# Priority and Abstraction in Process Algebra*

V. Natarajan** I. Christoff*** L. Christoff** R. Cleaveland*

**Abstract.** This paper defines a behavioral equivalence based on the notion of *weak observational equivalence* for processes in which actions may take priority over other actions. The largest congruence contained in this relation is then identified and given a complete equational axiomatization for finite processes. An example illustrates the utility of this equivalence which, in contrast with other equivalences for systems with priorities, abstracts from internal computation.

## 1 Introduction

Over the past 15 years, significant progress has been made in the development and application of algebraic theories [3, 12, 13, 15], called *process algebras*, for reasoning about reactive systems. These theories focus on equational reasoning as a means for verifying systems; given a behavioral equivalence relating processes on the basis of their observable behavior, one typically formulates a specification as a process describing the desired behavior of a system and then proves that an implementation is equivalent to it. An advantage of this approach is that, provided the equivalence is a congruence for the algebra, one may reason about the correctness of a system on the basis of a compositional analysis of its components. A variety of examples and case studies point to the practical utility of such equivalence-based verification [2, 7, 9, 16, 20].

Traditional process algebras enable users to analyze the potential nondeterminism systems may exhibit during their execution. More recently, researchers have looked into enriching the forms of system behavior that one may capture algebraically. Real-time and probabilistic extensions to existing theories have been proposed, and languages have been extended to allow the modeling of different *priority levels* among the actions that processes may perform [1, 5, 6, 8, 10, 11, 18], so that systems in which some actions, e.g. interrupts, take precedence over others may be modeled correctly. In this latter work, equivalences have also been proposed and axiomatized; however, these equivalences do not attempt any abstraction from internal computation. This lack of abstraction imposes severe restrictions on the use of these equivalences in verification, since a user wishing to establish that two systems exhibit the same behavior must account for the precise level of internal computation that the systems exhibit.

In this paper, we introduce a congruence on processes having actions with priority that is in some sense *as insensitive as possible* to internal computation. This equivalence thus permits a variety of implementations which differ only in their internal details to be related to the same specification, and hence affords system designers more latitude in devising appropriate implementations. The rest of the paper is organized as follows. Section 2 introduces the syntax and the semantics of the process algebra used in this paper, reviews the strong congruence of [8] and presents an axiomatization for all the operators in [8]. The next section introduces a behavioral equivalence analogous to the observational equivalence of CCS and characterizes the largest congruence contained in it. Section 4 provides an equational characterization of the congruence over finite processes, while Section 5 gives an example showing the use of the equivalence in verifying a system. The last section contains our conclusions, remarks on related research and directions for future work.

## 2 Priorities and Strong Equivalence

This section reviews the language and equivalence given in [8] for modeling systems in which actions have different priorities. It also extends the axiomatization for this language and equivalence to

** Authors supported by NSF/DARPA grant CCR-9014775, NSF grant CCR-9120995, ONR Young Investigator Award N00014-92-J-1582, and NSF Young Investigator Award CCR-9257963. Authors' address: Department of Computer Science, North Carolina State University, Raleigh, NC 27695-8206, USA, e-mail: {rance, raj}@science.csc.ncsu.edu.

*** Authors' address: Department of Computer Science, Uppsala University, Uppsala, Sweden.

* A version of this paper will appear in P.S. Thiagarajan, editor, *Foundations of Software Technology and Theoretical Computer Science*, Madras, India, December 1994.

$$
\begin{array}{lll}
\alpha.t \xrightarrow{\alpha} t & & \\
s \xrightarrow{\underline{a}} s' & \Rightarrow & s+t \xrightarrow{\underline{a}} s',\quad t+s \xrightarrow{\underline{a}} s',\quad s|t \xrightarrow{\underline{a}} s'|t,\quad t|s \xrightarrow{\underline{a}} t|s' \\
s \xrightarrow{a} s', t \not\xrightarrow{\underline{\tau}} & \Rightarrow & s+t \xrightarrow{a} s',\quad t+s \xrightarrow{a} s' \\
s \xrightarrow{\underline{a}} s', t \xrightarrow{\overline{\underline{a}}} t' & \Rightarrow & s|t \xrightarrow{\underline{\tau}} s'|t' \\
s \xrightarrow{a} s', s|t \not\xrightarrow{\underline{\tau}} & \Rightarrow & s|t \xrightarrow{a} s'|t,\quad t|s \xrightarrow{a} t|s' \\
s \xrightarrow{a} s', t \xrightarrow{\overline{a}} t', s|t \not\xrightarrow{\underline{\tau}} & \Rightarrow & s|t \xrightarrow{\tau} s'|t' \\
t \xrightarrow{\alpha} t', \alpha \neq \lambda, \overline{\lambda} & \Rightarrow & t\backslash\lambda \xrightarrow{\alpha} t'\backslash\lambda \\
s \xrightarrow{\mu} s' & \Rightarrow & s\lceil\mu \xrightarrow{\underline{\mu}} s'\lceil\mu \\
s \xrightarrow{\alpha} s', \mu \neq \alpha & \Rightarrow & s\lceil\mu \xrightarrow{\alpha} s'\lceil\mu \\
s \xrightarrow{\underline{\mu}} s', s \not\xrightarrow{\underline{\tau}} & \Rightarrow & s\lfloor\underline{\mu} \xrightarrow{\mu} s'\lfloor\underline{\mu} \\
s \xrightarrow{\underline{\mu}} s', s \xrightarrow{\underline{\tau}} s'' & \Rightarrow & s\lfloor\underline{\mu} \xrightarrow{\underline{\mu}} s'\lfloor\underline{\mu} \\
s \xrightarrow{\alpha} s', \underline{\mu} \neq \alpha & \Rightarrow & s\lfloor\underline{\mu} \xrightarrow{\alpha} s'\lfloor\underline{\mu} \\
t \xrightarrow{\alpha} t' & \Rightarrow & \mathit{fix}(x:t) \xrightarrow{\alpha} t'[\mathit{fix}(x:t)/x],\quad t[S] \xrightarrow{S(\alpha)} t'[S]
\end{array}
$$

$t \not\xrightarrow{\underline{\tau}}$ if and only if $\neg\exists t' :\ t \xrightarrow{\underline{\tau}} t'$

**Fig. 1.** Operational semantics.

incorporate operators that the treatment in [8] does not address.

### 2.1 Language and Strong Equivalence

The language we consider in this paper provides constructs for building processes up from atomic actions, which exhibit a two-level priority structure, with certain ones being designated as "prioritized" and others as "unprioritized". Intuitively, actions represent potential synchronizations that a process may be willing to engage in with its environment; given a choice between a prioritized and an unprioritized synchronization, a process should choose the former.

Formally, let $\Lambda$ denote a countably infinite set of labels; intuitively, $\Lambda$ contains the (unprioritized) "ports" that processes may synchronize over. Then the set of *unprioritized actions* $A$ may be defined by $A = \Lambda \cup \overline{\Lambda} \cup \{\tau\}$, where $\overline{\Lambda} = \{\, \overline{\lambda} \mid \lambda \in \Lambda \,\}$. Action $\lambda \in \Lambda$ may be thought of as representing the receipt of an input on port $\lambda$, while $\overline{\lambda} \in \overline{\Lambda}$ constitutes the deposit of an output on $\lambda$; $\tau \notin \Lambda$ represents an internal (unprioritized) computation step. To define the prioritized actions, let $\underline{\Lambda} = \{\, \underline{\lambda} \mid \lambda \in \Lambda \,\}$ be the set of "prioritized" ports, with $\overline{\underline{\Lambda}} = \{\, \overline{\underline{\lambda}} \mid \underline{\lambda} \in \underline{\Lambda} \,\}$. Then $\underline{A} = \underline{\Lambda} \cup \overline{\underline{\Lambda}} \cup \{\underline{\tau}\}$ is the set of *prioritized actions*, with $\underline{\tau}$ the *internal prioritized action*. We use $\mathit{Act} = A \cup \underline{A}$ to denote the set of all actions. In what follows we use $\alpha, \beta, \ldots$ to range over $\mathit{Act}$, $a, b, c, \ldots$ to range over $A$, and $\underline{a}, \underline{b}, \underline{c}, \ldots$ to range over $\underline{A}$. We also use $\lambda$ to represent elements in $\mathit{Act} \setminus \{\tau, \underline{\tau}\}$, $\mu$ to range over $A \setminus \{\tau\}$, and $\underline{\mu}$ to range over $\underline{A} \setminus \{\underline{\tau}\}$. We sometimes use the symbol '1' to stand for either $\tau$ or $\underline{\tau}$. We also extend $\bar{}$ to all non-$\{\tau, \underline{\tau}\}$ actions as follows: $\overline{\overline{\lambda}} = \lambda$, and if $L \subseteq \mathit{Act} \setminus \{\tau, \underline{\tau}\}$ then $\overline{L} = \{\, \overline{\lambda} \mid \lambda \in L \,\}$.

The terms of the language may now be defined as follows.

$$t ::= \mathit{nil} \mid \alpha.t \mid t+t \mid t|t \mid t\backslash\lambda \mid t[R] \mid t\lceil\mu \mid t\lfloor\underline{\mu} \mid x \mid \mathit{fix}(x:t)$$

Here $R$ is a *relabeling*, a mapping from $\mathit{Act}$ to $\mathit{Act}$ preserving $\tau, \underline{\tau}$ and $\bar{}$ and such that $R(a) \in A$ and $R(\underline{a}) \in \underline{A}$. We also assume that relabelings satisfy the following property: if $R$ is a relabeling then the set $\{\, a \in A \mid a \neq R(a) \,\}$ is finite. Note, however, that we do not require that $R(\underline{a}) = \underline{R(a)}$. We use $\mathit{Exp}$ to stand for the set of all terms of the language, and we adopt the usual definitions for *sort* of a process, *free* and *bound* variables, *open* and *closed* terms, *guarded recursion* and *contexts*. We call the closed guarded terms *processes*; $\mathcal{P}$ represents the set of all processes and will be ranged over by $p, q, \ldots$.

The operational semantics of processes is given as a relation $\rightarrow \subseteq \mathcal{P} \times \mathit{Act} \times \mathcal{P}$; we write $p \xrightarrow{\alpha} q$ in lieu of $\langle p, \alpha, q\rangle \in \rightarrow$ and $p \xrightarrow{\alpha}$ if $p \xrightarrow{\alpha} q$ holds for some $q$. The formal definition is given inductively in Figure 1 and is taken from [8]; note that even though the rules employ negative premises, the semantics is well-defined [4, 8]. Intuitively, $p \xrightarrow{\alpha} q$ holds if $p$ may engage in action $\alpha$ and thereafter behave like $q$; the relative priorities of actions are represented by the fact that a process can engage in an unprioritized action only if it is *patient*, i.e. initially incapable of executing the prioritized action

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

**Fig. 2.** Axioms for finite processes without $\lceil$ and $\lfloor$.

$\underline{\tau}$. One may wonder why, among the prioritized actions, only $\underline{\tau}$ has this preemptive power. The reason is that when $p \xrightarrow{\underline{\mu}} q$ holds, $p$ is signaling its *potential* for synchronizing on $\underline{\mu}$, whereas when $p \xrightarrow{\underline{\tau}} q$ holds $p$ is indicating, in essence, that a complete prioritized synchronization is possible. Accordingly, $\alpha.p$ may engage in $\alpha$ and then behave like $p$. The operator $+$ constitutes a choice construct: $p + q$ may perform the the prioritized actions of $p$ or $q$ and then behave like the process from which the action was chosen. Note that unprioritized actions are possible only if both $p$ and $q$ are patient. Process $p \mid q$ represents the parallel composition of $p$ and $q$, with concurrent execution modeled via interleaving and synchronization between $p$ and $q$ allowed if they are capable of complementary actions; the result of such a synchronization is a $\tau$ if the actions are unprioritized and a $\underline{\tau}$ otherwise. Note that $p \mid q$ may perform unprioritized actions only if $p \mid q$ is patient. The operator $\backslash\lambda$ defines a scope for actions using the port mentioned in action $\lambda$, while $p[R]$ relabels the actions of $p$ in the appropriate manner. The process $p\lceil\mu$ *prioritizes* the (unprioritized) $\mu$ actions that $p$ may engage in (note that $p$ may only engage in such actions if it is patient), while $p\lfloor\underline{\mu}$ *deprioritizes* $\underline{\mu}$ actions, but only if in so doing, the resulting $\mu$ actions remain possible (i.e. only if $p$ is patient). Finally $fix(x : t)$ represents a recursively defined process that is a distinguished "solution" to the equation $x = t$.

We now turn to defining the equivalence introduced in [8], which is given in terms of *prioritized strong bisimulations.*

**Definition 1.** $R \subseteq \mathcal{P} \times \mathcal{P}$ is a *prioritized strong bisimulation* if for every $\langle p, q\rangle$ in $R$ and $\alpha$ in $Act$ the following conditions hold:

$$
\begin{array}{l}
p \xrightarrow{\alpha} p' \Rightarrow \exists q' : \ q \xrightarrow{\alpha} q' \wedge p' R q'. \\
q \xrightarrow{\alpha} q' \Rightarrow \exists p' : \ p \xrightarrow{\alpha} p' \wedge p' R q'.
\end{array}
$$

Then $p \simeq q$ holds if there is a prioritized strong bisimulation relating $p$ and $q$. $\simeq$ is itself a prioritized strong bisimulation, and is in fact the largest such relation. The fact that $\simeq$ is a congruence relation over $\mathcal{P}$ has been proved in [8].

## 2.2 Axiomatizing $\simeq$

We now give an equational axiomatization of $\simeq$ for *finite* processes—that is, for processes including no subprocesses of the form $fix(x : t)$. Figure 2 contains axioms for finite processes that include no occurrences of $\lceil$ and $\lfloor$. Cleaveland and Hennessy [8] prove that these equations constitute a sound and complete axiomatization for the sublanguage that does not include these operators. They do not, however, provide axioms for $\lceil$ and $\lfloor$. In order to cover the full language of finite processes, and axiomatize $\simeq$ over *all* finite processes, we extend the axiomatization of Figure 2 with the rules in Figure 3. Note that the axioms $\lceil 3$ and $\lceil 4$ cannot be replaced by a single axiom of the form "$(x+y)\lceil\mu = x\lceil\mu + y\lceil\mu$", as this is not sound in general: for example $(\underline{\tau}.x+\mu.y)\lceil\mu \not\simeq (\underline{\tau}.x)\lceil\mu + (\mu.y)\lceil\mu$. Similarly, axioms $\lfloor 3$ and $\lfloor 4$ cannot be replaced by an axiom of the form "$(x + y)\lfloor\underline{\mu} = x\lfloor\underline{\mu} + y\lfloor\underline{\mu}$", as $(\underline{\tau}+\underline{\mu})\lfloor\underline{\mu} \not\simeq \underline{\tau}+\mu$. Let $E_{\simeq}$ denote the set of axioms provided in Figure 2 and Figure 3. The following theorem states that $E_{\simeq}$ equationally characterizes $\simeq$ over all finite processes.

$$
\begin{array}{lrl}
\lceil 1 & nil\lceil\mu = & nil \\
\lceil 2 & (\alpha.x)\lceil\mu = & \begin{cases} \underline{\mu}.(x\lceil\mu) \text{ if } \alpha = \mu \\ \alpha.(x\lceil\mu) \text{ otherwise} \end{cases} \\
\lceil 3 & (x + \underline{\tau}.y + \underline{a}.z)\lceil\mu = & (x + \underline{\tau}.y)\lceil\mu + \underline{a}.(z\lceil\mu) \\
\lceil 4 & (x + \nu_1.y + \nu_2.z)\lceil\mu = & (x + \nu_1.y)\lceil\mu + (x + \nu_2.z)\lceil\mu \quad \nu_1, \nu_2 \in Act \setminus \{\underline{\tau}\} \\
\lfloor 1 & nil\lfloor\underline{\mu} = & nil \\
\lfloor 2 & (\alpha.x)\lfloor\underline{\mu} = & \begin{cases} \mu.(x\lfloor\underline{\mu}) \text{ if } \alpha = \underline{\mu} \\ \alpha.(x\lfloor\underline{\mu}) \text{ otherwise} \end{cases} \\
\lfloor 3 & (x + \underline{\tau}.y + \underline{a}.z)\lfloor\underline{\mu} = & (x + \underline{\tau}.y)\lfloor\underline{\mu} + \underline{a}.(z\lfloor\underline{\mu}) \\
\lfloor 4 & (x + \nu_1.y + \nu_2.z)\lfloor\underline{\mu} = & (x + \nu_1.y)\lfloor\underline{\mu} + (x + \nu_2.z)\lfloor\underline{\mu} \quad \nu_1, \nu_2 \in Act \setminus \{\underline{\tau}\}
\end{array}
$$

**Fig. 3.** Axioms for $\lceil$ and $\lfloor$.

**Theorem 2.** *Let $p$ and $q$ be finite processes. Then $p \simeq q$ if and only if $\vdash_{E_{\simeq}} p = q$.*

## 3 Observational Equivalence and Congruence

The equivalence $\simeq$ treats $\underline{\tau}$ and $\tau$ actions no differently from other actions. This complicates the use of $\simeq$ for verification purposes, since equivalent processes must therefore exhibit the same "levels" of internal behavior. For example, $a.\underline{\tau}.b$ and $a.b$ are not related by $\simeq$. However, to an external observer, they should be indistingishable, since $\underline{\tau}$ is an internal action, and thus $\simeq$ may be seen as too discriminating. In this section our aim is to define define a congruence relation over processes which abstracts away from the internal computation that processes engage in and thus relates processes if they have the same *external* behavior. Our approach follows that of [15] for the definition of observational congruence. We first define a straightforward modification of $\simeq$ by introducing a new transition relation $\Rightarrow_p$ that "absorbs" internal computation steps. This new equivalence turns out not to be a congruence. However, it does contain a largest congruence; the bulk of this section is devoted to the very challenging task of characterizing this relation behaviorally.

In order to define our reference equivalence, we introduce the following definitions.

**Definition 3.**
1. $\stackrel{\epsilon}{\Rightarrow}_p = (\stackrel{\tau}{\rightarrow} \cup \stackrel{\underline{\tau}}{\rightarrow})^*$.
2. Let $\alpha \in Act$. Then $\stackrel{\alpha}{\Rightarrow}_p = \stackrel{\epsilon}{\Rightarrow}_p \circ \stackrel{\alpha}{\rightarrow} \circ \stackrel{\epsilon}{\Rightarrow}_p$.
3. Let $\alpha \in Act$. Then $\hat{\alpha} = \epsilon$ if $\alpha \in \{\tau, \underline{\tau}\}$ and $\alpha$ otherwise.

Intuitively, $p \stackrel{\epsilon}{\Rightarrow}_p q$ if $p$ may engage in a sequence of prioritized and unprioritized internal transitions and evolve to $q$, while $p \stackrel{\alpha}{\Rightarrow}_p q$ holds if $p$ may engage in internal computation, and then $\alpha$, and then more internal computation and evolve to $q$. $\hat{\alpha}$ represents the "visible content" of action $\alpha$.

Now we define *naive weak bisimulations* as follows.

**Definition 4.** A relation $R \subseteq \mathcal{P} \times \mathcal{P}$ is called a *naive weak bisimulation* if whenever $pRq$ then for every $\alpha \in Act$ the following conditions hold:

$$p \stackrel{\alpha}{\rightarrow} p' \Rightarrow \exists q' : \; q \stackrel{\hat{\alpha}}{\Rightarrow}_p q' \wedge p' R q'. \tag{1}$$

$$q \stackrel{\alpha}{\rightarrow} q' \Rightarrow \exists p' : \; p \stackrel{\hat{\alpha}}{\Rightarrow}_p p' \wedge p' R q'. \tag{2}$$

Define $p \approx_p q$ to hold if $p \, R \, q$ holds for some naive weak bisimulation $R$. It is straightforward to establish that $\approx_p$ is the largest naive weak bisimulation and that it is an equivalence relation over processes.

In order for an equivalence relation over the set of processes to be a satisfactory semantic equivalence it is imperative that the relation be a congruence with respect to all the process constructors present in the language. This is due to the fact that compositional reasoning is the main tool for system verification in the process–algebraic framework. Unfortunately the relation $\approx_p$ is *not* a congruence. Unsurprisingly, $\approx_p$ is not preserved by the choice operator $+$. More disturbingly, $\approx_p$ is also not preserved by parallel composition. Consider the following examples. In each case $p \approx_p q$ but $p \mid r \not\approx_p q \mid r$.

*Example 1.*
1. $p = \underline{\mu}.\tau.\underline{\mu}, \quad q = \underline{\mu}.\underline{\mu}, \quad r = fix(x : \underline{\tau}.x)$.
2. $p = \underline{a} + \tau.\underline{a}, \quad q = \tau.\underline{a}, \quad r = \overline{a} + \mu$.
3. $p = a.(\underline{b} + \tau.\mu), \quad q = a.(\underline{b} + \tau.\mu) + a.\mu, \quad r = \overline{b} + \underline{\mu}$.
4. $p = nil, \quad q = fix(x : \underline{\tau}.x), \quad r = \mu$.

In the first example $q \mid r \xrightarrow{\underline{\mu}} \underline{\mu} \mid r \xrightarrow{\underline{\mu}}$, while no $\stackrel{\underline{\mu}}{\Rightarrow}_p$-derivative of $p|r$ can perform a $\underline{\mu}$ (note that this shows that the first $\tau$-law of Milner [15] is in general not valid). In the second, $q \mid r \xrightarrow{\underline{\mu}}$ whereas $p \mid r \not\stackrel{\underline{\mu}}{\Rightarrow}_p$; this shows that in general, Milner's second $\tau$-law is not valid. In the third, $q \mid r$ can perform the action $a$ to reach a state in which both actions $\mu$ and $\underline{\mu}$ are enabled. However $p \mid r$ can never to evolve to an equivalent state; hence Milner's third $\tau$-law does not hold. Finally, in the fourth example, $p \mid r$ can perform the action $\mu$, but $q \mid r$ can never a reach a state from which it can perform the action $\mu$.

One reason that $\mid$ does not preserve $\approx_p$ is that $p \approx_p q$ may hold even though $p$ may reach a state that is patient through $\underline{\tau}$-transitions while $q$ cannot (see *Example* 1(4)). Another is (see 1 through 3 of *Example* 1) that although $q$ and $r$ may both be patient, $q \mid r$ may not be because of the possibility of synchronizations between them involving prioritized actions. Thus, it may be the case that when $p \approx_p q$, a $\xrightarrow{\alpha}$-transition of $p$ might only be matched by $\stackrel{\hat{\alpha}}{\Rightarrow}_p$-transitions that include at least one $\xrightarrow{\tau}$-transition. When put in parallel with $r$, however, it may be the case that potential prioritized synchronizations between $r$ and some derivative of $q$ will prevent the possibility of these $\tau$-transitions in $q$. Therefore, in order to characterize the largest congruence contained in $\approx_p$, we must take account of the possibility of reaching patient states through $\underline{\tau}$-transitions and potential preemptability of $\tau$-transitions that processes engage in. With this motivation, we introduce the following definitions.

**Definition 5.** Let $p \in \mathcal{P}$.

1. $I(p) = \{\, \lambda \in Act \setminus \{\tau, \underline{\tau}\} \mid p \xrightarrow{\lambda} \}$ is the set of visible actions $p$ may initially perform. We also use $I_A(p) = I(p) \cap A$ and $I_{\underline{A}}(p) = I(p) \cap \underline{A}$ to denote the visible unprioritized and prioritized actions initially available to $p$.
2. $p \Downarrow$ holds if for some $q$, $p(\xrightarrow{\underline{\tau}})^* q$ and $q$ is patient.

Intuitively, $p \Downarrow$ means that it is *possible* for the process $p$ to evolve to a patient process through a sequence of zero or more $\underline{\tau}$ actions. Note that this definition differs from the notion of *convergence* that one often finds in the literature in that if $p \Downarrow$ holds, then it is still possible for $p$ to engage in an infinite sequence of $\underline{\tau}$ actions.

**Definition 6.**
1. $\stackrel{\epsilon}{\Rightarrow} \subseteq \mathcal{P} \times \mathcal{P} = (\xrightarrow{\underline{\tau}})^*$. We redefine $\hat{\underline{\tau}} = \underline{\epsilon}$.
2. Let $\alpha \in Act \setminus \{\tau\}$. Then $\stackrel{\alpha}{\Rightarrow} = \stackrel{\epsilon}{\Rightarrow} \circ \xrightarrow{\alpha} \circ \stackrel{\epsilon}{\Rightarrow}$
3. Let $L \subseteq Act \setminus \{\tau, \underline{\tau}\}$. Then $\underset{L}{\stackrel{\epsilon}{\Longrightarrow}}$ is defined inductively as follows. $p \underset{L}{\stackrel{\epsilon}{\Longrightarrow}} q$ if
   - $p = q$ or
   - $\exists r : p \underset{L}{\stackrel{\epsilon}{\Longrightarrow}} r \wedge r \xrightarrow{\underline{\tau}} q$ or
   - $\exists r : p \underset{L}{\stackrel{\epsilon}{\Longrightarrow}} r \wedge r \xrightarrow{\tau} q \wedge I(r) \subseteq L$.

   We write $p \underset{L}{\stackrel{\tau}{\Longrightarrow}} q$ if $\exists p', q' : p \underset{L}{\stackrel{\epsilon}{\Longrightarrow}} p' \xrightarrow{\tau} q' \underset{L}{\stackrel{\epsilon}{\Longrightarrow}} q$ with $I(p') \subseteq L$.

Intuitively, $p \underset{L}{\stackrel{\epsilon}{\Longrightarrow}} q$ holds if there exists a sequence of processes $p_0, \cdots, p_n$ such that $p \equiv p_0 \xrightarrow{1} p_1 \xrightarrow{1} \cdots \xrightarrow{1} p_n \equiv q$ and such that if $p_i$ is patient (meaning $p_i \xrightarrow{\tau} p_{i+1}$) then the initial visible actions available to $p_i$ must be contained in $L$. For example, $\underline{\tau}.(\underline{b} + \tau.c) \underset{L}{\stackrel{\epsilon}{\Longrightarrow}} c$ holds for those $L$ containing $\underline{b}$. The next lemma remarks on several straightforward properties of $\underset{L}{\stackrel{\epsilon}{\Longrightarrow}}$.

**Lemma 7.** *For any $L \subseteq Act \setminus \{\tau, \underline{\tau}\}$ and $p, q \in \mathcal{P}$ the following are true.*

1. $\underset{L}{\stackrel{\tau}{\Longrightarrow}} \circ \stackrel{\epsilon}{\Rightarrow} = \stackrel{\epsilon}{\Rightarrow} \circ \underset{L}{\stackrel{\tau}{\Longrightarrow}} = \underset{L}{\stackrel{\tau}{\Longrightarrow}}$.
2. *If* $p \stackrel{\epsilon}{\Rightarrow} q$ *then* $p \underset{L}{\stackrel{\epsilon}{\Longrightarrow}} q$.

3. *If* $p \stackrel{\tau}{\rightarrow} q$ *and* $I(p) \subseteq L$ *then* $p \underset{L}{\stackrel{\tau}{\Rightarrow}} q$.
4. $\underset{L}{\stackrel{\epsilon}{\Rightarrow}} \subseteq \underset{M}{\stackrel{\epsilon}{\Rightarrow}}$ *if* $L \subseteq M$.

To understand the import of $\underset{L}{\stackrel{\epsilon}{\Rightarrow}}$ suppose that $p \underset{L}{\stackrel{\epsilon}{\Rightarrow}} q$, and suppose further that $r$ is patient and is incapable of synchronizing with any prioritized action in $L$. We then can assert that $p \mid r \stackrel{\epsilon}{\Rightarrow}_{\mathcal{P}} q \mid r$. This fact is a consequence of the following lemma.

**Lemma 8.** *Assume that* $p, q, r$ *and* $L$ *are such that* $p \underset{L}{\stackrel{\epsilon}{\Rightarrow}} q$ , $r \stackrel{\underline{\tau}}{\not\rightarrow}$ *and* $I_{\underline{A}}(r) \cap \overline{L} = \emptyset$. *Then* $p \mid r \underset{M}{\stackrel{\epsilon}{\Rightarrow}} q \mid r$ *where* $M = L \cup I(r)$.

One may also wonder why the definition of $p \underset{L}{\stackrel{\epsilon}{\Rightarrow}} q$ constrains the unprioritized as well as the prioritized actions of the patient intermediate states between $p$ and $q$ (i.e. why "$I(r) \subseteq L$" rather than "$I_{\underline{A}}(r) \subseteq L$" is used in the definition). This requirement is needed to ensure that the equivalence we are about to define is a congruence for the prioritization operator $\lceil$.

We may introduce ***prioritized weak bisimulations*** as follows.

**Definition 9.** A relation $R \subseteq \mathcal{P} \times \mathcal{P}$ is called a *prioritized weak bisimulation (pwb* in short) if for all $\langle p, q\rangle \in R$ and $\alpha \in Act$ the following conditions hold.

$$p \Downarrow \;\Leftrightarrow\; q \Downarrow \tag{3}$$

$$p \stackrel{\alpha}{\rightarrow} p' \;\Rightarrow\; \begin{cases} \exists q' : \; q \stackrel{\hat{\alpha}}{\Rightarrow} q' \wedge p' R q' & \textit{if } \alpha \neq \tau \\ \exists q' : \; q \underset{I(p)}{\stackrel{\epsilon}{\Rightarrow}} q' \wedge p' R q' & \textit{if } \alpha = \tau \end{cases} \tag{4}$$

$$q \stackrel{\alpha}{\rightarrow} q' \;\Rightarrow\; \begin{cases} \exists p' : \; p \stackrel{\hat{\alpha}}{\Rightarrow} p' \wedge p' R q' & \textit{if } \alpha \neq \tau \\ \exists p' : \; p \underset{I(q)}{\stackrel{\epsilon}{\Rightarrow}} p' \wedge p' R q' & \textit{if } \alpha = \tau \end{cases} \tag{5}$$

The next lemma establishes several important properties enjoyed by pwb's.

**Lemma 10.** *Let* $R$ *be a pwb with* $pRq$, *and let* $L \subseteq Act \setminus \{\tau, \underline{\tau}\}$. *Then*

$$p \underset{L}{\stackrel{\epsilon}{\Rightarrow}} p' \;\Rightarrow\; \exists q' : \; q \underset{L}{\stackrel{\epsilon}{\Rightarrow}} q' \;\wedge\; p' R q'. \tag{6}$$

$$p \stackrel{\underline{\tau}}{\not\rightarrow} \;\Rightarrow\; \exists q' : \; q \stackrel{\epsilon}{\Rightarrow} q' \;\wedge\; pRq' \;\wedge\; q' \stackrel{\underline{\tau}}{\not\rightarrow}. \tag{7}$$

$$p \stackrel{\underline{\tau}}{\not\rightarrow} \;\Rightarrow\; I(q) \subseteq I(p). \tag{8}$$

Define $p \gtrsim q$ to hold if there is a pwb $R$ with $pRq$. It is straightforward to establish that $\gtrsim$ is the largest pwb and that it is also an equivalence relation. In addition, we have that $\simeq$ is strictly contained in $\gtrsim$. The following lemma states that $\gtrsim$ is preserved under all operators but non-deterministic choice and recursion.

**Lemma 11.** *Let* $p, q, r \in \mathcal{P}$ *and* $p \gtrsim q$. *Then the following hold.*

$$\begin{array}{lll} \alpha.p \gtrsim \alpha.q & \quad p \mid r \gtrsim q \mid r & \quad p \backslash \lambda \gtrsim q \backslash \lambda \\ p[f] \gtrsim q[f] & \quad p \lceil \mu \gtrsim q \lceil \mu & \quad p \lfloor \underline{\mu} \gtrsim q \lfloor \underline{\mu} \end{array}$$

Because $\gtrsim$ is insensitive to initial internal computation, it is not preserved by nondeterministic choice. This problem can be fixed in the usual fashion by requiring that any silent action of one process be matched by at least one silent action of the equivalent process besides the other requirements for observational equivalence.

**Definition 12.** Define $p \gtrsim^{+} q$ to hold if for every $\alpha \in Act$ the following conditions hold.

$$p \stackrel{\alpha}{\rightarrow} p' \;\Rightarrow\; \begin{cases} \exists q' : \; q \stackrel{\alpha}{\Rightarrow} q' \wedge p' \gtrsim q' & \textit{if } \alpha \neq \tau \\ \exists q' : \; q \underset{I(p)}{\stackrel{\tau}{\Rightarrow}} q' \wedge p' \gtrsim q' & \textit{if } \alpha = \tau \end{cases} \tag{9}$$

$$q \stackrel{\alpha}{\rightarrow} q' \;\Rightarrow\; \begin{cases} \exists p' : \; p \stackrel{\alpha}{\Rightarrow} p' \wedge p' \gtrsim q' & \textit{if } \alpha \neq \tau \\ \exists p' : \; p \underset{I(q)}{\stackrel{\tau}{\Rightarrow}} p' \wedge p' \gtrsim q' & \textit{if } \alpha = \tau \end{cases} \tag{10}$$

```
Input: X, an equivalence relation over processes.
Output: The largest congruence contained in X.
Let Y = X.
while Y is not a congruence do {
 Given processes p and q, choose a specific context C(p,q)
        such that Z = {<p,q>|C(p,q)[p] Y C(p,q)[q]} is contained in Y.
 Replace Y by an equivalence lying between Y and Z.
}
Y is the largest congruence contained in X
```

**Fig. 4.** Methodology for identifying the largest congruence

We now have the following.

**Lemma 13.** *$p \gtrapprox^+ q$ if and only if $\forall r \in \mathcal{P}: \ p + r \gtrapprox q + r$.*

To show that $\gtrapprox^+$ is also preserved by recursive definition, we first extend $\gtrapprox^+$ to open terms in the usual way. Let $free(e)$ denote the set of all *free* variables in the expression $e$ and let $e\{p/x\}$ denote the expression obtained from $e$ by textually substituting every free occurrence of the variable $x$ by the process $p$.

**Definition 14.** Let $e, f \in Exp$ with $free(e) \subseteq \{x\}$ and $free(f) \subseteq \{x\}$. Then $e \gtrapprox^+ f$ if $\forall p \in \mathcal{P}$ : $e\{p/x\} \gtrapprox^+ f\{q/x\}$.

**Lemma 15.** *Let $e, f \in Exp$ with $free(e) \subseteq \{x\}$ and $free(f) \subseteq \{x\}$, and suppose $e \gtrapprox^+ f$. Then $fix(x : e) \gtrapprox^+ fix(x : f)$.*

The proof of this lemma involves the definition of the concept of *prioritized weak bisimulation up to* along the lines given in Proposition 1 of Sangiorgi and Milner [17].

We now have the results needed to prove the following theorem.

**Theorem 16.** *$\gtrapprox^+$ is a congruence with respect to all operators in the language, and it is the largest such relation contained in $\gtrapprox$.*

### Relating $\gtrapprox^+$ and $\approx_p$

In this subsection we establish that $\gtrapprox^+$ coincides with the largest congruence ${\approx_p}^C$ contained in $\approx_p$. We first recall the following well-known result from universal algebra.

**Theorem 17.** *Let $=$ be an equivalence relation over some algebra, and define $t =^C u$ to hold if $C[t] = C[u]$ for any context $C[\,]$. Then $=^C$ is the largest congruence contained in $=$.*

Thus we know that the largest congruence ${\approx_p}^C \subseteq \approx_p$ exists. The theorem we wish to establish is the following.

**Theorem 18.** *${\approx_p}^C$ coincides with $\gtrapprox^+$.*

The technique we use to establish this result may be seen as an instance of a general methodology given in Figure 4. The validity of the method can be established by proving $X^C = Y^C$ is an invariant of the while-loop. Clearly the predicate is true before the first iteration, as $X$ and $Y$ are the same. During an iteration $Y$ gets replaced by a set which lies between $Z$ and $Y$, and as $X^C \subseteq Z$ must hold, we have that $X^C \subseteq Y$ must also be true. Then the following lemma can be used to conclude that the invariant is maintained.

**Lemma 19.** *Let $X$ and $Y$ be equivalences such that $X^C \subseteq Y \subseteq X$. Then $X^C = Y^C$.*

If $Y$ is a congruence then $Y^C = Y$. As the while loop is exited only when $Y$ is congruence, it follows that $Y = X^C$. Intuitively, the method identifies the largest congruence by generating a decreasing sequence of approximations.

To use this methodology to identify ${\approx_p}^C$ we first introduce a specific context $\mathcal{K}_{p,q}[\,]$ given a pair of processes $p$ and $q$ (see Definition 20). Then we define ${\approx_p}^1 \; = \; \{ \langle p, q \rangle \mid \mathcal{K}_{p,q}[p] \approx_p \mathcal{K}_{p,q}[q] \}$ ; this relation plays the role of "Z" in the methodology. Then we introduce a new equivalence $\underline{\approx}^f$ which can be shown to lie between ${\approx_p}^1$ and $\approx_p$; this relation becomes the new value of "Y" in the methodology. So we have that ${\approx_p}^C \subseteq {\approx_p}^1 \subseteq \underline{\approx}^f \subseteq \approx_p$, and thus from Lemma 19 we know that ${\approx_p}^C = \underline{\approx}^{fC}$.

As $\underline{\approx}^f$ is not a congruence, we now must repeat the body of the while loop; our goal now is to isolate $\underline{\approx}^{fC}$, since we know that it coincides with ${\approx_p}^C$. So given a pair of processes $p$ and $q$, we introduce the context $\mathcal{C}_{p,q}[\,] = [\,] \mid c.nil$, where $c$ is an unprioritized visible action which does not belong to either of sort of $p$ or sort of $q$, and define ${\approx_p}^2 \; = \; \{ \langle p, q \rangle \mid \mathcal{C}_{p,q}[p] \; \underline{\approx}^f \; \mathcal{C}_{p,q}[q] \}$. This relation now plays the role of "Z" in the methodology. We have that $\underline{\approx}^{fC} \subseteq {\approx_p}^2$. But it can be shown that ${\approx_p}^2$ is a pwb, which means that ${\approx_p}^2$ is contained in $\underline{\approx}$. From the definition of $\underline{\approx}^f$, it is also easy to establish that $\underline{\approx} \subseteq \underline{\approx}^f$, and thus in our methodology "Y" is assigned the value $\underline{\approx}$. Then from Lemma 19, it follows that $\underline{\approx}^{fC} = \underline{\approx}^C$. This concludes the second iteration of the while-loop in the methodology.

As $\underline{\approx}$ is not a congruence, we could iterate the while-loop once again. However, we instead appeal to the Theorem 16, which establishes that $\underline{\approx}^C = \underline{\approx}^+$. This implies that ${\approx_p}^C = \underline{\approx}^{fC} = \underline{\approx}^C = \underline{\approx}^+$, and the desired result then follows.

In order to complete the proof of the theorem we need to do the following: (1) define the context $\mathcal{K}_{p,q}$; (2) define $\underline{\approx}^f$, and show it is an equivalence lying between ${\approx_p}^1$ and $\approx_p$; (3) establish $\underline{\approx}^f$ is *not* a congruence; (4) show ${\approx_p}^2$ is a pwb; and (5) establish $\underline{\approx}$ is contained in $\underline{\approx}^f$.

We begin with the definition of the context $\mathcal{K}_{p,q}$. Intuitively, the role of this context serves as the link between $\Rightarrow_p$ and $\Rightarrow$; that is, we want the $\Rightarrow_p$-transitions of $\mathcal{K}_{p,q}[p]$ and $\mathcal{K}_{p,q}[q]$ to enable us to deduce information about the $\Rightarrow$-transitions of $p$ and $q$, respectively. The structure of the context $\mathcal{K}_{p,q}[\,]$ is somewhat complex, and its description demands some auxiliary notation. Let $S_U(p), S_P(p)$ be the unprioritized sort and prioritized sort, respectively, of the process $p$. These sets are finite for any process $p$ because of the restriction placed on relabelings. We use $S_{p,q}$ for $S_U(p) \cup S_U(q)$. Now define a context $U_{p,q}[\,]$ analogous to the context "$D_{p,q}[\,]$" from [8] as follows. $U_{p,q}[r] \; = \; ([r][L_{p,q}])\lceil S_{p,q}$ where the relabeling function $L_{p,q}$ is defined as

$$L_{p,q}(\alpha) = \begin{cases} \alpha & \text{if } \alpha \in A \\ \alpha & \text{if } \underline{\lambda} = \alpha \text{ and } \lambda \notin S_{p,q} \\ \underline{k}_\alpha & \text{otherwise} \end{cases}$$

where $\underline{k}_\alpha \in \underline{A}$, $\underline{k}_{\overline{\alpha}} = \overline{\underline{k}_\alpha}$, $\underline{k}_\alpha \neq \underline{k}_\beta$ if $\alpha \neq \beta$, and $k_\alpha \notin S_{p,q}$ and $\underline{k}_\alpha \notin S_P(p) \cup S_P(q)$. Since the set *Act* of actions is infinite, such a $k_\alpha$ is guaranteed to exist. The effect of this context $U_{p,q}[\,]$ on its "argument" is to prioritize uniquely those of its unprioritized actions which lie in $S_{p,q}$.

**Definition 20.** If L $= \{l_1 \cdots l_k\} \subseteq Act$, then $\langle L \rangle$ represents the process $l_1.nil + \cdots + l_k.nil$, with $\langle \emptyset \rangle \equiv nil$ and $\langle \{\alpha\} \rangle \equiv \alpha.nil$. For $p, q \in \mathcal{P}$, let $\mathcal{A}_{p,q}$ denote the set $S_P(U_{p,q}[p]) \cup S_P(U_{p,q}[q])$. Also let $\mathcal{Q}_{p,q}$ denote the process

$$fix(X : \underline{c} + \textstyle\sum_{L \subseteq \overline{\mathcal{A}_{p,q}}} \underline{\tau}.(\langle L \rangle + \underline{d}_L.X)),$$

where $\underline{c} \in \underline{A} \setminus \{\underline{\tau}\}$ and $\underline{d}_L \in \underline{A} \setminus \{\underline{\tau}\}$ do not belong to $S(p) \cup S(q)$, and $\underline{d}_L = \underline{d}_{L'}$ if and only if $L = L'$. Then the context $\mathcal{K}_{p,q}[\,]$ is defined as $\mathcal{Q}_{p,q} \mid U_{p,q}[\,]$

Note that we may use the $\Rightarrow_p$-transitions of $K_{p,q}[p]$ to infer $\Rightarrow$-transitions in $p$ as follows. For simplicity assume that the sorts of $p$ and $q$ contain no unprioritized actions. Then for $\alpha$ in $S_P(p) \cup S_P(q)$, $\mathcal{K}_{p,q}[p] \overset{\alpha}{\Rightarrow}_p \mathcal{K}_{p,q}\,[p']$ if and only if $p \overset{\alpha}{\Rightarrow} p'$, while $\mathcal{K}_{p,q}[r] \overset{d_L}{\Rightarrow}_p \mathcal{K}_{p,q}[r']$ if and only if $p \underset{M}{\overset{\epsilon}{\Rightarrow}} p'$ where $L \cap \overline{M} = \emptyset$.

*Example 2.* Let $p = \tau.(\overline{\underline{a}} + \tau.nil) + \overline{b}.nil$ and $q = p$. Then $\mathcal{A}_{p,q} = \{\overline{\underline{a}}, \overline{\underline{b}}\}$ and $\overline{\mathcal{A}_{p,q}} = \{\underline{a}, \underline{b}\}$. So,

$$\mathcal{Q}_{p,q} = fix(X : \underline{c} + \underline{\tau}.(\underline{a} + \underline{b} + \underline{d}_1.X) + \underline{\tau}.(\underline{a} + \underline{d}_2.X) + \underline{\tau}.(\underline{b} + \underline{d}_3.X) + \underline{\tau}.(\underline{d}_4.X))$$

Turning to the second item on the list of things needed to complete the proof of Thoerem 18, we define the relation $\underline{\approx}^f$ by means of a class of relations called *finite prioritized weak bisimulations.*

$$\begin{array}{ll} \text{iA1} & \alpha.x \sqsubseteq_i \alpha.y \\ \text{iA2} & \tau.nil \sqsubseteq_i nil \\ \text{iA3} & nil \sqsubseteq_i \nu.nil \quad \nu \in Act \setminus \{\underline{\tau}\} \end{array}$$

**Fig. 5.** Axioms for $\sqsubseteq_i$.

$$\begin{array}{lr} \underline{\tau}1 & \alpha.(1.x + x) = \alpha.x \\ \underline{\tau}2 & \underline{\tau}.x = \underline{\tau}.x + x \\ \underline{\tau}3 & \alpha(x + \underline{\tau}.y) = \alpha(x + \underline{\tau}.y) + \alpha.y \\ \tau 1 & z + \tau.(x + \tau.y) = z + \tau.(x + \tau.y) + \tau.y \quad (\vdash x \sqsubseteq_i z) \end{array}$$

**Fig. 6.** The $\tau$ laws.

**Definition 21.** A relation $R \subseteq \mathcal{P} \times \mathcal{P}$ is called a *finite prioritized weak bisimulation* (fpwb in short) if for all $\langle p, q\rangle \in R$ and $\alpha \in Act$ the following conditions hold.

$$p \xrightarrow{\alpha} p' \;\Rightarrow\; \begin{cases} \exists q' : q \overset{\hat{\alpha}}{\Rightarrow} q' \wedge p' R q' & if\ \alpha \neq \tau \\ \exists q' : q \underset{I(p)}{\overset{\epsilon}{\Longrightarrow}} q' \wedge p' R q' & if\ \alpha = \tau \end{cases} \tag{11}$$

$$q \xrightarrow{\alpha} q' \;\Rightarrow\; \begin{cases} \exists p' : p \overset{\hat{\alpha}}{\Rightarrow} p' \wedge p' R q' & if\ \alpha \neq \tau \\ \exists p' : p \underset{I(q)}{\overset{\epsilon}{\Longrightarrow}} p' \wedge p' R q' & if\ \alpha = \tau \end{cases} \tag{12}$$

The only difference between a pwb and a fpwb is that pwb's require related processes to satisfy: $p \Downarrow \iff q \Downarrow$. $\approx^f$ is defined as the largest fpwb, and it can be shown to be an equivalence relation by usual arguments. It is easy to see that every fpwb is a naive weak bisimulation and thus it follows that $\approx^f$ is contained in $\approx_p$. The fact that ${\approx_p}^1$ is contained in $\approx^f$ is obtained from the following lemma.

**Lemma 22.** *${\approx_p}^1$ is a fpwb.*

Now we remark on the last three items needed to finish the proof of Theorem 18. $\approx^f$ turns out not to be congruence, as in general parallel composition is not preserved; the processes in Example 1(4) provide an illustration of this, since it can be shown that $p \approx^f q$ but $p \mid r \not\approx^f q \mid r$. ${\approx_p}^2$ can be shown to be a pwb in a straightforward manner. In addition, one may easily show that any pwb is also a fpwb; consequently, $\approx$ is contained in $\approx^f$.

## 4 Algebraic Characterization of ${\approx_p}^C$

In this section we provide a sound and complete axiomatization of $\approx^+$ over finite processes: two finite processes are related by the prioritized observational congruence if and only if this can be derived *syntactically* from the axioms using the laws of equational reasoning.

Let $E$ denote the set of axioms given in Figure 6 plus those axioms which are present in $E_\simeq$. Our aim is to show $E$ is sound and complete with respect to $\approx^+$. The soundness of all the axioms except $\tau 1$ is clear. Axiom $\tau 1$ involves a side condition ($\vdash x \sqsubseteq_i z$). $\sqsubseteq_i$ is the precongruence on finite processes generated from the axioms presented in the Figure 5 using the laws of inequational reasoning. The behavioral interpretation of this relation is made precise in the following lemma.

**Lemma 23.** *Let $\vdash p \sqsubseteq_i q$. Then $I(p) \subseteq I(q)$, and $p$ is patient if and only if $q$ is patient. In addition, if $p$ and $q$ are finite processes such that $I(p) \subseteq I(q)$ and such that either both or neither is patient then there exists some $p'$ and $q'$ such that $p \simeq p'$ and $q \simeq q'$ and $\vdash p' \sqsubseteq_i q'$.*

In light of the above lemma, when ($\vdash x \sqsubseteq_i z$), $I(x) \subseteq I(z)$. Then it is immediate that axiom $\tau 1$ is sound.

Completeness of the axioms is proved along the same lines as that given in Section 7.4 of [15]. Any finite process can be converted into a *normal form* [8] by eliminating all static combinators using axioms $E_\simeq$. Then we need the following analogues of theorems from [15].

**Lemma 24.**

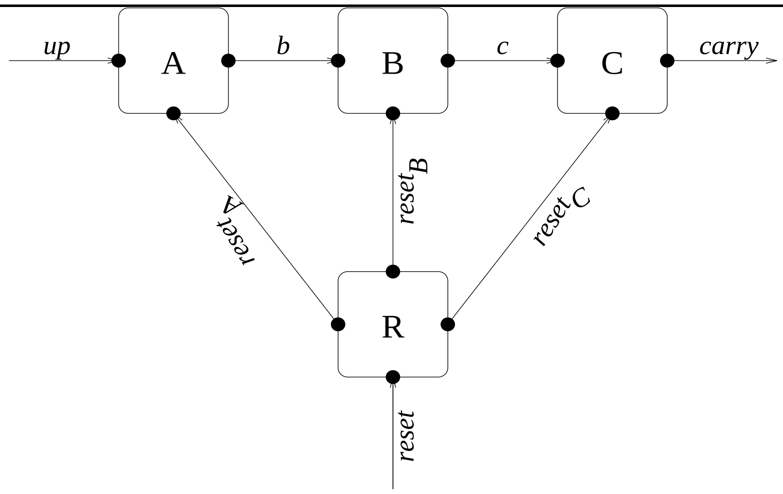


**Fig. 7.** Architecture of the 8-resettable counter implementation.

1. *Suppose that* $p \stackrel{\alpha}{\Rightarrow} q$ *for* $\alpha \neq \tau$. *Then* $\vdash\ p =_E p + \alpha.q$.
2. *Suppose that* $p \underset{I(p)}{\stackrel{\tau}{\Longrightarrow}} q$. *Then* $\vdash\ p =_E p + \tau.q$.

**Lemma 25.** *If* $p \gtrsim q$ *then* $(p \gtrsim^+ q)$ *or* $(p \gtrsim^+ q + 1.q)$ *or* $(p + 1.p \gtrsim^+ q)$.

**Theorem 26.** *Let* $p, q$ *be finite processes. Then* $p \gtrsim^+ q$ *if and only if* $\vdash p =_E q$.

## 5 An Example

In this section we apply our definition of $\gtrsim^+$ to reason about the relationship between a specification and an implementation of a system, which we call an 8-resettable counter, that can take two kinds of input, namely *up* and *reset*, and outputs *carry* immediately after 8 consecutive inputs of the kind *up*.

A formal specification the system is given by the process *Spec* defined below.

$$\begin{aligned} Spec &\Leftarrow S_0 \\ S_i &\Leftarrow up.S_{i+1} + reset.S_0 \quad 0 \leq i \leq 6 \\ S_7 &\Leftarrow up.\overline{carry}.S_0 + reset.S_0 \end{aligned}$$

Note that execution of *reset* causes the counter to become "zero".

Suppose now that we are asked to implement *Spec* using 2-resettable counters, which are specified as follows.

$$Cell \Leftarrow up.Cell' + reset.Cell, \quad Cell' \Leftarrow up.\overline{carry}.Cell + reset.Cell$$

A natural way to implement this specification is to compose three such Cells serially, connecting the *carry* ports of one cell to the *up* port of another cell (see Figure 7). The resulting process *Sys* is given below.

$$Sys \Leftarrow (A|B|C|R)\backslash\{b, c, reset_A, reset_B, reset_C\}$$

$$A \Leftarrow Cell[\overline{b}/\overline{carry},\ reset_A/reset], \quad B \Leftarrow Cell[b/up,\ \overline{c}/\overline{carry},\ reset_B/reset],$$

$$C \Leftarrow Cell[c/up,\ reset_C/reset] \quad R \Leftarrow reset.\overline{reset_A}.\overline{reset_B}.\overline{reset_C}.R$$

In order to prove that *Sys* is correct, we can try to establish that $Sys \gtrsim^+ Spec$. Unfortunately, this fails to be the case; in fact, the two systems are not even trace equivalent. To see this, note that *Sys* has a trace which begins with "*up up up up reset up up up up* $\overline{carry}$". But in any trace of *Spec* every occurrence of $\overline{carry}$ should be immediately preceded by 8 consecutive occurrences of *up*, with no intervening occurrences of *reset*. So *Sys* does not implement *Spec*.

The cause of the problem with *Sys* is that the process $A$ can choose to ignore the $\mathit{reset}_A$ and still perform the action *up*. To remedy this problem, we might try to prioritize the reset actions of $A$, $B$ and $C$. The resulting system, *Sys1*, is given as follows.

$$\mathit{Sys1} \Leftarrow (A1|B1|C1|R1)\backslash\{b, c, \underline{\mathit{reset}_A}, \underline{\mathit{reset}_B}, \underline{\mathit{reset}_C}\}$$

$$A1 \Leftarrow A\lceil\{\mathit{reset}_A\}, \quad B1 \Leftarrow B\lceil\{\mathit{reset}_B\}, \quad C1 \Leftarrow C\lceil\{\mathit{reset}_C\}, \quad R1 \Leftarrow R\lceil\{\mathit{reset}_A, \mathit{reset}_B, \mathit{reset}_C\}$$

However, it is still not the case that $\mathit{Sys1} \underset{\sim}{\approx}^{+} \mathit{Spec}$. The reason for this inequivalence is somewhat subtle. To see why it is not possible to construct a pwb containing a pair consisting of *Spec* and *Sys1*, suppose *Spec* evolves to $S_2$ by performing the sequence of actions "*up up*". Now there is only one process *Sys1* can evolve to by means of $\overset{up}{\Rightarrow} \circ \overset{up}{\Rightarrow}$, namely

$$Q = (b.(A\lceil\{\mathit{reset}_A\})|B1|C1|R1)\backslash\{b, c, \underline{\mathit{reset}_A}, \underline{\mathit{reset}_B}, \underline{\mathit{reset}_C}\}$$

So any attempt to construct a pwb relating *Spec* and *Sys1* must also relate $S_2$ and $Q$. The only initial actions available to $Q$ are $\tau$ and *reset*; $Q$ is incapable of a $\overset{up}{\Rightarrow}$-transition. Thus $S_2$ and $Q$, and hence *Spec* and *Sys1*, cannot be related by any pwb. In fact, the following context distinguishes them.

$$\mathcal{C} = ([\,]\lceil\{up, out\})|\underline{\tau} + \underline{a}$$

Intuitively, this context does not permit the low priority interaction between the process $A$ and $B$ through the port $b$ while action $\underline{a}$ is enabled.

This last observation suggests that to come up with a correct implementation, we should also prioritize the communications on $b$ and $c$. This leads to the system *Sys2*.

$$\mathit{Sys2} \Leftarrow (A2|B2|C2|R1)\backslash\{\underline{b}, \underline{c}, \underline{\mathit{reset}_A}, \underline{\mathit{reset}_B}, \underline{\mathit{reset}_C}\}$$
$$A2 \Leftarrow A1\lceil\{b\}, \quad B2 \Leftarrow B1\lceil\{b, c\}, \quad C2 \Leftarrow C1\lceil\{c\}.$$

It turns out that *Sys2* is indeed an implementation of *Spec*, as $\mathit{Sys2} \underset{\sim}{\approx}^{+} \mathit{Spec}$. To demonstrate this, one needs to build a pwb relating the two systems and then remark that as neither system is capable of initial internal actions, equivalence with respect to $\underset{\sim}{\approx}$ implies equivalence with respect to $\underset{\sim}{\approx}^{+}$.

## 6 Conclusions

In this paper, we investigated the problem of defining a behavioral equivalence over processes in which actions have different priorities that relates processes on the basis of their external observable behavior. Taking the framework developed in [8] as the foundation, we first extended a strong congruence by abstracting all internal computations in the straightforward fashion to obtain the equivalence $\approx_p$. Then we observed that this relation was not a congruence because of its insensitivity to the potential preemptability of transitions, and we defined a congruence $\underset{\sim}{\approx}^{+}$ that retained such sensitivity and showed that it was the largest such relation contained in $\approx_p$. Establishing $\underset{\sim}{\approx}^{+}$ is the largest congruence contained in $\approx_p$ needed more ingenuity than typically required to prove a similar theorem in other process algebras, owing to the fact that the "base relation", $\approx_p$, was not preserved under parallel composition. We also presented an algebraic characterization of our congruence over finite processes and gave an example illustrating the use of this congruence in system verification.

Regarding future work, we would like to extend the proof system given in Section 4 for finite-*state* processes. This would help us, for instance, in proving $\mathit{Sys2} \underset{\sim}{\approx}^{+} \mathit{Spec}$ by means of equational reasoning. It would also be useful to investigate the adaptation of other observational equivalences for processes with priorities. In particular, modifying branching bisimulation equivalence [19] appropriately appears to yield a somewhat simpler congruence than the CCS-style observational equivalence we used in this paper. Developing an equivalence based on the notion of *testing* [12] would additionally enhance the applicability of this theory. Another line of research is to develop efficient algorithms to compute prioritized weak bisimulations.

Two other groups of researchers have considered the question of observational congruences for languages with priority. Jensen [14] has defined an observational congruence over processes generated by the framework developed in [6]. In this approach, actions do not have priority ascribed to them; rather, the language includes prioritized choice and parallelism operators in which, the actions of one process have precedence over the actions available to the other. The observational congruence is not shown to be the largest one, although the author suggests that it is. However, we believe that the context provided in the Definition 20 can be adapted and used in the methodology given in the Figure 4 to prove this result. In [4] Bol and Groote define a weak observational congruence for a process algebra in which actions are assigned priority statically; the priorities do not take effect, however, until a special "prioritization" process constructor is applied, which has the effect of "turning on" the priorities. They accomplish this by adding rules that endow processes with arbitrary look-ahead along internal transitions; this reduces weak observational congruence to strong congruence. However, they make no attempt to establish the maximality of their congruence, and their framework would yield an undesirable semantics in our setting. For example, by following their approach it would become possible for the term $\underline{a} + \tau.\underline{b} \mid \overline{\underline{a}}$ to engage in a transition labeled $\underline{b}$, even though intuitively this should not be possible, since the $\underline{\tau}$ resulting from the synchronization of $\underline{a}$ and $\overline{\underline{a}}$ should pre-empt the $\tau$ guarding the $\underline{b}$.

## References


1. J. Baetan, J. Bergstra, and J. Klop. Syntax and defining equations for an interrupt mechanism in process algebra. Technical Report CS-R8503, Department of Computer Science, University of Amsterdam, 1985.
2. J. Baeten, editor. *Applications of Process Algebra*, volume 17 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, England, 1990.
3. J. Baeten and W. Weijland. *Process Algebra*, volume 18 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, England, 1990.
4. R. Bol and J. Groote. The meaning of negative premises in transition system specification. In *Proceedings of the International Colloquium on Automata, Languages and Programming*, pages 481–494, Madrid, Spain, July 1991. Springer-Verlag.
5. P. Brémonde-Grégoire, I. Lee, and R. Gerber. ACSR: An algebra of communicating shared resources with dense time and priorities. In *CONCUR '93*, volume 715 of *Lecture Notes in Computer Science*, pages 417–431, Hildesheim, Germany, August 1993. Springer-Verlag.
6. J. Camilleri and G. Winskel. CCS with priority choice. In *Proceedings of the Symposium on Logic in Computer Science*, pages 246–255, Amsterdam, July 1991. Computer Society Press.
7. R. Cleaveland. Analyzing concurrent systems using the Concurrency Workbench. In P. Lauer, editor, *Functional Programming, Concurrency, Simulation and Automated Reasoning*, volume 693 of *Lecture Notes in Computer Science*, pages 129–144. Springer-Verlag, 1993.
8. R. Cleaveland and M. Hennessy. Priorities in process algebra. *Information and Computation*, 87(1/2):58–77, July/August 1990.
9. R. Cleaveland, J. Parrow, and B. Steffen. The Concurrency Workbench: A semantics-based tool for the verification of finite-state systems. *ACM Transactions on Programming Languages and Systems*, 15(1):36–72, January 1993.
10. R. Gerber and I. Lee. CCSR: A calculus for communicating shared resources. In *CONCUR '90*, volume 458 of *Lecture Notes in Computer Science*, pages 263–277, Amsterdam, August 1990. Springer-Verlag.
11. H. Hansson and F. Orava. A process calculus with incomparable priorities. In *Proceedings of the North American Process Algebra Workshop*, 1992.
12. M. Hennessy. *Algebraic Theory of Processes*. MIT Press, Boston, 1988.
13. C. Hoare. *Communicating Sequential Processes*. Prentice-Hall, London, 1985.
14. C.-T. Jensen. Ccs with priorities. Technical report, Computer Science Department, Aarhus University, 1992.
15. R. Milner. *Communication and Concurrency*. Prentice-Hall, London, 1989.
16. J. Parrow. Verifying a CSMA/CD-protocol with CCS. In *Proceedings of the IFIP Symposium on Protocol Specification, Testing and Verification*, pages 373–387, Atlantic City, New Jersey, June 1988. North-Holland.
17. D. Sangiorgi and R. Milner. The problem of 'weak bisimulation up to'. In *Proceedings of CONCUR '92*, volume 630 of *Lecture Notes in Computer Science*, pages 32–46, Stony Brook, New York, August 1992. Springer-Verlag.
18. S. Smolka and B. Steffen. Priority as extremal probability. In *CONCUR '90*, volume 458 of *Lecture Notes in Computer Science*, pages 456–466, Amsterdam, August 1990. Springer-Verlag.
19. R. van Glabbeek and P. Weijland. Branching time and abstraction in bisimulation semantics. In *Information Processing 89*, pages 613–618. North-Holland, 1989.
20. D. Walker. Analysing mutual exclusion algorithms using CCS. *Formal Aspects of Computing*, 1:273–292, 1989.