Proc \times ('procS, 'procT)\ Proc)\ set$
    and $SRel :: ('procS \times 'procS)\ set$
    and $TRel :: ('procT \times 'procT)\ set$
  assumes *fullAbs*: $fully\text{-}abstract\ SRel\ TRel$
      and *encR*: $\forall S.\ (SourceTerm\ S,\ TargetTerm\ (\llbracket S \rrbracket)) \in Rel$
      and *encL*: $\forall S.\ (TargetTerm\ (\llbracket S \rrbracket),\ SourceTerm\ S) \in Rel$
      and *trans*: $trans\ Rel$
  shows $(\forall S1\ S2.\ (\llbracket S1 \rrbracket,\ \llbracket S2 \rrbracket) \in TRel \longleftrightarrow (TargetTerm\ (\llbracket S1 \rrbracket),\ TargetTerm\ (\llbracket S2 \rrbracket)) \in Rel)$
      $\longleftrightarrow (SRel = \{(S1,\ S2).\ (SourceTerm\ S1,\ SourceTerm\ S2) \in Rel\})$
$\langle proof \rangle$

Full abstraction ensures that SRel and TRel satisfy the same basic properties that can be defined on their pairs. In particular: (1) SRel is refl iff TRel reduced to translated source terms is refl (2) if the encoding is surjective then SRel is refl iff TRel is refl (3) SRel is sym iff TRel reduced to translated source terms is sym (4) SRel is trans iff TRel reduced to translated source terms is trans

lemma (**in** *encoding*) *full-abstraction-impl-SRel-iff-TRel-is-refl*:
  fixes $SRel :: ('procS \times 'procS)\ set$
    and $TRel :: ('procT \times 'procT)\ set$
  assumes *fullAbs*: *fully-abstract SRel TRel*
  shows $refl\ SRel \longleftrightarrow (\forall S.\ (⟦S⟧, ⟦S⟧) \in TRel)$
    ⟨*proof*⟩

lemma (**in** *encoding*) *full-abstraction-and-surjectivity-impl-SRel-iff-TRel-is-refl*:
  fixes $SRel :: ('procS \times 'procS)\ set$
    and $TRel :: ('procT \times 'procT)\ set$
  assumes *fullAbs*: *fully-abstract SRel TRel*
    and *surj*: $\forall T.\ \exists S.\ T = ⟦S⟧$
  shows $refl\ SRel \longleftrightarrow refl\ TRel$
⟨*proof*⟩

lemma (**in** *encoding*) *full-abstraction-impl-SRel-iff-TRel-is-sym*:
  fixes $SRel :: ('procS \times 'procS)\ set$
    and $TRel :: ('procT \times 'procT)\ set$
  assumes *fullAbs*: *fully-abstract SRel TRel*
  shows $sym\ SRel \longleftrightarrow sym\ \{(T1,\ T2).\ \exists S1\ S2.\ T1 = ⟦S1⟧ \wedge T2 = ⟦S2⟧ \wedge (T1,\ T2) \in TRel\}$
    ⟨*proof*⟩

lemma (**in** *encoding*) *full-abstraction-and-surjectivity-impl-SRel-iff-TRel-is-sym*:
  fixes $SRel :: ('procS \times 'procS)\ set$
    and $TRel :: ('procT \times 'procT)\ set$
  assumes *fullAbs*: *fully-abstract SRel TRel*
    and *surj*: $\forall T.\ \exists S.\ T = ⟦S⟧$
  shows $sym\ SRel \longleftrightarrow sym\ TRel$
    ⟨*proof*⟩

lemma (**in** *encoding*) *full-abstraction-impl-SRel-iff-TRel-is-trans*:
  fixes $SRel :: ('procS \times 'procS)\ set$
    and $TRel :: ('procT \times 'procT)\ set$
  assumes *fullAbs*: *fully-abstract SRel TRel*
  shows $trans\ SRel \longleftrightarrow trans\ \{(T1,\ T2).\ \exists S1\ S2.\ T1 = ⟦S1⟧ \wedge T2 = ⟦S2⟧ \wedge (T1,\ T2) \in TRel\}$
    ⟨*proof*⟩

lemma (**in** *encoding*) *full-abstraction-and-surjectivity-impl-SRel-iff-TRel-is-trans*:
  fixes $SRel :: ('procS \times 'procS)\ set$
    and $TRel :: ('procT \times 'procT)\ set$
  assumes *fullAbs*: *fully-abstract SRel TRel*
    and *surj*: $\forall T.\ \exists S.\ T = ⟦S⟧$
  shows $trans\ SRel \longleftrightarrow trans\ TRel$
    ⟨*proof*⟩

Similarly, a fully abstract encoding that respects a predicate ensures the this predicate is preserved, reflected, or respected by SRel iff it is preserved, reflected, or respected by TRel.

lemma (**in** *encoding*) *full-abstraction-and-enc-respects-pred-impl-SRel-iff-TRel-preserve*:
  fixes $SRel :: ('procS \times 'procS)\ set$
    and $TRel :: ('procT \times 'procT)\ set$
    and $Pred :: ('procS,\ 'procT)\ Proc \Rightarrow bool$
  assumes *fullAbs*: *fully-abstract SRel TRel*
    and *encP*: *enc-respects-pred Pred*
  shows $rel\text{-}preserves\text{-}pred\ \{(P,\ Q).\ \exists SP\ SQ.\ SP \in S\ P \wedge SQ \in S\ Q \wedge (SP,\ SQ) \in SRel\}\ Pred$
      $\longleftrightarrow rel\text{-}preserves\text{-}pred\ \{(P,\ Q).\ \exists SP\ SQ.\ ⟦SP⟧ \in T\ P \wedge ⟦SQ⟧ \in T\ Q \wedge (⟦SP⟧, ⟦SQ⟧) \in TRel\}\ Pred$
⟨*proof*⟩

lemma (**in** *encoding*) *full-abstraction-and-enc-respects-binary-pred-impl-SRel-iff-TRel-preserve*:
  fixes *SRel* :: ($'procS \times 'procS$) *set*
    and *TRel* :: ($'procT \times 'procT$) *set*
    and *Pred* :: ($'procS$, $'procT$) *Proc* $\Rightarrow$ $'b \Rightarrow$ *bool*
  assumes *fullAbs*: *fully-abstract SRel TRel*
      and *encP*:   *enc-respects-binary-pred Pred*
  shows *rel-preserves-binary-pred* $\{(P, Q). \exists SP\ SQ.\ SP \in S\ P \wedge SQ \in S\ Q \wedge (SP, SQ) \in SRel\}$ *Pred*
        $\longleftrightarrow$ *rel-preserves-binary-pred*
            $\{(P, Q). \exists SP\ SQ.\ [\![SP]\!] \in T\ P \wedge [\![SQ]\!] \in T\ Q \wedge ([\![SP]\!], [\![SQ]\!]) \in TRel\}$ *Pred*
⟨*proof*⟩

lemma (**in** *encoding*) *full-abstraction-and-enc-respects-pred-impl-SRel-iff-TRel-reflects*:
  fixes *SRel* :: ($'procS \times 'procS$) *set*
    and *TRel* :: ($'procT \times 'procT$) *set*
    and *Pred* :: ($'procS$, $'procT$) *Proc* $\Rightarrow$ *bool*
  assumes *fullAbs*: *fully-abstract SRel TRel*
      and *encP*:   *enc-respects-pred Pred*
  shows *rel-reflects-pred* $\{(P, Q). \exists SP\ SQ.\ SP \in S\ P \wedge SQ \in S\ Q \wedge (SP, SQ) \in SRel\}$ *Pred*
        $\longleftrightarrow$ *rel-reflects-pred* $\{(P, Q). \exists SP\ SQ.\ [\![SP]\!] \in T\ P \wedge [\![SQ]\!] \in T\ Q \wedge ([\![SP]\!], [\![SQ]\!]) \in TRel\}$ *Pred*
⟨*proof*⟩

lemma (**in** *encoding*) *full-abstraction-and-enc-respects-binary-pred-impl-SRel-iff-TRel-reflects*:
  fixes *SRel* :: ($'procS \times 'procS$) *set*
    and *TRel* :: ($'procT \times 'procT$) *set*
    and *Pred* :: ($'procS$, $'procT$) *Proc* $\Rightarrow$ $'b \Rightarrow$ *bool*
  assumes *fullAbs*: *fully-abstract SRel TRel*
      and *encP*:   *enc-respects-binary-pred Pred*
  shows *rel-reflects-binary-pred* $\{(P, Q). \exists SP\ SQ.\ SP \in S\ P \wedge SQ \in S\ Q \wedge (SP, SQ) \in SRel\}$ *Pred*
        $\longleftrightarrow$ *rel-reflects-binary-pred*
            $\{(P, Q). \exists SP\ SQ.\ [\![SP]\!] \in T\ P \wedge [\![SQ]\!] \in T\ Q \wedge ([\![SP]\!], [\![SQ]\!]) \in TRel\}$ *Pred*
⟨*proof*⟩

lemma (**in** *encoding*) *full-abstraction-and-enc-respects-pred-impl-SRel-iff-TRel-respects*:
  fixes *SRel* :: ($'procS \times 'procS$) *set*
    and *TRel* :: ($'procT \times 'procT$) *set*
    and *Pred* :: ($'procS$, $'procT$) *Proc* $\Rightarrow$ *bool*
  assumes *fullAbs*: *fully-abstract SRel TRel*
      and *encP*:   *enc-respects-pred Pred*
  shows *rel-respects-pred* $\{(P, Q). \exists SP\ SQ.\ SP \in S\ P \wedge SQ \in S\ Q \wedge (SP, SQ) \in SRel\}$ *Pred*
        $\longleftrightarrow$ *rel-respects-pred* $\{(P, Q). \exists SP\ SQ.\ [\![SP]\!] \in T\ P \wedge [\![SQ]\!] \in T\ Q \wedge ([\![SP]\!], [\![SQ]\!]) \in TRel\}$ *Pred*
      ⟨*proof*⟩

lemma (**in** *encoding*) *full-abstraction-and-enc-respects-binary-pred-impl-SRel-iff-TRel-respects*:
  fixes *SRel* :: ($'procS \times 'procS$) *set*
    and *TRel* :: ($'procT \times 'procT$) *set*
    and *Pred* :: ($'procS$, $'procT$) *Proc* $\Rightarrow$ $'b \Rightarrow$ *bool*
  assumes *fullAbs*: *fully-abstract SRel TRel*
      and *encP*:   *enc-respects-binary-pred Pred*
  shows *rel-respects-binary-pred* $\{(P, Q). \exists SP\ SQ.\ SP \in S\ P \wedge SQ \in S\ Q \wedge (SP, SQ) \in SRel\}$ *Pred*
        $\longleftrightarrow$ *rel-respects-binary-pred*
            $\{(P, Q). \exists SP\ SQ.\ [\![SP]\!] \in T\ P \wedge [\![SQ]\!] \in T\ Q \wedge ([\![SP]\!], [\![SQ]\!]) \in TRel\}$ *Pred*
      ⟨*proof*⟩

9.3 Full Abstraction w.r.t. Preorders

If there however exists a trans relation Rel that relates source terms and their literal translations in both directions, then the encoding is fully abstract with respect to the reduction of Rel to source terms and the reduction of Rel to target terms.

lemma (**in** *encoding*) *trans-source-target-relation-impl-full-abstraction*:

fixes $Rel :: (('procS, 'procT)\ Proc \times ('procS, 'procT)\ Proc)\ set$
assumes enc: $\forall S.\ (SourceTerm\ S,\ TargetTerm\ ([\![S]\!])) \in Rel$
$\wedge\ (TargetTerm\ ([\![S]\!]),\ SourceTerm\ S) \in Rel$
and $trans$: $trans\ Rel$
shows $fully\text{-}abstract\ \{(S1,\ S2).\ (SourceTerm\ S1,\ SourceTerm\ S2) \in Rel\}$
$\{(T1,\ T2).\ (TargetTerm\ T1,\ TargetTerm\ T2) \in Rel\}$
⟨*proof*⟩

lemma (**in** *encoding*) *source-target-relation-impl-full-abstraction-wrt-trans-closures*:
fixes $Rel :: (('procS, 'procT)\ Proc \times ('procS, 'procT)\ Proc)\ set$
assumes enc: $\forall S.\ (SourceTerm\ S,\ TargetTerm\ ([\![S]\!])) \in Rel$
$\wedge\ (TargetTerm\ ([\![S]\!]),\ SourceTerm\ S) \in Rel$
shows $fully\text{-}abstract\ \{(S1,\ S2).\ (SourceTerm\ S1,\ SourceTerm\ S2) \in Rel^+\}$
$\{(T1,\ T2).\ (TargetTerm\ T1,\ TargetTerm\ T2) \in Rel^+\}$
⟨*proof*⟩

lemma (**in** *encoding*) *quasi-trans-source-target-relation-impl-full-abstraction*:
fixes $Rel :: (('procS, 'procT)\ Proc \times ('procS, 'procT)\ Proc)\ set$
and $SRel :: ('procS \times 'procS)\ set$
and $TRel :: ('procT \times 'procT)\ set$
assumes enc: $\forall S.\ (SourceTerm\ S,\ TargetTerm\ ([\![S]\!])) \in Rel$
$\wedge\ (TargetTerm\ ([\![S]\!]),\ SourceTerm\ S) \in Rel$
and $srel$: $SRel = \{(S1,\ S2).\ (SourceTerm\ S1,\ SourceTerm\ S2) \in Rel\}$
and $trel$: $TRel = \{(T1,\ T2).\ (TargetTerm\ T1,\ TargetTerm\ T2) \in Rel\}$
and $trans$: $\forall P\ Q\ R.\ (P,\ Q) \in Rel \wedge (Q,\ R) \in Rel \wedge ((P \in ProcS \wedge Q \in ProcT)$
$\vee\ (P \in ProcT \wedge Q \in ProcS)) \longrightarrow (P,\ R) \in Rel$
shows $fully\text{-}abstract\ SRel\ TRel$
⟨*proof*⟩

If an encoding is fully abstract w.r.t. SRel and TRel, then we can conclude from a pair in indRelRTPO or indRelSTEQ on a pair in TRel and SRel.

lemma (**in** *encoding*) *full-abstraction-impl-indRelRSTPO-to-SRel-and-TRel*:
fixes $SRel :: ('procS \times 'procS)\ set$
and $TRel :: ('procT \times 'procT)\ set$
and $P\ Q :: ('procS, 'procT)\ Proc$
assumes $fullAbs$: $fully\text{-}abstract\ SRel\ TRel$
and rel: $P \lesssim[\![\cdot]\!]R{<}SRel,TRel{>}\ Q$
shows $\forall SP\ SQ.\ SP \in S\ P \wedge SQ \in S\ Q \longrightarrow ([\![SP]\!],\ [\![SQ]\!]) \in TRel^+$
and $\forall SP\ TQ.\ SP \in S\ P \wedge TQ \in T\ Q \longrightarrow ([\![SP]\!],\ TQ) \in TRel^*$
⟨*proof*⟩

lemma (**in** *encoding*) *full-abstraction-wrt-preorders-impl-indRelSTEQ-to-SRel-and-TRel*:
fixes $SRel :: ('procS \times 'procS)\ set$
and $TRel :: ('procT \times 'procT)\ set$
and $P\ Q :: ('procS, 'procT)\ Proc$
assumes fA: $fully\text{-}abstract\ SRel\ TRel$
and $transT$: $trans\ TRel$
and $reflS$: $refl\ SRel$
and rel: $P \sim[\![\cdot]\!]{<}SRel,TRel{>}\ Q$
shows $\forall SP\ SQ.\ SP \in S\ P \wedge SQ \in S\ Q \longrightarrow (SP,\ SQ) \in SRel$
and $\forall SP\ SQ.\ SP \in S\ P \wedge SQ \in S\ Q \longrightarrow ([\![SP]\!],\ [\![SQ]\!]) \in TRel$
and $\forall SP\ TQ.\ SP \in S\ P \wedge TQ \in T\ Q \longrightarrow ([\![SP]\!],\ TQ) \in TRel$
and $\forall TP\ SQ.\ TP \in T\ P \wedge SQ \in S\ Q \longrightarrow (TP,\ [\![SQ]\!]) \in TRel$
and $\forall TP\ TQ.\ TP \in T\ P \wedge TQ \in T\ Q \longrightarrow (TP,\ TQ) \in TRel$
⟨*proof*⟩

If an encoding is fully abstract w.r.t. a preorder SRel on the source and a trans relation TRel on the target, then there exists a trans relation, namely indRelSTEQ, that relates source terms and their literal translations in both direction such that its reductions to source terms is SRel and its reduction to target terms is TRel.

lemma (**in** *encoding*) *full-abstraction-wrt-preorders-impl-trans-source-target-relation*:
fixes *SRel* :: ('*procS* × '*procS*) *set*
and *TRel* :: ('*procT* × '*procT*) *set*
assumes *fullAbs*: *fully-abstract SRel TRel*
and *reflS*: *refl SRel*
and *transT*: *trans TRel*
shows $\exists$ *Rel*. ($\forall$ *S*. (*SourceTerm S*, *TargetTerm* ([[*S*]])) $\in$ *Rel*
$\wedge$ (*TargetTerm* ([[*S*]]), *SourceTerm S*) $\in$ *Rel*)
$\wedge$ *SRel* = {(*S1*, *S2*). (*SourceTerm S1*, *SourceTerm S2*) $\in$ *Rel*}
$\wedge$ *TRel* = {(*T1*, *T2*). (*TargetTerm T1*, *TargetTerm T2*) $\in$ *Rel*}
$\wedge$ *trans Rel*
⟨*proof*⟩

Thus an encoding is fully abstract w.r.t. a preorder SRel on the source and a trans relation TRel on the target iff there exists a trans relation that relates source terms and their literal translations in both directions and whose reduction to source/target terms is SRel/TRel.

theorem (**in** *encoding*) *fully-abstract-wrt-preorders-iff-source-target-relation-is-trans*:
fixes *SRel* :: ('*procS* × '*procS*) *set*
and *TRel* :: ('*procT* × '*procT*) *set*
shows (*fully-abstract SRel TRel* $\wedge$ *refl SRel* $\wedge$ *trans TRel*) =
($\exists$ *Rel*. ($\forall$ *S*. (*SourceTerm S*, *TargetTerm* ([[*S*]])) $\in$ *Rel*
$\wedge$ (*TargetTerm* ([[*S*]]), *SourceTerm S*) $\in$ *Rel*)
$\wedge$ *SRel* = {(*S1*, *S2*). (*SourceTerm S1*, *SourceTerm S2*) $\in$ *Rel*}
$\wedge$ *TRel* = {(*T1*, *T2*). (*TargetTerm T1*, *TargetTerm T2*) $\in$ *Rel*}
$\wedge$ *trans Rel*)
⟨*proof*⟩

9.4 Full Abstraction w.r.t. Equivalences

If there exists a relation Rel that relates source terms and their literal translations and whose sym closure is trans, then the encoding is fully abstract with respect to the reduction of the sym closure of Rel to source/target terms.

lemma (**in** *encoding*) *source-target-relation-with-trans-symcl-impl-full-abstraction*:
fixes *Rel* :: (('*procS*, '*procT*) *Proc* × ('*procS*, '*procT*) *Proc*) *set*
assumes *enc*: $\forall$ *S*. (*SourceTerm S*, *TargetTerm* ([[*S*]])) $\in$ *Rel*
and *trans*: *trans* (*symcl Rel*)
shows *fully-abstract* {(*S1*, *S2*). (*SourceTerm S1*, *SourceTerm S2*) $\in$ *symcl Rel*}
{(*T1*, *T2*). (*TargetTerm T1*, *TargetTerm T2*) $\in$ *symcl Rel*}
⟨*proof*⟩

If an encoding is fully abstract w.r.t. the equivalences SRel and TRel, then there exists a preorder, namely indRelRSTPO, that relates source terms and their literal translations such that its reductions to source terms is SRel and its reduction to target terms is TRel.

lemma (**in** *encoding*) *fully-abstract-wrt-equivalences-impl-symcl-source-target-relation-is-preorder*:
fixes *SRel* :: ('*procS* × '*procS*) *set*
and *TRel* :: ('*procT* × '*procT*) *set*
assumes *fullAbs*: *fully-abstract SRel TRel*
and *reflT*: *refl TRel*
and *symmT*: *sym TRel*
and *transT*: *trans TRel*
shows $\exists$ *Rel*. ($\forall$ *S*. (*SourceTerm S*, *TargetTerm* ([[*S*]])) $\in$ *Rel*)
$\wedge$ *SRel* = {(*S1*, *S2*). (*SourceTerm S1*, *SourceTerm S2*) $\in$ *symcl Rel*}
$\wedge$ *TRel* = {(*T1*, *T2*). (*TargetTerm T1*, *TargetTerm T2*) $\in$ *symcl Rel*}
$\wedge$ *preorder* (*symcl Rel*)
⟨*proof*⟩

lemma (**in** *encoding*) *fully-abstract-impl-symcl-source-target-relation-is-preorder*:
fixes *SRel* :: ('*procS* × '*procS*) *set*
and *TRel* :: ('*procT* × '*procT*) *set*

assumes *fullAbs*: *fully-abstract* $((symcl\ (SRel^{=}))^{+})\ ((symcl\ (TRel^{=}))^{+})$
shows $\exists Rel.\ (\forall S.\ (SourceTerm\ S,\ TargetTerm\ (\llbracket S \rrbracket)) \in Rel)$
$\wedge\ ((symcl\ (SRel^{=}))^{+}) = \{(S1,\ S2).\ (SourceTerm\ S1,\ SourceTerm\ S2) \in symcl\ Rel\}$
$\wedge\ ((symcl\ (TRel^{=}))^{+}) = \{(T1,\ T2).\ (TargetTerm\ T1,\ TargetTerm\ T2) \in symcl\ Rel\}$
$\wedge\ preorder\ (symcl\ Rel)$
⟨*proof*⟩

lemma (**in** *encoding*) *fully-abstract-wrt-preorders-impl-source-target-relation-is-trans*:
fixes *SRel* :: (*'procS* × *'procS*) *set*
and *TRel* :: (*'procT* × *'procT*) *set*
assumes *fullAbs*: *fully-abstract SRel TRel*
shows $\exists Rel.\ (\forall S.\ (SourceTerm\ S,\ TargetTerm\ (\llbracket S \rrbracket)) \in Rel)$
$\wedge\ SRel = \{(S1,\ S2).\ (SourceTerm\ S1,\ SourceTerm\ S2) \in Rel\}$
$\wedge\ TRel = \{(T1,\ T2).\ (TargetTerm\ T1,\ TargetTerm\ T2) \in Rel\}$
$\wedge\ ((refl\ SRel \wedge trans\ TRel)$
$\longleftrightarrow trans\ (Rel \cup \{(P,\ Q).\ \exists S.\ \llbracket S \rrbracket \in T\ P \wedge S \in S\ Q\}))$
⟨*proof*⟩

lemma (**in** *encoding*) *fully-abstract-wrt-preorders-impl-source-target-relation-is-trans-B*:
fixes *SRel* :: (*'procS* × *'procS*) *set*
and *TRel* :: (*'procT* × *'procT*) *set*
assumes *fullAbs*: *fully-abstract SRel TRel*
and *reflT*: *refl TRel*
and *transT*: *trans TRel*
shows $\exists Rel.\ (\forall S.\ (SourceTerm\ S,\ TargetTerm\ (\llbracket S \rrbracket)) \in Rel)$
$\wedge\ SRel = \{(S1,\ S2).\ (SourceTerm\ S1,\ SourceTerm\ S2) \in Rel\}$
$\wedge\ TRel = \{(T1,\ T2).\ (TargetTerm\ T1,\ TargetTerm\ T2) \in Rel\}$
$\wedge\ trans\ (Rel \cup \{(P,\ Q).\ \exists S.\ \llbracket S \rrbracket \in T\ P \wedge S \in S\ Q\})$
⟨*proof*⟩

Thus an encoding is fully abstract w.r.t. an equivalence SRel on the source and an equivalence TRel on the target iff there exists a relation that relates source terms and their literal translations, whose sym closure is a preorder such that the reduction of this sym closure to source/target terms is SRel/TRel.

lemma (**in** *encoding*) *fully-abstract-wrt-equivalences-iff-symcl-source-target-relation-is-preorder*:
fixes *SRel* :: (*'procS* × *'procS*) *set*
and *TRel* :: (*'procT* × *'procT*) *set*
shows (*fully-abstract SRel TRel* ∧ *equivalence TRel*) =
$(\exists Rel.\ (\forall S.\ (SourceTerm\ S,\ TargetTerm\ (\llbracket S \rrbracket)) \in Rel)$
$\wedge\ SRel = \{(S1,\ S2).\ (SourceTerm\ S1,\ SourceTerm\ S2) \in symcl\ Rel\}$
$\wedge\ TRel = \{(T1,\ T2).\ (TargetTerm\ T1,\ TargetTerm\ T2) \in symcl\ Rel\}$
$\wedge\ preorder\ (symcl\ Rel))$
⟨*proof*⟩

lemma (**in** *encoding*) *fully-abstract-iff-symcl-source-target-relation-is-preorder*:
fixes *SRel* :: (*'procS* × *'procS*) *set*
and *TRel* :: (*'procT* × *'procT*) *set*
shows *fully-abstract* $((symcl\ (SRel^{=}))^{+})\ ((symcl\ (TRel^{=}))^{+})$ =
$(\exists Rel.\ (\forall S.\ (SourceTerm\ S,\ TargetTerm\ (\llbracket S \rrbracket)) \in Rel)$
$\wedge\ (symcl\ (SRel^{=}))^{+} = \{(S1,\ S2).\ (SourceTerm\ S1,\ SourceTerm\ S2) \in symcl\ Rel\}$
$\wedge\ (symcl\ (TRel^{=}))^{+} = \{(T1,\ T2).\ (TargetTerm\ T1,\ TargetTerm\ T2) \in symcl\ Rel\}$
$\wedge\ preorder\ (symcl\ Rel))$
⟨*proof*⟩

9.5 Full Abstraction without Relating Translations to their Source Terms

Let Rel be the result of removing from indRelSTEQ all pairs of two source or two target terms that are not contained in SRel or TRel. Then a fully abstract encoding ensures that Rel is trans iff SRel is refl and TRel is trans.

lemma (**in** *encoding*) *full-abstraction-impl-indRelSTEQ-is-trans*:

fixes *SRel* :: $('procS \times 'procS)$ *set*
and *TRel* :: $('procT \times 'procT)$ *set*
and *Rel* :: $(('procS, 'procT)\ Proc \times ('procS, 'procT)\ Proc)$ *set*
assumes *fullAbs*: *fully-abstract SRel TRel*
and *rel*: $Rel = ((indRelSTEQ\ SRel\ TRel)$
$- \{(P, Q).\ (P \in ProcS \wedge Q \in ProcS) \vee (P \in ProcT \wedge Q \in ProcT)\})$
$\cup \{(P, Q).\ (\exists SP\ SQ.\ SP \in S\ P \wedge SQ \in S\ Q \wedge (SP, SQ) \in SRel)$
$\vee (\exists TP\ TQ.\ TP \in T\ P \wedge TQ \in T\ Q \wedge (TP, TQ) \in TRel)\}$
shows (*refl SRel* $\wedge$ *trans TRel*) = *trans Rel*
⟨*proof*⟩

Whenever an encoding induces a trans relation that includes SRel and TRel and relates source terms to their literal translations in both directions, the encoding is fully abstract w.r.t. SRel and TRel.

lemma (**in** *encoding*) *trans-source-target-relation-impl-fully-abstract*:
fixes *Rel* :: $(('procS, 'procT)\ Proc \times ('procS, 'procT)\ Proc)$ *set*
and *SRel* :: $('procS \times 'procS)$ *set*
and *TRel* :: $('procT \times 'procT)$ *set*
assumes *enc*: $\forall S.\ (SourceTerm\ S, TargetTerm\ (\llbracket S \rrbracket)) \in Rel$
$\wedge (TargetTerm\ (\llbracket S \rrbracket), SourceTerm\ S) \in Rel$
and *srel*: $SRel = \{(S1, S2).\ (SourceTerm\ S1, SourceTerm\ S2) \in Rel\}$
and *trel*: $TRel = \{(T1, T2).\ (TargetTerm\ T1, TargetTerm\ T2) \in Rel\}$
and *trans*: *trans Rel*
shows *fully-abstract SRel TRel*
⟨*proof*⟩

Assume TRel is a preorder. Then an encoding is fully abstract w.r.t. SRel and TRel iff there exists a relation that relates add least all source terms to their literal translations, includes SRel and TRel, and whose union with the relation that relates exactly all literal translations to their source terms is trans.

lemma (**in** *encoding*) *source-target-relation-with-trans-impl-full-abstraction*:
fixes *Rel* :: $(('procS, 'procT)\ Proc \times ('procS, 'procT)\ Proc)$ *set*
assumes *enc*: $\forall S.\ (SourceTerm\ S, TargetTerm\ (\llbracket S \rrbracket)) \in Rel$
and *trans*: $trans\ (Rel \cup \{(P, Q).\ \exists S.\ \llbracket S \rrbracket \in T\ P \wedge S \in S\ Q\})$
shows $fully\text{-}abstract\ \{(S1, S2).\ (SourceTerm\ S1, SourceTerm\ S2) \in Rel\}$
$\{(T1, T2).\ (TargetTerm\ T1, TargetTerm\ T2) \in Rel\}$
⟨*proof*⟩

lemma (**in** *encoding*) *fully-abstract-wrt-preorders-iff-source-target-relation-is-transB*:
fixes *SRel* :: $('procS \times 'procS)$ *set*
and *TRel* :: $('procT \times 'procT)$ *set*
assumes *preord*: *preorder TRel*
shows *fully-abstract SRel TRel* =
$(\exists Rel.\ (\forall S.\ (SourceTerm\ S, TargetTerm\ (\llbracket S \rrbracket)) \in Rel)$
$\wedge SRel = \{(S1, S2).\ (SourceTerm\ S1, SourceTerm\ S2) \in Rel\}$
$\wedge TRel = \{(T1, T2).\ (TargetTerm\ T1, TargetTerm\ T2) \in Rel\}$
$\wedge trans\ (Rel \cup \{(P, Q).\ \exists S.\ \llbracket S \rrbracket \in T\ P \wedge S \in S\ Q\}))$
⟨*proof*⟩

The same holds if to obtain transitivity the union may contain additional pairs that do neither relate two source nor two target terms.

lemma (**in** *encoding*) *fully-abstract-wrt-preorders-iff-source-target-relation-union-is-trans*:
fixes *SRel* :: $('procS \times 'procS)$ *set*
and *TRel* :: $('procT \times 'procT)$ *set*
shows (*fully-abstract SRel TRel* $\wedge$ *refl SRel* $\wedge$ *trans TRel*) =
$(\exists Rel.\ (\forall S.\ (SourceTerm\ S, TargetTerm\ (\llbracket S \rrbracket)) \in Rel)$
$\wedge SRel = \{(S1, S2).\ (SourceTerm\ S1, SourceTerm\ S2) \in Rel\}$
$\wedge TRel = \{(T1, T2).\ (TargetTerm\ T1, TargetTerm\ T2) \in Rel\}$
$\wedge (\exists Rel'.\ (\forall (P, Q) \in Rel'.\ P \in ProcS \longleftrightarrow Q \in ProcT)$
$\wedge trans\ (Rel \cup \{(P, Q).\ \exists S.\ \llbracket S \rrbracket \in T\ P \wedge S \in S\ Q\} \cup Rel')))$

⟨*proof*⟩

end
theory *CombinedCriteria*
imports *DivergenceReflection SuccessSensitiveness FullAbstraction OperationalCorrespondence*
begin

10 Combining Criteria

So far we considered the effect of single criteria on encodings. Often the quality of an encoding is prescribed by a set of different criteria. In the following we analyse the combined effect of criteria. This way we can compare criteria as well as identify side effects that result from combinations of criteria. We start with some technical lemmata. To combine the effect of different criteria we combine the conditions they induce. If their effect can be described by a predicate on the pairs of the relation, as in the case of success sensitiveness or divergence reflection, combining the effects is simple.

lemma (**in** *encoding*) *criterion-iff-source-target-relation-impl-indRelR*:
fixes $Cond :: ('procS \Rightarrow 'procT) \Rightarrow bool$
and $Pred :: (('procS, 'procT)\ Proc \times ('procS, 'procT)\ Proc)\ set \Rightarrow bool$
assumes $Cond\ enc = (\exists Rel.\ (\forall S.\ (SourceTerm\ S,\ TargetTerm\ (\llbracket S \rrbracket)) \in Rel) \wedge Pred\ Rel)$
shows $Cond\ enc = (\exists Rel'.\ Pred\ (indRelR \cup Rel'))$
⟨*proof*⟩

lemma (**in** *encoding*) *combine-conditions-on-pairs-of-relations*:
fixes $RelA\ RelB :: (('procS, 'procT)\ Proc \times ('procS, 'procT)\ Proc)\ set$
and $CondA\ CondB :: (('procS, 'procT)\ Proc \times ('procS, 'procT)\ Proc) \Rightarrow bool$
assumes $\forall (P, Q) \in RelA.\ CondA\ (P, Q)$
and $\forall (P, Q) \in RelB.\ CondB\ (P, Q)$
shows $(\forall (P, Q) \in RelA \cap RelB.\ CondA\ (P, Q)) \wedge (\forall (P, Q) \in RelA \cap RelB.\ CondB\ (P, Q))$
⟨*proof*⟩

lemma (**in** *encoding*) *combine-conditions-on-sets-of-relations*:
fixes $Rel\ RelA :: (('procS, 'procT)\ Proc \times ('procS, 'procT)\ Proc)\ set$
and $Cond :: (('procS, 'procT)\ Proc \times ('procS, 'procT)\ Proc)\ set \Rightarrow bool$
and $CondA :: (('procS, 'procT)\ Proc \times ('procS, 'procT)\ Proc) \Rightarrow bool$
assumes $\forall (P, Q) \in RelA.\ CondA\ (P, Q)$
and $Cond\ Rel \wedge Rel \subseteq RelA$
shows $Cond\ Rel \wedge (\forall (P, Q) \in Rel.\ CondA\ (P, Q))$
⟨*proof*⟩

lemma (**in** *encoding*) *combine-conditions-on-sets-and-pairs-of-relations*:
fixes $Rel\ RelA\ RelB :: (('procS, 'procT)\ Proc \times ('procS, 'procT)\ Proc)\ set$
and $Cond :: (('procS, 'procT)\ Proc \times ('procS, 'procT)\ Proc)\ set \Rightarrow bool$
and $CondA\ CondB :: (('procS, 'procT)\ Proc \times ('procS, 'procT)\ Proc) \Rightarrow bool$
assumes $\forall (P, Q) \in RelA.\ CondA\ (P, Q)$
and $\forall (P, Q) \in RelB.\ CondB\ (P, Q)$
and $Cond\ Rel \wedge Rel \subseteq RelA \wedge Rel \subseteq RelB$
shows $Cond\ Rel \wedge (\forall (P, Q) \in Rel.\ CondA\ (P, Q)) \wedge (\forall (P, Q) \in Rel.\ CondB\ (P, Q))$
⟨*proof*⟩

We mapped several criteria on conditions on relations that relate at least all source terms and their literal translations. The following lemmata help us to combine such conditions by switching to the witness indRelR.

lemma (**in** *encoding*) *combine-conditions-on-relations-indRelR*:
fixes $RelA\ RelB :: (('procS, 'procT)\ Proc \times ('procS, 'procT)\ Proc)\ set$
and $Cond :: (('procS, 'procT)\ Proc \times ('procS, 'procT)\ Proc)\ set \Rightarrow bool$
and $CondA\ CondB :: (('procS, 'procT)\ Proc \times ('procS, 'procT)\ Proc) \Rightarrow bool$
assumes $A1: \forall S.\ (SourceTerm\ S,\ TargetTerm\ (\llbracket S \rrbracket)) \in RelA$
and $A2: \forall (P, Q) \in RelA.\ CondA\ (P, Q)$

and *A3*: $\forall S.$ (*SourceTerm S*, *TargetTerm* ($[\![S]\!]$)) $\in$ *RelB*
and *A4*: $\forall$ (*P*, *Q*) $\in$ *RelB*. *CondB* (*P*, *Q*)
shows $\exists$ *Rel*. ($\forall S.$ (*SourceTerm S*, *TargetTerm* ($[\![S]\!]$)) $\in$ *Rel*) $\wedge$ ($\forall$ (*P*, *Q*) $\in$ *Rel*. *CondA* (*P*, *Q*))
$\wedge$ ($\forall$ (*P*, *Q*) $\in$ *Rel*. *CondB* (*P*, *Q*))
and *Cond indRelR* $\Longrightarrow$ ($\exists$ *Rel*. ($\forall S.$ (*SourceTerm S*, *TargetTerm* ($[\![S]\!]$)) $\in$ *Rel*)
$\wedge$ ($\forall$ (*P*, *Q*) $\in$ *Rel*. *CondA* (*P*, *Q*)) $\wedge$ ($\forall$ (*P*, *Q*) $\in$ *Rel*. *CondB* (*P*, *Q*)) $\wedge$ *Cond Rel*)
⟨*proof*⟩

lemma (**in** *encoding*) *indRelR-cond-respects-predA-and-reflects-predB*:
fixes *PredA PredB* :: (*'procS*, *'procT*) *Proc* $\Rightarrow$ *bool*
shows (($\exists$ *Rel*. ($\forall S.$ (*SourceTerm S*, *TargetTerm* ($[\![S]\!]$)) $\in$ *Rel*) $\wedge$ *rel-respects-pred Rel PredA*)
$\wedge$ ($\exists$ *Rel*. ($\forall S.$ (*SourceTerm S*, *TargetTerm* ($[\![S]\!]$)) $\in$ *Rel*) $\wedge$ *rel-reflects-pred Rel PredB*))
= ($\exists$ *Rel*. ($\forall S.$ (*SourceTerm S*, *TargetTerm* ($[\![S]\!]$)) $\in$ *Rel*) $\wedge$ *rel-respects-pred Rel PredA*
$\wedge$ *rel-reflects-pred Rel PredB*)
⟨*proof*⟩

10.1 Divergence Reflection and Success Sensitiveness

We combine results on divergence reflection and success sensitiveness to analyse their combined effect on an encoding function. An encoding is success sensitive and reflects divergence iff there exists a relation that relates source terms and their literal translations that reflects divergence and respects success.

lemma (**in** *encoding-wrt-barbs*) *WSS-DR-iff-source-target-rel*:
fixes *success* :: *'barbs*
shows (*enc-weakly-respects-barb-set* {*success*} $\wedge$ *enc-reflects-divergence*)
= ($\exists$ *Rel*. ($\forall S.$ (*SourceTerm S*, *TargetTerm* ($[\![S]\!]$)) $\in$ *Rel*)
$\wedge$ *rel-weakly-respects-barb-set Rel* (*STCalWB SWB TWB*) {*success*}
$\wedge$ *rel-reflects-divergence Rel* (*STCal Source Target*))
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *SS-DR-iff-source-target-rel*:
fixes *success* :: *'barbs*
shows (*enc-respects-barb-set* {*success*} $\wedge$ *enc-reflects-divergence*)
= ($\exists$ *Rel*. ($\forall S.$ (*SourceTerm S*, *TargetTerm* ($[\![S]\!]$)) $\in$ *Rel*)
$\wedge$ *rel-respects-barb-set Rel* (*STCalWB SWB TWB*) {*success*}
$\wedge$ *rel-reflects-divergence Rel* (*STCal Source Target*))
⟨*proof*⟩

10.2 Adding Operational Correspondence

The effect of operational correspondence includes conditions (TRel is included, transitivity) that require a witness like indRelRTPO. In order to combine operational correspondence with success sensitiveness, we show that if the encoding and TRel (weakly) respects barbs than indRelRTPO (weakly) respects barbs. Since success is only a specific kind of barbs, the same holds for success sensitiveness.

lemma (**in** *encoding-wrt-barbs*) *enc-and-TRel-impl-indRelRTPO-weakly-respects-success*:
fixes *success* :: *'barbs*
and *TRel* :: (*'procT* $\times$ *'procT*) *set*
assumes *encRS*: *enc-weakly-respects-barb-set* {*success*}
and *trelPS*: *rel-weakly-preserves-barb-set TRel TWB* {*success*}
and *trelRS*: *rel-weakly-reflects-barb-set TRel TWB* {*success*}
shows *rel-weakly-respects-barb-set* (*indRelRTPO TRel*) (*STCalWB SWB TWB*) {*success*}
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *enc-and-TRel-impl-indRelRTPO-weakly-respects-barbs*:
fixes *TRel* :: (*'procT* $\times$ *'procT*) *set*
assumes *encRS*: *enc-weakly-respects-barbs*
and *trelPS*: *rel-weakly-preserves-barbs TRel TWB*
and *trelRS*: *rel-weakly-reflects-barbs TRel TWB*
shows *rel-weakly-respects-barbs* (*indRelRTPO TRel*) (*STCalWB SWB TWB*)

⟨proof⟩

lemma (**in** *encoding-wrt-barbs*) *enc-and-TRel-impl-indRelRTPO-respects-success*:
  fixes *success* :: *'barbs*
    and *TRel* :: (*'procT* × *'procT*) *set*
  assumes *encRS*: *enc-respects-barb-set* {*success*}
      and *trelPS*: *rel-preserves-barb-set TRel TWB* {*success*}
      and *trelRS*: *rel-reflects-barb-set TRel TWB* {*success*}
  shows *rel-respects-barb-set* (*indRelRTPO TRel*) (*STCalWB SWB TWB*) {*success*}
⟨proof⟩

lemma (**in** *encoding-wrt-barbs*) *enc-and-TRel-impl-indRelRTPO-respects-barbs*:
  fixes *TRel* :: (*'procT* × *'procT*) *set*
  assumes *encRS*: *enc-respects-barbs*
      and *trelPS*: *rel-preserves-barbs TRel TWB*
      and *trelRS*: *rel-reflects-barbs TRel TWB*
  shows *rel-respects-barbs* (*indRelRTPO TRel*) (*STCalWB SWB TWB*)
⟨proof⟩

An encoding is success sensitive and operational corresponding w.r.t. a bisimulation TRel that respects success iff there exists a bisimultion that includes TRel and respects success. The same holds if we consider not only success sensitiveness but barb sensitiveness in general.

lemma (**in** *encoding-wrt-barbs*) *OC-SS-iff-source-target-rel*:
  fixes *success* :: *'barbs*
    and *TRel* :: (*'procT* × *'procT*) *set*
  shows (*operational-corresponding* ($TRel^*$)
        ∧ *weak-reduction-bisimulation* ($TRel^+$) *Target*
        ∧ *enc-weakly-respects-barb-set* {*success*}
        ∧ *rel-weakly-respects-barb-set TRel TWB* {*success*})
        = (∃ *Rel*. (∀ *S*. (*SourceTerm S*, *TargetTerm* (⟦*S*⟧)) ∈ *Rel*)
          ∧ (∀ *T1 T2*. (*T1*, *T2*) ∈ *TRel* ⟶ (*TargetTerm T1*, *TargetTerm T2*) ∈ *Rel*)
          ∧ (∀ *T1 T2*. (*TargetTerm T1*, *TargetTerm T2*) ∈ *Rel* ⟶ (*T1*, *T2*) ∈ $TRel^+$)
          ∧ (∀ *S T*. (*SourceTerm S*, *TargetTerm T*) ∈ *Rel* ⟶ (⟦*S*⟧, *T*) ∈ $TRel^*$)
          ∧ *weak-reduction-bisimulation Rel* (*STCal Source Target*)
          ∧ *rel-weakly-respects-barb-set Rel* (*STCalWB SWB TWB*) {*success*})
⟨proof⟩

lemma (**in** *encoding-wrt-barbs*) *OC-SS-RB-iff-source-target-rel*:
  fixes *success* :: *'barbs*
    and *TRel* :: (*'procT* × *'procT*) *set*
  shows (*operational-corresponding* ($TRel^*$)
        ∧ *weak-reduction-bisimulation* ($TRel^+$) *Target*
        ∧ *enc-weakly-respects-barbs* ∧ *enc-weakly-respects-barb-set* {*success*}
        ∧ *rel-weakly-respects-barbs TRel TWB* ∧ *rel-weakly-respects-barb-set TRel TWB* {*success*})
        = (∃ *Rel*. (∀ *S*. (*SourceTerm S*, *TargetTerm* (⟦*S*⟧)) ∈ *Rel*)
          ∧ (∀ *T1 T2*. (*T1*, *T2*) ∈ *TRel* ⟶ (*TargetTerm T1*, *TargetTerm T2*) ∈ *Rel*)
          ∧ (∀ *T1 T2*. (*TargetTerm T1*, *TargetTerm T2*) ∈ *Rel* ⟶ (*T1*, *T2*) ∈ $TRel^+$)
          ∧ (∀ *S T*. (*SourceTerm S*, *TargetTerm T*) ∈ *Rel* ⟶ (⟦*S*⟧, *T*) ∈ $TRel^*$)
          ∧ *weak-reduction-bisimulation Rel* (*STCal Source Target*)
          ∧ *rel-weakly-respects-barbs Rel* (*STCalWB SWB TWB*)
          ∧ *rel-weakly-respects-barb-set Rel* (*STCalWB SWB TWB*) {*success*})
⟨proof⟩

lemma (**in** *encoding-wrt-barbs*) *OC-SS-wrt-preorder-iff-source-target-rel*:
  fixes *success* :: *'barbs*
    and *TRel* :: (*'procT* × *'procT*) *set*
  shows (*operational-corresponding TRel* ∧ *preorder TRel* ∧ *weak-reduction-bisimulation TRel Target*
        ∧ *enc-weakly-respects-barb-set* {*success*}
        ∧ *rel-weakly-respects-barb-set TRel TWB* {*success*})
        = (∃ *Rel*. (∀ *S*. (*SourceTerm S*, *TargetTerm* (⟦*S*⟧)) ∈ *Rel*)

$$\wedge\ TRel = \{(T1,\ T2).\ (TargetTerm\ T1,\ TargetTerm\ T2) \in Rel\}$$
$$\wedge\ (\forall\, S\ T.\ (SourceTerm\ S,\ TargetTerm\ T) \in Rel \longrightarrow ([\![S]\!],\ T) \in TRel)$$
$$\wedge\ weak\text{-}reduction\text{-}bisimulation\ Rel\ (STCal\ Source\ Target) \wedge preorder\ Rel$$
$$\wedge\ rel\text{-}weakly\text{-}respects\text{-}barb\text{-}set\ Rel\ (STCalWB\ SWB\ TWB)\ \{success\})$$
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *OC-SS-RB-wrt-preorder-iff-source-target-rel*:
fixes *success* :: *'barbs*
and *TRel* :: *('procT × 'procT) set*
shows $(operational\text{-}corresponding\ TRel \wedge preorder\ TRel \wedge weak\text{-}reduction\text{-}bisimulation\ TRel\ Target$
$\wedge\ enc\text{-}weakly\text{-}respects\text{-}barbs \wedge rel\text{-}weakly\text{-}respects\text{-}barbs\ TRel\ TWB$
$\wedge\ enc\text{-}weakly\text{-}respects\text{-}barb\text{-}set\ \{success\}$
$\wedge\ rel\text{-}weakly\text{-}respects\text{-}barb\text{-}set\ TRel\ TWB\ \{success\})$
$= (\exists\, Rel.\ (\forall\, S.\ (SourceTerm\ S,\ TargetTerm\ ([\![S]\!])) \in Rel)$
$\wedge\ TRel = \{(T1,\ T2).\ (TargetTerm\ T1,\ TargetTerm\ T2) \in Rel\}$
$\wedge\ (\forall\, S\ T.\ (SourceTerm\ S,\ TargetTerm\ T) \in Rel \longrightarrow ([\![S]\!],\ T) \in TRel)$
$\wedge\ weak\text{-}reduction\text{-}bisimulation\ Rel\ (STCal\ Source\ Target) \wedge preorder\ Rel$
$\wedge\ rel\text{-}weakly\text{-}respects\text{-}barbs\ Rel\ (STCalWB\ SWB\ TWB)$
$\wedge\ rel\text{-}weakly\text{-}respects\text{-}barb\text{-}set\ Rel\ (STCalWB\ SWB\ TWB)\ \{success\})$
⟨*proof*⟩

An encoding is success sensitive and weakly operational corresponding w.r.t. a correspondence simulation TRel that respects success iff there exists a correspondence simultion that includes TRel and respects success. The same holds if we consider not only success sensitiveness but barb sensitiveness in general.

lemma (**in** *encoding-wrt-barbs*) *WOC-SS-wrt-preorder-iff-source-target-rel*:
fixes *success* :: *'barbs*
and *TRel* :: *('procT × 'procT) set*
shows $(weakly\text{-}operational\text{-}corresponding\ TRel \wedge preorder\ TRel$
$\wedge\ weak\text{-}reduction\text{-}correspondence\text{-}simulation\ TRel\ Target$
$\wedge\ enc\text{-}weakly\text{-}respects\text{-}barb\text{-}set\ \{success\}$
$\wedge\ rel\text{-}weakly\text{-}respects\text{-}barb\text{-}set\ TRel\ TWB\ \{success\})$
$= (\exists\, Rel.\ (\forall\, S.\ (SourceTerm\ S,\ TargetTerm\ ([\![S]\!])) \in Rel)$
$\wedge\ TRel = \{(T1,\ T2).\ (TargetTerm\ T1,\ TargetTerm\ T2) \in Rel\}$
$\wedge\ (\forall\, S\ T.\ (SourceTerm\ S,\ TargetTerm\ T) \in Rel \longrightarrow ([\![S]\!],\ T) \in TRel)$
$\wedge\ weak\text{-}reduction\text{-}correspondence\text{-}simulation\ Rel\ (STCal\ Source\ Target) \wedge preorder\ Rel$
$\wedge\ rel\text{-}weakly\text{-}respects\text{-}barb\text{-}set\ Rel\ (STCalWB\ SWB\ TWB)\ \{success\})$
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *WOC-SS-RB-wrt-preorder-iff-source-target-rel*:
fixes *success* :: *'barbs*
and *TRel* :: *('procT × 'procT) set*
shows $(weakly\text{-}operational\text{-}corresponding\ TRel \wedge preorder\ TRel$
$\wedge\ weak\text{-}reduction\text{-}correspondence\text{-}simulation\ TRel\ Target$
$\wedge\ enc\text{-}weakly\text{-}respects\text{-}barbs \wedge enc\text{-}weakly\text{-}respects\text{-}barb\text{-}set\ \{success\}$
$\wedge\ rel\text{-}weakly\text{-}respects\text{-}barbs\ TRel\ TWB \wedge rel\text{-}weakly\text{-}respects\text{-}barb\text{-}set\ TRel\ TWB\ \{success\})$
$= (\exists\, Rel.\ (\forall\, S.\ (SourceTerm\ S,\ TargetTerm\ ([\![S]\!])) \in Rel)$
$\wedge\ TRel = \{(T1,\ T2).\ (TargetTerm\ T1,\ TargetTerm\ T2) \in Rel\}$
$\wedge\ (\forall\, S\ T.\ (SourceTerm\ S,\ TargetTerm\ T) \in Rel \longrightarrow ([\![S]\!],\ T) \in TRel)$
$\wedge\ weak\text{-}reduction\text{-}correspondence\text{-}simulation\ Rel\ (STCal\ Source\ Target) \wedge preorder\ Rel$
$\wedge\ rel\text{-}weakly\text{-}respects\text{-}barbs\ Rel\ (STCalWB\ SWB\ TWB)$
$\wedge\ rel\text{-}weakly\text{-}respects\text{-}barb\text{-}set\ Rel\ (STCalWB\ SWB\ TWB)\ \{success\})$
⟨*proof*⟩

An encoding is strongly success sensitive and strongly operational corresponding w.r.t. a strong bisimulation TRel that strongly respects success iff there exists a strong bisimultion that includes TRel and strongly respects success. The same holds if we consider not only strong success sensitiveness but strong barb sensitiveness in general.

lemma (**in** *encoding-wrt-barbs*) *SOC-SS-wrt-preorder-iff-source-target-rel*:
fixes *success* :: *'barbs*

and *TRel* :: $('procT \times 'procT)$ *set*
shows (*strongly-operational-corresponding TRel* ∧ *preorder TRel*
∧ *strong-reduction-bisimulation TRel Target*
∧ *enc-respects-barb-set* {*success*} ∧ *rel-respects-barb-set TRel TWB* {*success*})
= (∃ *Rel*. (∀ *S*. (*SourceTerm S*, *TargetTerm* (⟦*S*⟧)) ∈ *Rel*)
∧ *TRel* = {(*T1*, *T2*). (*TargetTerm T1*, *TargetTerm T2*) ∈ *Rel*}
∧ (∀ *S T*. (*SourceTerm S*, *TargetTerm T*) ∈ *Rel* ⟶ (⟦*S*⟧, *T*) ∈ *TRel*)
∧ *strong-reduction-bisimulation Rel* (*STCal Source Target*) ∧ *preorder Rel*
∧ *rel-respects-barb-set Rel* (*STCalWB SWB TWB*) {*success*})
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *SOC-SS-RB-wrt-preorder-iff-source-target-rel*:
fixes *success* :: *'barbs*
and *TRel* :: $('procT \times 'procT)$ *set*
shows (*strongly-operational-corresponding TRel* ∧ *preorder TRel*
∧ *strong-reduction-bisimulation TRel Target*
∧ *enc-respects-barbs* ∧ *rel-respects-barbs TRel TWB*
∧ *enc-respects-barb-set* {*success*} ∧ *rel-respects-barb-set TRel TWB* {*success*})
= (∃ *Rel*. (∀ *S*. (*SourceTerm S*, *TargetTerm* (⟦*S*⟧)) ∈ *Rel*)
∧ *TRel* = {(*T1*, *T2*). (*TargetTerm T1*, *TargetTerm T2*) ∈ *Rel*}
∧ (∀ *S T*. (*SourceTerm S*, *TargetTerm T*) ∈ *Rel* ⟶ (⟦*S*⟧, *T*) ∈ *TRel*)
∧ *strong-reduction-bisimulation Rel* (*STCal Source Target*) ∧ *preorder Rel*
∧ *rel-respects-barbs Rel* (*STCalWB SWB TWB*)
∧ *rel-respects-barb-set Rel* (*STCalWB SWB TWB*) {*success*})
⟨*proof*⟩

Next we also add divergence reflection to operational correspondence and success sensitiveness.

lemma (**in** *encoding*) *enc-and-TRelimpl-indRelRTPO-reflect-divergence*:
fixes *TRel* :: $('procT \times 'procT)$ *set*
assumes *encRD*: *enc-reflects-divergence*
and *trelRD*: *rel-reflects-divergence TRel Target*
shows *rel-reflects-divergence* (*indRelRTPO TRel*) (*STCal Source Target*)
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *OC-SS-DR-iff-source-target-rel*:
fixes *success* :: *'barbs*
and *TRel* :: $('procT \times 'procT)$ *set*
shows (*operational-corresponding* ($TRel^*$)
∧ *weak-reduction-bisimulation* ($TRel^+$) *Target*
∧ *enc-weakly-respects-barb-set* {*success*}
∧ *rel-weakly-respects-barb-set TRel TWB* {*success*}
∧ *enc-reflects-divergence* ∧ *rel-reflects-divergence TRel Target*)
= (∃ *Rel*. (∀ *S*. (*SourceTerm S*, *TargetTerm* (⟦*S*⟧)) ∈ *Rel*)
∧ (∀ *T1 T2*. (*T1*, *T2*) ∈ *TRel* ⟶ (*TargetTerm T1*, *TargetTerm T2*) ∈ *Rel*)
∧ (∀ *T1 T2*. (*TargetTerm T1*, *TargetTerm T2*) ∈ *Rel* ⟶ (*T1*, *T2*) ∈ $TRel^+$)
∧ (∀ *S T*. (*SourceTerm S*, *TargetTerm T*) ∈ *Rel* ⟶ (⟦*S*⟧, *T*) ∈ $TRel^*$)
∧ *weak-reduction-bisimulation Rel* (*STCal Source Target*)
∧ *rel-weakly-respects-barb-set Rel* (*STCalWB SWB TWB*) {*success*}
∧ *rel-reflects-divergence Rel* (*STCal Source Target*))
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *WOC-SS-DR-wrt-preorder-iff-source-target-rel*:
fixes *success* :: *'barbs*
and *TRel* :: $('procT \times 'procT)$ *set*
shows (*weakly-operational-corresponding TRel* ∧ *preorder TRel*
∧ *weak-reduction-correspondence-simulation TRel Target*
∧ *enc-weakly-respects-barb-set* {*success*}
∧ *rel-weakly-respects-barb-set TRel TWB* {*success*}
∧ *enc-reflects-divergence* ∧ *rel-reflects-divergence TRel Target*)
= (∃ *Rel*. (∀ *S*. (*SourceTerm S*, *TargetTerm* (⟦*S*⟧)) ∈ *Rel*)
∧ *TRel* = {(*T1*, *T2*). (*TargetTerm T1*, *TargetTerm T2*) ∈ *Rel*}

∧ (∀ *S T*. (*SourceTerm S*, *TargetTerm T*) ∈ *Rel* ⟶ (⟦*S*⟧, *T*) ∈ *TRel*)
∧ *weak-reduction-correspondence-simulation Rel* (*STCal Source Target*) ∧ *preorder Rel*
∧ *rel-weakly-respects-barb-set Rel* (*STCalWB SWB TWB*) {*success*}
∧ *rel-reflects-divergence Rel* (*STCal Source Target*))
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *OC-SS-DR-wrt-preorder-iff-source-target-rel*:
fixes *success* :: *'barbs*
and *TRel* :: (*'procT* × *'procT*) *set*
shows (*operational-corresponding TRel* ∧ *preorder TRel* ∧ *weak-reduction-bisimulation TRel Target*
∧ *enc-weakly-respects-barb-set* {*success*}
∧ *rel-weakly-respects-barb-set TRel TWB* {*success*}
∧ *enc-reflects-divergence* ∧ *rel-reflects-divergence TRel Target*)
= (∃ *Rel*. (∀ *S*. (*SourceTerm S*, *TargetTerm* (⟦*S*⟧)) ∈ *Rel*)
∧ *TRel* = {(*T1*, *T2*). (*TargetTerm T1*, *TargetTerm T2*) ∈ *Rel*}
∧ (∀ *S T*. (*SourceTerm S*, *TargetTerm T*) ∈ *Rel* ⟶ (⟦*S*⟧, *T*) ∈ *TRel*)
∧ *weak-reduction-bisimulation Rel* (*STCal Source Target*) ∧ *preorder Rel*
∧ *rel-weakly-respects-barb-set Rel* (*STCalWB SWB TWB*) {*success*}
∧ *rel-reflects-divergence Rel* (*STCal Source Target*))
⟨*proof*⟩

lemma (**in** *encoding-wrt-barbs*) *SOC-SS-DR-wrt-preorder-iff-source-target-rel*:
fixes *success* :: *'barbs*
and *TRel* :: (*'procT* × *'procT*) *set*
shows (*strongly-operational-corresponding TRel* ∧ *preorder TRel*
∧ *strong-reduction-bisimulation TRel Target*
∧ *enc-respects-barb-set* {*success*} ∧ *rel-respects-barb-set TRel TWB* {*success*}
∧ *enc-reflects-divergence* ∧ *rel-reflects-divergence TRel Target*)
= (∃ *Rel*. (∀ *S*. (*SourceTerm S*, *TargetTerm* (⟦*S*⟧)) ∈ *Rel*)
∧ *TRel* = {(*T1*, *T2*). (*TargetTerm T1*, *TargetTerm T2*) ∈ *Rel*}
∧ (∀ *S T*. (*SourceTerm S*, *TargetTerm T*) ∈ *Rel* ⟶ (⟦*S*⟧, *T*) ∈ *TRel*)
∧ *strong-reduction-bisimulation Rel* (*STCal Source Target*) ∧ *preorder Rel*
∧ *rel-respects-barb-set Rel* (*STCalWB SWB TWB*) {*success*}
∧ *rel-reflects-divergence Rel* (*STCal Source Target*))
⟨*proof*⟩

10.3 Full Abstraction and Operational Correspondence

To combine full abstraction and operational correspondence we consider a symmetric version of the induced relation and assume that the relations SRel and TRel are equivalences. Then an encoding is fully abstract w.r.t. SRel and TRel and operationally corresponding w.r.t. TRel such that TRel is a bisimulation iff the induced relation contains both SRel and TRel and is a transitive bisimulation.

lemma (**in** *encoding*) *FS-OC-modulo-equivalences-iff-source-target-relation*:
fixes *SRel* :: (*'procS* × *'procS*) *set*
and *TRel* :: (*'procT* × *'procT*) *set*
assumes *eqS*: *equivalence SRel*
and *eqT*: *equivalence TRel*
shows *fully-abstract SRel TRel*
∧ *operational-corresponding TRel* ∧ *weak-reduction-bisimulation TRel Target*
⟷ (∃ *Rel*.
(∀ *S*. (*SourceTerm S*, *TargetTerm* (⟦*S*⟧)) ∈ *Rel* ∧ (*TargetTerm* (⟦*S*⟧), *SourceTerm S*) ∈ *Rel*)
∧ *SRel* = {(*S1*, *S2*). (*SourceTerm S1*, *SourceTerm S2*) ∈ *Rel*}
∧ *TRel* = {(*T1*, *T2*). (*TargetTerm T1*, *TargetTerm T2*) ∈ *Rel*}
∧ *trans Rel* ∧ *weak-reduction-bisimulation Rel* (*STCal Source Target*))
⟨*proof*⟩

lemma (**in** *encoding*) *FA-SOC-modulo-equivalences-iff-source-target-relation*:
fixes *SRel* :: (*'procS* × *'procS*) *set*
and *TRel* :: (*'procT* × *'procT*) *set*
assumes *eqS*: *equivalence SRel*

and *eqT*: *equivalence TRel*
shows *fully-abstract SRel TRel* $\wedge$ *strongly-operational-corresponding TRel*
$\wedge$ *strong-reduction-bisimulation TRel Target* $\longleftrightarrow$ ($\exists$ *Rel*.
($\forall$ *S*. (*SourceTerm S*, *TargetTerm* ($[\![S]\!]$)) $\in$ *Rel* $\wedge$ (*TargetTerm* ($[\![S]\!]$), *SourceTerm S*) $\in$ *Rel*)
$\wedge$ *SRel* = {(*S1*, *S2*). (*SourceTerm S1*, *SourceTerm S2*) $\in$ *Rel*}
$\wedge$ *TRel* = {(*T1*, *T2*). (*TargetTerm T1*, *TargetTerm T2*) $\in$ *Rel*} $\wedge$ *trans Rel*
$\wedge$ *strong-reduction-bisimulation Rel* (*STCal Source Target*))
⟨*proof*⟩

An encoding that is fully abstract w.r.t. the equivalences SRel and TRel and operationally corresponding w.r.t. TRel ensures that SRel is a bisimulation iff TRel is a bisimulation.

lemma (**in** *encoding*) *FA-and-OC-and-TRel-impl-SRel-bisimulation*:
fixes *SRel* :: (*'procS* $\times$ *'procS*) *set*
and *TRel* :: (*'procT* $\times$ *'procT*) *set*
assumes *fullAbs*: *fully-abstract SRel TRel*
and *opCom*: *operational-complete TRel*
and *opSou*: *operational-sound TRel*
and *symmT*: *sym TRel*
and *transT*: *trans TRel*
and *bisimT*: *weak-reduction-bisimulation TRel Target*
shows *weak-reduction-bisimulation SRel Source*
⟨*proof*⟩

lemma (**in** *encoding*) *FA-and-SOC-and-TRel-impl-SRel-strong-bisimulation*:
fixes *SRel* :: (*'procS* $\times$ *'procS*) *set*
and *TRel* :: (*'procT* $\times$ *'procT*) *set*
assumes *fullAbs*: *fully-abstract SRel TRel*
and *opCom*: *strongly-operational-complete TRel*
and *opSou*: *strongly-operational-sound TRel*
and *symmT*: *sym TRel*
and *transT*: *trans TRel*
and *bisimT*: *strong-reduction-bisimulation TRel Target*
shows *strong-reduction-bisimulation SRel Source*
⟨*proof*⟩

lemma (**in** *encoding*) *FA-and-OC-impl-SRel-iff-TRel-bisimulation*:
fixes *SRel* :: (*'procS* $\times$ *'procS*) *set*
and *TRel* :: (*'procT* $\times$ *'procT*) *set*
assumes *fullAbs*: *fully-abstract SRel TRel*
and *opCor*: *operational-corresponding TRel*
and *symmT*: *sym TRel*
and *transT*: *trans TRel*
and *surj*: $\forall$ *T*. $\exists$ *S*. *T* = $[\![S]\!]$
shows *weak-reduction-bisimulation SRel Source* $\longleftrightarrow$ *weak-reduction-bisimulation TRel Target*
⟨*proof*⟩

end