Equating Simulation with Refinement

He Jifeng, Tony Hoare, Cedric Fournet, Paul Gardiner,
Sriram Rajamani, Jakob Rehof, Bill Roscoe

November, 2003

Summary. Two-thirds simulation is equivalent to failures refinement if two redundant transition rules are added to the operational semantics of CCS.

1 Introduction

In the study of CCS [10, 11] and related process calculi, there are two standard approaches to the definition of similarity or equivalence of processes. The first is bisimulation, an equivalence relation based on the structural operational semantics of the calculus. The second is refinement, an ordering defined as inclusion of the sets of observations that may be made of the behaviour of each process. The original forms of bisimulation (strong and weak) were intended to give the strongest reasonable definition of process expressions. The original forms of refinement (traces and failures) were intended to make mutual refinement the weakest definition of equivalence. There are many other definitions of equivalence that lie between these two extremes. The selection between them is often made to match the needs of each particular application.

This paper shows how to define a process calculus in which the concepts of simulation and mutual refinement coincide with each other : thus the combined benefits of two approaches are available uniformly to all applications. The calculus is identical with CCS, except for a few additional transition rules. Each additional rule can be interpreted as permission (but not a compulsion) for an implementation to perform a transition that is to the benefit of the user, for example, enabling the user to avoid deadlock. It is shown that two-thirds simulation is a valid asymmetric form of bisimulation, and corresponds exactly to failures refinement.

The technical results of this paper, together with a treatment of divergence refinement, are reported more fully in [7].

2 CCS Summary

Following the CCS [10, 11], we adopt an infinite set $\mathcal{N}$ of *names*, and the set $\bar{\mathcal{N}} = \{\bar{a} \mid a \in \mathcal{N}\}$ of *co-names*. Let $\bar{\bar{a}}$ to stand for action a. It is assumed that $\mathcal{N}$ and $\bar{\mathcal{N}}$ are disjoint, and their union $\mathcal{L}$ comprises the labels representing *observable* actions. The *unobservable* action is denoted by τ, which, as an internal action, has no complement. The full class of actions, both observable and internal, consists of $Act \stackrel{def}{=} \mathcal{L} \cup \{\tau\}$. We denote members of Act by α and members of $\mathcal{L}$ by λ.

We need a little notation. We shall write $\vec{a}$ for a sequence $a_1, .., a_n$ of names. If $\vec{a}$ and $\vec{b}$ are name sequences of length n where all elements of $\vec{a}$ are distinct, and P is a process expression, then $\{\vec{b}/\vec{a}\}P$ means the result of replacing a_i by b_i in P $(1 \leq i \leq n)$. The set of *free* names which occur in P is denoted by $fn(P)$. For any subset X of $\mathcal{L}$, define

$\bar{X} \stackrel{def}{=} \{\bar{a} \mid a \in X\}$ and $X^\tau \stackrel{def}{=} X \cup \{\tau\}$.

Definition 2.1 (Process expression)

The set $\mathcal{P}$ of process expressions is defined by the following syntax:

$$P ::= \Sigma_{i\in I}\alpha_i.P_i \mid A < a_1, .., a_n > \mid (P_1|P_2) \mid \mathtt{new}\, a\, P$$

where $\alpha_i \in Act$ and I is any *finite indexing set.* (In [10], I can also be an infinite set.) If $I = \emptyset$ then $\Sigma_{i\in I}\alpha_i.P_i$ is the empty sum, written 0. We shall often use M, N to stand for summations. We assume that every process identifier A has a *defining equation* of the form $A(\vec{a}) \stackrel{def}{=} P_A$ where P_A is a summation, and the names $\vec{a} = a_1, .., a_n$ (all distinct) include all the free names of P_A. In general P_A may contain process identifiers, including A. Any resulting recursion is guaranteed syntactically to be guarded. The notation $(P_1|P_2)$ stands for the parallel composition of P_1 and P_2, and the restriction $\mathtt{new}\, a\, P$ internalises both a and $\bar{a}$ by preventing either of them from happening.

The following labelled transition system for concurrent processes was defined in [11].

Definition 2.2 (The LTS of concurrent processes)

The labelled transition system $(\mathcal{P}, \mathcal{T})$ of concurrent processes over the action set Act has all process expressions $\mathcal{P}$ as its states, and its transitions $\mathcal{T}$ are exactly those which can be inferred from the rules listed below, together with alpha-conversion.

$$SUM : M + \alpha.P + N \stackrel{\alpha}{\rightarrow} P$$

$$REACT : \frac{P \stackrel{\lambda}{\rightarrow} P' \quad Q \stackrel{\bar{\lambda}}{\rightarrow} Q'}{P|Q \stackrel{\tau}{\rightarrow} P'|Q'}$$

$$L-PAR : \frac{P \stackrel{\alpha}{\rightarrow} P'}{P|Q \stackrel{\alpha}{\rightarrow} P'|Q} \qquad R-PAR : \frac{Q \stackrel{\alpha}{\rightarrow} Q'}{P|Q \stackrel{\alpha}{\rightarrow} P|Q'}$$

$$RES : \frac{P \stackrel{\alpha}{\rightarrow} P'}{\mathtt{new}\, a\, P \stackrel{\alpha}{\rightarrow} \mathtt{new}\, a\, P'} \quad if\ \alpha \notin \{a, \bar{a}\}$$

$$IDENT : \frac{\{\vec{b}/\vec{a}\}P_A \stackrel{\alpha}{\rightarrow} P'}{A < \vec{b} > \stackrel{\alpha}{\rightarrow} P'} \quad if\ A(\vec{a}) \stackrel{def}{=} P_A$$

In the rules $REACT$, $L-PAR$, $R-PAR$ and RES a transition of a composite process is inferred from transitions of its components. SUM is the basic rule to which all transitions are ultimately traced.

The following properties of transitions given in [11] will be used later. The first says that the branching of a process behaviour is finite; the second says that a process can only use the names which it already contains.

Lemma 2.3

(1) Given P, there are only finitely many transitions $P \stackrel{\alpha}{\rightarrow} P'$.

(2) If $P \stackrel{\alpha}{\rightarrow} P'$ then $fn(P') \cup \{\alpha\} \subseteq fn(P)$. □

3 Failures refinement

A process P can perform both internal actions and observable actions. We use $\mathtt{init}(P)$ to denote the set of actions in which P can engage at the very beginning.

$$\mathtt{init}(P) \stackrel{def}{=} \{\alpha \mid \exists P' \bullet P \stackrel{\alpha}{\rightarrow} P'\}$$

Lemma 3.1
(1) $\mathbf{init}(\Sigma_{i\in I}\alpha_i.P_i) = \{\alpha_i \mid i \in I\}$
(2) $\mathbf{init}(P|Q) = \mathbf{init}(P) \cup \mathbf{init}(Q) \cup \{\tau \mid \exists\lambda \bullet \lambda \in \mathbf{init}(P) \wedge \bar{\lambda} \in \mathbf{init}(Q)\}$
(3) $\mathbf{init}(\mathbf{new}\, a\, P) = \mathbf{init}(P) \setminus \{a, \bar{a}\}$
(4) $\mathbf{init}(A\langle\vec{b}\rangle) = \mathbf{init}(\{\vec{b}/\vec{a}\}P_A)$ □

Definition 3.2 (stability)
A process P is stable if $\tau \notin \mathbf{init}(P)$. □

Lemma 3.3
(1) $\Sigma_{i\in I}\alpha_i.P_i$ is stable iff $a_i \neq \tau$ for all $i \in I$.
(2) $P|Q$ is stable iff both P and Q are stable, and $\mathbf{init}(P) \cap \{\bar{a} \mid a \in \mathbf{init}(Q)\} = \emptyset$.
(3) $(\mathbf{new}\, a\, P)$ is stable iff P is stable,
(4) $A\langle\vec{b}\rangle$ is stable iff $\{\vec{b}/\vec{a}\}P_A$ is stable.

Definition 3.4 (refusal)
Let X be a subset of $\mathcal{L}$, and $\xrightarrow{\tau}{}^*$ the reflexive and transitive closure of $\xrightarrow{\tau}$. Then

$$P \text{ can refuse } X \stackrel{def}{=} \exists Q \bullet P \xrightarrow{\tau}{}^* Q \wedge \mathbf{init}(Q) \cap \bar{X}^\tau = \emptyset$$ □

Lemma 3.5 (subset closure of refusals)
If P can refuse X then it can also refuse every subset of X. □

Theorem 3.6 (refusals of process expressions)
(1) $\Sigma_{i\in I}\lambda_i.P_i$ can refuse X iff $\bar{\lambda}_i \notin X$ for all $i \in I$.
(2) $\Sigma_{i\in I}\lambda_i.P_i + \Sigma_{j\in J}\tau.Q_j$ can refuse X iff Q_j can refuse X for some $j \in J$.
(3) $\mathbf{new}\, a\, P$ can refuse X iff P can refuse $X \setminus \{a, \bar{a}\}$.
(4) $A\langle\vec{b}\rangle$ can refuse X iff $\{\vec{b}/\vec{a}\}P_A$ does so. □

We will come back to the refusals of parallel processes after introduction of the notions of traces and failures for process expressions.

Definition 3.7 (traces and failures)
Let $\stackrel{\lambda}{\Rightarrow} \stackrel{def}{=} \xrightarrow{\tau}{}^* \circ \xrightarrow{\lambda}$ for $\lambda \in \mathcal{L}$. Define

$$traces(P) \stackrel{def}{=} \{<>\} \cup \{<\lambda> s \mid \exists Q \bullet P \stackrel{\lambda}{\Rightarrow} Q \wedge s \in traces(Q)\}$$

$$failures(P) \stackrel{def}{=} \{(<>, X) \mid P \text{ can refuse } X\} \cup \{(<\lambda> s, X) \mid \exists Q \bullet P \stackrel{\lambda}{\Rightarrow} Q \wedge (s, X) \in failures(Q)\}$$

P is a *failures refinement* [6] of Q if

$$traces(P) \subseteq traces(Q) \text{ and } failures(P) \subseteq failures(Q)$$ □

Remark: In general, $traces(P) \neq \{s \mid \exists X \bullet (s, X) \in failures(P)\}$. For example, let **top** be defined by the defining equation $\mathbf{top} \stackrel{def}{=} \tau.\mathbf{top}$, then

$$traces(\mathbf{top}) = \{\langle\rangle\} \qquad failures(\mathbf{top}) = \emptyset$$ □

Theorem 3.8 (refusals of parallel composition)
$(P|Q)$ can refuse X iff there exist $(s, X_P) \in failures(P)$ and $(\bar{s}, X_Q) \in failures(Q)$ such that $X \subseteq (X_P \cap X_Q)$ and $\forall\lambda \in \mathcal{L} \bullet (\lambda \in X_P \vee \bar{\lambda} \in X_Q)$ □

Theorem 3.9 (τ-transition implies failures refinement)
If $P \xrightarrow{\tau} P'$, then P' is a failures refinement of P. □

The failures of a summation can be calculated as follows.

Theorem 3.10
$failures(\Sigma_{i\in I}\alpha_i.P_i) = \bigcup_{\alpha_i=\tau} failures(P_i) \cup$
$\bigcup_{\alpha_j\neq\tau}(\alpha_j)\cdot failures(P_j) \cup \{(\langle\rangle, X) \mid \bar{X}^\tau \cap \{\alpha_i \| i \in I\} = \emptyset\}$
where $\langle\lambda\rangle \cdot failures(P)$ denotes the set $\{\langle\lambda\rangle s, X) \mid (s, X) \in failures(P)\}$ □

Let P be a process and $\lambda \in \mathcal{L}$. From Lemma 2.3 we can obtain the following finite sets:

$$\{Q1, .., Qn\} \stackrel{def}{=} \{Q \mid P \stackrel{\lambda}{\rightarrow} Q\}$$
$$\{R1, ..., Rm\} \stackrel{def}{=} \{R \mid P \stackrel{\tau}{\rightarrow} R \wedge R \stackrel{\lambda}{\Rightarrow}\}$$

When $P \stackrel{\lambda}{\Rightarrow}$, we have $n + m > 0$.

Definition 3.11 (Derivative)
Let P be a process and $\lambda \in \mathcal{L}$. Then the process P_λ is called the λ-derivative of P, whose defining equation can be presented syntactically as follows

$$P_\lambda \stackrel{def}{=} \begin{cases} \tau.Q1 + ... + \tau.Qn + \tau.R1_\lambda + ... + \tau.Rm_\lambda & m > 0 \wedge n > 0 \\ \tau.Q1 + ... + \tau.Qn & m = 0 \wedge n > 0 \\ \tau.R1_\lambda + ... + \tau.Rm_\lambda & n = 0 \wedge m > 0 \\ \tau.\mathbf{top} & m = 0 \wedge n = 0 \end{cases}$$

where P_λ, $R1_\lambda$,..., and Rm_λ are process identifiers, and the notations used in the above defining equation is similar to those of the Boolean buffer given in [11]. □

Theorem 3.12
If $P \stackrel{\lambda}{\Rightarrow} P'$, then P' is a failures refinement of P_λ.

Proof We proceed by induction on the length of transitions. In case of $P \stackrel{\lambda}{\rightarrow} P'$, the conclusion follows from Theorem 3.10. Otherwise, assume that $P \stackrel{\tau}{\rightarrow} R$ and $R \stackrel{\lambda}{\Rightarrow} P'$. From the inductive hypothesis we have $failures(P') \subseteq failures(R_\lambda)$. The conclusion follows from the definition of P_λ and Theorem 3.10. □

Let s be a sequence of observable actions, and $\stackrel{s}{\Longrightarrow} \stackrel{def}{=} (\stackrel{first(s)}{\Longrightarrow} \circ \stackrel{tail(s)}{\Longrightarrow})$ and $\stackrel{\langle\rangle}{\Longrightarrow} \stackrel{def}{=} \stackrel{\tau}{\rightarrow}^*$.

Theorem 3.13
(1) $failures(P) = \{(s, X) \mid \exists Q \bullet (P \stackrel{s}{\Longrightarrow} Q \wedge \mathbf{init}(Q) \cap \bar{X}^\tau = \emptyset\}$
(2) $traces(P) = \{s \mid \exists Q \bullet P \stackrel{s}{\Longrightarrow} Q\}$

Proof By induction on the length of s. □

Let s and t be sequences of observable actions. The set $s|t$ is defined inductively

$$s|t \stackrel{def}{=} \begin{cases} \{s\} & t = \langle\rangle \\ \{t\} & s = \langle\rangle \\ \langle first(s)\rangle(tail(s)|t) \cup \langle first(t)\rangle(s|tail(t)) & s \neq \langle\rangle \text{ and } t \neq \langle\rangle \end{cases}$$

4 Simulation

The two-thirds simulation defined by Larsen and Skou [9] is adopted here.

Definition 4.1 (Two-thirds simulation)
Let $\mathcal{S}$ be a binary relation over the set $\mathcal{P}$ of process expressions. Then $\mathcal{S}$ is called a two-third simulation over $(\mathcal{P}, \mathcal{T})$ if, whenever $P\,\mathcal{S}\,Q$

(1) if $P \stackrel{\lambda}{\Rightarrow} P'$ then there exists $Q \in \mathcal{P}$ such that $Q \stackrel{\lambda}{\Rightarrow} Q'$ and $P' \mathcal{S} Q'$.

(2) if P can refuse X, so can Q.

We say Q simulates P, denoted by $P \leq_{2/3} Q$, if there exists a two-third simulation $\mathcal{S}$ such that $P \mathcal{S} Q$. □

Remark: top $\leq_{2/3} Q$ for all Q. □

Lemma 4.2 (τ-**transition implies two thirds simulation**)
If $P \stackrel{\tau}{\rightarrow} P'$ then $P' \leq_{2/3} P$. □

Lemma 4.3
$\leq_{2/3}$ is reflexive and transitive. □

Theorem 4.4 (Two thirds simulation is a pre-congruence)
Two thirds simulation is a pre-congruence, in other words, $P \leq_{2/3} Q$ implies for all $\alpha \in Act$

(1) $\alpha.P + M \leq_{2/3} \alpha.Q + M$

(2) $\mathbf{new}\, a\, P \leq_{2/3} \mathbf{new}\, a\, Q$

(3) $P|R \leq_{2/3} Q|R$

(4) $R|P \leq_{2/3} R|Q$ □

Theorem 4.5 (simulation implies refinement)
If $P \leq_{2/3} Q$ then P is a failures refinement of Q.

Proof

	$(\langle\rangle, X) \in failures(P)$	{Def of failures}
$\Longrightarrow$	P can refuse X	$\{P \leq_{2/3} Q\}$
$\Longrightarrow$	Q can refuse X	{Def of failures}
$\Longrightarrow$	$(\langle\rangle, X) \in failures(Q)$	
$\Longleftarrow$	$(\langle\lambda\rangle \cdot s, X) \in failures(P)$	{Def of failures}
$\Longrightarrow$	$\exists P' \bullet (P \stackrel{\lambda}{\Rightarrow} P') \wedge (s, X) \in failures(P')$	$\{P \leq_{2/3} Q\}$
$\Longrightarrow$	$\exists Q' \bullet (Q \stackrel{\lambda}{\Rightarrow} Q' \wedge P' \leq_{2/3} Q') \wedge (s, X) \in failures(P')$	{induction hypothesis}
$\Longrightarrow$	$\exists Q' \bullet (Q \stackrel{\lambda}{\Rightarrow} Q') \wedge (s, X) \in failures(Q')$	{Def of failures}
$\Longrightarrow$	$(\langle\lambda\rangle \cdot s, X) \in failures(Q)$	

□

5 Equating simulation with refinement

The introduction of failures into the definition of simulation and refinement enables us to distinguish $\tau.M + \tau.N$ from $M + N$. In the labelled transition system presented in Section 2, the summation $(a.b + a.c)$ cannot simulate the process $a.(\tau.b + \tau.c)$, even though they have the same trace and failure sets. This is why we need to add some non-standard transitions.

$$RECON-1: \frac{P \stackrel{\lambda}{\rightarrow} P_i \quad \text{for all } i \in I}{P \stackrel{\lambda}{\rightarrow} \Sigma_{i \in I} \tau.P_i} \quad I \text{ is finite}$$

$$RECON-2: \frac{P \stackrel{\tau}{\rightarrow} Q \quad Q \stackrel{\lambda}{\rightarrow} R}{P \stackrel{\lambda}{\rightarrow} R}$$

The rule $RECON-1$ states that if there is a non-deterministic choice between transitions with the same label, then this choice may be exercised after the transition has taken

place. $RECON-2$ states that if an action can occur after a τ-transition, an implementer is permitted to optimise the execution by omitting the tau-transition, and allowing the visible action to occur immediately. Actually, in $RECON-2$, the λ could be replaced by α as in [11], allowing an optimisation that replaces two internal steps by one. In the following we use LTS_1 to stand for the extended transition systems, and $\xrightarrow{\alpha}_1$ for the transitions in LTS_1. Define $\mathbf{init}_1(P) \stackrel{def}{=} \{\alpha \mid \exists Q \bullet P \xrightarrow{\alpha}_1 Q\}$.

Lemma 5.1
P is stable in LTS_1 iff it is stable in LTS. Furthermore if P is stable then $\mathbf{init}_1(P) = \mathbf{init}(P)$.

Proof From the fact that $\xrightarrow{\alpha} \subseteq \xrightarrow{\alpha}_1$ we are only required to prove the if-part.

(1) $P = \Sigma_{i\in I}\alpha_i.P_i$. From Lemma 3.3(1) we have $\tau \notin \{\alpha_i \mid i \in I\}$. Clearly P has no τ-transition in LTS_1, and $\mathbf{init}_1(P) = \{\alpha_i | i \in I\}$.

(2) $P = \mathbf{new}\, a\, Q$. From Lemma 3.3(3) it follows that Q is stable. By induction hypothesis we conclude that Q is stable in LTS_1. From Lemmas 3.3(3) and 3.1(3) follows

$$\mathbf{init}_1(\mathbf{new}\, a\, Q) = \mathbf{init}_1(Q) \setminus \{a, \bar{a}\} = \mathbf{init}(Q) \setminus \{a, \bar{a}\} = \mathbf{init}(\mathbf{new}\, a\, Q)$$

(3) $P = Q|R$. From Lemma 3.3(2) it follows that both Q and R are stable and $\mathbf{init}(Q) \cap \{\bar{a}|a \in \mathbf{init}(R)\} = \emptyset$. From induction hypothesis it follows that $\mathbf{init}_1(Q) = \mathbf{init}(Q)$ and $\mathbf{init}_1(R) = \mathbf{init}(R)$ and both Q and R stable in LTS_1. The conclusion follows from Lemmas 3.3(2) and 3.1(2).

(4) $P = A\langle\vec{b}\rangle$ where $A\langle\vec{a}\rangle \stackrel{def}{=} P_A$. From Lemma 3.3(4) it follows that $\{\vec{b}/\vec{a}\}P_A$ is stable. From induction hypothesis and Lemma 3.3(4) we conclude that P is also stable, and

$$\mathbf{init}_1(P) = \mathbf{init}_1(\{\vec{b}/\vec{a}\}P_A) = \mathbf{init}(\{\vec{b}/\vec{a}\}P_A) = \mathbf{init}(P) \qquad \square$$

Lemma 5.2
If $(\tau.P_1 + \tau.P_2)(\stackrel{s}{\Longrightarrow}_1 \circ \xrightarrow{\tau}_1^*) Q$ and Q is stable, then there is i such that $P_i(\stackrel{s}{\Longrightarrow}_1 \circ \xrightarrow{\tau}_1^*) Q$ with no more *recon* rules is used.

Proof The proof proceeds by induction on the number of the *recon* rules used in the inference.
(1) The first step is a τ-transition. In this case from the rule SUM we conclude that there is i such that $P \xrightarrow{\tau}_1 P_i$ and $P_i(\stackrel{s}{\Longrightarrow}_1 \circ \xrightarrow{\tau}_1^*) Q$ which implies the conclusion directly.

(2) The first step is a λ-transition $(\tau.P_1 + \tau.P_2) \xrightarrow{\lambda}_1 R$. There are two possible cases:

(2.1) It is inferred by the rule $RECON-1$

$$\frac{\dfrac{(\tau.P_1+\tau.P_2)\xrightarrow{\tau}P_1,\ P_1\xrightarrow{\lambda}_1 Q_1}{(\tau.P_1+\tau.P_2)\xrightarrow{\lambda}_1 Q_1} \qquad \dfrac{(\tau.P_1+\tau.P_2)\xrightarrow{\tau}P_2,\ P_2\xrightarrow{\lambda}_1 Q_2}{(\tau.P_1+\tau.P_2)\xrightarrow{\lambda}_1 Q_2}}{(\tau.P_1 + \tau.P_2) \xrightarrow{\lambda}_1 (\tau.Q_1 + \tau.Q_2)}$$

By induction hypothesis we can find j such that $Q_j(\stackrel{tail(s)}{\Longrightarrow}_1 \circ \xrightarrow{\tau}_1^*) Q$. Now the inference $P_j(\stackrel{s}{\Longrightarrow}_1 \circ \xrightarrow{\tau}_1^*) Q$ uses less *recon* rules as required.

(2.2) It is inferred by the rule $RECON-2$, i.e., there is i such that

$$\frac{(\tau.P_1 + \tau.P_2) \xrightarrow{\tau} P_i, \quad P_i \xrightarrow{\lambda}_1 R}{(\tau.P_1 + \tau.P_2) \xrightarrow{\lambda}_1 R}$$

In this case we have $P_i(\stackrel{s}{\Longrightarrow}_1 \circ \xrightarrow{\tau}^*) Q$ which uses less *recon* rules. $\square$.

Theorem 5.3

If $P\,(\stackrel{s}{\Longrightarrow}_1 \circ \stackrel{\tau}{\rightarrow}_1^*)\,Q$ and Q is stable, then $P\,(\stackrel{s}{\Longrightarrow} \circ \stackrel{\tau}{\rightarrow}^*)\,Q$.

Proof The proof uses induction on the structure of P and the number of the *recon* rules used in the inference. When the number of steps of the inference is zero, the conclusion follows from Theorem 5.1. If the inference does not use the *recon* rules, the conclusion is obvious. In the following we are going to treat all possible cases for the first step of the inference based on structural induction.

(1) $P = \Sigma_{i\in I}\alpha_i.P_i$. We consider three cases.

(1.1) The first step of the inference is inferred by the use of SUM, i.e., there exists $i \in I$ such that

$$P \stackrel{\alpha_i}{\rightarrow}_1 P_i \quad \text{and} \quad P_i(\stackrel{s'}{\Longrightarrow}_1 \circ \stackrel{\tau}{\rightarrow}_1^*)\,Q$$

where $s' = s$ if $\alpha_i = \tau$ otherwise $s' = tail(s)$. By induction hypothesis we conclude that $P_i\,(\stackrel{s'}{\Longrightarrow} \circ \stackrel{\tau}{\rightarrow}^*)\,Q$ which implies that $P\,(\stackrel{s}{\Longrightarrow} \circ \stackrel{\tau}{\rightarrow}^*)\,Q$

(1.2) The first step is inferred by $RECON-2$ where $\alpha_i = \tau$

$$\frac{P \stackrel{\tau}{\rightarrow} P_i,\ \ P_i \stackrel{\lambda}{\rightarrow}_1 W}{P \stackrel{\lambda}{\rightarrow}_1 W}$$

which implies that $P_i\,(\stackrel{s}{\Longrightarrow}_1 \circ \stackrel{\tau}{\rightarrow}_1^*)\,Q$. By induction hypothesis we conclude

$$P_i\,(\stackrel{s}{\Longrightarrow} \circ \stackrel{\tau}{\rightarrow}^*)\,Q$$

which together with $P \stackrel{\tau}{\rightarrow} P_i$ implies the conclusion.

(1.3) The first step is inferred by $RECON-1$, i.e., there exist $i_1, i_2 \in I$

$$\frac{\dfrac{P \stackrel{\tau}{\rightarrow} P_{i_1},\ P_{i_1} \stackrel{\lambda}{\rightarrow}_1 Q_1}{P \stackrel{\lambda}{\rightarrow}_1 Q_1} \qquad \dfrac{P \stackrel{\tau}{\rightarrow} P_{i_2},\ P_{i_2} \stackrel{\lambda}{\rightarrow}_1 Q_2}{P \stackrel{\lambda}{\rightarrow} Q_2}}{P \stackrel{\lambda}{\rightarrow}_1 (\tau.Q_1 + \tau.Q_2)}$$

In this case we have $(\tau.Q_1 + \tau.Q_2)\,(\stackrel{tail(s)}{\Longrightarrow}_1 \circ \stackrel{\tau}{\rightarrow}_1^*)\,Q$. From Lemma 5.2 we can find k such that $P \stackrel{\lambda}{\rightarrow}_1 Q_k(\stackrel{tail(s)}{\Longrightarrow}_1 \circ \stackrel{\tau}{\rightarrow}_1^*)Q$ where the first step uses less *recon* rules:

$$\frac{P \stackrel{\tau}{\rightarrow} P_{i_k},\ P_{i_k} \stackrel{\lambda}{\rightarrow}_1 Q_k}{P \stackrel{\lambda}{\rightarrow}_1 Q_k}$$

which together the induction hypothesis leads to the conclusion.

(2) $P = \mathtt{new}\,a\,P'$. From the rule RES and Lemma 3.3 we can find a stable process Q' such that $Q = \mathtt{new}\,a\,Q'$ and

$$P'\,(\stackrel{s}{\Longrightarrow}_1 \circ \stackrel{\tau}{\rightarrow}_1^*)\,Q'$$

By induction hypothesis we conclude $P'\,(\stackrel{s}{\Longrightarrow} \circ \stackrel{\tau}{\rightarrow}^*)\,Q'$, which implies the conclusion.

(3) $P = P_1|P_2$. From $L-PAR$ and $R-PAR$ it follows that there exist stable processes Q_1 and Q_2 and sequences s_1 and s_2 such that $Q = Q_1|Q_2$ and $s \in (s_1|s_2)$, and $P_i\,(\stackrel{s_1}{\Longrightarrow}_1 \circ \stackrel{\tau}{\rightarrow}_1^*)\,Q_i$ for $i = 1, 2$. By induction hypothesis we conclude $P_i\,(\stackrel{s_1}{\Longrightarrow} \circ \stackrel{\tau}{\rightarrow})^*\,Q_i$ (for $i = 1, 2$) from which and Theorem 5.1 follows the conclusion.

(4) $P = A\langle\vec{b}\rangle$ where $A\langle\vec{a}\rangle \stackrel{def}{=} P_A$. From the rule $IDENT$ it follows that

$$\{\vec{b}/\vec{a}\}P_A\,(\stackrel{s}{\Longrightarrow}_1 \circ \stackrel{\tau}{\rightarrow}_1^*)\,Q$$

By induction hypothesis we have $\{\vec{b}/\vec{a}\}P_A\,(\stackrel{s}{\Longrightarrow} \circ \stackrel{\tau}{\rightarrow}^*)\,Q$, which together with the rule *IDENT* implies the conclusion. □

Theorem 5.4
$P \stackrel{s}{\Longrightarrow}_1$ iff $P \stackrel{s}{\Longrightarrow}$

Proof Similar to Theorem 5.3. □

Theorem 5.5
Adding the rules $RECON-1$ and $RECON-2$ to the labelled transition system (given by Definition 2.2) has no effect on $traces(P)$ and $failures(P)$.

Proof Direct from Theorems 3.13, 5.3 and 5.4. □

Corollary
Theorem 3.12 remains valid in presence of the *recon* rules. □

Theorem 5.6 (closure)

If $P \stackrel{\lambda}{\Rightarrow}_1 P_i$ for $i = 1, 2$, then $P \stackrel{\lambda}{\rightarrow}_1 (\tau.P_1 + \tau.P_2)$. □

Corollary

If $P \stackrel{\lambda}{\Rightarrow}_1$ then $P \stackrel{\lambda}{\rightarrow}_1 P_\lambda$,

where the definition of P_λ applies only to the original transition system, without the recon rules. □

Theorem 5.7 (failures refinement implies simulation)
If P is a failures refinement of Q then $P \leq_{2/3} Q$.

Proof Define

$$\mathcal{S} \stackrel{def}{=} \{(P, Q) \mid traces(P) \subseteq traces(Q) \ \wedge \ failures(P) \subseteq failures(Q)\}$$

We are going to show that $\mathcal{S}$ is a two-third simulation.

Assume that $P\,\mathcal{S}\,Q$

(1) If $P \stackrel{\lambda}{\Rightarrow}_1 P'$, then one has $\langle\lambda\rangle \in traces(P) \subseteq traces(Q)$ which implies that $Q \stackrel{\lambda}{\Rightarrow}_1$. From Corollary of Theorem 5.6 it follows that $Q \stackrel{\lambda}{\Rightarrow}_1 Q_\lambda$, which implies

$$traces(P') \subseteq traces(P_\lambda) \subseteq traces(Q_\lambda) \text{ and}$$
$$failures(P') \subseteq failures(P_\lambda) \subseteq failures(Q_\lambda)$$

as required.

(2) If P can refuse X, then from the definition of failures one has $(\langle\rangle, X) \in failures(P)$. From the assumption $P\,\mathcal{S}\,Q$ one has $(\langle\rangle, X) \in failures(Q)$, i.e., Q can also refuse X. □

6 Conclusion

An explicit goal in the definition of bisimulation as the notion of process equivalence in CCS is to maintain an explicit distinction between processes which resolve there internal non-determinism at different times. The redundant transitions suggested in this paper deliberately obscure these distinctions. They give explicit freedom to the implementer of the calculus to resolve non-determinism whenever it is most convenient to do so. They thereby raise the level of abstraction of CCS closer to that of the standard model of CSP [6].

But they still differ in the treatment of divergence, which is defined as the possibility of an infinite sequence of invisible tau-transitions. It is sometimes called live-lock, and is

a common goal of denial-of-service attackers on a computer system. In CCS, the responsibility for avoidance of divergence (wherever possible) is placed upon the implementation of the calculus; it is required to make an ultimately fair choice between a possible visible action and an invisible one. That is why CCS is so useful in modelling and analysis of distributed scheduling algorithms, where fairness is needed to break symmetry, and is easily implemented by probabilistic means.

CSP, on the other hand, places responsibility for the avoidance of divergence on the designer of a process. It does this by interpreting recursion in the sense of Dijkstra [2] as the greatest fixed point rather than the least fixed point of the defining equation. Thus the possibility of an infinite tau-sequence gives rise to the universal set of failures. As a result, a divergent process cannot refine any specification, except one that is also divergent. For example, the equation defining **top** makes it denote the bottom of the simulation ordering rather than the top. CSP is therefore an appropriate calculus where avoidance of live-lock is an important concern of system design.

There is a variety of different methods of achieving equivalence of simulation with refinement. The simplest one is just to postulate certain basic algebraic equations as axioms or structural equivalences of the calculus [5, 12]. A second method is to define a version of simulation between sets of processes rather than between single processes [3]. A third method uses the modal mu-calculus [8] as a specification language, and then makes a syntactic restriction on the expressive power of the specification, so that two processes with the same failure set cannot be distinguished. The fact that mathematics provides so many ways of achieving the same goal is indicative of robustness in a theory, and maybe in practice too.

Redundant transitions have been used before to explore, adapt and improve the properties of bisimulation [1, 4, 16]. Their use to equate simulation with refinement is probably original.

The primary goal for this investigation was to simplify the theoretical foundation for the design of a new model-checker [14]. It gave confidence that model checking would be effective in treating more general forms of refinement.

References

[1] B. Bloom. "*Structural operational semantics for weak bisimulation.*" Theoretical Computer Science 146, 25–68, (1995).

[2] E.W. Dijkstra. "*A discipline of programming.*" Prentice Hall, (1976).

[3] Paul Gardiner. "*Bisumation on sets of processes.*" Private communication, (2002)

[4] R.J. Glabbeek. "*Bounded non-determinism and the approximation induction principle in process algebra (extended abstract).* Proceedings STACS, LNCS 247, pp 336–347, (1987).

[5] M. Hennessy. "*Algebraic theory of processes.*" The MIT Press, (1988).

[6] C.A.R. Hoare. "*Communicating sequential processes.*" Prentice Hall, (1985).

[7] C.A.R. Hoare, Cedric Fournet, He Jifeng, Paul Gardiner, Robin Milner, Sriram Rajamani, Jakob Rehof, Bill Roscoe. "*Bisimulation and Refinement Reconciled*" to appear as Microsoft Research Report (2003).

[8] D. Kozen. "*Results on propositional mu-calculus.*" Theoretical Computer Science, (1983).

[9] K.G. Larsen and A. Skou. "*Bisimulation through probabilistic testing.*" Information and control 94 (1), (1991)

[10] R. Milner. "*Communication and concurrency.*" Prentice Hall, (1989).

[11] R. Milner. "*Communicating and mobile systems: the π-calculus.*" Cambridge University Press, (1999).

[12] R. De Nicola. "*A complete set of axioms for a theory of communicating sequential processes.*" Foundations of Computing Theory, LNCS 158, pp 115–126, (1983).

[13] R. De Nicola and M. Hennessy. "*Testing equivalence for processes.*" Theoretical Computer Science 34, pp 83–133, (1983).

[14] S. Rajamani, J. Rehof. "*Zing: A new model checker.*" to appear as Microsoft Research Report, (2003).

[15] A.W. Roscoe. "*The theory and practice of concurrency.*" Prentice Hall, (1998)

[16] Simone Tini. "*Rule Formats for Non-interference.*" ESOP'2003, LNCS 2618, pp 129–143, (2003)