Priorities in Process Algebras

Rance Cleaveland
Matthew Hennessy
Computer Science Department
University of Sussex
Falmer, Brighton BN1 9QH, ENGLAND

Abstract

An operational semantics for an algebraic theory of concurrency is developed that incorporates a notion of priority into the definition of the execution of actions. An equivalence based on strong observational equivalence is defined and shown to be a congruence, and a complete axiomatization is given for finite terms. Several examples highlight the novelty and usefulness of our approach.

1 Introduction

Much has been written in recent years about the semantic basis for communicating processes and reactive systems. [12,9,15] Intuitively, these are systems which evolve by interacting, or communicating, with their environment. Systems may perform a variety of actions, some of which may interact with the environment in a particular way; communication is modeled in terms of such interactions.

Most semantic theories for these systems are *operational.* Processes are interpreted using ***labeled transition systems***, which are triples of the form $\langle P, \rightarrow, Act\rangle$, where P is the set of processes, Act is the set of actions, and $\rightarrow$ is a relation on $P \times Act \times P$ defining the behaviour of processes. The statement "$p \xrightarrow{a} q$" means that process p may evolve to process q by performing action a. This form of operational semantics has given rise to a variety of behavioural equivalences on processes [12,9,6] and has proven to be a convenient way of defining process behaviour when actions are all of equal importance. However, the approach has a well recognized flaw when one tries to assign more importance to some actions than others. Obvious examples of actions which require special treatment include interrupts in hardware systems and time outs in communications protocols. In addition, certain programming language constructs, such as the "delay" commands in OCCAM [10] and ADA [16] and the disabling construct in LOTOS [4], embody the notion of priority between actions. No satisfactory semantic theory exists for these constructs and for actions with priority in general.

In this paper we wish to develop such a theory. To do so we introduce the notion of priority on actions and modify the use of labeled transition systems to develop an operational semantics reflecting these priorities. We then develop and axiomatize a new behavioural equivalence analogous to the strong observational equivalence of [12]. We believe that this new semantic theory provides a sound basis for the language constructs and actions mentioned previously.

The paper is organized as follows. In section 2 we give an example highlighting the problem with existing theories and introduce the idea of prioritized actions. Section 3 defines the language we use and gives our new operational semantics. The language is based on CCS, but our method for giving operational semantics can equally well be applied to other languages such as LOTOS [4] and Statecharts [7]. In section 4 we define a new behavioural equivalence based on strong observational equivalence, and in section 5 we discuss proof techniques and give a complete axiomatization for finite terms and a proof rule for recursively defined terms. Section 6 gives some examples emphasizing the novelty of our approach, and the final section discusses our conclusions and future plans. The proofs of most theorems are omitted; the interested reader is referred to [5].

2 Prioritized Actions

The following example is intended to illustrate the inadequacy of existing theories in the presence of interrupts, or, more generally, in the presence of language features which enable and disable actions. Consider the system of figure 1. Component C acts as a counter, while INT is designed to halt C when the environment issues a ***shut_down*** request. In CCS this could be defined by having an internal connection i between INT and C which is used when ***shut_down*** is performed. In this case the

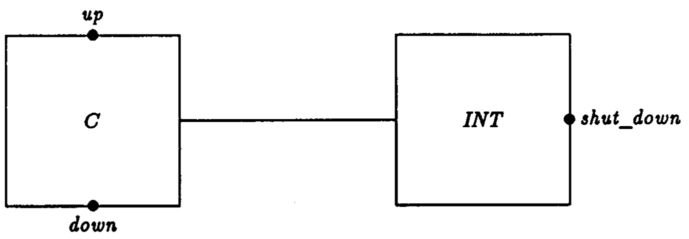


Figure 1: A two-process system.

definition of the system would be as follows.

$$\begin{aligned} SYS &\Leftarrow (C|INT)\backslash i \\ INT &\Leftarrow shut\_down.\bar{i}.nil \\ C &\Leftarrow up.C_1 + i.nil \\ C_{n+1} &\Leftarrow up.C_{n+2} + down.C_n + \bar{i}.nil \end{aligned}$$

However, in the standard operational semantics based on labeled transition systems the following is a valid possible sequence of actions for *SYS*.

$$up\ up\ shut\_down\ up\ up\ \ldots$$

Although the action *shut_down* is performed C_n may choose to ignore indefinitely the offer of communication from *INT* along channel i; the operational semantics merely states that actions *up* and i are always possible for C_n. Because of this defect in the operational semantics, the resulting behavioural theories [12,9,3,6] give an inadequate account of this system. This semantic shortcoming is not confined to CCS; any language whose operational semantics are given in terms of traditionally defined labeled transition systems will not be able to describe this system correctly.

Intuitively, the desired behaviour of *SYS* can be captured if we associate *priorities* to actions. By assigning a higher priority to i than to *up* or *down* we can ensure that when *SYS* reaches the state $(C|\bar{i}.nil)\backslash i$, the internal synchronization on port i must take place next, since it has a higher priority than any of the other possible actions. This is the basis of our approach; we modify the usual operational semantics to take account of priorities and use it to develop what we feel is an adequate semantic theory of such processes.

3 The Language

The syntax of our language is essentially that of pure CCS, although later in the paper we introduce additional operators. For simplicity we assume a two-level hierarchy of priorities; an action is either prioritized or unprioritized. (We should point out that there is no theoretical difficulty is extending our results to sets of actions with a range of discrete priorities.) To model communication we adopt the usual structure of actions used in CCS. Let Λ be a set of action labels, and let $A = \Lambda \cup \overline{\Lambda} \cup \{\tau\}$. Additionally, each $a \in A$ has a prioritized version, $\underline{a}$. Let $\underline{A}$ be the set of prioritized actions. Then $Act = A \cup \underline{A}$ is our set of actions.

In the remainder of the paper, let $a, b \in A$, $\underline{a}, \underline{b} \in \underline{A}$, and $\alpha, \beta \in Act$, with $\lambda \in Act - \{\tau, \underline{\tau}\}$. The terms of our language are defined as follows.

$$t ::= nil \mid \alpha.t \mid t|t \mid t+t \mid t\backslash\lambda \mid t[R] \mid x \mid fix(x.t)$$

R, it should be noted, is a *relabeling*, a mapping from Act to Act preserving $\tau, \underline{\tau}$ and $^-$ and such that $R(a) \in A$ and $R(\underline{a}) \in \underline{A}$. (Note that we do not require that $R(\underline{a}) = \underline{R(a)}$.) We adopt the usual definitions for free and bound variables, open and closed terms, and guarded recursion. In what follows, p, q and r range over closed terms, which we shall often call *processes*.

The operational semantics of this language is given in two stages. The first stage ignores priorities and is called the *a priori* operational semantics; the relations $\xrightarrow{\alpha}$ are defined by structural induction on terms in figure 2. This definition takes no account of the special properties we wish to assign to prioritized actions. The second stage defines the relations $\overset{\alpha}{\longrightarrow\!\!\triangleright}$, representing the actions which are actually possible. These relations are defined by:

1. if $p \xrightarrow{\underline{a}} q$ then $p \overset{\underline{a}}{\longrightarrow\!\!\triangleright} q$;

2. if $p \xrightarrow{a} q$ and for no $q', \underline{b}$ does $p \xrightarrow{\underline{b}} q'$ then $p \overset{a}{\longrightarrow\!\!\triangleright} q$.

Intuitively, prioritized actions are unconstrained, while unprioritized actions can only happen if no unprioritized actions are possible. Thus $\longrightarrow\!\!\triangleright$ defines an operational semantics reflecting our intuitions about prioritized actions. Formally, we have de-

$$
\begin{array}{lcl}
\alpha.s \xrightarrow{\alpha} s & & \\
s \xrightarrow{\alpha} s' & \Rightarrow & s + t \xrightarrow{\alpha} s' \\
t \xrightarrow{\alpha} t' & \Rightarrow & s + t \xrightarrow{\alpha} t' \\
s \xrightarrow{\alpha} s' & \Rightarrow & s|t \xrightarrow{\alpha} s'|t \\
t \xrightarrow{\alpha} t' & \Rightarrow & s|t \xrightarrow{\alpha} s|t' \\
s \xrightarrow{a} s', t \xrightarrow{\overline{a}} t' & \Rightarrow & s|t \xrightarrow{\tau} s'|t' \\
s \xrightarrow{\underline{a}} s', t \xrightarrow{\underline{\overline{a}}} t' & \Rightarrow & s|t \xrightarrow{\underline{\tau}} s'|t' \\
s \xrightarrow{\alpha} s', \alpha \neq \beta & \Rightarrow & s\backslash\beta \xrightarrow{\alpha} s'\backslash\beta \\
s \xrightarrow{\alpha} s' & \Rightarrow & s[R] \xrightarrow{R(\alpha)} s'[R] \\
t \xrightarrow{\alpha} t' & \Rightarrow & \mathit{fix}(x.t) \xrightarrow{\alpha} t'[\mathit{fix}(x.t)/x]
\end{array}
$$

Figure 2: The *a priori* semantics.

fined a labeled transition system $\langle P, \rightarrowtriangle, Act\rangle$, where P is the set of closed terms.

4 The Behavioural Equivalence

A variety of behavioural equivalences have been defined on the basis of labeled transition systems. One such equivalence, ***strong observational equivalence***, can be defined in terms of relations on processes called ***bisimulations*** [14]. Given the labeled transition system $\langle P, \rightarrow, Act\rangle$ a bisimulation $R \subseteq P \times P$ is a symmetric relation satisfying:

> $\langle p, q\rangle \in R$ and $p \xrightarrow{\alpha} p'$ implies $q \xrightarrow{\alpha} q'$ for some q' where $\langle p', q'\rangle \in R$.

Strong observational equivalence, $\sim$, is defined as the largest such bisimulation. It is guaranteed to exist and to be an equivalence relation [14].

This definition can be applied to $\langle P, \rightarrowtriangle, Act\rangle$ to obtain a new equivalence, $\sim_p$. Unfortunately, $\sim_p$ is *not* an adequate behavioural equivalence, because it identifies processes that can intuitively be distinguished. For example,

$$a.p + \underline{b}.q \sim_p \underline{b}.q,$$

since the only transition available to $a.p + \underline{b}.q$ is $a.p + \underline{b}.q \overset{\underline{b}}{\rightarrowtriangle} q$, and yet

$$(a.p + \underline{b}.q)\backslash\underline{b} \not\sim_p (\underline{b}.q)\backslash\underline{b}$$

since $(a.p + \underline{b}.q)\backslash\underline{b}$ may perform an a action while $(\underline{b}.q)\backslash\underline{b}$ may not.

In CCS terms, $\sim_p$ is not a *congruence*, since terms that are $\sim_p$-equivalent may nonetheless give rise to $\sim_p$-inequivalent processes when substituted into a context (a *context* being a term with a "hole"). In the previous example, the context $[]\backslash\underline{b}$ distinguishes $a.p + \underline{b}.q$ and $\underline{b}.q$. Relation $\sim_p$ does contain a largest congruence, $\sim_p^C$, where $p \sim_p^C q$ if and only if $C[p] \sim_p C[q]$ for all contexts C. However, there are disadvantages to this type of definition. It does not give rise to the elegant proof techniques normally associated with bisimulation equivalences, and it is dependent on the linguistic constructs allowed in the language, which may not coincide with all programming environments that appear in practice.

Given that we desire an equivalence that is a congruence and that can also be defined in terms of bisimulations, we define another operational semantics based on a new arrow, $\succ\!\!\rightarrowtriangle$. For convenience, let us say that p is *patient* if $p \xrightarrow{\underline{\tau}} q$ for no q. Then $\succ\!\!\rightarrowtriangle$ is defined by the following two clauses.

1. If $p \xrightarrow{\underline{a}} q$ then $p \overset{\underline{a}}{\succ\!\!\rightarrowtriangle} q$.

2. If $p \xrightarrow{a} q$ and p is patient then $p \overset{a}{\succ\!\!\rightarrowtriangle} q$.

As before, prioritized actions are not constrained. However, unprioritized actions are pre-empted by $\underline{\tau}$; they can only be performed by patient processes.

Let $\simeq$ be the strong equivalence generated by the labeled transition system $\langle P, \succ\!\!\rightarrowtriangle, Act\rangle$. Then we have the following.

Theorem 4.1 *$\simeq$ is a congruence with respect to the operators of CCS.*

It also turns out that $\simeq$ is a congruence with respect to several other language constructs. Of particular interest are the operations corresponding to prioritization of an action (written $p\lceil a$ for $a \neq \tau$) and deprioritization of an action (written $p\lfloor \underline{a}$ for $\underline{a} \neq \underline{\tau}$). Intuitively, $p\lceil a$ prioritizes all a actions p can perform, while $p\lfloor \underline{a}$ deprioritizes all $\underline{a}$ actions p can perform, provided the newly deprioritized action is also available to p. In particular, properties like

$$p \overset{a}{\rightarrowtail} q \Rightarrow p\lceil a \overset{\underline{a}}{\rightarrowtail} q\lceil a$$

hold. Defining these operators precisely is somewhat subtle, since their semantics must be defined first in terms of the *a priori* semantics even though the actions that p may ultimately perform are not defined until the second stage of the semantic specification. The required additions to the *a priori* semantics are as follows.

1. Prioritization

   (a) If $p \xrightarrow{a} q$ and p is patient then $p\lceil a \xrightarrow{\underline{a}} q\lceil a$.

   (b) If $p \xrightarrow{a} q$ and p is not patient then $p\lceil a \xrightarrow{a} q\lceil a$.

   (c) If $p \xrightarrow{\alpha} q$ and $\alpha \neq a$ then $p\lceil a \xrightarrow{\alpha} q\lceil a$.

2. Deprioritization

   (a) If $p \xrightarrow{\underline{a}} q$ and p is patient then $p\lfloor \underline{a} \xrightarrow{a} q\lfloor \underline{a}$.

   (b) If $p \xrightarrow{\underline{a}} q$ and p is not patient then $p\lfloor \underline{a} \xrightarrow{\underline{a}} q\lfloor \underline{a}$.

   (c) If $p \xrightarrow{\alpha} q$ and $\alpha \neq \underline{a}$ then $p\lfloor \underline{a} \xrightarrow{\alpha} q\lfloor \underline{a}$.

From these definitions it is easy to see that $\lceil a$ and $\lfloor \underline{a}$ enjoy the following properties with respect to $\rightarrowtail$, in addition to the one mentioned above.

- If $p\lceil a \overset{\underline{a}}{\rightarrowtail} q\lceil a$ then either $p \overset{a}{\rightarrowtail} q$ or $p \overset{\underline{a}}{\rightarrowtail} q$.
- If $p \overset{\underline{a}}{\rightarrowtail} q$ and p is patient then $p\lfloor \underline{a} \overset{a}{\rightarrowtail} q\lfloor \underline{a}$.
- If $p\lfloor \underline{a} \overset{a}{\rightarrowtail} q\lfloor \underline{a}$ then either $p \overset{\underline{a}}{\rightarrowtail} q$ and p is patient or $p \overset{a}{\rightarrowtail} q$.

In particular, it should be noted that if $p \xrightarrow{a} q$, $p \overset{\underline{a}}{\not\rightarrow} q$, and $p \overset{a}{\not\rightarrowtail} q$ (owing to p not being patient) then $p\lceil a \overset{\underline{a}}{\not\rightarrowtail} q\lceil a$. That is, an action is prioritized only if it is "visible" with respect to $\rightarrowtail$. Similarly, an action is deprioritized only if it remains visible with respect to $\rightarrowtail$.

We now have the following result.

Theorem 4.2 $\simeq$ *is a congruence with respect to prioritization and deprioritization.*

In the augmented language $\sim_p^C$ and $\simeq$ turn out to be the same relation. In one direction the result is straightforward, as the following theorem demonstrates.

Theorem 4.3 *If* $p \simeq q$ *then* $p \sim_p^C q$.

Proving the converse of the previous theorem is somewhat more involved. Intuitively, the difference between $\longrightarrow$ and $\rightarrowtail$ arises in their treatments of unprioritized actions in the presence of external (e.g. non-$\underline{\tau}$) prioritized actions. Therefore, if we define a context $D[]$ that deprioritizes all external actions in an appropriate way, then $\longrightarrow$ transitions for $D[p]$ correspond to $\rightarrowtail$ transitions for p, and we can relate $\sim_p^C$ and $\simeq$. To this end, let p and q be processes, and let $D_{p,q}[]$ be the context defined as follows. First, let $S_U(p)$ and $S_P(p)$ be the "unprioritized" and "prioritized" sorts, respectively, of p. Now define the relabeling $L_{p,q}$ as

$$L_{p,q}(\alpha) = \begin{cases} \alpha & \text{if } \alpha \in \underline{A} \\ \alpha & \text{if } \alpha \in A \text{ and } \underline{\alpha} \notin S_P(p) \cup S_P(q) \\ c_\alpha & \text{otherwise} \end{cases}$$

where $c_\alpha \in A$, $c_{\overline{\alpha}} = \overline{c_\alpha}$, $c_\alpha \neq c_\beta$ if $\alpha \neq \beta$, and $\underline{c_\alpha} \notin S_P(p) \cup S_P(q) \cup \underline{S_U(p)} \cup \underline{S_U(q)}$. $L_{p,q}$ essentially maps unprioritized actions in p and q whose prioritized versions also exist in p and q to unique unprioritized actions with no prioritized counterparts in the processes. Now define $S_{p,q} = S_P(p) \cup S_P(q)$, and let $D_{p,q}[] = ([][L_{p,q}])\lfloor S_{p,q}$, where $\lfloor S$ is the obvious generalization of the deprioritization operator to sets of actions. This context uniquely deprioritizes actions in p and q.

The definition of $D_{p,q}[]$ gives rise to an equivalence between processes in much the same way that the set of all contexts $C[]$ gives rise to the equivalence $\sim_p^C$. Define $p \sim_p^D q$ to hold exactly when $D_{p,q}(p) \sim_p D_{p,q}(q)$. Clearly, if $p \sim_p^C q$ then $p \sim_p^D q$.

Several properties of $D_{p,q}[]$ and $\sim_p^C$ deserve comment, since they will be used in the proof of the next theorem. To begin with, if p is patient then for no $\underline{a}$, q and r does $D_{p,q}[p] \overset{\underline{a}}{\longrightarrow} r$; this results from the fact that because p is patient, the deprioritization operator in $D_{p,q}[]$ deprioritizes all prioritized actions that p may initially perform. Also, if $\alpha \in A$ and $D_{p,q}[p] \overset{L_{p,q}(\alpha)}{\longrightarrow} D_{p,q}[p']$ then $p \overset{\alpha}{\rightarrowtail} p'$, since in this case p must be patient. Finally, if $S_P(p') \subseteq S_P(p), S_U(p') \subseteq S_U(p), S_P(q') \subseteq S_P(q)$, and $S_U(q') \subseteq S_U(q)$, then $D_{p',q'}[p'] \sim_p D_{p',q'}[q']$ (and thus $p' \sim_p^D q'$) if and only if $D_{p,q}[p'] \sim_p D_{p,q}[q']$. This follows from the fact that both $D_{p,q}$ and $D_{p',q'}$ uniquely deprioritze actions in p' and q'. It should be noted that if $p \xrightarrow{\alpha} p'$ then $S_P(p') \subseteq S_P(p)$ and $S_U(p') \subseteq S_U(p)$.

We are now able to prove the next theorem.

Theorem 4.4 *If* $p \sim_p^C q$ *then* $p \simeq q$.

Proof. Since $\sim_p^C \subseteq \sim_p^D$ it suffices to show that $\sim_p^D$ is a bisimulation for $\langle P, \rightarrowtail, Act \rangle$. Clearly $\sim_p^D$ is symmetric. Assume

$$\begin{array}{llcl}
\text{A1} & x+x & = & x \\
\text{A2} & x+y & = & y+x \\
\text{A3} & x+(y+z) & = & (x+y)+z \\
\text{A4} & x+\mathit{nil} & = & x \\
\text{P} & a.x+\underline{\tau}.y & = & \underline{\tau}.y \\
\text{INT} & \multicolumn{3}{l}{\text{Let } p,q \text{ denote } \sum \alpha_i.p_i, \sum \beta_j.q_j, \text{ respectively. Then}} \\
 & p|q & = & \sum \alpha_i.(p_i|q) + \sum \beta_j.(p|q_j) + \sum_{\alpha_i=\overline{\beta_j}\in A} \tau.(p_i|q_j) + \sum_{\alpha_i=\overline{\beta_j}\in \underline{A}} \underline{\tau}.(p_i|q_j) \\
\text{RES1} & \mathit{nil}\backslash\lambda & = & \mathit{nil} \\
\text{RES2} & (\alpha.x)\backslash\lambda & = & \begin{cases} \mathit{nil} & \text{if } \alpha\in\{\lambda,\overline{\lambda}\} \\ \alpha.(x\backslash\lambda) & \text{otherwise} \end{cases} \\
\text{RES3} & (x+y)\backslash\lambda & = & (x\backslash\lambda)+(y\backslash\lambda) \\
\text{REL1} & \mathit{nil}[R] & = & \mathit{nil} \\
\text{REL2} & (\alpha.x)[R] & = & R(\alpha).(x[R]) \\
\text{REL3} & (x+y)[R] & = & x[R]+y[R]
\end{array}$$

Figure 3: The equational characterization of $\simeq$ for CCS.

$p \sim_p^D q$ and $p \overset{\alpha}{\rightarrowtail} p'$; we must show that there is a q' such that $q \overset{\alpha}{\rightarrowtail} q'$ and $p' \sim_p^D q'$. There are three cases.

- $\alpha = \underline{a} \in \underline{A}$ and p is patient. Then $D_{p,q}[p]$ is patient, and $D_{p,q}[p] \xrightarrow{\underline{a}} D_{p,q}[p']$. From the properties above it follows that $D_{p,q}[p] \overset{\underline{a}}{\twoheadrightarrow} D_{p,q}[p']$, and this implies that there is a q' such that $D_{p,q}[q] \overset{\underline{a}}{\twoheadrightarrow} D_{p,q}[q']$ and $D_{p,q}[p'] \sim_p D_{p,q}[q']$. Again from the properties mentioned above, $p' \sim_p^D q'$, and it is easy to establish that $q \overset{\underline{a}}{\rightarrowtail} q'$.

- $\alpha \in \underline{A}$ and p is impatient. This case is routine because the deprioritization in $D_{p,q}[]$ has no effect.

- $\alpha \in A$. Then p, and hence $D_{p,q}[p]$, $D_{p,q}[q]$ and q, must be patient. This implies that $D_{p,q}[p] \overset{L_{p,q}(\alpha)}{\twoheadrightarrow} D_{p,q}[p']$, and there is therefore a q' such that $D_{p,q}[q] \overset{L_{p,q}(\alpha)}{\twoheadrightarrow} D_{p,q}[q']$ and $D_{p,q}[p'] \sim_p D_{p,q}[q']$, meaning that $p' \sim_p^D q'$. Moreover, as $D_{p,q}[q] \overset{L_{p,q}(\alpha)}{\twoheadrightarrow} D_{p,q}[q']$, by the above properties it must be the case that $q \overset{\alpha}{\rightarrowtail} q'$.

□

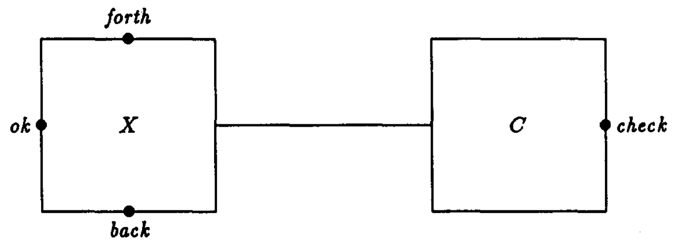
5 Proof Techniques

In this section we briefly sketch the proof techniques we have for deriving equivalences. Because $\simeq$ is defined as a bisimulation equivalence a natural and often effective way of proving $p \simeq q$ is to exhibit a bisimulation R (with respect to $\rightarrowtail$) which contains the pair $\langle p, q\rangle$. Indeed, the well-known bisimulation construction algorithms for deciding bisimulation equivalence [11] may be adapted to our semantics.

Alternatively, there is a set of equivalence-preserving syntactic transformations based on the well-known ones for strong bisimulation equivalence [8]; these are listed in figure 3. The usual laws for $+, \mathit{nil}, \backslash\lambda$ and $[R]$ remain valid; however, the presence of $\underline{\tau}$ introduces the new law P. This is readily seen to be satisfied by $\simeq$ because of the preemptive power of $\underline{\tau}$. The interleaving law from [8] also needs a slight modification because of the presence of two kinds of synchronization actions, τ and $\underline{\tau}$; this law appears in figure 3 as INT. Let E denote the set of equations in figure 3, and let $p =_E q$ mean that p can be transformed into q by application of the equations in E. The next theorem says that these equations completely characterize the new equivalence for finite terms.

Theorem 5.1 *For finite CCS terms p, q, $p \simeq q$ if and only if $p =_E q$.*

We now consider proof rules for recursive terms. The development of the standard results is complicated somewhat by the fact that the basic *a priori* moves, $\xrightarrow{\alpha}$, are defined by structural induction on (open) terms instead of by the more usual inductive definition. This is necessary because the clauses in the definition of $\xrightarrow{\alpha}$ for the prioritization and deprioritization operators have negative antecedents and therefore cannot be used in an inductive definition specified as the least relation satisfying a collection of clauses. One consequence of this is that we must confine ourselves to guarded recursions, since in



$$\begin{aligned} X &\Leftarrow back.X' + \underline{i}.ok.\bar{\underline{i}}.X \\ X' &\Leftarrow forth.X + \underline{i}.ok.\bar{\underline{i}}.X' \\ C &\Leftarrow check.\bar{\underline{i}}.\underline{i}.C \\ Sys &\Leftarrow (X|C)\backslash\underline{i} \end{aligned}$$

Figure 4: A two-process system.

the more general case certain desirable properties of recursively defined processes–such as $fix(x.t) \simeq t[fix(x.t)/x]$–do not hold. This restriction, however, is a standard one adopted in the literature and is indeed a reasonable one. The relevant property of guarded expressions, which we leave to the reader to verify, is the following. Assume that x is guarded in t. Then,

$$t[u/x] \xrightarrow{\alpha} r \iff r \text{ is } t'[u/x] \text{ and } t \xrightarrow{\alpha} t'. \tag{1}$$

Another useful property, which is trivial to prove, is:

$$(p \xrightarrow{\alpha} r \iff q \xrightarrow{\alpha} r) \text{ implies } p \simeq q. \tag{2}$$

More generally, let R be any bisimulation in the *a priori* semantics $\langle P, \rightarrow, Act\rangle$. Then

$$\langle p, q\rangle \in R \text{ implies } p \simeq q. \tag{3}$$

This follows easily from the fact that such an R is also a bisimulation in the actual operational semantics $\langle P, \rightarrowtail, Act\rangle$.

With these results we can derive many of the expected properties of the fixed point operator. The first states that $fix(x.t)$ is indeed a fixed point of the equation $x \simeq t$.

Theorem 5.2 $fix(x.t) \simeq t[fix(x.t)/x]$.

It is possible to extend $\simeq$ to open terms in a natural way. Let a *substitution* σ be a mapping from variables to P, and for a term t let $t\sigma$ represent the obvious (closed) term. Then define $t \simeq u$ to hold exactly when for all substitutions σ, $t\sigma \simeq u\sigma$. The next result states that this extended $\simeq$ behaves properly with respect to fix considered as an operator on terms.

Theorem 5.3 *If* $t \simeq u$ *then* $fix(x.t) \simeq fix(x.u)$.

The final result we show is a form of induction that is often called "unique fixed point induction."

Theorem 5.4 *If* $p \simeq t[p/x]$ *then* $p \simeq fix(x.t)$.

6 Examples

In this section we present two examples that illustrate the usefulness of our approach. The first example defines a system consisting of two processes: a process X that flips back and forth between two states and a process C that checks that the first process is running properly. The implementation of this system in our language appears in figure 4.

One desirable property of this system would be that each *check* action is followed by an *ok*, an acknowledgement that X is running. In pure CCS, this is not the case; indeed, the (infinite) sequence of actions

$$check\ back\ forth\ back\ forth\ \ldots$$

is possible, owing to the fact that X is not required to synchronize with C after C performs a *check*. In our framework, this cannot happen, since i is prioritized. In fact, it is the case that

$$Sys \simeq Spec,$$

where *Spec* is defined as follows:

$$\begin{aligned} Spec &\Leftarrow back.Spec' + check.\underline{\tau}.ok.\underline{\tau}.Spec \\ Spec' &\Leftarrow forth.Spec + check.\underline{\tau}.ok.\underline{\tau}.Spec'. \end{aligned}$$

To prove this, let $Sys' = (X'|C)\backslash i$. It suffices to show that

$$\begin{aligned} Sys &= Spec \\ Sys' &= Spec'. \end{aligned}$$

$$
\begin{aligned}
Sys &= [\mathit{back}.(X'|C) + \underline{i}.((\mathit{ok}.\overline{\underline{i}}.X)|C) + \mathit{check}.(X|\overline{\underline{i}}.\underline{i}.C)]\backslash\underline{i} \text{ by Int}\\
&= \mathit{back}.[(X'|C)\backslash\underline{i}] + \mathit{check}.[(X|\overline{\underline{i}}.\underline{i}.C)\backslash\underline{i}] \text{ by RES3, RES2 and A4}\\
&= \mathit{back}.[(X'|C)\backslash\underline{i}] +\\
&\quad \mathit{check}.[[\mathit{back}.(X'|\overline{\underline{i}}.\underline{i}.C) + \underline{i}.((\mathit{ok}.\overline{\underline{i}}.X)|\overline{\underline{i}}.\underline{i}.C) + \overline{\underline{i}}.(X|\underline{i}.C) + \underline{\tau}.((\mathit{ok}.\overline{\underline{i}}.X)|\underline{i}.C)]\backslash\underline{i}] \text{ by INT}\\
&= \mathit{back}.[(X'|C)\backslash\underline{i}] + \mathit{check}.[\underline{\tau}.((\mathit{ok}.\overline{\underline{i}}.X)|\underline{i}.C)\backslash\underline{i}] \text{ by P, RES3, RES2 and A4}\\
&\ \vdots
\end{aligned}
$$

Figure 5: A fragment of the proof that $Sys =_E Spec$.

Using the unique fixed point rule, it is sufficient to show that

$$
\begin{aligned}
Sys &= \mathit{back}.Sys' + \mathit{check}.\underline{\tau}.\mathit{ok}.\underline{\tau}.Sys\\
Sys' &= \mathit{forth}.Sys + \mathit{check}.\underline{\tau}.\mathit{ok}.\underline{\tau}.Sys'.
\end{aligned}
$$

Figure 5 contains a fragment of the proof of (4) using the axioms of figure 3 as rewrite rules. Readers familiar with similar-style proofs from [14] should have no trouble completing this proof.

The second example uses priorities in a slightly different fashion. Here we present a development of the alternating bit protocol [1] that is correct in pure CCS when the medium may lose messages but is incorrect when the medium is reliable. We show how the introduction of priorities resolves this anomaly. We should note that the use of priorities here is only partially successful. The inadequacy of the example is discussed more fully after its presentation, but the problem arises from the fact that only the prioritized internal move can preempt unprioritized actions.

The alternating bit protocol provides a means of ensuring reliable communication over half-duplex lines. In this protocol, the sending and receiving processes alternate between two states in response to the receipt of messages (in the case of the receiving process) and acknowledgements (in the case of the sending process). Senders and receivers may also time out while waiting for acknowledgements and messages, respectively. A full account of the protocol may be found in [1].

Figure 6 presents the development (in pure CCS) of the protocol in the context of a lossy medium. It can be proven correct – after every *send* action, the only next possible non-τ action is a *receive*, and vice versa, and the system does not deadlock (i.e. wind up in a state where no actions are possible). However, if we replace M_{lossy} with M_{safe}, a medium that does not lose messages, the protocol is no longer correct. Consider the definition of M_{safe}.

$$
\begin{aligned}
M_{safe} \Leftarrow\ & s_0.\overline{r_0}.M_{safe} + s_1.\overline{r_1}.M_{safe}\\
& + s_{ack_0}.\overline{r_{ack_0}}.M_{safe} + s_{ack_1}.\overline{r_{ack_1}}.M_{safe}
\end{aligned}
$$

Since every s-action is followed by an r-action, this medium delivers every message it receives for sending. With M_{safe} replacing M_{lossy}, however, Sys may deadlock; the state

$$(\overline{s_{ack_1}}.R_0|\overline{r_0}.M_{safe}|S_0')\backslash\{r_0, r_1, s_0, s_1, r_{ack_0}, r_{ack_1}, s_{ack_0}, s_{ack_1}\}$$

is reachable via the sequence of actions

$$\mathit{send}\ \tau\ \tau\ \tau,$$

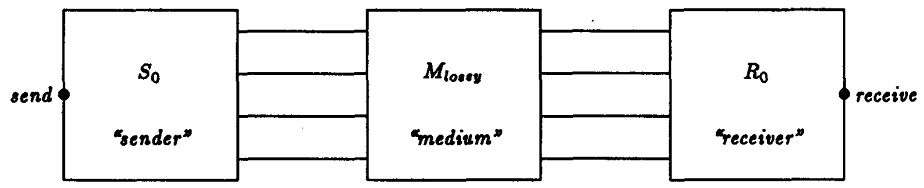
and no actions are possible from this state. Intuitively, this problem results from the fact that the receiver R_0 can elect to time out (by executing its τ action) even though a message is available for it to receive from the medium.

Using prioritized actions, this situation can be prevented. By prioritizing all actions except the τ actions in S_0', S_1', R_0 and R_1 (the time-out actions), interactions with the medium that are possible are required to happen; the above state is therefore not reachable, and the protocol will behave correctly. In fact, one can prove that $Sys \simeq Spec$, where $Spec$ is defined as follows.

$$
\begin{aligned}
Spec &\Leftarrow \underline{\mathit{send}}.\underline{\tau}.\underline{\tau}.Spec' + \underline{\tau}.Spec\\
Spec' &\Leftarrow \underline{\mathit{receive}}.\underline{\tau}.\underline{\tau}.Spec + \underline{\tau}.Spec'
\end{aligned}
$$

Figure 7 presents a tabular representation of a relation whose symmetric closure is a bisimulation containing $\langle Sys, Spec\rangle$.

It is worth noting here that our implementation of the alternating bit protocol uses busy waiting. That is, S_0, S_1, R_0' and R_1' each offer $\underline{\mathit{send}}$ and $\underline{\mathit{receive}}$ actions in the context of $\underline{\tau}$-loops. It is certainly more natural to imagine implementing the protocol without this busy waiting; however, in this case, prioritizing all actions except the τ's corresponding to time-outs does not fix the anomaly that results from the substitution of a safe medium for a lossy one. The reason is that in our semantics, only prioritized internal actions have preemptive power;



$$
\begin{aligned}
S_0 &\Leftarrow send.S_0' + \tau.S_0\\
S_0' &\Leftarrow \overline{s_0}.(r_{ack_0}.S_1 + r_{ack_1}.S_0' + \tau.S_0')\\
S_1 &\Leftarrow send.S_1' + \tau.S_1\\
S_1' &\Leftarrow \overline{s_1}.(r_{ack_1}.S_0 + r_{ack_0}.S_1' + \tau.S_1')
\end{aligned}
$$

$$
\begin{aligned}
M_{lossy} \Leftarrow\ & s_0.(\overline{r_0}.M_{lossy} + M_{lossy})\\
&+s_1.(\overline{r_1}.M_{lossy} + M_{lossy})\\
&+s_{ack_0}.(\overline{r_{ack_0}}.M_{lossy} + M_{lossy})\\
&+s_{ack_1}.(\overline{r_{ack_1}}.M_{lossy} + M_{lossy})
\end{aligned}
$$

$$
\begin{aligned}
R_0 &\Leftarrow r_0.R_0' + r_1.\overline{s_{ack_1}}.R_0 + \tau.\overline{s_{ack_1}}.R_0\\
R_0' &\Leftarrow receive.\overline{s_{ack_0}}.R_1 + \tau.R_0'\\
R_1 &\Leftarrow r_1.R_1' + r_0.\overline{s_{ack_0}}.R_1 + \tau.\overline{s_{ack_0}}.R_1\\
R_1' &\Leftarrow receive.\overline{s_{ack_1}}.R_0 + \tau.R_1'
\end{aligned}
$$

$$
Sys \Leftarrow (S_0|M_{lossy}|R_0)\backslash\{r_0, r_1, s_0, s_1, r_{ack_0}, r_{ack_1}, s_{ack_0}, s_{ack_1}\}
$$

Subscripted s and r actions denote sends and receives to and from the medium, respectively.
τ represents the time out action in the R_i and S_i'.

Figure 6: The alternating bit protocol.

$$
\begin{array}{ll}
\{\ \langle Sys, & Spec\rangle,\\
\langle [S_0'|M_{safe}|R_0]\backslash I, & \underline{\tau}.\underline{\tau}.Spec'\rangle,\\
\langle [(\underline{r_{ack_0}}.S_1 + \underline{r_{ack_1}}.S_0' + \tau.S_0')|\overline{\underline{r_0}}.M_{safe}|R_0]\backslash I, & \underline{\tau}.Spec'\rangle,\\
\langle [(\underline{r_{ack_0}}.S_1 + \underline{r_{ack_1}}.S_0' + \tau.S_0')|M_{safe}|R_0']\backslash I, & Spec'\rangle,\\
\langle [(\underline{r_{ack_0}}.S_1 + \underline{r_{ack_1}}.S_0' + \tau.S_0')|M_{safe}|\overline{\underline{s_{ack_0}}}.R_1]\backslash I, & \underline{\tau}.\underline{\tau}.Spec\rangle,\\
\langle [(\underline{r_{ack_0}}.S_1 + \underline{r_{ack_1}}.S_0' + \tau.S_0')|\overline{\underline{r_{ack_0}}}.M_{safe}|R_1]\backslash I, & \underline{\tau}.Spec\rangle,\\
\langle [S_1|M_{safe}|R_1]\backslash I, & Spec\rangle,\\
\langle [S_1'|M_{safe}|R_1]\backslash I, & \underline{\tau}.\underline{\tau}.Spec'\rangle,\\
\langle [(\underline{r_{ack_1}}.S_0 + \underline{r_{ack_0}}.S_1' + \tau.S_1')|\overline{\underline{r_1}}.M_{safe}|R_1]\backslash I, & \underline{\tau}.Spec'\rangle,\\
\langle [(\underline{r_{ack_1}}.S_0 + \underline{r_{ack_0}}.S_1' + \tau.S_1')|M_{safe}|R_1']\backslash I, & Spec'\rangle,\\
\langle [(\underline{r_{ack_1}}.S_0 + \underline{r_{ack_0}}.S_1' + \tau.S_1')|M_{safe}|\overline{\underline{s_{ack_1}}}.R_0]\backslash I, & \underline{\tau}.\underline{\tau}.Spec\rangle,\\
\langle [(\underline{r_{ack_1}}.S_0 + \underline{r_{ack_0}}.S_1' + \tau.S_1')|\overline{\underline{r_{ack_1}}}.M_{safe}|R_1]\backslash I, & \underline{\tau}.Spec\rangle\ \ \}
\end{array}
$$

I is the set $\{\underline{r_0}, \underline{r_1}, \underline{s_0}, \underline{s_1}, \underline{r_{ack_0}}, \underline{r_{ack_1}}, \underline{s_{ack_0}}, \underline{s_{ack_1}}\}$.

Figure 7: A relation whose symmetric closure is a bisimulation.

prioritized external (i.e. non-$\underline{\tau}$) actions cannot override non-prioritized actions. Thus, a process can time-out when the versions of S_0, S_1, R_0' and R_1' without busy waiting offer a $\underline{send}$ or $\underline{receive}$, even though these actions are prioritized.

This phenomenon merits more study; one idea to get around it would be to extend the language with a special type of prioritized actions having the preemptive power of $\underline{\tau}$. This would imply that these special actions could not be restricted or deprioritized, as otherwise $\simeq$ would cease to be a congruence. Nevertheless, such actions could play a useful role, for example, as the external actions used in specifications. In the example of the alternating bit protocol, if the $\underline{send}$ and $\underline{receive}$ actions were of this type the $\underline{\tau}$-loops representing busy waiting could be eliminated.

7 Conclusions

In this paper we have developed an operational semantics for processes having actions that take priority over other actions. Using the semantics as a basis we have developed a behavioural equivalence based on strong observational equivalence and shown that it is a congruence, and we have presented a complete axiomatization of the equivalence for finite terms and a development of proof rules for recursive proof rules. We have also developed several examples that show the usefulness of our approach.

Much research remains to be done in order to show that our approach to priorities is indeed of general interest and applicability. We have already indicated that a process algebra with prioritized actions should also contain associated combinators such as proritization and deprioritization and that certain other characteristics such as "strongly prioritized" actions are desirable. In general, what is a reasonable set of new combinators? We would also like to develop and axiomatise a weak observational equivalence or testing equivalence based on our semantics; the interaction of prioritization and abstraction from internal details may prove to be difficult. Another related line of research is to examine the semantics of programming language constructs that incorporate notions of prioritization, like those mentioned in the introduction.

An alternative approach to priorities in process algebras may be found in [2]. There the emphasis is on equational reasoning; more specifically, the authors examine the consistency of sets of equations obtained by adding to existing equational theories additional equations that the authors feel prioritization operators should satisfy. Their approach is purely algebraic in the sense that they do not give an operational semantics or behavioural equivalence that underlies their theory.

References

[1] Bartlett, K.A., R.A. Scantlebury and P.T. Wilkinson. "A Note on Reliable Full-Duplex Transmission over Half-Duplex Links." *Communications of the ACM* 12, n. 5, May 1969, pp. 260-261.

[2] Baeten, J.C.M, J.A. Bergstra and J.W. Klop. "Syntax and Defining Equations for an Interrupt Mechanism in Process Algebra." Technical Report CS-R8503, Department of Computer Science, Center for Mathematics and Computer Science, Amsterdam, February 1985.

[3] Bergstra, J.A., and J.W. Klop. "Process Algebra for Synchronous Communication." *Information and Control* 60, 1984, pp. 109-137.

[4] Brinksma, E. "A Tutorial on LOTOS." *Proceedings of IFIP Workshop on Protocol Specification, Testing and Verification V*, M. Diaz, ed., pp. 73-84. North-Holland, Amsterdam, 1986.

[5] Cleaveland, R. and M. Hennessy. "Priorities in Process Algebras." Technical Report 2/88, Department of Computer Science, University of Sussex, March 1988.

[6] DeNicola, R. and M. Hennessy. "Testing Equivalences for Processes." *Theoretical Computer Science* 24, 1984, pp. 83-113.

[7] Harel, D. "Statecharts: A Visual Formalism for Complex Systems." *Science of Computer Programming* 8, 1987.

[8] Hennessy, M. and R. Milner. "Algebraic Laws for Nondeterminism and Concurrency." *Journal of the ACM* 32, n. 1, January 1985, pp. 137-161.

[9] Hoare, C.A.R. *Communicating Sequential Processes.* Prentice-Hall International, London, 1985.

[10] INMOS Limited. OCCAM *Programming Manual.* Prentice-Hall International, London, 1984.

[11] Kanellakis, P.C. and S.A. Smolka. "CCS Expressions, Finite State Processes, and Three Problems of Equivalence." Technical Report, Department of Computer Science, Brown University, 1983.

[12] Milner, R. *A Calculus of Communicating Systems.* Lecture Notes in Computer Science 92. Springer-Verlag, Berlin, 1980.

[13] Milner, R. "A Complete Inference System for a Class of Regular Behaviours." *Journal of Computer and System Sciences* 28, n. 3, June 1984, pp. 439-466.

[14] Milner, R. "Calculi for Synchrony and Asynchrony." *Theoretical Computer Science* 25, n. 3, July 1983, pp. 267-310.

[15] Pnueli, A. "Linear and Branching Structures in the Semantics and Logics of Reactive Systems." Lecture Notes in Computer Science 194, pp. 14-32. Springer-Verlag, Berlin, 1985.

[16] U.S. Department of Defense. "Reference Manual for the Ada Programming Language." ANSI/MIL-STD 1815 A, January 1983.