The United Nations University

UNU/IIST

International Institute for Software Technology

Equating Bisimulation with Refinement

He Jifeng and Tony Hoare

August, 2003

UNU/IIST Report No. 282

UNU/IIST and UNU/IIST Reports

UNU/IIST (United Nations University International Institute for Software Technology) is a Research and Training Centre of the United Nations University (UNU). It is based in Macau, and was founded in 1991. It started operations in July 1992. UNU/IIST is jointly funded by the Governor of Macau and the governments of the People's Republic of China and Portugal through a contribution to the UNU Endownment Fund. As well as providing two-thirds of the endownment fund, the Macau authorities also supply UNU/IIST with its office premises and furniture and subsidise fellow accommodation.

The mission of UNU/IIST is to assist developing countries in the application and development of software technology.

UNU/IIST contributes through its programmatic activities:

1. Advanced development projects, in which software techniques supported by tools are applied,
2. Research projects, in which new techniques for software development are investigated,
3. Curriculum development projects, in which courses of software technology for universities in developing countries are developed,
4. University development projects, which complement the curriculum development projects by aiming to strengthen all aspects of computer science teaching in universities in developing countries,
5. Schools and Courses, which typically teach advanced software development techniques,
6. Events, in which conferences and workshops are organised or supported by UNU/IIST, and
7. Dissemination, in which UNU/IIST regularly distributes to developing countries information on international progress of software technology.

Fellows, who are young scientists and engineers from developing countries, are invited to actively participate in all these projects. By doing the projects they are trained.

At present, the technical focus of UNU/IIST is on formal methods for software development. UNU/IIST is an internationally recognised center in the area of formal methods. However, no software technique is universally applicable. We are prepared to choose complementary techniques for our projects, if necessary.

UNU/IIST produces a report series. Reports are either Research R, Technical T, Compendia C or Administrative A. They are records of UNU/IIST activities and research and development achievements. Many of the reports are also published in conference proceedings and journals.

Please write to UNU/IIST at P.O. Box 3058, Macau or visit UNU/IIST's home page: http://www.iist.unu.edu, if you would like to know more about UNU/IIST and its report series.

Chris George, Acting Director

The United Nations University

UNU/IIST

International Institute for Software Technology

P.O. Box 3058
Macau

Equating Bisimulation with Refinement

He Jifeng and Tony Hoare

Abstract

Two-thirds simulation is equivalent to failures refinement if two redundant transition rules are added to the operational semantics of CCS. This paper also shows how to define a process calculus in which the concepts of simulation and mutual refinement coincide with testing equivalence: thus the combined benefits of all three approaches are available uniformly to all applications.

He Jifeng is a senior research-fellow of UNU/IIST. He is also a professor of computer science at the Software Engineering Institute of East China Normal University. His research interests include the theories of programming languages, formal methods for safety-critical systems, parallel and distributed systems, component-oriented computing, co-design techniques for real-time embedded systems. Email: jifeng@iist.unu.edu

Professor Sir Tony Hoare held the James Martin Chair in Computing at the Oxford University Computing Laboratory from 1979 to 1999. Following his retirement from Oxford, he has taken up an appointment as Senior Researcher with Microsoft Research at Cambridge, where he is pursuing his long-term interest in programming theory. He is a fellow of the Royal Society of London. As one of the foremost experts in the field of computer science, he was knighted in 2000 for his achievements in software engineering. In further recognition of this, Sir Tony was awarded the Kyoto Prize for Advanced Technology in November 2000. Email: thoare@microsoft.com

Contents

1 Introduction

In the study of CCS 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, and also with testing equivalence: thus the combined benefits of all three 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

$$\bar{X} \stackrel{def}{=} \{\bar{a} \mid a \in X\}$$

$$X^\tau \stackrel{def}{=} X \cup \{\tau\}$$

Definition 2.1 (**Process expression**)

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

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

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 $\texttt{new}\, a\, P$ internalises both a and $\bar{a}$.

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.

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

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

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

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

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

In the rules $REACT_t$, $L-PAR_t$, $R-PAR_t$ and RES_t a transition of a composite process is inferred from transitions of its components. SUM_t 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 $P \xrightarrow{\alpha} P'$.

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

3 Failures refinement

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

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

Lemma 3.1

(1) $\mathbf{init}(\Sigma_i \alpha_i.P_i) = \{\alpha_i \mid i \in I\}$

(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) $\mathbf{init}(\mathtt{new}\, a\, P) = \mathbf{init}(P) \setminus \{a, \bar{a}\}$

(4) $\mathbf{init}(A < \vec{b} >) = \mathbf{init}(\{\vec{b}/\vec{a}\}P_A)$ □

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) $(\mathtt{new}\, a\, P)$ is stable iff P is stable,

(4) $A < \vec{b} >$ 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

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

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) $\texttt{new}\, a\, P$ can refuse X iff P can refuse $X \setminus \{a, \bar{a}\}$.

(4) $A < \vec{b} >$ 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 $\stackrel{\lambda}{\Rightarrow} \stackrel{def}{=} \stackrel{\tau}{\rightarrow}^* \circ \stackrel{\lambda}{\rightarrow}$ for $\lambda \in \mathcal{L}$. Define

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

$$failures(P) \stackrel{def}{=} \begin{array}{l} \{(<>, X) \mid P \textit{ can refuse } X\} \cup \\ \{(< \lambda > s, X) \mid P \stackrel{\lambda}{\Rightarrow} P' \wedge (s, X) \in failures(P')\} \end{array}$$

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

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

In the following we shall use $< \lambda > \cdot failures(P)$ to stand for the set

$$\{< \lambda > s, X) \mid (s, X) \in failures(P)\}$$

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

$$traces(\mathbf{top}) = \{<>\} \qquad failures(\mathbf{top}) = \emptyset$$ □

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

$$X \subseteq (X_P \cap X_Q) \text{ and } \forall \lambda \in \mathcal{L} \bullet (\lambda \in X_P \vee \bar{\lambda} \in X_Q)$$ □

Theorem 3.9 (τ-transition implies failures refinement)

If $P \stackrel{\tau}{\rightarrow} P'$, then P' is a failures refinement of P. □

The traces and failures of a summation can be calculated as follows.

Theorem 3.10

(1) $traces(\Sigma_{i\in I}\alpha_i.P_i) \;=\; \bigcup_{i\in I} traces(\alpha_i.P_i)$

(2) $failures(\Sigma_{i\in I}\alpha_i.P_i) \;=\; \bigcup_{\alpha_i=\tau} failures(P_i) \cup$
$\bigcup_{\alpha_j\neq\tau} <\alpha_j> \cdot failures(P_j) \;\cup\; \{(<>, X) \mid \bar{X}^{\tau} \cap \{\alpha_i \mid i \in I\} \;=\; \emptyset\}$ □

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 Let $P \stackrel{\tau}{\rightarrow}^n \circ \stackrel{\lambda}{\rightarrow} P'$. We proceed by induction on n.

(1) Base case $n = 0$: In this case we conclude that $P' \in \{Q1, .., Qn\}$. The conclusion follows from Theorems 3.10.

(2) Inductive step $n = k + 1$: It is clear that there is a process $R \in \{R1, .., Rm\}$ such that

$$P \stackrel{\tau}{\rightarrow} R\,(\stackrel{\tau}{\rightarrow}^k \circ \stackrel{\lambda}{\rightarrow})\,P'$$

From the induction hypothesis and Theorem 3.10 it follows the conclusion.

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) $\texttt{new}\,a\,P \;\leq_{2/3}\; \texttt{new}\,a\,Q$

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

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

Proof Here we shall prove the conclusion (3).

Let $\mathcal{S} \stackrel{def}{=} \{(P|R,\, Q|R) \mid P \leq_{2/3} Q\}$.

Condition 1 of the two-thirds simulation: Assume that $(P|R) \stackrel{\lambda}{\Rightarrow} U$.

From the rules $L-PAR_t$, $REACT_t$ and $R-PAR_t$ we can find P' and R' such that $U = P'|R'$, where there are following four cases:

(1) $P \stackrel{\lambda}{\Rightarrow} P'$ and $R \stackrel{\tau}{\rightarrow}{}^{*} R'$

(2) $R \stackrel{\lambda}{\Rightarrow} R'$ and $P \stackrel{\tau}{\rightarrow}{}^{*} P'$

(3) There is a sequence $< \lambda_1, .., \lambda_n >$ of visible actions such that

$$P \stackrel{<\lambda_1,...,\lambda_n,\lambda>}{\Rightarrow} P' \text{ and } R\,(\stackrel{<\bar{\lambda}_1,...,\bar{\lambda}_n>}{\Rightarrow} \circ \stackrel{\tau}{\rightarrow}^*)\,R'$$

(4) There is a sequence $< \lambda_1, .., \lambda_n >$ of visible actions such that

$$R \stackrel{<\lambda_1,...,\lambda_n,\lambda>}{\Rightarrow} R' \text{ and } P\,(\stackrel{<\bar{\lambda}_1,...,\bar{\lambda}_n>}{\Rightarrow} \circ \stackrel{\tau}{\rightarrow}^*)\,P'$$

Case (1). Since $P \leq_{2/3} Q$ we can find Q' such that $Q \stackrel{\lambda}{\Rightarrow} Q'$ and $P' \leq_{2/3} Q'$. From the rules $L - PAR_t$ and $R - PAR_t$ and the definition $\mathcal{S}$ it follows that

$$(Q|R) \stackrel{\lambda}{\Rightarrow} (Q'|R') \text{ and } (P'|R')\,\mathcal{S}\,(Q'|R')$$

Case (2). From Lemma 4.2 and the assumption $P \leq_{2/3} Q$ we conclude $P' \leq_{2/3} Q$. From the rule $R - PAR_t$ and the definition $\mathcal{S}$, it follows that

$$(Q|R) \stackrel{\lambda}{\Rightarrow} (Q|R') \text{ and } (P'|R')\,\mathcal{S}\,(Q|R')$$

Case (3). Similar to Case (1).

Case (4): Similar to Case (2).

Condition 2 of the two-thirds simulation: Assume that $(P|R)$ can refuse X, i.e., there is a stable process $U = (P'|R')$ such that

$$(P|R) \stackrel{\tau}{\rightarrow}^* U \text{ and } \mathbf{init}(U) \cap \bar{X} = \emptyset$$

There are two cases:

(1) $P \stackrel{\tau}{\rightarrow}^* P'$ and $R \stackrel{\tau}{\rightarrow}^* R'$.

(2) There is a sequence $< \lambda_1, .., \lambda_n >$ of visible actions such that

$$P\,(\stackrel{<\lambda_1,...,\lambda_n>}{\Rightarrow} \circ \stackrel{\tau}{\rightarrow}^*)\,P' \text{ and } R\,(\stackrel{<\bar{\lambda}_1,...,\bar{\lambda}_n>}{\Rightarrow} \circ \stackrel{\tau}{\rightarrow}^*)\,R'.$$

Case (1) From Lemma 3.1(2) it follows that $\mathbf{init}(P'|R') = \mathbf{init}(P') \cup \mathbf{init}(R')$ and

$$\mathbf{init}(P'|R') \cap \bar{X} = \emptyset$$

From Lemmas 4.2 and 4.3 we have $P' \leq_{2/3} Q$. Because P' can refuse $Y = \mathcal{L} \setminus \{\lambda \mid \bar{\lambda} \in \mathbf{init}(P')\}$, as a result, we can find a stable process Q' such that $Q \stackrel{\tau}{\rightarrow}^* Q'$ and it rejects every action of Y which indicates that

$$\mathbf{init}(Q') \subseteq \mathbf{init}(P')$$

Now from Lemma 3.1(2) it follows that $(Q'|R')$ is also stable, and $\mathbf{init}(Q'|R') = \mathbf{init}(Q') \cup \mathbf{init}(R')$ and

$$\mathbf{init}(Q'|R') \cap \bar{X} = \emptyset$$

which implies that $(Q|R)$ can also refuse X.

Case (2). Similar to Case (1). □

Theorem 4.5 (**simulation implies refinement**)

If $P \leq_{2/3} Q$ then P is a failures refinement of Q.

Proof By induction on the length of traces.

Base case :	$(<>, X) \in failures(P)$	
$\Longrightarrow$	{Definition of failures}	
	P can refuse X	
$\Longrightarrow$	{Condition 2 of the two-thirds simulation}	
	Q can refuse X	
$\Longrightarrow$	{Definition of failures}	
	$(<>, X) \in failures(Q)$	
Inductive step :	$(<\lambda> s, X) \in failures(P)$	
$\Longrightarrow$	{Definition of failures}	
	$\exists P' \bullet P \stackrel{\lambda}{\Rightarrow} P' \ \wedge \ (s, X) \in failures(P')$	
$\Longrightarrow$	$\{P \leq_{2/3} Q\}$	
	$\exists Q' \bullet Q \stackrel{\lambda}{\Rightarrow} Q' \ \wedge \ P' \leq_{2/3} Q' \ \wedge \ (s, X) \in failures(P')$	
$\Longrightarrow$	{Inductive hypothesis}	
	$\exists Q' \bullet Q \stackrel{\lambda}{\Rightarrow} Q' \ \wedge \ (s, X) \in traces(Q')$	
$\Longrightarrow$	{Definition of failures}	
	$(<\lambda> s, X) \in failures(Q)$	

□

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.

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

$$RECON-2: \quad \frac{P \xrightarrow{\tau} Q \qquad Q \xrightarrow{\lambda} R}{P \xrightarrow{\lambda} R}$$

The rule $RECOM-1$ 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 tau-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.

Theorem 5.1 (**closure**)

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

Proof Assume that

$$P \xrightarrow{\tau} Q_1 \xrightarrow{\tau} ... \xrightarrow{\tau} Q_m \xrightarrow{\lambda} P_1$$

$$P \xrightarrow{\tau} R_1 \xrightarrow{\tau} ... \xrightarrow{\tau} R_n \xrightarrow{\lambda} P_2$$

Case 1: $m = 0$ and $n = 0$. Direct from the rule $RECON-1$.

Case 2: $m > 0$ and $n = 0$. Applying the rule $RECON-2$ m times it follows that $P \xrightarrow{\lambda} P_1$. Following the proof of Case 1 we can establish the conclusion.

Case 3: $m = 0$ and $n > 0$. Similar to Case 2.

Case 4: $m > 0$ and $n > 0$. Applying the rule $RECON-2$ to $P \xrightarrow{\tau}{}^{m} \circ \xrightarrow{\lambda} P_1$ and $P \xrightarrow{\tau}{}^{n} \circ \xrightarrow{\lambda} P_2$ leads to the conclusion

$$P \xrightarrow{\lambda} P_1 \quad \text{and} \quad P \xrightarrow{\lambda} P_2$$

which induces the transition $P \xrightarrow{\lambda} (\tau.P_1 + \tau.P_2)$. □

Corollary

If $P \stackrel{\lambda}{\Rightarrow}$ then $P \stackrel{\lambda}{\rightarrow} P_\lambda$. □

Theorem 5.2

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 Let

$$\mathcal{P}_1 \stackrel{def}{=} \{P' \mid P \stackrel{\lambda}{\Rightarrow} P' \text{ in the original } \mathbf{LTS}\}$$

$$\mathcal{P}_2 \stackrel{def}{=} \{P' \mid P \stackrel{\lambda}{\Rightarrow} P' \text{ in the presence of new rules}\}$$

From Theorem 5.1 it follows that

$$\mathcal{P}_2 \;=\; \mathcal{P}_1 \;\cup\; \{\Sigma_{i\in I}\tau.Q_i \mid \forall i \in I \bullet Q_i \in \mathcal{P}_2\}$$

Let $F(\mathcal{W}) \stackrel{def}{=} P_1 \cup \{\Sigma_{i\in I}\tau.Q_i \mid \forall i \in I \bullet Q_i \in \mathcal{W}\}$. From the continuity of F it follows that its least fixed point $\mathcal{P}_2$ can be calculated by

$$\mathcal{P}_2 \;=\; \cup_{n\geq 1} F^n(\emptyset)$$

Now we are going to prove that for all $n \geq 1$

$$\bigcup\{traces(Q) \mid Q \in \mathcal{P}_1\} \;=\; \bigcup\{traces(Q) \mid Q \in F^n(\emptyset)\} \qquad (1)$$

$$\bigcup\{failures(Q) \mid Q \in \mathcal{P}_1\} \;=\; \bigcup\{failures(Q) \mid Q \in F^n(\emptyset)\} \qquad (2)$$

Base case $n = 1$: The conclusion follows from the fact that

$$F^1(\emptyset) \;=\; \mathcal{P}_1$$

Inductive step:

$\bigcup\{traces(Q) \mid Q \in F^{n+1}(\emptyset)\}$

$=$ {definition of F}

$\bigcup\{traces(Q) \mid Q \in \mathcal{P}_1\} \ \cup \ \bigcup\{traces(\Sigma_{i\in I}\tau.Q_i) \mid \forall i \in I \bullet Q_i \in F^n(\emptyset)\}$

$=$ {Theorem 5.2.10(1)}

$\bigcup\{traces(Q) \mid Q \in \mathcal{P}_1\} \ \cup \ \bigcup\{\bigcup_{i\in I} traces(Q_i) \mid \forall i \in I \bullet Q_i \in F^n(\emptyset)\}$

$=$ {idempotency of $\cup$}

$\bigcup\{traces(Q) \mid Q \in \mathcal{P}_1\} \ \cup \ \bigcup\{traces(Q) \mid Q \in F^n(\emptyset)\}$

$=$ {induction hypothesis}

$\bigcup\{traces(Q) \mid Q \in \mathcal{P}_1\}$

In a similar way we can establish the conclusion (2). □

Corollary

Theorem 3.12 remains valid in presence of the new rules $RECON-1$ and $RECON-2$. □

Theorem 5.3 (**failures refinement implies simulation**)

If P is a failures refinement of Q then $P \leq_{2/3} Q$.

Proof In the following proof, the definitions of P_λ and Q_λ apply only to the original transition system, without the recon rules.

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 $P \stackrel{\lambda}{\Rightarrow} P'$, then one has

$$< \lambda > \in traces(P) \subseteq traces(Q)$$

which implies that $Q \stackrel{\lambda}{\Rightarrow}$. From Corollary of Theorem 5.1 it follows that $Q \stackrel{\lambda}{\Rightarrow} Q_\lambda$, which implies

$$traces(P') \ \subseteq \ traces(P_\lambda) \ \subseteq \ traces(Q_\lambda)$$

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 $(<>, X) \in failures(P)$. From the assumption $P \mathcal{S} Q$ one has $(<>, X) \in failures(Q)$, i.e., Q can also refuse X. □

6 Testing refinement

One of the main problems of design of concurrent communicating systems is the avoidance of deadlock. A process is said to be stuck if it is non-deterministically possible for it behave like 0, the archetypically deadlocked process.

Definition 6.1

P can deadlock if $0 \leq_{2/3} P$. □

The occurrence of deadlock of a process P may be conditional on the contexts within which P is executed. A context may be defined by a term F, where F is written solely in the notations of the language. Now stuck-free refinement is defined to hold between P and Q if all contexts $F()$ in which P can deadlock are also contexts in which Q can deadlock. So P is always going to be better than Q, if we want to avoid deadlock. Stuck-free refinement is one of the test preorders defined by De Nicola and Hennessy in [13], where P is called a *test refinement* of Q if P can pass all the tests which Q satisfies.

Theorem 6.2 (**simulation implies stuck-free refinement**)

If $P \leq_{2/3} Q$ then $0 \leq_{2/3} F(P)$ implies $0 \leq_{2/3} F(Q)$ for all contexts F.

Proof $0 \leq_{2/3} F(P)$

$\Longrightarrow$ {Theorem 4.5}

$failures(0) \subseteq failures(F(P))$

$\Longrightarrow$ $\{failures(P) \subseteq failures(Q)\}$

$failures(0) \subseteq failures(F(Q))$

$\Longrightarrow$ $\{traces(0) = \{<>\} \subseteq traces(F(Q))$ and Theorem 5.4.5$\}$

$0 \leq_{2/3} F(Q)$ □

Theorem 6.3 (**stuck-free refinement implies simulation**)

If $failures(P)$ is not a subset of $failures(Q)$, then there exists a context F such that $F(P)$ can deadlock but $F(Q)$ cannot.

Proof Let $s =< a_1, .., a_n >$ and $X = \{b_1, .., b_m\}$. Suppose that $(s, X) \in failures(P) \setminus failures(Q)$. Define

$$T_i \stackrel{def}{=} \bar{a}_i.T_{i+1} + succ \quad 1 \leq i \leq n$$

$$T_{n+1} \stackrel{def}{=} \Sigma_{1\le i\le m} b_i.succ$$

where $succ$ is a fresh name not occurring in P nor Q.

Let $F(X) \stackrel{def}{=} \mathtt{new}\, t\, (X|T_1)$ where t contains all free names of P. We are going to establish that $0 \le_{2/3} F(P)$.

From the assumption that $(s,\, X) \in failures(P)$, then there exists a family $\{P_i \mid 0 \le i \le 2n+1\}$ of processes such that $P_0 = P$ and

$$P_{2k} \stackrel{\tau}{\to}{}^* P_{2k+1} \stackrel{a_{k+1}}{\to} P_{2k+2} \quad \text{for all } 0 \le k < n$$

and $\quad P_{2n} \stackrel{\tau}{\to}{}^* P_{2n+1}$

and $\quad \mathbf{init}(P_{2n+1}) \cap \{\bar{b}_1, .., \bar{b}_m\}^\tau = \emptyset$

$\mathtt{new}\, t\, (P|T_1)$

$\stackrel{\tau}{\to}{}^*$ $\{P \stackrel{\tau}{\to}{}^* P_1, \text{ and the rules } L - PAR_t \text{ and } RES_t\}$

$\mathtt{new}\, t\, (P_1|T_1)$

$\stackrel{\tau}{\to}$ $\{P_1 \stackrel{a_1}{\to} P_2,\ T_1 \stackrel{\bar{a}_1}{\to} T_2, \text{ and the rules } REACT_t \text{ and } RES_t\}$

$\mathtt{new}\, t\, (P_2|T_2)$

$\stackrel{\tau}{\to}{}^*$ {similar argument as previous one}

$\mathtt{new}\, t\, (P_{2n-1}|T_n)$

$\stackrel{\tau}{\to}$ $\{P_{2n-1} \stackrel{a_n}{\to} P_{2n},\ T_n \stackrel{\bar{a}_n}{\to} T_{n+1}, \text{ and the rules } REACT_t \text{ and } RES_t\}$

$\mathtt{new}\, t\, (P_{2n}|T_{n+1})$

$\stackrel{\tau}{\to}{}^*$ $\{P_{2n} \stackrel{\tau}{\to}{}^* P_{2n+1} \text{ and the rules } L - PAR_t \text{ and } RES_t\}$

$\mathtt{new}\, t\, (P_{2n+1}|T_{n+1})$

Notice that

$\mathbf{init}(\mathtt{new}\, t\, (P_{2n+1}|T_n))$

$=$ $\{\mathbf{init}(P) \cap \{\bar{b}_1, .., \bar{b}_m\}^\tau = \emptyset \text{ and Lemma 3.1(2) and (3)}\}$

$(\mathbf{init}(P_{2n+1}) \cup \{b_1, .., b_m\}) \setminus \{a,\, \bar{a} \mid a \in t\}$

$=$ {Definition of t}

$\emptyset$

which states the process $\mathtt{new}\, t\, (P_{2n+1}|T_n)$ is stable and can refuse every visible action. This indicates $0 \le_{2/3} \mathtt{new}\, t\, (P|T_1)$.

Now we are going to show that whenever $F(Q) \stackrel{\tau}{\to}{}^* R$ and R is stable then R cannot simulate 0. The proof proceeds by induction on the length of trace $< a_1, .., a_n >$.

Base case $n = 0$: $F(Q) = \mathtt{new}\, t\, (Q|\Sigma_{1\le i\le m} b_i.succ)$

From the rules $L - PAR_t$, RES_t and $REACT_t$, it follows that $F(Q) \stackrel{\tau}{\rightarrow}^* R$ iff

(1) either there exists Q' such that $Q \stackrel{\tau}{\rightarrow}^* Q'$ and Q' is stable, and

$$F(Q) \stackrel{\tau}{\rightarrow}^* F(Q') \stackrel{\tau}{\rightarrow}^* R$$

(2) or there exists Q' such that $Q \stackrel{\bar{b}_i}{\Rightarrow} Q'$, and

$$F(Q) \stackrel{\tau}{\rightarrow}^* (\mathtt{new}\, t\, (Q'|succ)) \stackrel{\tau}{\rightarrow}^* R$$

In the case (1), we conclude that Q' cannot refuse $\{b_1, .., b_m\}$ because $(<>, \{b_1, .., b_m\})$ does not lie in $failures(Q)$. Let

$$\{\bar{b}_{k_1}, .., \bar{b}_{k_n}\} \;=\; \mathbf{init}(Q') \cap \{\bar{b}_1, .., \bar{b}_m\} \text{ and } \; Q' \stackrel{\bar{b}_{k_i}}{\rightarrow} Q_i \;\; \text{for } 1 \leq i \leq n$$

From the rules $REACT_t$ and RES_t we conclude that that $F(Q')$ can only perform the following τ transitions

$$F(Q') \stackrel{\tau}{\rightarrow} \mathtt{new}\, t \bullet (Q_i|succ) \quad \text{for } 1 \leq i \leq n$$

From Theorem 3.8 it follows that $(Q_i|succ)$ cannot refuse $\{s\bar{u}cc\}$, which together with Theorem 3.6(3) implies that

$$(<>, \{s\bar{u}cc\}) \notin failures(\mathtt{new}\, t\, (Q_i|succ))$$

i.e., $\mathtt{new}\, t\, (Q_i|succ)$ cannot simulate 0. This completes the proof for the case (1).

Case (2): Similar to Case (1).

Inductive step: Assume that the conclusion is true when $n = k$. Now consider the case $n = k+1$. From the rules $L - PAR_t$, RES_t and $REACT_t$, it follows that $F(Q) = \mathtt{new}\, t\, (Q|T_1) \stackrel{\tau}{\rightarrow}^* R$ iff

(3) either there exists Q' such that $Q \stackrel{\tau}{\rightarrow}^* Q'$ and Q' is stable, and

$$F(Q) \stackrel{\tau}{\rightarrow}^* F(Q') \stackrel{\tau}{\rightarrow}^* R$$

(4) or there exists Q' such that $Q \stackrel{a_1}{\Rightarrow} Q'$, and

$$F(Q) \stackrel{\tau}{\rightarrow}^* (\mathtt{new}\, t\, (Q'|T_2)) \stackrel{\tau}{\rightarrow}^* R$$

Case (3.a) $< a_1 > \notin traces(Q')$: In this case we have $R = \mathtt{new}\, t\, (Q'|T_1)$ and

$$\tau \notin \mathbf{init}(\mathtt{new}\, t\, (Q'|T_1)) \quad \text{and} \quad succ \in \mathbf{init}(\mathtt{new}\, t\, (Q'|T_1))$$

which indicates that R cannot simulate 0.

Case (3.b) $< a_1 > \in traces(Q')$: let $Q' \xrightarrow{a_1} R'$. Obviously

$$(< a_2, .., a_n >, \{b_1, .., b_m\}) \notin failures(R')$$

By induction hypothesis we know that $\mathtt{new}\, t\, (R'|T_2)$ cannot simulate 0. Because $\mathtt{new}\, t\, (Q'|T_1) \xrightarrow{\tau} \mathtt{new}\, t\, (R'|T_2)$ is the only τ transition which $\mathtt{new}\, t\, (Q'|T_1)$ can perform, we conclude that $\mathtt{new}\, t\, (Q'|T_1)$ cannot simulate 0.

Case (4): Similar to Case (3b). □

7 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. Dijksra.
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)