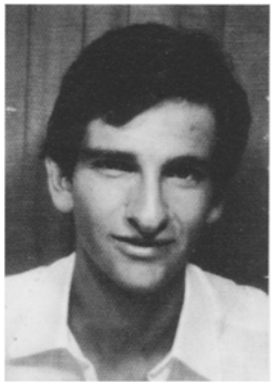
Distributed Computing (1988) 3:9–18

DISTRIBUTED COMPUTING

A state-based approach to communicating processes

Mark B. Josephs*
IBM Research Division, Yorktown



Mark Josephs joined the Programming Research Group at Oxford University in 1983, upon graduating from London University with a degree in Mathematics. One year later he was awarded the Master's degree in Computation. He received the doctorate in 1986 for his work in functional programming and took up a Visiting Scientist post at IBM Yorktown Heights in their Specification and Design Languages Group. He has now returned to the P.R.G. as a S.E.R.C. Research Officer.

Abstract. Communicating processes, which may exhibit nondeterministic behaviour, are specified as state-transition systems. Equivalence and refinement relations are defined in terms of the failures model of processes. Downward and upward simulation are considered as proof methods for refinement. Various operators on processes are defined and their refinement rules established.

Key words: Communicating processes – State-transition systems – Refinement – Simulation – Failures model

1 Introduction

In (Hoare 1985), a theory of communicating processes (CSP) is expounded. It introduces a notation in which it is possible to specify and reason about distributed systems. In particular, a number of processes operating concurrently may be looked upon as a single process. Furthermore, no mention is made of internal actions in describing the behaviour of a process. Another feature of CSP is that explicit modelling of the state of a process is avoided.

CSP provides us with a large collection of algebraic laws with which we are able to prove processes equivalent. A mathematical model of processes, called the *failures model* (Brookes et al. 1984; Brookes and Roscoe 1984), has been used to establish the correctness of these laws. (CCS (Milner 1980) and ACP_τ (Baeten et al. 1985) also support proof by algebraic transformation and are similar to CSP in several other respects.)

The fact that CSP does not facilitate modelling of the state may not, however, be advantageous. Experience with the specification languages VDM (Jones 1986) and Z (Hayes 1987), for example, has shown that such modelling can be a very convenient way of describing complex systems.

It also seems reasonable to criticize CSP on the following grounds. Although the laws enable us to prove that one process is equivalent to another, we are in fact at liberty to make design decisions that require us to prove that one process is *refined* by another. Refinement is more difficult to prove than equivalence in CSP.

In contrast to CSP, the approach taken in this paper is to make process state explicit and to use state-transitions as a means of specifying how a process behaves. Machines are defined in a similar way in automata theory. There is also a strong resemblance to the synchronisation trees of (Milner 1980).

The state-based approach is, nevertheless, just a matter of convenience. We are primarily concerned with the *events* (communications) in which processes engage. Two processes may communi-

* Current address: M.B. Josephs, Oxford University Computing Laboratory, Programming Research Group, Keble Road, Oxford OX1 3QD, UK

cate with their environment in the same way, even though their state-based specifications seem very different. Indeed, the failures model remains the ultimate model of a process. That is, every state-based specification can be identified with a process in the failures model.

The main contribution of this paper is the provision of a number of simple rules by means of which one process can be shown to be a refinement of another. These rules are based on the idea of *downward* and *upward simulation* (He et al. 1986). That paper was concerned with the refinement of abstract data types into concrete data types. Jifeng He (1988) has independently adapted the method to process refinement, with similar results to our own.

This paper is organized as follows:

Section 2 applies a state-based approach to the specification and refinement of non-divergent processes. A collection of refinement rules is presented and various CSP operators on processes (Hoare 1985) are investigated.

The theoretical underpinning is provided in Sect. 3. There, the concept of failures is introduced and equivalence and refinement relations between processes are defined. Soundness and completeness results are obtained, the latter being achieved by exploring a normal form for processes.

2 Specification and refinement

In this section we consider a way of specifying non-divergent systems, together with a set of refinement rules. A theoretical foundation for this approach is developed in Sect. 3.

We begin with the definition of a process:

Definition 2.1. A process is a tuple $(A, S, \longrightarrow, R)$ with **alphabet** A – the set of (all possible) events in which it may engage; **state-space** S – the set of states of the system; **transition relation** $\longrightarrow \subseteq S \times A \times S$ – the set of transitions $\sigma \xrightarrow{e} \sigma'$; **region** $R \subseteq S$, $R \neq \phi$ – a specification of the initial state of the system. (A and S need not be finite.) □

Thus, in specifying a process, if we do not care whether it is initially in state σ_0, σ_1 or σ_2, we simply define R to be $\{\sigma_0, \sigma_1, \sigma_2\}$. Note also that a process in state σ can only engage in event e if a transition $\sigma \xrightarrow{e} \sigma'$ to some state σ' is possible. In fact, there may be several such transitions, for example, $\sigma \xrightarrow{e} \sigma'$ and $\sigma \xrightarrow{e} \sigma''$, if it does not matter whether the new state is σ' or σ''.

Before considering some examples of processes, we define two useful concepts: the next possible events in which a process P may engage when in a particular state σ are given by $\text{next}_P(\sigma)$; those events in which P cannot next engage are given by its complement, $\overline{\text{next}_P}(\sigma)$. Formally:

Definition 2.2. The functions $\text{next}_P, \overline{\text{next}_P}: S \longrightarrow \mathbf{P}A$ are defined by

$$\text{next}_P(\sigma) \stackrel{\text{def}}{=} \{e \in A \mid \exists \sigma' \in S.\ \sigma \xrightarrow{e} \sigma'\}$$

$$\overline{\text{next}_P}(\sigma) \stackrel{\text{def}}{=} A - \text{next}_P(\sigma). \quad \square$$

It is also convenient to extend the transition relation $\longrightarrow$ to sequences of events. That is, $\longrightarrow \subseteq S \times A^* \times S$, where

$$\sigma \xrightarrow{\varepsilon} \sigma' \Leftrightarrow \sigma = \sigma'$$

$$\sigma \xrightarrow{st} \sigma'' \Leftrightarrow \exists \sigma' \in S.\ \sigma \xrightarrow{s} \sigma' \wedge \sigma' \xrightarrow{t} \sigma''$$

with ε denoting the empty sequence.

2.1 Examples

The following examples have been adapted from (Hoare 1985).

1. The process STOP_A never engages in any event. One possible definition is $(A, \{0\}, \phi, \{0\})$. We have that $\text{next}(0) = \phi$ for this process.
2. The process $(A, \{0, 1\}, \{0 \xrightarrow{a} 1\}, \{0\})$ first engages in the event $a \in A$ and then stops. In CSP this would be expressed as $a \longrightarrow \text{STOP}_A$. Here, $\text{next}(0) = \{a\}$ and $\text{next}(1) = \phi$.
3. The process CLOCK ticks for ever. A suitable definition is

   $(\{\text{tick}\}, \{\text{on}\}, \{\text{on} \xrightarrow{\text{tick}} \text{on}\}, \{\text{on}\})$

   and we have that $\text{next}(\text{on}) = \{\text{tick}\}$.
4. An object starts on the ground, and may move up. At any time thereafter it may move up or down, except that when it is on the ground it cannot move any further down. But when it is on the ground, it may move around. This behaviour is specified by a process with alphabet {up, down, around}, state-space **N** (the natural numbers), region $\{0\}$ and transition relation defined by

$i \xrightarrow{\text{up}} i' \Leftrightarrow i' = i+1$

$i \xrightarrow{\text{down}} i' \Leftrightarrow i > 0 \wedge i' = i-1$

$i \xrightarrow{\text{around}} i' \Leftrightarrow i = 0 \wedge i' = 0.$

For this process, $\text{next}(0) = \{\text{up, around}\}$ and $\text{next}(i+1) = \{\text{up, down}\}$.

5. A variable, taking values in some set V, supports read and write operations. Suppose we decide to leave unspecified its initial value and its value after each (destructive) read operation. A process with this behaviour is defined by

$(\{\text{read. } v, \text{ write. } v \mid v \in V\}, V, \longrightarrow, V)$

where

$x \xrightarrow{\text{read.}v} x' \Leftrightarrow v = x$

$x \xrightarrow{\text{write.}v} x' \Leftrightarrow v = x'.$

We have that

$\text{next}(x) = \{\text{read. } x\} \cup \{\text{write. } v \mid v \in V\}.$

2.2 Traces

From the specification of a process it is possible to determine its *traces*, those sequences of events in which it may engage. In the following let $P = (A, S, \longrightarrow, R)$. Then:

Definition 2.3. The traces of a process P are given by

$\text{traces}(P) \stackrel{\text{def}}{=} \{s \in A^* \mid \exists \sigma \in R, \sigma' \in S.\ \sigma \xrightarrow{s} \sigma'\}$. □

Proposition 2.1. *The set of traces of a process has the following properties:*

1. $\varepsilon \in \text{traces}(P)$.
2. $st \in \text{traces}(P) \Rightarrow s \in \text{traces}(P)$. □

The above properties are in fact taken as axioms in the traces model (Hoare 1980).

It is also useful to be able to reason about the behaviour of a process after it has engaged in a particular sequence of events.

Definition 2.4. For any $s \in \text{traces}(P)$,

$P \text{ after } s \stackrel{\text{def}}{=} (A, S, \longrightarrow, R')$

where $\sigma' \in R' \Leftrightarrow \exists \sigma \in R.\ \sigma \xrightarrow{s} \sigma'$. □

Proposition 2.2. *For any* $s \in \text{traces}(P)$,

$t \in \text{traces}(P \text{ after } s) \Leftrightarrow st \in \text{traces}(P)$. □

2.3 Refinement

In designing a system, we proceed through a succession of refinement steps. A simple example involves the design of a change-giving machine (Hoare 1985). We might begin with a specification that permits several different combinations of change to be returned, over which the user of the machine has no control. A refinement step could then involve the elimination of this nondeterminism: we might decide that a particular combination of change should always be returned.

We record the refinement of a process P_1 into a process P_2 by writing $P_1 \sqsubseteq P_2$. One condition that must be met is that the two processes have the same alphabet. The other condition can be stated informally as follows: every behaviour that is possible for P_2 must also be possible for P_1. This means that if s is a trace of P_2, then it is also a trace of P_1. It also means that if P_2 is offered some choice of events by its environment, but it cannot engage in any of them next, then it is possible for P_1 to behave likewise. Fortunately, it is not necessary to try to analyse the behaviour of the processes in this way. Instead, one of the following refinement rules should be applied at each step.

We now present three methods for proving refinements correct:

Our first rule may be used when refining a process into a second process which has been defined over the same state space, that is, $P_i = (A, S, \longrightarrow_i, R_i)$, $i = 1, 2$.

Rule 2.1 (Strengthening a specification). $P_1 \sqsubseteq P_2$ *if*

1. $\forall \sigma \in S.\ \text{next}_{P_1}(\sigma) = \text{next}_{P_2}(\sigma)$
   In any state, the processes must be able to engage in the same events.
2. $\longrightarrow_1 \supseteq \longrightarrow_2$
   Every transition of P_2 must also be a transition of P_1.
3. $R_1 \supseteq R_2$
   Every possible initial state of P_2 must also be a possible initial state of P_1. □

For example, noting that in state 0 only event a can happen next and in state 1 only b, it is easy to see that

$(\{a, b\}, \{0, 1\}, \{0 \xrightarrow{a} 0, 0 \xrightarrow{a} 1, 1 \xrightarrow{b} 0, 1 \xrightarrow{b} 1\}, \{0, 1\})$
$\sqsubseteq (\{a, b\}, \{0, 1\}, \{0 \xrightarrow{a} 1, 1 \xrightarrow{b} 0\}, \{0\})$.

The other two rules enable us to change the state-space in performing the refinement. To apply these rules, we must establish a correspondence between the states of the two processes. In the following let $P_i=(A, S_i, \longrightarrow_i, R_i)$, $i=1, 2$.

Rule 2.2 (Downward simulation). *$P_1 \sqsubseteq P_2$ if there is a relation $D \subseteq S_1 \times S_2$ such that*

1. $\forall \sigma_1 \in S_1, \sigma_2 \in S_2.$
   $\sigma_1 D \sigma_2 \Rightarrow \text{next}_{P_1}(\sigma_1) = \text{next}_{P_2}(\sigma_2)$.
   If the processes are in corresponding states, they must be able to engage in the same events.
2. $\forall \sigma_1 \in S_1, \sigma_2, \sigma_2' \in S_2, e \in A.$
   $\sigma_1 D \sigma_2 \wedge \sigma_2 \xrightarrow{e}_2 \sigma_2'$
   $\Rightarrow \exists \sigma_1' \in S_1 . \sigma_1 \xrightarrow{e}_1 \sigma_1' \wedge \sigma_1' D \sigma_2'$.
   If the processes are in corresponding states and P_2 can engage in e, P_1 must be able to engage in e in such a way that the processes remain in corresponding states.
3. $\forall \sigma_2 \in R_2 . \exists \sigma_1 \in R_1 . \sigma_1 D \sigma_2$.
   Every possible initial state of P_2 must correspond to a possible initial state of P_1. □

For example, by taking iDj to be $i=0 \vee i=1$ and checking conditions 1–3 above, we can see that

$$(\{a\}, \{0, 1, 2\}, \{0 \xrightarrow{a} 1, 1 \xrightarrow{a} 0\}, \{0\})$$
$$\sqsubseteq (\{a\}, \{0\}, \{0 \xrightarrow{a} 0\}, \{0\}).$$

The basic idea in the last rule was that P_1 can simulate the behaviour of P_2 so that they always remain in corresponding states. The following rule is far less obvious. This time the idea is that if P_2 has reached some state after engaging in a sequence of events, then it is possible to find a corresponding state of P_1; from this state P_1 can simulate P_2 *backwards*, retracing the sequence of events until it has reached an initial state.

Rule 2.3 (Upward simulation). *$P_1 \sqsubseteq P_2$ if there is a relation $U \subseteq S_2 \times S_1$ such that*

1. $\forall \sigma_2 \in S_2 . \exists \sigma_1 \in S_1 .$
   $\sigma_2 U \sigma_1 \wedge \text{next}_{P_1}(\sigma_1) \subseteq \text{next}_{P_2}(\sigma_2)$.
   For every state of P_2, there is a corresponding state of P_1 such that, if P_1 can next engage in e, so can P_2.
2. $\forall \sigma_1' \in S_1, \sigma_2, \sigma_2' \in S_2, e \in A.$
   $\sigma_2 \xrightarrow{e}_2 \sigma_2' \wedge \sigma_2' U \sigma_1'$
   $\Rightarrow \exists \sigma_1 \in S_1 . \sigma_2 U \sigma_1 \wedge \sigma_1 \xrightarrow{e}_1 \sigma_1'$.
   If the processes are in corresponding states and P_2 could have reached its state by engaging in e, then so could P_1 in such a way that their previous states also corresponded.
3. $\forall \sigma_1 \in S_1, \sigma_2 \in R_2 . \sigma_2 U \sigma_1 \Rightarrow \sigma_1 \in R_1$.
   If the processes are in corresponding states and P_2 may have been in its state initially, then P_1 may have been in its state initially. □

For example, by taking jUi to be *true* and checking the above conditions, we can see that

$$(\{a, b\}, \{0, 1\}, \{0 \xrightarrow{a} 0, 0 \xrightarrow{a} 1, 1 \xrightarrow{b} 0, 1 \xrightarrow{b} 1\}, \{0, 1\})$$
$$\sqsubseteq (\{a, b\}, \{0\}, \{0 \xrightarrow{a} 0, 0 \xrightarrow{b} 0\}, \{0\}).$$

Note that this refinement could not have been proved by Rule 2.2. Conversely, in the previous example, the refinement could not have been proved by Rule 2.3.

It can be seen that Rule 2.1 is just a special case of Rules 2.2 and 2.3 in which D or U is the identity relation.

2.4 Operators

We now consider three of Hoare's CSP operators: the first operator ($|||$) allows several processes, executing independently and concurrently, to be regarded as a single process; the second ($||$) deals similarly with processes that synchronize over certain events; the third ($\backslash$) permits hiding of events, so that they may occur as internal, unobservable state-transitions of the system. Definitions are provided for each of these operators. Refinement rules (derived from Rule 2.2) are also given.

With these operators, we are able to express the refinement of a specification into a distributed system. For example, a first refinement step may be of a process P, specifying a system with an input and an output communication channel, into a process Q which has an internal communication channel C in addition to the input and output channels. This design decision would be recorded as

$$P \sqsubseteq Q \backslash C$$

and be subject to a proof of correctness. Q itself might then be refined into two processes R and S which communicate over the channel C:

$$Q \sqsubseteq R \,\|\, S.$$

Here, R may handle the input channel and S the output channel. Thus, the original single process

P has been decomposed into a distributed system, specified as $(R \| S)\backslash C$. That is:

$P \sqsubseteq (R \| S)\backslash C$.

The processes R and S are now themselves candidates for refinement.

2.4.1 The interleave operator

For any processes $P_i = (A_i, S_i, \longrightarrow_i, R_i)$, $i = 1, 2$, we can interleave their behaviours as follows:

Definition 2.5. $P_1 ||| P_2 \stackrel{\text{def}}{=} (A_1 \cup A_2,\ S_1 \times S_2,\ \longrightarrow,\ R_1 \times R_2)$ where $\forall\, \sigma_1, \sigma_1' \in S_1, \sigma_2, \sigma_2' \in S_2, e \in A_1 \cup A_2$.

$$\sigma_1 \sigma_2 \xrightarrow{e} \sigma_1' \sigma_2' \Leftrightarrow (e \in A_1 \wedge \sigma_1 \xrightarrow{e}_1 \sigma_1' \wedge \sigma_2 = \sigma_2') \vee (e \in A_2 \wedge \sigma_2 \xrightarrow{e}_2 \sigma_2' \wedge \sigma_1 = \sigma_1').$$ □

To prove that $P \sqsubseteq P_1 ||| P_2$, we could first construct $P_1 ||| P_2$ as above and then apply a refinement rule. However, it may often be easier to apply the following rule directly. Let $P = (A_1 \cup A_2, S, \longrightarrow, R)$. Then:

Rule 2.4. *$P \sqsubseteq P_1 ||| P_2$ if there is a relation $D \subseteq S \times (S_1 \times S_2)$ satisfying*

1. $\forall \sigma \in S, \sigma_1 \in S_1, \sigma_2 \in S_2$.

   $\sigma D \sigma_1 \sigma_2 \Rightarrow \text{next}_P(\sigma) = \text{next}_{P_1}(\sigma_1) \cup \text{next}_{P_2}(\sigma_2)$.

2. *(a)* $\forall \sigma \in S, \sigma_1, \sigma_1' \in S_1, \sigma_2 \in S_2, e \in A_1$.

   $\sigma D \sigma_1 \sigma_2 \wedge \sigma_1 \xrightarrow{e}_1 \sigma_1'$

   $\Rightarrow \exists\, \sigma' \in S.\ \sigma \xrightarrow{e} \sigma' \wedge \sigma' D \sigma_1' \sigma_2$.

   (b) $\forall \sigma \in S, \sigma_1 \in S_1, \sigma_2, \sigma_2' \in S_2, e \in A_2$.

   $\sigma D \sigma_1 \sigma_2 \wedge \sigma_2 \xrightarrow{e}_2 \sigma_2'$

   $\Rightarrow \exists\, \sigma' \in S.\ \sigma \xrightarrow{e} \sigma' \wedge \sigma' D \sigma_1 \sigma_2'$.

3. $\forall \sigma_1 \in R_1, \sigma_2 \in R_2.\ \exists \sigma \in R.\ \sigma D \sigma_1 \sigma_2$. □

For example, it is easy to show that $P_{a,b} \sqsubseteq P_{a,b} ||| P_{a,b}$ where, for any events x, y,

$P_{x,y} = (\{x, y\}, \mathbf{N}, \{i \xrightarrow{x} i+1, i+1 \xrightarrow{y} i \mid i \in \mathbf{N}\}, \{0\})$.

(Take $iD(j, k)$ to be $i = j + k$.)

2.4.2 The parallel operator

The parallel composition of P_1 and P_2 may be constructed as follows:

Definition 2.6. $P_1 \| P_2 \stackrel{\text{def}}{=} (A_1 \cup A_2,\ S_1 \times S_2,\ \longrightarrow,\ R_1 \times R_2)$ where $\forall\, \sigma_1, \sigma_1' \in S_1, \sigma_2, \sigma_2' \in S_2, e \in A_1 \cup A_2$.

$$\sigma_1 \sigma_2 \xrightarrow{e} \sigma_1' \sigma_2'$$
$$\Leftrightarrow (e \in A_1 - A_2 \wedge \sigma_1 \xrightarrow{e}_1 \sigma_1' \wedge \sigma_2 = \sigma_2') \vee$$
$$(e \in A_2 - A_1 \wedge \sigma_2 \xrightarrow{e}_2 \sigma_2' \wedge \sigma_1 = \sigma_1') \vee$$
$$(e \in A_1 \cap A_2 \wedge \sigma_1 \xrightarrow{e}_1 \sigma_1' \wedge \sigma_2 \xrightarrow{e}_2 \sigma_2').$$ □

For example, with $P_{x,y}$ defined as above, the parallel composition of $P_{a,c}$ and $P_{c,b}$ is given by

$$P_{a,c} \| P_{c,b} \stackrel{\text{def}}{=}$$
$$(\{a, b, c\}, \mathbf{N} \times \mathbf{N},$$
$$\{(j, k) \xrightarrow{a} (j+1, k),$$
$$(j, k+1) \xrightarrow{b} (j, k),$$
$$(j+1, k) \xrightarrow{c} (j, k+1) \mid j, k \in \mathbf{N}\},$$
$$\{(0, 0)\}).$$

Again, there is a special refinement rule that can be used. Let $P = (A_1 \cup A_2, S, \longrightarrow, R)$. Then:

Rule 2.5. *$P \sqsubseteq P_1 \| P_2$ if there is a relation $D \subseteq S \times (S_1 \times S_2)$ satisfying*

1. $\forall \sigma \in S, \sigma_1 \in S_1, \sigma_2 \in S_2$.

   $\sigma D \sigma_1 \sigma_2 \Rightarrow \overline{\text{next}}_P(\sigma) = \overline{\text{next}}_{P_1}(\sigma_1) \cup \overline{\text{next}}_{P_2}(\sigma_2)$.

2. *(a)* $\forall \sigma \in S, \sigma_1, \sigma_1' \in S_1, \sigma_2 \in S_2, e \in A_1 - A_2$.

   $\sigma D \sigma_1 \sigma_2 \wedge \sigma_1 \xrightarrow{e}_1 \sigma_1'$

   $\Rightarrow \exists\, \sigma' \in S.\ \sigma \xrightarrow{e} \sigma' \wedge \sigma' D \sigma_1' \sigma_2$.

   (b) $\forall \sigma \in S, \sigma_1 \in S_1, \sigma_2, \sigma_2' \in S_2, e \in A_2 - A_1$.

   $\sigma D \sigma_1 \sigma_2 \wedge \sigma_2 \xrightarrow{e}_2 \sigma_2'$

   $\Rightarrow \exists\, \sigma' \in S.\ \sigma \xrightarrow{e} \sigma' \wedge \sigma' D \sigma_1 \sigma_2'$.

   (c) $\forall \sigma \in S, \sigma_1, \sigma_1' \in S_1, \sigma_2, \sigma_2' \in S_2, e \in A_1 \cap A_2$.

   $\sigma D \sigma_1 \sigma_2 \wedge \sigma_1 \xrightarrow{e}_1 \sigma_1' \wedge \sigma_2 \xrightarrow{e}_2 \sigma_2'$

   $\Rightarrow \exists\, \sigma' \in S.\ \sigma \xrightarrow{e} \sigma' \wedge \sigma' D \sigma_1' \sigma_2'$.

3. $\forall \sigma_1 \in R_1, \sigma_2 \in R_2, \exists \sigma \in R, \sigma D \sigma_1 \sigma_2$. □

2.4.3 The hiding operator

Finally, we provide a definition and a refinement rule for the hiding operator. (Hiding a set C of events can be thought of as declaring a local channel C.)

For any process $P = (A, S, \longrightarrow, R)$ and set of events $C \subseteq A$:

Definition 2.7. $P\backslash C$ is well-defined only if

$\forall s \in \text{traces}(P).\ \neg \forall n \in \mathbf{N}.\ \exists t \in C^*.$
$\#t > n \wedge st \in \text{traces}(P).$

If this condition is met, then $P\backslash C$ is defined by

$$P\backslash C \stackrel{\text{def}}{=} (A', S, \longrightarrow', R')$$

where

1. $A' = A - C.$
2. $\forall \sigma, \sigma' \in S, e \in A'.$
$$\sigma \stackrel{e}{\longrightarrow}' \sigma' \Leftrightarrow \exists s, t \in C^*.\ \sigma \stackrel{set}{\longrightarrow} \sigma'.$$
3. $\forall \sigma' \in S.$
$$\sigma' \in R' \Leftrightarrow \exists \sigma \in R, s \in C^*.\ \sigma \stackrel{s}{\longrightarrow} \sigma'. \quad \square$$

In the above definition, the restriction on P ensures that $P\backslash C$ can never diverge (engage indefinitely in hidden events).

Let $P_i = (A_i, S_i, \longrightarrow_i, R_i)$, $i = 1, 2$, and $C \subseteq A_2$ such that $A_1 = A_2 - C$ and $P_2\backslash C$ is well-defined. Then:

Rule 2.6. *$P_1 \sqsubseteq P_2\backslash C$ if there is a relation $D \subseteq S_1 \times S_2$ satisfying*

1. $\forall \sigma_1 \in S_1, \sigma_2 \in S_2, e \in A_1.$
$$\sigma_1 D \sigma_2 \wedge e \in \text{next}_{P_1}(\sigma_1)$$
$$\Rightarrow \exists \sigma_2' \in S_2, s \in C^*.\ \sigma_2 \stackrel{se}{\longrightarrow}_2 \sigma_2'.$$
2. $\forall \sigma_1 \in S_1, \sigma_2, \sigma_2' \in S_2, s, t \in C^*, e \in A_1.$
$$\sigma_1 D \sigma_2 \wedge \sigma_2 \stackrel{set}{\longrightarrow}_2 \sigma_2'$$
$$\Rightarrow \exists \sigma_1' \in S_1.\ \sigma_1 \stackrel{e}{\longrightarrow}_1 \sigma_1' \wedge \sigma_1' D \sigma_2'.$$
3. $\forall \sigma_2 \in R_2, \sigma_2' \in S_2, s \in C^*.$
$$\sigma_2 \stackrel{s}{\longrightarrow}_2 \sigma_2' \Rightarrow \exists \sigma_1 \in R_1.\ \sigma_1 D \sigma_2'. \quad \square$$

By adding an *invariant*, condition 2. (a) below, the above rule may be smplified:

Rule 2.7. *$P_1 \sqsubseteq P_2\backslash C$ if there is a relation $D \subseteq S_1 \times S_2$ satisfying*

1. $\forall \sigma_1 \in S_1, \sigma_2 \in S_2, e \in A_1.$
$$\sigma_1 D \sigma_2 \wedge e \in \text{next}_{P_1}(\sigma_1)$$
$$\Rightarrow \exists \sigma_2' \in S_2, s \in C^*.\ \sigma_2 \stackrel{se}{\longrightarrow}_2 \sigma_2'.$$
2. *(a)* $\forall \sigma_1 \in S_1, \sigma_2, \sigma_2' \in S_2, e \in C.$
$$\sigma_1 D \sigma_2 \wedge \sigma_2 \stackrel{e}{\longrightarrow}_2 \sigma_2' \Rightarrow \sigma_1 D \sigma_2'.$$
(b) $\forall \sigma_1 \in S_1, \sigma_2, \sigma_2' \in S_2, e \in A_1.$
$$\sigma_1 D \sigma_2 \wedge \sigma_2 \stackrel{e}{\longrightarrow}_2 \sigma_2'$$
$$\Rightarrow \exists \sigma_1' \in S_1.\ \sigma_1 \stackrel{e}{\longrightarrow}_1 \sigma_1' \wedge \sigma_1' D \sigma_2'.$$
3. $\forall \sigma_2 \in R_2.\ \exists \sigma_1 \in R_1.\ \sigma_1 D \sigma_2. \quad \square$

For example, in order to prove that

$$P_{a,b} \sqsubseteq (P_{a,c} \parallel P_{c,b})\backslash \{c\}$$

take $iD(j, k)$ to be $i = j + k$ and check conditions 1–3 above. (We can show that hiding c is well-defined, as follows. After any trace s, j is bounded from above by $\#s$. Each event c decreases j. Well-definedness follows by taking $n > \#s$.)

Rules 2.4–7 are only a small selection of the refinement rules available. Similar rules for upward simulation can be formulated. Rules can also be provided for other special cases, for example, $P_1 \parallel P_2 \sqsubseteq P$ and $P_1\backslash C_1 \sqsubseteq P_2\backslash C_2$.

3 Theoretical foundations

In this section, we introduce the failures model (for non-divergent processes). The refinement relation is formally defined as an ordering on failures. The operators defined in Sect. 2 are shown to have the usual failures semantics.

We are then in a position to analyse the proof methods of downward and upward simulation. (These methods can be compared with the use of *bi-simulation* (Park 1981) to prove the equivalence of processes in CCS (Milner 1985).)

The soundness of the methods is demonstrated first. Next, they are employed in the derivation of a normal form for processes. This concept of normal form enables us to prove that the methods are together complete.

3.1 Failures

A process P might engage in some sequence s of events and then refuse to engage in any event from some set X offered by its environment. Such a refusal is possible if the process has reached a state in which it cannot next engage in any event in X. The pair sX is called a failure of P.

Definition 3.1. The failures of a process P are given by

$$\text{failures}(P) \stackrel{\text{def}}{=} \{sX \in A^* \times \mathbf{P}A \mid \exists \sigma \in R, \sigma' \in S. \ \sigma \xrightarrow{s} \sigma' \wedge X \subseteq \overline{\text{next}}_P(\sigma')\}. \ \square$$

Proposition 3.1. *The failures of a process enjoy the following properties:*

1. $s \in \text{traces}(P) \Leftrightarrow s\phi \in \text{failures}(P)$.
2. $X \subseteq Y \wedge sY \in \text{failures}(P) \Rightarrow sX \in \text{failures}(P)$.
3. $sX \in \text{failures}(P) \wedge (\forall e \in Y.\ se \notin \text{trace}(P)) \Rightarrow s(X \cup Y) \in \text{failures}(P)$. $\square$

These properties are axioms in the failures model (Brookes and Roscoe 1984).

Two processes are equivalent if they have the same failures. That is:

Definition 3.2. $P_1 \equiv P_2$ if $\text{failures}(P_1) = \text{failures}(P_2)$. $\square$

P_1 is refined by P_2 if the behaviour of P_2 is always consistent with that of P_1. That is:

Definition 3.3. $P_1 \sqsubseteq P_2$ if $\text{failures}(P_1) \supseteq \text{failures}(P_2)$. $\square$

Note that $\sqsubseteq$ is a partial order over processes with alphabet A.

Finally, we examine the three operators on processes. Their meaning, in terms of failures, is consistent with that given in (Hoare 1985). The operators are monotonic and, hence, can be used in stepwise refinement.

Theorem 3.1.

$$\text{failures}(P_1 \,|||\, P_2) = \{sX \in A^* \times \mathbf{P}A \mid \exists s_1 X_1 \in \text{failures}(P_1), s_2 X_2 \in \text{failures}(P_2).\ s \text{ interleaves } (s_1, s_2) \wedge X \cap A_1 \subseteq X_1 \wedge X \cap A_2 \subseteq X_2\}. \ \square$$

(Informally, s interleaves (t, u) whenever the trace s is some interleaving of the traces t and u.)

From Theorem 3.1, we can see that $|||$ is a monotonic operator. That is:

Corollary 3.1. *If* $P_1 \sqsubseteq P_1'$ *and* $P_2 \sqsubseteq P_2'$ *then* $P_1 \,|||\, P_2 \sqsubseteq P_1' \,|||\, P_2'$. $\square$

Theorem 3.2.

$$\text{failures}(P_1 \parallel P_1) = \{sX \in A^* \times \mathbf{P}A \mid \exists s_1 X_1 \in \text{failures}(P_1), s_2 X_2 \in \text{failures}(P_2).\ s_1 = s|_{A_1} \wedge s_2 = s|_{A_2} \wedge X \subseteq X_1 \cup X_2\}. \ \square$$

(The trace $s|_A$ is formed by restricting the trace s to those events in A.)

Corollary 3.2. *If* $P_1 \sqsubseteq P_1'$ *and* $P_2 \sqsubseteq P_2'$ *then* $P_1 \parallel P_2 \sqsubseteq P_1' \parallel P_2'$. $\square$

Theorem 3.3. *If* $P \backslash C$ *is well-defined then*

$$\text{failures}(P \backslash C) = \{s' X' \in A'^* \times \mathbf{P}A' \mid \exists sX \in \text{failures}(P).\ s' = s|_{A'} \wedge X = X' \cup C\}.$$

Proof. Suppose $s'X' \in \text{failures}(P \backslash C)$. Then, P can engage in a sequence of hidden events interleaved with s' to arrive at some state σ from which it cannot engage in any event in X', even after performing further hidden events first. Because $P \backslash C$ is well-defined, by engaging in hidden events, P can reach from σ a state σ' in which all hidden events are refused. Thus, $X' \cup C \subseteq \overline{\text{next}}_{P \backslash C}(\sigma')$ and so $\exists sX \in \text{failures}(P).\ s' = s|_{A'} \wedge X = X' \cup C$.

Suppose instead $sX \in \text{failures}(P)$, $s' = s|_{A'}$ and $X = X' \cup C$, $X' \subseteq A'$. Then, P can reach after s a state σ with $X \subseteq \overline{\text{next}}_P(\sigma)$. Since σ refuses all hidden events, $X' \subseteq \overline{\text{next}}_{P \backslash C}(\sigma)$. Hence, $s'X' \in \text{failures}(P \backslash C)$. $\square$

Corollary 3.3. *If* $P \sqsubseteq P'$ *and* $P \backslash C$ *is well-defined, then* $P' \backslash C$ *is well-defined and* $P \backslash C \sqsubseteq P' \backslash C$. $\square$

3.2 Downward simulation

Let $P_i = (A, S_i, \longrightarrow_i, R_i)$, $i = 1, 2$:

Definition 3.4. A relation $D \subseteq S_1 \times S_2$ is a downward simulation between P_1 and P_2 if

1. $\forall \sigma_1 \in S_1, \sigma_2 \in S_2.$
   $\sigma_1 D \sigma_2 \Rightarrow \overline{\text{next}}_{P_1}(\sigma_1) \supseteq \overline{\text{next}}_{P_2}(\sigma_2)$.
2. $\forall \sigma_1 \in S_1, \sigma_2, \sigma_2' \in S_2, e \in A.$
   $\sigma_1 D \sigma_2 \wedge \sigma_2 \xrightarrow{e}_2 \sigma_2'$
   $\Rightarrow \exists \sigma_1' \in S_1.\ \sigma_1 \xrightarrow{e}_1 \sigma_1' \wedge \sigma_1' D \sigma_2'$.
3. $\forall \sigma_2 \in R_2.\ \exists \sigma_1 \in R_1.\ \sigma_1 D \sigma_2$. $\square$

Suppose D is a downward simulation. Then:

Lemma 3.1. $\forall \sigma_1 \in S_1, \sigma_2, \sigma_2' \in S_2, s \in A^*.$

$$\sigma_1 D \sigma_2 \wedge \sigma_2 \xrightarrow{s}_2 \sigma_2' \Rightarrow \exists \sigma_1' \in S_1 . \sigma_1 \xrightarrow{s}_1 \sigma_1' \wedge \sigma_1' D \sigma_2'. \quad \square$$

Theorem 3.4. *$P_1 \sqsubseteq P_2$ if there is a downward simulation between P_1 and P_2.*

Proof.

$$s X \in \text{failures}(P_2)$$
$$\Leftrightarrow \exists \sigma_2 \in R_2, \sigma_2' \in S_2 . \sigma_2 \xrightarrow{s}_2 \sigma_2' \wedge X \subseteq \overline{\text{next}}_{P_2}(\sigma_2')$$
$$\Rightarrow \exists \sigma_1 \in R_1, \sigma_2 \in R_2, \sigma_2' \in S_2 . \sigma_1 D \sigma_2 \wedge \sigma_2 \xrightarrow{s}_2 \sigma_2' \wedge X \subseteq \overline{\text{next}}_{P_2}(\sigma_2')$$

by Def. 3.4, condition 3.

$$\Rightarrow \exists \sigma_1 \in R^1, \sigma_1' \in S_1, \sigma_2' \in S_2 . \sigma_1 \xrightarrow{s}_1 \sigma_1' \wedge \sigma_1' D \sigma_2' \wedge X \subseteq \overline{\text{next}}_{P_2}(\sigma_2')$$

by Lemma 3.1

$$\Rightarrow \exists \sigma_1 \in R_1, \sigma_1' \in S_1 . \sigma_1 \xrightarrow{s}_1 \sigma_1' \wedge X \subseteq \overline{\text{next}}_{P_1}(\sigma_1')$$

by Def. 3.4, condition 1

$$\Leftrightarrow s X \in \text{failures}(P_1). \quad \square$$

Rule 2.2 follows from this theorem.

3.3 Upward simulation

Let $P_i = (A, S_i, \longrightarrow_i, R_i)$, $i = 1, 2$.

Definition 3.5. A relation $U \subseteq S_2 \times S_1$ is an upward simulation between P_2 and P_1 if

1. $\forall \sigma_2 \in S_2 .$
   $\exists \sigma_1 \in S_1 . \sigma_2 U \sigma_1 \wedge \overline{\text{next}}_{P_1}(\sigma_1) \supseteq \overline{\text{next}}_{P_2}(\sigma_2).$
2. $\forall \sigma_1' \in S_1, \sigma_2, \sigma_2' \in S_2, e \in A.$
   $\sigma_2 \xrightarrow{e}_2 \sigma_2' \wedge \sigma_2' U \sigma_1' \Rightarrow \exists \sigma_1 \in S_1 . \sigma_2 U \sigma_1 \wedge \sigma_1 \xrightarrow{e}_1 \sigma_1'.$
3. $\forall \sigma_1 \in S_1, \sigma_2 \in R_2 . \sigma_2 U \sigma_1 \Rightarrow \sigma_1 \in R_1. \quad \square$

Suppose U is an upward simulation. Then:

Lemma 3.2. $\forall \sigma_1' \in S_1, \sigma_2 \in R_2, \sigma_2' \in S_2, s \in A^*.$

$$\sigma_2 \xrightarrow{s}_2 \sigma_2' \wedge \sigma_2' U \sigma_1' \Rightarrow \exists \sigma_1 \in R_1 . \sigma_1 \xrightarrow{s}_1 \sigma_1'. \quad \square$$

Theorem 3.5. *$P_1 \sqsubseteq P_2$ if there is an upward simulation between P_2 and P_1.*

Proof.

$$s X \in \text{failures}(P_2)$$
$$\Leftrightarrow \exists \sigma_2 \in R_2, \sigma_2' \in S_2 . \sigma_2 \xrightarrow{s}_2 \sigma_2' \wedge X \subseteq \overline{\text{next}}_{P_2}(\sigma_2')$$
$$\Rightarrow \exists \sigma_1' \in S_1, \sigma_2 \in R_2, \sigma_2' \in S_2 . \sigma_2 \xrightarrow{s}_2 \sigma_2' \wedge \sigma_2' U \sigma_1' \wedge X \subseteq \overline{\text{next}}_{P_1}(\sigma_1')$$

by Def. 3.5, condition 1

$$\Rightarrow \exists \sigma_1 \in R_1, \sigma_1' \in S_1 . \sigma_1 \xrightarrow{s}_1 \sigma_1' \wedge X \subseteq \overline{\text{next}}_{P_1}(\sigma_1')$$

by Lemma 3.2

$$\Leftrightarrow s X \in \text{failures}(P_1). \quad \square$$

Rule 2.3 follows from this theorem.

3.4 Readiness and normal form

A process P might engage in a trace s and arrive in a state for which Y is the set of events that are possible next. The set of all such pairs sY defines the readiness of P (Olderog and Hoare 1986).

Definition 3.6. The readiness of a process P is defined by

$$\text{readiness}(P) \stackrel{\text{def}}{=} \{ s Y \in A^* \times \mathbf{P} A \mid \exists \sigma \in R, \sigma' \in S . \sigma \xrightarrow{s} \sigma' \wedge Y = \text{next}_P(\sigma') \}. \quad \square$$

The following theorem states an important relationship between readiness and failures.

Theorem 3.6. $\text{failures}(P_1) \supseteq \text{failures}(P_2)$ *if and only if*

$$\forall s_2\ Y_2 \in \text{readiness}(P_2).$$
$$\exists s_1\ Y_1 \in \text{readiness}(P_1) . s_1 = s_2 \wedge Y_1 \subseteq Y_2. \quad \square$$

Another theorem provides further insight into downward simulation.

Theorem 3.7. $\text{readiness}(P_1) \supseteq \text{readiness}(P_2)$ *if there is a downward simulation between P_1 and P_2.* $\square$

Our reason for introducing readiness, however, is that it turns out to be useful in defining a normal form for a process, as we see next.

The idea is that each sY in the readiness of a process P can be taken as a state of its normal

form $P^\natural$. This state can only be reached after the trace s. The effect of this is to eliminate all the unreachable states of P. States of P are identified that can be reached after the same trace and which are able to engage in the same events next. Also, a state of P that can be reached after different traces gives rise to one state in $P^\natural$ for each such trace.

Definition 3.7. The normal form $P^\natural$ of a process P is given by

$$P^\natural \stackrel{\text{def}}{=} (A, S^\natural, \longrightarrow^\natural, R^\natural)$$

where

1. $S^\natural = \text{readiness}(P)$.
2. $\forall\, sY, s'Y' \in S^\natural, e \in A.$

   $sY \xrightarrow{e}{}^\natural s'Y' \Leftrightarrow s' = se \wedge e \in Y.$
3. $\forall\, sY \in S^\natural.$

   $sY \in R^\natural \Leftrightarrow s = \varepsilon.$ □

Lemma 3.3.

$\text{next}_{P^\natural}(sY) = Y.$ □

Lemma 3.4. *There is an upward simulation between $P^\natural$ and P.*

Proof. $sYU\sigma' = \exists\, \sigma \in R.\ \sigma \xrightarrow{s} \sigma'$ is such a simulation. □

Lemma 3.5. *There is a downward simulation between $P^\natural$ and P.*

Proof. $sYD\sigma' = \exists\, \sigma \in R.\ \sigma \xrightarrow{s} \sigma' \wedge Y = \text{next}_P(\sigma')$ is such a simulation. □

The normal form theorem follows immediately from the last two lemmas:

Theorem 3.8.

$P \equiv P^\natural.$ □

3.5 Completeness

Suppose it is the case that $P_1 \sqsubseteq P_2$. The question arises as to whether it is possible to prove the correctness of this refinement by exhibiting suitable simulations.

In fact, such a proof is always possible. $P_1 \sqsubseteq P_1^\natural$ can be proved by upward simulation. $P_1^\natural \sqsubseteq P_2^\natural$ can also be proved by upward simulation – see below. Finally, $P_2^\natural \sqsubseteq P_2$ can be proved by downward simulation. $P_1 \sqsubseteq P_2$ follows from the fact that refinement is transitive.

Lemma 3.6. *If $P_1 \sqsubseteq P_2$ then there is an upward simulation between $P_2^\natural$ and $P_1^\natural$.*

Proof. $s_2\, Y_2 U s_1\, Y_1 = (s_1 = s_2)$ is such a simulation. □

Thus, we have the following completeness result:

Theorem 3.9. *Rules 2.2 and 2.3 are sufficient to prove that P_1 is refined by P_2 whenever this is in fact the case.* □

(A similar result has been obtained for data refinement by He et al. (1986).)

4 Conclusion

A state-based approach has been taken to the specification and refinement of communicating processes. This might form the basis of a design methodology for distributed systems. In particular, Rules 2.1–7 may be of some assistance in developing a distributed implementation that must satisfy a given specification.

On the theoretical side, we have demonstrated the soundness (and completeness) of these methods with respect to the failures model of communicating processes. In so doing, the relationships between state-transition systems and the failures model and between simulation and refinement have been clarified.

Acknowledgements. I am most grateful to Tony Hoare for encouraging me to undertake this research and for providing me with much helpful advice. Jifeng He was kind enough to point out a mistake in a previous draft of this paper. I also wish to thank Charles Barton and Carroll Morgan for their comments and Jim Woodcock and Nils Klarlund for their interest. The financial support of IBM is acknowledged.

References

Baetan JCM, Bergstra JA, Klop JW (1985) Conditional axioms and α/β calculus in process algebra. Report CS-R8502. Centre for Mathematics and Computer Science, Amsterdam

Brookes SD, Hoare CAR, Roscoe AW (1984) A theory of communicating sequential processes. J Assoc Comput Mach 31:560–599

Brookes SD, Roscoe AW (1984) An improved failures model for communicating sequential processes. Lect Notes Comp Sci 197:281–305

Hayes IJ (1987) Specification case studies. Prentice-Hall International, London

He J (1988) Process refinement. Refinement Workshop, University of York

He J, Hoare CAR, Sanders JW (1986) Data refinement refined. Lect Notes Comp Sci 213:187–196

Hoare CAR (1980) A model for communicating sequential processes. In: McKeag RM, McNaghton AM (eds) On the construction of programs. Cambridge University Press, Cambridge, UK, pp 229–243

Hoare CAR (1985) Communicating sequential processes. Prentice-Hall International, London

Jones CB (1986) Systematic software development using VDM. Prentice-Hall International, London

Milner AJRG (1980) A calculus of communicating systems. Lect Notes Comp Sci 92

Milner AJRG (1985) Lectures on a calculus for communicating systems. In: Broy M (ed) Control flow and data flow. Springer, Berlin Heidelberg New York Tokyo

Olderog E-R, Hoare CAR (1986) Specification-oriented semantics for communicating processes. Acta Informatica 23:9–66

Park D (1981) Concurrency and automata on infinite sequences. Lect Notes Comp Sci 104:167–183