INFORMATION AND COMPUTATION **138**, 124–159 (1997)
ARTICLE NO. IC972657

A Complete Axiomatization of Finite-state

Patrice Brémond-Grégoire, Jin-Young Choi,[†] and Insup Lee

Department of Computer and Information Science, University of Pennsylvania, Philadelphia, Pennsylvania 19104-6389

A real-time process algebra, called ACSR, has been developed to facilitate the specification and analysis of real-time systems. ACSR supports synchronous timed actions and asynchronous instantaneous events. Timed actions are used to represent the usage of resources and to model the passage of time. Events are used to capture synchronization between processes. To be able to specify real-time systems accurately, ACSR supports a notion of priority that can be used to arbitrate among timed actions competing for the use of resources and among events that are ready for synchronization. In addition to operators common to process algebra, ACSR includes the scope operator, which can be used to model timeouts and interrupts. Equivalence between ACSR terms is based on the notion of strong bisimulation. This paper briefly describes the syntax and semantics of ACSR and then presents a set of algebraic laws that can be used to prove equivalence of ACSR processes. The contribution of this paper is the soundness and completeness proofs of this set of laws. The completeness proof is for finite-state ACSR processes, which are defined to be processes without free variables under parallel operator or scope operator. © 1997 Academic Press

Contents

* This research was supported in part by AFOSR F49620-95-1-0508, ARO DAAH04-95-1-0092, NSF CCR 93-11622, and NSF CCR 94-15346.

[†] Department of Computer Science and Engineering, Korea University, Seoul, Korea.

1. INTRODUCTION

Process algebras, such as CCS (Milner, 1989), CSP (Hoare, 1985), Acceptance Trees (Hennessy, 1988), and ACP (Bergstra and Klop, 1985), have been developed to describe and analyze communicating, concurrently executing systems. They are based on the premise that the two essential notions in understanding complex dynamic systems are concurrency and communication. Process algebras without the notion of time are being used widely in specifying and verifying concurrent systems. To expand their usefulness to real-time systems, several real-time process algebras have been developed by adding the notion of time and including a set of timing operators; *e.g.*, Baeten and Bergstra (1991, 1992), Davies and Schneider (1995), Hennessy and Regan (1990, 1995), Moller and Tofts (1990), Nicollin and Sifakis (1994), Reed and Roscoe (1987), and Yi (1991).

The most salient aspect of process algebras is that they support the *modular* specification and verification of a system. This is due to the algebraic laws that form a compositional proof system, and thus, it is possible to verify the whole system by reasoning about its parts. The reasoning on processes expressed by process algebras is usually based on the notion of bisimulation. One advantage of using bisimulation is that bisimulation can be characterized by a set of axioms (i.e., equational laws). Applying such a set of axioms, one can show that two processes are equal up to bisimulation. Milner (1984) provides the complete axiomatization of bisimulation for regular CCS processes. Since then, the complete axiomatizations of bisimulation for subsets of timed process algebras have been presented and have shown that bisimulations which deal with timed behaviors are as mathematically tractable as the standard untimed ones; *e.g.*, Klusener (1991), Moller and Tofts (1990), and Yi (1991).

Algebra of Communicating Shared Resource (ACSR), introduced by Lee *et al.* (1994), is a timed process algebra which can be regarded as an extension of CCS. ACSR supports the notions of resources, priorities, interrupt, timeout, and process structure. The notion of real time in ACSR is quantative and discrete and is accommodated using the concept of timed actions. The execution of a timed action takes one time unit and consumes a set of resources defined in the timed action during that one time unit period. The execution of a timed action is subject to the availability of resources it uses. The contention for resources is arbitrated according to the priorities of competing actions. To ensure the uniform progression of time, processes execute timed actions synchronously.

Similar to CCS, the execution of an event is instantaneous and never consumes any resource. The notion of communication is modeled using events through the execution of complementary events, which are then converted into an internal event. As with timed actions, priorities are also used to arbitrate the choice of several events that are possible at the same time. Although the concurrency model of CCS-like process algebras is based on interleaving semantics, ACSR includes interleaving semantics for events as well as lock-step parallelism for timed actions. This is because we assume that events occur instantaneously, whereas the execution of a timed action takes one time unit and consumes a set of resources during that time. Therefore, the ACSR axioms for bisimulation deal with both interleaving behavior of instantaneous events as well as lock-step parallelism of timed actions.

The main purpose of this paper is to show the complete axiomatization of a subset of ACSR processes. Since a process expressed by ACSR can be of infinite states, completeness does not hold in general. However, a complete axiomatization can be obtained by restricting the syntax to ensure finite state processes. The complete axiomatization has been studied by Milner (1984, 1989a) for a subset of CCS processes called "finite state agents," which are processes coded without the parallel operator. Since the restriction operator becomes unnecessary without the parallel operator, it has been omitted as well. Although the main idea of our proof is essentially similar to that of Milner's proof of completeness for an axiom system, it is possible to generate infinite state processes in ACSR even without the parallel operator, in contrast to CCS. For example, the use of the recursion operator combined with real-time operators such as timeout and interrupt can result in an infinite state process. Thus, we consider a subset of ACSR processes that do not recur through the parallel or real-time operators and prove that our axiomatization is complete for this subset. Since CCS is a subset of ACSR, our result can also be applied to CCS with the restriction and parallel operators, and thus extends the completeness result of CCS (Milner, 1989a).

Our soundness proof differs from Milner's proof (Milner, 1989), which uses the definition of bisimulation in the case analysis of possible behaviors. Our proof is based on the set of derivations of a process: whenever two processes have the same derivation set, the two processes are bisimilar. Our soundness proof seems to be clear and compact.

ACSR is an extension of another real-time process algebra, called CCSR (Gerber and Lee, 1994), which shares many aspects of ACSR. In particular, CCSR was the first process algebra to support the notions of both resources and priorities. CCSR, however, lacks instantaneous synchronization since all actions take exactly one time unit. Lee *et al.* (1994) extended CCSR by introducing the notion of instantaneous events and synchronization, and also developed a set of laws complete for finite processes. This paper extends the set of laws to finite state processes. We note that there are other process algebras that support the notion of priorities, including Baeten *et al.* (1987), Camilleri and Winskel (1991), and Cleaveland and Hennessy (1990). Their approaches are overviewed and compared by Gerber and Lee (1994).

The rest of the paper is organized as follows. In Section 2, we briefly introduce the computation model. In Section 3, we present the syntax of the algebra and describe the operational semantics. Section 4 defines the notion of equivalence and describes a set of equational laws that can be used to show the equivalence of two terms through syntactic manipulation. Section 5 contains a proof of soundness of the ACSR laws, followed in Section 6 by a proof of completeness for a syntactically characterized subset of the finite state processes. We conclude in Section 7 by discussing possible extensions of this work.

2. THE COMPUTATION MODEL

In our algebra there are two types of actions: those which consume time, and those which are instantaneous. The time-consuming actions represent the progress of one time unit of a global clock. These actions may also represent the consumption of resources, e.g., CPUs in the system configuration. On the other hand, the instantaneous actions (or *events*) provide a basic mechanism for synchronization and communication between concurrent processes.

Timed Actions. We consider a system to be composed of a finite set of serially reusable resources, denoted by $\mathscr{R}$. An action that consumes one "tick" of time is drawn from the domain $\mathbb{P}(\mathscr{R} \times \mathbb{N})$, with the restriction that each resource is represented at most once. As an example, the singleton action, $\{(r, p)\}$, denotes the use of some resource $r \in \mathscr{R}$ running at the priority level p. The action $\varnothing$ represents idling for one time unit, since no resuable resource is consumed.

We use $\mathscr{D}_R$ to denote the domain of timed actions, and we let A, B, C range over $\mathscr{D}_R$. We define $\rho(A)$ to be the set of resources used by the action A; e.g., $\rho(\{(r_1, p_1), (r_2, p_2)\}) = \{r_1, r_2\}$. We also use $\pi_r(A)$ to denote the priority level of the action A in the resource r; e.g., $\pi_{r_1}(\{(r_1, p_1), (r_2, p_2)\}) = p_1$. By convention, if r is not in $\rho(A)$, then $\pi_r(A) = 0$.

Instantaneous Events. We call instantaneous actions *events*, which provide the basic synchronization in our process algebra. An event is denoted by a pair (a, p), where a is the *label* of the event, and p is its *priority*. Labels are drawn from the set $\mathscr{L} \cup \bar{\mathscr{L}} \cup \{\tau\}$, where if a is a given label, we say that $\bar{a}$ is its *inverse* label; i.e., $\bar{\bar{a}} = a$. As in CCS, the special identity label, τ, arises when two events with inverse labels are executed in parallel.

We use $\mathscr{D}_E$ to denote the domain of events and let e, f, and g range over $\mathscr{D}_E$. We use $l(e)$ and $\pi(e)$ to represent the label and priority, respectively, of the event e.

Finally, the entire domain of actions is $\mathscr{D} = \mathscr{D}_R \cup \mathscr{D}_E$, and we let α and β range over $\mathscr{D}$.

The executions of a process are defined by a timed labelled transition system (timed LTS). A timed LTS M is defined as $\langle \mathscr{P}, \mathscr{D}, \rightarrow \rangle$, where (1) $\mathscr{P}$ is a set of ACSR processes, ranged over by P, Q; (2) $\mathscr{D}$ is a set of actions, and (3) $\rightarrow$ is a labeled transition relation such that $P \xrightarrow{\alpha} Q$ if the process P may perform an instantaneous event or timed action α and then it behaves as Q.

For example, a process P_1 may have the following behavior:

$$P_1 \xrightarrow{\alpha_1} P_2 \xrightarrow{\alpha_2} P_3 \xrightarrow{\alpha_3} \cdots.$$

That is, P_1 first executes α_1 and evolves into P_2, which executes α_2, etc. It takes no time to execute an instantaneous event. But in the case of a timed action, the action is executed for exactly one unit of time, during which P consumes the resources specified in the action, and then P evolves to Q.

3. THE SYNTAX AND OPERATIONAL SEMANTICS

The following grammar describes the syntax of processes:

$$P ::= \mathrm{NIL} \mid A{:}P \mid e.P \mid P + P \mid P \parallel P \mid$$

$$P \triangle_t^b (P, P, P) \mid [P]_I \mid P \backslash F \mid rec\ X.P \mid X$$

Table 1 shows the unprioritized operational semantics of ACSR defined as a structured transition system.

NIL is a process that executes no action (i.e., it is initially deadlocked). Since NIL executes no action, it has no transition rule.

TABLE 1

Unprioritized Transition Relation →

ActT	$A{:}P \xrightarrow{A} P$		ActI	$e.P \xrightarrow{e} P$	
ChoiceL	$\dfrac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}$		ChoiceR	$\dfrac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'}$	
ParT	$\dfrac{P \xrightarrow{A_1} P', Q \xrightarrow{A_2} Q'}{P \Vert Q \xrightarrow{A_1 \cup A_2} P' \Vert Q'}$	$(\rho(A_1) \cap \rho(A_2) = \emptyset)$	ParCom	$\dfrac{P \xrightarrow{(a,m)} P', Q \xrightarrow{(\bar{a},n)} Q'}{P \Vert Q \xrightarrow{(\tau,m+n)} P' \Vert Q'}$	
ParIL	$\dfrac{P \xrightarrow{e} P'}{P \Vert Q \xrightarrow{e} P' \Vert Q}$		ParIR	$\dfrac{Q \xrightarrow{e} Q'}{P \Vert Q \xrightarrow{e} P \Vert Q'}$	
ScopeCT	$\dfrac{P \xrightarrow{A} P'}{P \triangle_t^b (Q, R, S) \xrightarrow{A} P' \triangle_{t-1}^b (Q, R, S)}$	$(t > 0)$			
ScopeCI	$\dfrac{P \xrightarrow{e} P'}{P \triangle_t^b (Q, R, S) \xrightarrow{e} P' \triangle_t^b (Q, R, S)}$	$(l(e) \neq \bar{b}, t > 0)$			
ScopeE	$\dfrac{P \xrightarrow{(\bar{b},n)} P'}{P \triangle_t^b (Q, R, S) \xrightarrow{(\tau,n)} Q}$	$(t > 0)$			
ScopeT	$\dfrac{R \xrightarrow{\alpha} R'}{P \triangle_t^b (Q, R, S) \xrightarrow{\alpha} R'}$	$(t = 0)$	ScopeI	$\dfrac{S \xrightarrow{\alpha} S'}{P \triangle_t^b (Q, R, S) \xrightarrow{\alpha} S'}$	$(t > 0)$
CloseT	$\dfrac{P \xrightarrow{A_1} P'}{[P]_I \xrightarrow{A_1 \cup A_2} [P']_I}$	$(A_2 = \{(r, 0) \mid r \in I - \rho(A_1)\})$	CloseI	$\dfrac{P \xrightarrow{e} P'}{[P]_I \xrightarrow{e} [P']_I}$	
ResT	$\dfrac{P \xrightarrow{A} P'}{P \backslash F \xrightarrow{A} P' \backslash F}$		ResI	$\dfrac{P \xrightarrow{e} P'}{P \backslash F \xrightarrow{e} P' \backslash F}$	$(l(e) \notin F)$
Rec	$\dfrac{P[rec\ X.P/X] \xrightarrow{\alpha} P'}{rec\ X.P \xrightarrow{\alpha} P'}$				

There are two prefix operators, corresponding to the two types of actions. Rule **ActT** is for a time-consuming action, and rule **ActI** is for an instantaneous action. For example, the process $\{(r_1, p_1), (r_2, p_2)\}:P$ simultaneously uses resources r_1 and r_2 for one time unit, and then executes P. Alternatively, the process $(a, p).P$ executes the event "(a, p)" and proceeds to P. There are times when we do not want to distinguish between timed and untimed prefixes; in those cases we will use juxtaposition with a generic action. For example, αP stands for $\alpha{:}P$ if $\alpha \in \mathscr{D}_R$ and for $\alpha.P$ if $\alpha \in \mathscr{D}_E$.

The Choice operator $P + Q$ allows either of the processes to be chosen to execute, subject to the constraints of the environment. Rule **ChoiceL** and **ChoiceR** are for the choice operator and identical for both timed actions and instantaneous events (and hence we use "α" as the label).

The operator $P \| Q$ is the parallel composition of P and Q. The first rule, **ParT**, is for two time-consuming transitions. Note that timed transitions are truly synchronous, in that the resulting process advances only if both of the constituents take a step. The condition $\rho(A_1) \cap \rho(A_2) = \varnothing$ mandates that each resource be truly sequential, and that only one process may use a given resource during any time step. The next rules, **ParCom**, **ParIL**, and **ParIR**, are for event transitions. As opposed to timed actions, events may occur asynchronously (as in CCS and related interleaving models). The two rules **ParIL** and **ParIR** show that events may be arbitrarily interleaved. The rule **ParCom** is for two synchronizing processes; that is, P executes an event with the label a, while Q executes an event with the inverse label $\bar{a}$. Note that when the two events synchronize, their resulting priority is the sum of their constituent priorities. This definition of **ParCom** helps to ensure that the parallel composition is associative and also that the *relative* priority ordering among events with the same labels remains consistent even after communication takes places.

The Scope construct $P \,\triangle_t^b\, (Q, R, S)$ binds the process P by a temporal scope (Lee and Gehlot, 1985), and incorporates both the features of timeouts and interrupts. We call t the *time bound*, where $t \in \mathbb{N}^+ \cup \{\infty\}$ (i.e., t is either a non-negative integer or infinity). P executes for a maximum of t time units. The first two rules, **ScopeCT** and **ScopeCI**, show that as long as $t > 0$ and P fails to execute an event labelled with $\bar{b}$, the executions of P continue. The scope may be exited in a number of ways. First, by the rule **ScopeE**, if P successfully terminates within time t by executing an event labeled with $\bar{b}$, then control proceeds to the "success-handler" Q (here, b may be any label other than τ). Second, by the rule **ScopeT**, if P fails to terminate within time t, then control proceeds to the "timeout exception-handler" R. Last, by the **ScopeI**, at any time while P is executing it may be interrupted by S's execution of a timed action or instantaneous event, and the scope is then departed.

The Close operator, $[P]_I$, produces a process P that monopolizes the resources in $I \subseteq \mathscr{R}$. While Restriction assigns dedicated channels to processes, the Close operator assigns dedicated resources. Rules **CloseI** and **CloseT** describe the behaviors of the close operator. When a process P is embedded in a closed context such as $[P]_I$, we ensure that there is no further sharing of the resources in I. Assume that P executes a time-consuming action A. If A utilizes less than the full resource set I, the action is augmented with $(r, 0)$ pairs for each unused resource $r \in I - \rho(A)$. The way to interpret Close is as follows. A process may idle in two ways—it may release its resources during the idle time (represented by $\emptyset$), or it may hold them. Close ensures that the resources are held. (Instantaneous events are not affected.)

The Restriction operator, PF, limits the behavior of P: events with labels in F are permitted to execute only if they synchronize and become the internal event τ. The Restriction operator defines a subset of instantaneous events that are excluded from the behavior of the system as shown in **ResI** and **ResT**. This is done by establishing a set of labels, F $(\tau \notin F)$, and deriving only those behaviors that do not involve events with those labels. Time-consuming actions, on the other hand, remain unaffected.

The process *rec X.P* denotes standard recursion, allowing the specification of infinite behaviors. The term X, without a "*rec*" binding, is a free variable that belongs to the infinite set FV. Rule **Rec** shows that the operator *rec X.P* denotes recursion, allowing the specification of infinite behaviors. In rule **Rec**, $P[{}^{rec\,X.P}/_{X}]$ is the standard notation for substitution of *rec X.P* for each free occurrence of X in P.

The semantics is defined in two steps. First, we develop the *unconstrained* transition system, where a transition is denoted as $P \xrightarrow{\alpha} P'$. Within "$\rightarrow$" no priority arbitration is made between actions; rather, we subsequently refine "$\rightarrow$" to define our prioritized transition system, "$\rightarrow_\pi$." The bisimulation introduced later is based on the prioritized transition system, and called prioritized strong bisimulation.

3.1. Preemption and Prioritized Transitions

The prioritized transition system is based on the notion of *preemption*, which incorporates our treatment of synchronization, resource-sharing, and priority. The definition of preemption is based on priorites assigned to events and resources. We define a binary relation $\prec$ such that for two actions α and β, $\alpha \prec \beta$ iff α is preempted by β. That means if a process P can execute either α or β at a time, P will never execute α, because the priority of β is higher than that of α.

DEFINITION 3.1 (Preemption Relation). For two actions, α, β, we say that β preempts α $(\alpha \prec \beta)$, if one of the following cases hold:

(1) Both α and β are timed actions in $\mathscr{D}_R$, where

  (i) $\rho(\beta) \subseteq \rho(\alpha)$,

  (ii) $\forall r. \pi_r(\alpha) \leqslant \pi_r(\beta)$, and

  (iii) $\exists r. \pi_r(\alpha) < \pi_r(\beta)$

(2) Both α and β are events in $\mathscr{D}_E$, where $\pi(\alpha) < \pi(\beta) \wedge l(\alpha) = l(\beta)$

(3) $\alpha \in \mathscr{D}_R$ and $\beta \in \mathscr{D}_E$, with $l(\beta) = \tau$ and $\pi(\beta) > 0$.

Case (1) shows that the two timed actions, α and β, compete for common resources, and in fact, the preempted action α may use a superset of β's resources. However, β uses all the resources at least the same priority level as α (recall that $\pi_r(B)$ is, by convention, 0 when r is not in B). Thus, for any resource r in $\rho(\alpha) - \rho(\beta)$, the priority of r in α must be zero in order that β may preempt α. Also, β uses at least one resource at a higher level. For instance, $\{(r_1, 2), (r_2, 0)\} \prec \{(r_1, 7)\}$ but $\{(r_1, 2), (r_2, 1)\} \not\prec \{(r_1, 7)\}$.

Case (2) shows that an event may be preempted by another event sharing the same label, but with a higher priority. For example, $(\tau, 1) \prec (\tau, 2)$, $(a, 2) \prec (a, 5)$, and $(a, 1) \not\prec (b, 2)$ if $a \neq b$.

Finally, case (3) shows the single case in which an event and a timed action are comparable under "$\prec$." That is, if $n > 0$ in an event (τ, n), we let the event preempt any timed action. For instance, $\{(r_1, 2), (r_2, 5)\} \prec (\tau, 2)$, but $\{(r_1, 2), (r_2, 5)\} \not\prec (\tau, 0)$.

We define the prioritized transition system "$\rightarrow_\pi$," which simply refines "$\rightarrow$" to account for preemption.

Definition 3.2. The labelled transition system "$\rightarrow_\pi$" is defined as follows: $P \xrightarrow{\alpha} P'$ if and only if

(*a*) $P \xrightarrow{\alpha} P'$ is an unprioritized transition, and

(*b*) There is no unprioritized transition $P \xrightarrow{\beta} P''$ such that $\alpha \prec \beta$.

3.2. *Preemptive Static Priority Scheduler*

To illustrate the expressiveness of ACSR, we describe a set of periodic tasks which are to execute on a single processor. The deadline of a task defines the latest time by which the execution of the task must finish and is assumed to equal to its period. To resolve contention over the processor, we use a rate-monotonic scheduler, which assigns to each task a static priority that is inversely proportional to its period; that is, a task with a shorter period is assigned a higher priority than a task with a longer period. To make the example more readable, we augment the ACSR syntax with a definition operator as well as indexed processes and events. We use $X \overset{\text{def}}{=} P$ to refer to the process expression P by the process name X. We use subscripts to define indexed processes and events, e.g., P_1 and (a_2, p).

Figure 1 shows the ACSR specification of such a system. The system consists of a dispatcher, *Dispatch*, and three tasks, T_1, T_2, and T_3. The dispatcher instantiates the tasks at the beginning of their periods. Each task, T_i, is characterized by two constants: the computation time, c_i, and the period, p_i, where $c_i \leqslant p_i$. We assume task T_i has a longer period than task T_{i+1}, i.e., $p_1 \geqslant p_2 \geqslant p_3$. Consequently, task T_i is assigned the priority i, for $i = 1, 2, 3$. Furthermore, to simplify the presentation, we use the following assumptions about the system: tasks become ready at the beginning of their periods, and they can be preempted instantly with no preemption overhead.

The dispatcher periodically instantiates a task T_i using a process D_i as follows: D_i first signals the task T_i to start by sending the event $\overline{s_i}$. It then idles for the

$$System \overset{\text{def}}{=} [\,(Dispatch \,||\, T_1 \,||\, T_2 \,||\, T_3)\backslash\{s_1, s_2, s_3\}\,]_{\{cpu\}}$$

$$\begin{array}{rcll}
Dispatch & \overset{\text{def}}{=} & D_1 \,||\, D_2 \,||\, D_3 & \\
D_1 & \overset{\text{def}}{=} & (\overline{s_1}, 1).D_{1,0} & \\
D_2 & \overset{\text{def}}{=} & (\overline{s_2}, 1).D_{2,0} & \\
D_3 & \overset{\text{def}}{=} & (\overline{s_3}, 1).D_{3,0} & \\
D_{1,j} & \overset{\text{def}}{=} & \emptyset : D_{1,j+1} & 0 \le j < p_1 \\
D_{1,p_1} & \overset{\text{def}}{=} & D_1 & \\
D_{2,j} & \overset{\text{def}}{=} & \emptyset : D_{2,j+1} & 0 \le j < p_2 \\
D_{2,p_2} & \overset{\text{def}}{=} & D_2 & \\
D_{3,j} & \overset{\text{def}}{=} & \emptyset : D_{3,j+1} & 0 \le j < p_3 \\
D_{3,p_1} & \overset{\text{def}}{=} & D_3 &
\end{array}$$

$$\begin{array}{rcll}
T_1 & \overset{\text{def}}{=} & (s_1, 1).C_{1,0} & \\
T_2 & \overset{\text{def}}{=} & (s_2, 2).C_{2,0} & \\
T_3 & \overset{\text{def}}{=} & (s_3, 3).C_{3,0} & \\
C_{1,j} & \overset{\text{def}}{=} & \emptyset : C_{1,j} + \{(cpu, 1)\} : C_{1,j+1} & 0 \le j < c_1 \\
C_{1,c_1} & \overset{\text{def}}{=} & \emptyset : C_{1,c_1} + T_1 & \\
C_{2,j} & \overset{\text{def}}{=} & \emptyset : C_{2,j} + \{(cpu, 2)\} : C_{2,j+1} & 0 \le j < c_2 \\
C_{2,c_2} & \overset{\text{def}}{=} & \emptyset : C_{2,c_2} + T_2 & \\
C_{3,j} & \overset{\text{def}}{=} & \emptyset : C_{3,j} + \{(cpu, 3)\} : C_{3,j+1} & 0 \le j < c_3 \\
C_{3,c_3} & \overset{\text{def}}{=} & \emptyset : C_{3,c_3} + T_3 &
\end{array}$$

FIG. 1. Prioritized multitask system.

duration of the task's period p_i before sending $\overline{s_i}$ again. Once signaled to start, each task T_i tries to compute for c_i time units using the *cpu* resource. This is described by the processes $C_{i,j}$ which keep track of how much computation time is used. When a task T_i does not have access to the *cpu* resource, it idles by executing the process $\varnothing : C_{i,j}$. This means that T_i is preempted. When it finishes its computation, T_i idles waiting for the next periodic instantion; this is described by process $C_{i,\,c_i}$. On the other hand, if T_i does not finish its computation within its period p_i, it will not be able to synchronize with the dispatcher which tries to send the starting event $\overline{s_i}$. Such a missed synchronization leads the process D_i of the dispatcher to deadlock. This in turn leads the whole system to deadlock.

We note that because ACSR associates priorities with actions, it is straightforward to change the rate-monotonic scheduler to another scheduler by modifying action priorities.

Let S be a system after each task T_i has synchronized on the event s_i:

$$S = D_{1,\,0} \,||\, D_{2,\,0} \,||\, D_{3,\,0} \,||\, C_{1,\,0} \,||\, C_{2,\,0} \,||\, C_{3,\,0}.$$

By applying the rules of the operational semantics, we see that there are three unprioritized transitions that the system S can take:

$$(1)\quad S \xrightarrow{\{(cpu,1)\}} D_{1,1} \| D_{2,1} \| D_{3,1} \| C_{1,1} \| C_{2,0} \| C_{3,0}$$

$$(2)\quad S \xrightarrow{\{(cpu,2)\}} D_{1,1} \| D_{2,1} \| D_{3,1} \| C_{1,0} \| C_{2,1} \| C_{3,0}$$

$$(3)\quad S \xrightarrow{\{(cpu,3)\}} D_{1,1} \| D_{2,1} \| D_{3,1} \| C_{1,0} \| C_{2,0} \| C_{3,1}.$$

Only transition (3) remains admitted by the prioritized transition system. This allows T_3 to execute while T_1 and T_2 idle.

4. PRIORITIZED STRONG EQUIVALENCE AND LAWS

Equivalence between two ACSR processes is based on the concept of *strong bisimulation* (Park, 1981), which compares the computation trees of the two processes.

DEFINITION 4.1. For a given transition system "$\rightsquigarrow$", any binary relation r is a strong bisimulation if, for $(P, Q) \in r$ and $\alpha \in \mathscr{D}$,

1. if $P \overset{\alpha}{\rightsquigarrow} P'$ then, for some Q', $Q \overset{\alpha}{\rightsquigarrow} Q'$ and $(P', Q') \in r$, and
2. if $Q \overset{\alpha}{\rightsquigarrow} Q'$ then, for some P', $P \overset{\alpha}{\rightsquigarrow} P'$ and $(P', Q') \in r$.

In other words, if P (or Q) can take a step on α, then Q (or P) must also be able to take a step on α with both of the next states also bisimilar. There are some very obvious bisimulation relations; e.g., $\varnothing$ (which certainly adheres to the above rules) or syntactic identity. However, using the theory found in Milner (1980, 1983, 1989) it is straightforward to show that there exists a largest such bisimulation over "$\rightarrow$," which we denote as "$\sim$." This relation is an equivalence relation, and is a congruence with respect to the operators (Gerber, 1991). Similarly, "$\sim_\pi$" is the largest strong bisimulation over "$\rightarrow_\pi$," and we call it *prioritized strong equivalence.*

LEMMA 4.1. *$\sim_\pi$ is a congruence with the ACSR operators.*

Proof. The proof is very similar to that found in (Garber, 1991). ∎

Table 2 presents a set of equivalence-preserving laws for ACSR, $\mathscr{A}$. In the sequel, wherever we use the equality symbol "$=$" in showing that two processes are equivalent, it means that we have used the laws $\mathscr{A}$ along with the standard laws for substitution to construct the proof. The bisimilarity of the processes follows from the soundness of the laws.

Note the use of the summation symbol $\sum$ in Par(3). The interpretation is as follows: Let I be an index set representing processes, such that for each $i \in I$, there is some corresponding process P_i. If $I = \{i_1, ..., i_n\}$, because of Choice (3) and Choice (4) we are able to neglect parentheses and use the notation

$$\sum_{i \in I} P_i \overset{\text{def}}{=} P_{i_1} + \cdots + P_{i_n}$$

where $\sum_{i \in \varnothing} P_i \overset{\text{def}}{=} NIL.$

TABLE 2

The Set of ACSR Laws, $\mathscr{A}$

Choice(1)	$P + \text{NIL} = P$
Choice(2)	$P + P = P$
Choice(3)	$P + Q = Q + P$
Choice(4)	$(P + Q) + R = P + (Q + R)$
Choice(5)	$\alpha P + \beta Q = \beta Q \quad \text{if } \alpha \prec \beta$
Par(1)	$P \parallel Q = Q \parallel P$
Par(2)	$(P \parallel Q) \parallel R = P \parallel (Q \parallel R)$
Par(3)	$(\sum_{i\in I} A_i{:}P_i + \sum_{j\in J} e_j.Q_j) \parallel (\sum_{k\in K} B_k{:}R_k + \sum_{l\in L} f_l.S_l) = \left[\begin{array}{l} \sum_{i\in I,k\in K,\ \rho(A_i)\cap\rho(B_k)=\emptyset} (A_i\cup B_k){:}(P_i \parallel R_k) \\ + \sum_{j\in J} e_j.(Q_j \parallel (\sum_{k\in K} B_k{:}R_k + \sum_{l\in L} f_l.S_l)) \\ + \sum_{l\in L} f_l.((\sum_{i\in I} A_i{:}P_i + \sum_{j\in J} e_j.Q_j) \parallel S_l) \\ + \sum_{j\in J,l\in L,\ l(e_j)=l(f_l)} (\tau,\pi(e_j)+\pi(f_l)).(Q_j \parallel S_l) \end{array}\right.$
Scope(1)	$A{:}P \,\Delta^{b}_{t}\, (Q,R,S) = A{:}(P \,\Delta^{b}_{t-1}\, (Q,R,S)) + S \quad \text{if } t > 0$
Scope(2)	$e.P \,\Delta^{b}_{t}\, (Q,R,S) = e.(P \,\Delta^{b}_{t}\, (Q,R,S)) + S \quad \text{if } t > 0 \wedge \overline{l(e)} \neq b$
Scope(3)	$e.P \,\Delta^{b}_{t}\, (Q,R,S) = (\tau,\pi(e)).Q + S \quad \text{if } t > 0 \wedge \overline{l(e)} = b$
Scope(4)	$P \,\Delta^{b}_{0}\, (Q,R,S) = R$
Scope(5)	$(P_1 + P_2) \,\Delta^{b}_{t}\, (Q,R,S) = P_1 \,\Delta^{b}_{t}\, (Q,R,S) + P_2 \,\Delta^{b}_{t}\, (Q,R,S)$
Scope(6)	$\text{NIL} \,\Delta^{b}_{t}\, (Q,R,S) = S \quad \text{if } t > 0$
Res(1)	$\text{NIL}\backslash F = \text{NIL}$
Res(2)	$(P + Q)\backslash F = (P\backslash F) + (Q\backslash F)$
Res(3)	$(A{:}P)\backslash F = A{:}(P\backslash F)$
Res(4)	$((a,n).P)\backslash F = (a,n).(P\backslash F) \quad \text{if } a, \bar{a} \notin F$
Res(5)	$((a,n).P)\backslash F = \text{NIL} \quad \text{if } a, \bar{a} \in F$
Res(6)	$P\backslash E\backslash F = P\backslash E \cup F$
Res(7)	$P\backslash\emptyset = P$
Close(1)	$[\text{NIL}]_I = \text{NIL}$
Close(2)	$[P + Q]_I = [P]_I + [Q]_I$
Close(3)	$[A_1{:}P]_I = (A_1 \cup A_2){:}[P]_I \quad \text{where } A_2 = \{(r,0) \mid r \in I - \rho(A_1)\}$
Close(4)	$[e.P]_I = e.[P]_I$
Close(5)	$[[P]_I]_J = [P]_{I\cup J}$
Close(6)	$[P]_\emptyset = P$
Close(7)	$[P\backslash E]_I = [P]_I\backslash E$
Rec(1)	$rec\ X.P = P[rec\ X.P/X]$
Rec(2)	If $P = Q[P/X]$ and X is guarded in Q then $P = rec\ X.Q$
Rec(3)	$rec\ X.(P + \sum_{i\in I}[X\backslash E_i]_{U_i}) = rec\ X.(\sum_{J\subseteq I}[P\backslash E_J]_{U_J})$ where $E_J = \bigcup_{i\in J} E_i$, $U_J = \bigcup_{i\in J} U_i$, I is finite and X is guarded in P

5. SOUNDNESS OF THE LAWS

In order to prove soundness of the ACSR laws, we make use of two functions:

$$\mathscr{T}(P)=\{\langle \alpha, P'\rangle \mid P \xrightarrow{\alpha} P'\} \quad \text{and} \quad \mathscr{T}_\pi(P)=\{\langle \alpha, P'\rangle \mid P \xrightarrow{\alpha}_\pi P'\}.$$

Traditionally, researchers in process algebra have proved the soundness of algebraic axioms by using the bisimulation definition as described by Milner (1989): one must first pick up a possible derivation of a process and show that such a derivation can be found in the other process. Unlike the Milner's method based on the bisimulation definition, our soundness proof in this section is based on two functions, $\mathscr{T}$ and $\mathscr{T}_\pi$, such that given an ACSR process P, $\mathscr{T}(P)$ ($\mathscr{T}_\pi(P)$) is a set of pairs $\langle a, P'\rangle$, where $P \xrightarrow{a} P'$ ($P \xrightarrow{a}_\pi P'$). Furthermore, these are complete sets of derivations of a process. The main use of these functions is as follows: given two ACSR finite processes P and Q, if $\mathscr{T}(P)$ is the same as $\mathscr{T}(Q)$, then $\mathscr{T}_\pi(P)$ is the same as $\mathscr{T}_\pi(Q)$, and hence, P and Q are prioritized bisimilar. Hence, the soundness of an axiom $P=Q$ can be proved by showing that $\mathscr{T}(P)=\mathscr{T}(Q)$ or, if $\mathscr{T}(P)\neq\mathscr{T}(Q)$ then showing that $\mathscr{T}_\pi(P)=\mathscr{T}_\pi(Q)$. This approach simplifies the soundness proof, and seems to increase the degree of understandability for the intuitive meaning of the soundness proof. Now we need the following lemmas.

LEMMA 5.1. *For any two ACSR terms, P and Q, the followings hold*: (1) $\mathscr{T}(P)=\mathscr{T}(Q)$ *implies* $\mathscr{T}_\pi(P)=\mathscr{T}_\pi(Q)$, *and* (2) $\mathscr{T}_\pi(P)=\mathscr{T}_\pi(Q)$ *implies* $P \sim_\pi Q$

Proof. It follows from the definition of the prioritized transition system that $\mathscr{T}_\pi(P)$ can be calculated from $\mathscr{T}(P)$:

$$\mathscr{T}_\pi(P)=\{\langle \alpha, P'\rangle \in \mathscr{T}(P) \mid \nexists \langle \beta, Q\rangle \in \mathscr{T}(P).\alpha \prec \beta\}.$$

Thus, $\mathscr{T}(P)=\mathscr{T}(Q)$ implies $\mathscr{T}_\pi(P)=\mathscr{T}_\pi(Q)$.

From the definition of $\mathscr{T}_\pi$ we have:

$$\text{if } \mathscr{T}_\pi(P)=\mathscr{T}_\pi(Q) \text{ then } \begin{cases} \forall\alpha\colon P \xrightarrow{\alpha} P' \Rightarrow Q \xrightarrow{\alpha} P' \text{ and} \\ \forall\alpha\colon Q \xrightarrow{\alpha} Q' \Rightarrow P \xrightarrow{\alpha} Q'. \end{cases}$$

Since the identity is a bisimulation, we conclude that $P \sim_\pi Q$. ▮

The lemma above plays an important role in proving the soundness of the laws. It may be shown that all the laws, except Choice (5), that are sound for unprioritized strong equivalence are also sound for prioritized equivalence. Hence all soundness proofs, except for Choice (5), use $\mathscr{T}$.

LEMMA 5.2. *If $\mathscr{R}$ is a relation such that all the pairs $(P, Q)\in\mathscr{R}$ are such that*

$$\forall\langle \alpha, P'\rangle \in \mathscr{T}_\pi(P)\colon \exists Q'\colon \langle \alpha, Q'\rangle \in \mathscr{T}_\pi(Q) \wedge (P', Q')\in\mathscr{R}$$

and

$$\forall\langle \alpha, Q'\rangle \in T_\pi(Q)\colon \exists P'\colon \langle \alpha, P'\rangle \in \mathscr{T}_\pi(P) \wedge (P', Q')\in\mathscr{R}$$

then the relation $\mathscr{R}$ is a strong bisimulation.

Proof. Follows directly from the definitions of the strong bisimulation and of the functions $\mathcal{T}$ and $\mathcal{T}_\pi$. ▌

Milner (1989) proved the soundness of CCS axioms using the bisimulation definition: one must first pick up a possible derivation of a process and show that such a deriviation can be found in the other process. That is, Milner's proof is based on case analysis of every individual derivation. Contrary to Milner's method based on the bisimulation definition, our proof is based on the complete set of derivations of a process: when two sets are the same, the processes are bisimilar. Having the simple fact that $\mathcal{T}(P) = \mathcal{T}(Q)$ implies P and Q are bisimilar, our soundness proof seems clear and short.

In order to prove most of the laws, we apply the appropriate formula to calculate either $\mathcal{T}$ or $\mathcal{T}_\pi$ for the processes in both sides and verify that the resulting sets are equal or related in a way that satisfies Lemma 5.2. For instance, since $\mathcal{T}(P+Q) = \{\langle \alpha, P' \rangle \mid P + Q \xrightarrow{\alpha} P'\}$ is the same as the set of $\{\{\langle \alpha, P' \rangle \mid P \xrightarrow{\alpha} P'\} \cup \{\{\langle \alpha, P' \rangle \mid Q \xrightarrow{\alpha} P'\}$, $\mathcal{T}(P+Q)$ is the set $\mathcal{T}(P) \cup \mathcal{T}(Q)$. Since $\cup$ is a commutative set operator, we have $\mathcal{T}(P+Q) = \mathcal{T}(P) \cup \mathcal{T}(Q) = \mathcal{T}(Q) \cup \mathcal{T}(P)$. Hence, by definition of $\mathcal{T}$, we conclude $\mathcal{T}(P+Q) = \mathcal{T}(Q+P)$.

Since the behavior of a process must be derived from the rules of the operational semantics, for any process P the set $\mathcal{T}(P)$ is the union of all the sets that can be derived from each rule of the operational semantics that applies. From this we can derive the following sets of equations (the operational rule applied to calculate each term is shown in brackets):

$$\mathcal{T}(\mathrm{NIL}) = \varnothing$$

$$\mathcal{T}(\alpha P) = \{\langle \alpha, P \rangle\} \qquad \text{[ActT and ActI]}$$

$$\mathcal{T}(P+Q) = \mathcal{T}(P) \cup \mathcal{T}(Q) \qquad \text{[ChoiceL and ChoiceR]}$$

$$\begin{aligned}
\mathcal{T}(P \| Q) = {} & \{\langle A \cup B, P' \| Q' \rangle \mid \langle A, P' \rangle \in \mathcal{T}(P) \wedge \langle B, Q' \rangle \in \mathcal{T}(Q) \\
& \quad \wedge \rho(A) \cap \rho(B) = \varnothing\} && \text{[ParT]} \\
& \cup \{\langle e, P' \| Q \rangle \mid \langle e, P' \rangle \in \mathcal{T}(P)\} && \text{[ParIL]} \\
& \cup \{\langle e, P \| Q' \rangle \mid \langle e, Q' \rangle \in \mathcal{T}(Q)\} && \text{[ParIR]} \\
& \cup \{\langle (\tau, n+m), P' \| Q' \rangle \mid \langle (a, n), P' \rangle \in \mathcal{T}(P) \\
& \quad \wedge \langle (\bar{a}, m), Q' \rangle \in \mathcal{T}(Q)\} && \text{[ParCom]}
\end{aligned}$$

$$\mathcal{T}(P \mathbin{\triangle}_0^b (Q, R, S)) = \mathcal{T}(R) \qquad \text{[ScopeT]}$$

$$\begin{aligned}
\mathcal{T}(P \mathbin{\triangle}_{t>0}^b (Q, R, S)) = {} & \{\langle A, P' \mathbin{\triangle}_{t-1}^b (Q, R, S) \rangle \mid \langle A, P' \rangle \in \mathcal{T}(P)\} && \text{[ScopeCT]} \\
& \cup \{\langle e, P' \mathbin{\triangle}_b^t (Q, R, S) \rangle \mid \langle e, P' \rangle \in \mathcal{T}(P) \wedge l(e) \neq \bar{b}\} && \text{[ScopeCI]} \\
& \cup \{\langle (\tau, n), Q \rangle \mid \langle (\bar{b}, n), P' \rangle \in \mathcal{T}(P)\} && \text{[ScopeE]} \\
& \cup \mathcal{T}(S) && \text{[ScopeI]}
\end{aligned}$$

$$\begin{aligned}
\mathscr{T}(P\backslash F) &= \{\langle A, P'\backslash F\rangle \mid \langle A, P'\rangle \in \mathscr{T}(P)\} && [\text{ResT}]\\
&\quad \cup \{\langle (a,n), P'\backslash F\rangle \mid \langle (a,n)\, P'\rangle \in \mathscr{T}(P) \wedge a, \bar{a} \notin F\} && [\text{ResI}]\\
\mathscr{T}([P]_I) &= \{\langle A \cup B, [P']_I\rangle \\
&\qquad \mid \langle A, P'\rangle \in \mathscr{T}(P) \wedge B = \{(r,0) \mid r \in I - \rho(A)\}\} && [\text{CloseT}]\\
&\quad \cup \{\langle e, [P']_I\rangle \mid \langle e, P'\rangle \in \mathscr{T}(P)\} && [\text{CloseI}]\\
\mathscr{T}(rec\ X.P) &= \{\langle \alpha, P'\rangle \mid \langle \alpha, P'\rangle \in \mathscr{T}(P[rec\ X.P/X])\} && [\text{Rec}]\\
&= \mathscr{T}(P[rec\ X.P/X]).
\end{aligned}$$

To save space, we show only those that are representative or non-trivial.

Choice (1). $\mathscr{T}(P + \text{NIL}) = \mathscr{T}(P) \cup \mathscr{T}(\text{NIL}) = \mathscr{T}(P) \cup \varnothing = \mathscr{T}(P)$.

Choice (5). $\mathscr{T}(\alpha P + \beta Q) = \{\langle \alpha, P\rangle, \langle \beta, Q\rangle\}$ and therefore we have $\mathscr{T}_\pi(\alpha P + \beta Q) = \{\langle \beta, Q\rangle\} = \mathscr{T}_\pi(\beta Q)$.

Par (3). Let us call

$$P = \sum_{i \in I} A_i : P_i$$

$$Q = \sum_{i \in J} e_j . Q_j$$

$$R = \sum_{k \in K} B_k : R_k$$

$$S = \sum_{l \in L} f_l . S_l .$$

We can calculate

$$\begin{aligned}
\mathscr{T}(P + Q \parallel R + S) &= \{\langle A_i \cup B_k, P_i \parallel Q_k\rangle \mid i \in I \wedge k \in K \wedge \rho(A_i) \cap \rho(B_k) = \varnothing\}\\
&\quad \cup \{\langle e_j, Q_j \parallel (R+S)\rangle \mid j \in J\}\\
&\quad \cup \{\langle f_l, (P+Q) \parallel S_l\rangle \mid l \in L\}\\
&\quad \cup \{\langle (\tau, \pi(e_j) + \pi(f_l)), Q_j \parallel S_l\rangle \mid j \in J \wedge l \in L \wedge l(e_j) = \overline{l(f_l)}\}\\
&= \mathscr{T}\left(\begin{array}{l}
\displaystyle\sum_{\substack{i \in I,\, k \in K,\\ \rho(A_i) \cap \rho(B_k) = \varnothing}} (A_i \cup B_k) : (P_i \parallel R_k)\\
\displaystyle + \sum_{j \in J} e_j . (Q_j \parallel (R+S))\\
\displaystyle + \sum_{l \in L} f_l . ((P+Q) \parallel S_l)\\
\displaystyle + \sum_{\substack{j \in J,\, l \in L,\\ l(e_j) = \overline{l(f_l)}}} (\tau, \pi(e_j) + \pi(f_l)) . (Q_j \parallel S_l)
\end{array}\right)
\end{aligned}$$

Scope (1). This follows from the fact that $A{:}P$ has a single transition that:

$$\begin{aligned}\mathscr{T}(A{:}P \vartriangle^{b}_{t>0} (Q, R, S)) &= \{\langle A, P \vartriangle^{b}_{t-1} (Q, R, S)\} \cup \mathscr{T}(S)\\ &= \mathscr{T}(A{:}(P \vartriangle^{b}_{t-1} (Q, R, S)) + S).\end{aligned}$$

Scope (5). We distinguish two different cases:

(i) When $t = 0$

$$\begin{aligned}\mathscr{T}(P_1 + P_2 \vartriangle^{b}_{0} (Q, R, S)) &= \mathscr{T}(R) \cup \mathscr{T}(R)\\ &= \mathscr{T}(P_1 \vartriangle^{b}_{0} (Q, R, S)) \cup \mathscr{T}(P_2 \vartriangle^{b}_{0} (Q, R, S))\\ &= \mathscr{T}(P_1 \vartriangle^{b}_{0} (Q, R, S)) + \mathscr{T}(P_2 \vartriangle^{b}_{0} (Q, R, S)).\end{aligned}$$

(ii) When $t > 0$

$$\begin{aligned}&\mathscr{T}(P_1 + P_2 \vartriangle^{b}_{t} (Q, R, S))\\
&\quad= \{\langle A, P' \vartriangle^{b}_{t-1} (Q, R, S)\rangle \mid \langle A, P'\rangle \in \mathscr{T}(P_1 + P_2)\}\\
&\qquad \cup \{\langle e, P' \vartriangle^{b}_{t} (Q, R, S)\rangle \mid \langle e, P'\rangle \in \mathscr{T}(P_1 + P_2) \wedge l(e) \neq \bar{b}\}\\
&\qquad \cup \{\langle (\tau, n), Q\rangle \mid \langle (\bar{b}, n), P'\rangle \in \mathscr{T}(P_1 + P_2)\}\\
&\qquad \cup \mathscr{T}(S)\\
&\quad= \{\langle A, P' \vartriangle^{b}_{t-1} (Q, R, S)\rangle \mid \langle A, P'\rangle \in \mathscr{T}(P_1)\}\\
&\qquad \cup \{\langle A, P' \vartriangle^{b}_{t-1} (Q, R, S) \mid \langle A, P'\rangle \in \mathscr{T}(P_2)\}\\
&\qquad \cup \{\langle e, P' \vartriangle^{b}_{t} (Q, R, S)\rangle \mid \langle e, P'\rangle \in \mathscr{T}(P_1) \wedge l(e) \neq \bar{b}\}\\
&\qquad \cup \{\langle e, P' \vartriangle^{b}_{t} (Q, R, S)\rangle \mid \langle e, P'\rangle \in \mathscr{T}(P_2) \wedge l(e) \neq \bar{b}\}\\
&\qquad \cup \{\langle (\tau, n), Q\rangle \mid \langle (\bar{b}, n), P'\rangle \in \mathscr{T}(P_1)\}\\
&\qquad \cup \{\langle (\tau, n),\ Q\rangle \mid \langle (\bar{b}, n), P'\rangle \in \mathscr{T}(P_2)\}\\
&\qquad \cup \mathscr{T}(S)\\
&\quad= \{\langle A, P' \vartriangle^{b}_{t-1} (Q, R, S)\rangle \mid \langle A, P'\rangle \in \mathscr{T}(P_1)\}\\
&\qquad \cup \{\langle e, P'\rangle \vartriangle^{b}_{t} (Q, R, S)\rangle \mid \langle e, P'\rangle \in \mathscr{T}(P_1) \wedge l(e) \neq \bar{b}\}\\
&\qquad \cup \{\langle (\tau, n), Q\rangle \mid \langle (\bar{b}, n), P'\rangle \in \mathscr{T}(P_1)\}\\
&\qquad \cup \mathscr{T}(S)\\
&\qquad \cup \{\langle A, P' \vartriangle^{b}_{t-1} (Q, R, S)\rangle \mid \langle A, P'\rangle \in \mathscr{T}(P_2)\}\\
&\qquad \cup \{\langle e, P' \vartriangle^{b}_{t} (Q, R, S)\rangle \mid \langle e, P'\rangle \in \mathscr{T}(P_2) \wedge l(e) \neq \bar{b}\}\\
&\qquad \cup \{\langle (\tau, n), Q\rangle \mid \langle (\bar{b}, n), P'\rangle \in \mathscr{T}(P_2)\}\\
&\qquad \cup \mathscr{T}(S)\\
&\quad= \mathscr{T}(P_1 \vartriangle^{b}_{t} (Q, R, S)) \cup \mathscr{T}(P_2 \vartriangle^{b}_{t} (Q, R, S))\\
&\quad= \mathscr{T}(P_1 \vartriangle^{b}_{t} (Q, R, S) + P_2 \vartriangle^{b}_{t} (Q, R, S)).\end{aligned}$$

Res (2).

$$\begin{aligned}\mathscr{T}((P+Q)\backslash F) &= \{\langle A, P'\backslash F\rangle \mid \langle A, P'\rangle \in \mathscr{T}(P+Q)\}\\ &\quad \cup \{\langle (a,n), P'\backslash F\rangle \mid \langle (a,n), P'\rangle \in \mathscr{T}(P+Q) \wedge a, \bar{a} \notin F\}\\ &= \{\langle A, P'\backslash F\rangle \mid \langle A, P'\rangle \in \mathscr{T}(P)\}\\ &\quad \cup \{\langle A, P'\backslash F\rangle \mid \langle A, P'\rangle \in \mathscr{T}(Q)\}\\ &\quad \cup \{\langle (a,n), P'\backslash F\rangle \mid \{(a,n), P'\rangle \in \mathscr{T}(P) \wedge a, \bar{a} \notin F\}\\ &\quad \cup \{\langle (a,n), P'\backslash F\rangle \mid \langle (a,n), P'\rangle \in \mathscr{T}(Q) \wedge a, \bar{a} \notin F\}\\ &= \mathscr{T}(P\backslash F) \cup \mathscr{T}(Q\backslash F) = \mathscr{T}(P\backslash F + Q\backslash F).\end{aligned}$$

Res (6).

$$\begin{aligned}\mathscr{T}(P\backslash E\backslash F) &= \{\langle A, P'\backslash F\rangle \mid \langle A, P'\rangle \in \mathscr{T}(P\backslash E)\}\\ &\quad \cup \{\langle (a,n), P'\backslash F\rangle \mid \langle (a,n), P'\rangle \in \mathscr{T}(P\backslash E) \wedge a, \bar{a} \notin F\}\\ &= \{\langle A, P''\backslash E\backslash F\rangle \mid \langle A, P''\rangle \in \mathscr{T}(P)\}\\ &\quad \cup \{\langle (a,n), P''\backslash E'F\rangle \mid \langle (a,n), P''\rangle \in \mathscr{T}(P) \wedge a, \bar{a} \notin E \wedge a, \bar{a} \notin F\}\\ &= \{\langle A, P''\backslash E\backslash F\rangle \mid \langle A, P''\rangle \in \mathscr{T}(P)\}\\ &\quad \cup \{\langle (a,n), P''\backslash E\backslash F\rangle \mid \langle (a,n), P''\rangle \in \mathscr{T}(P) \wedge a, \bar{a} \notin E \cup F\}.\end{aligned}$$

However,

$$\begin{aligned}\mathscr{T}(P\backslash(E\cup F)) &= \{\langle A, P'\backslash E\cup F\rangle \mid \langle A, P'\rangle \in \mathscr{T}(P)\}\\ &\quad \cup \{\langle (a,n), P'\backslash E\cup F\rangle \mid \langle (a,n), P'\rangle \in \mathscr{T}(P) \wedge a, \bar{a} \notin E\cup F\}.\end{aligned}$$

It follows from Lemma 5.2 that the relation $\{(X\backslash E\backslash F, X\backslash E\cup F) \mid X, Y \in ACSR \wedge E, F \subseteq \mathscr{L}\}$ is a bisimulation.

Close (5).

$$\begin{aligned}&\mathscr{T}([[P]_I]_J)\\ &\quad = \{\langle A' \cup A_3, [P']_J\rangle \mid \langle A', P'\rangle \in \mathscr{T}([P]_I) \wedge A_3 = \{(r,0) \mid r \in J - \rho(A')\}\}\\ &\qquad \cup \{\langle e, [P']_J\rangle \mid \langle e, P'\rangle \in \mathscr{T}([P]_I)\}\\ &\quad = \{\langle (A'' \cup A_2) \cup A_3, [[P'']_I]_J\rangle \mid \langle A'', P''\rangle \in \mathscr{T}(P) \wedge A_2\\ &\quad = \{(r,0) \mid r \in I - \rho(A'')\} \wedge A_3 = \{(r,0) \mid r \in J - \rho(A'' \cup A_2)\}\}\\ &\qquad \cup \{\langle e, [[P'']_I]_J\rangle \mid \langle e, P''\rangle \in \mathscr{T}(P)\}\\ &\quad = \{\langle A'' \cup B, [[P'']_I]_J\rangle \mid \langle A'', P''\rangle \in \mathscr{T}(P) \wedge B\\ &\quad = \{(r,0) \mid r \in (I \cup J) - \rho(A'')\}\}\\ &\qquad \cup \{\langle e, [[P'']_I]_J\rangle \mid \langle e, P''\rangle \in \mathscr{T}(P)\}.\end{aligned}$$

However,

$$\begin{aligned}&\mathscr{T}([P]_{I\cup J})\\ &\quad=\{\langle A''\cup B, [P'']_{I\cup J}\rangle \mid \langle A'', P''\rangle \in \mathscr{T}(P) \wedge B=\{(r,0)\mid r\in (I\cup J)-\rho(A'')\}\}\\ &\qquad\cup\{\langle e, [P''_{I\cup J}\rangle \mid \langle e, P''\rangle \in \mathscr{T}(P)\}.\end{aligned}$$

It follows from Lemma 5.2 that the relation $\{([[X]_I]_J, [X]_{I\cup J}) \mid X\in ACSR \wedge I, J\subseteq \mathscr{R}\}$ is a bisimulation.

Close (7).

$$\begin{aligned}&\mathscr{T}([P\backslash E]_I)\\ &\quad=\{\langle A_1\cup A_2, [P']_I\rangle \mid \langle A_1, P'\rangle \in \mathscr{T}(P\backslash E) \wedge A_2=\{(r,0)\mid r\in I-\rho(A_1)\}\}\\ &\qquad\cup\{\langle (a,n), [P']_I\rangle \mid \langle (a,n), P'\rangle \in \mathscr{T}(P\backslash E)\}\\ &\quad=\{\langle A_1\cup A_2, [P''\backslash E]_I\rangle \mid \langle A_1, P''\rangle \in \mathscr{T}(P) \wedge A_2=\{(r,0)\mid r\in I-\rho(A_1)\}\}\\ &\qquad\cup\{\langle (a,n), [P''\backslash E]_I\rangle \mid \langle (a,n), P''\rangle \in \mathscr{T}(P) \wedge a, \bar{a}\notin E\}.\end{aligned}$$

However,

$$\begin{aligned}&\mathscr{T}([P]_I\backslash E)\\ &\quad=\{\langle A_1, P'\backslash E\rangle \mid \langle A_1, P'\rangle \in \mathscr{T}([P]_I)\}\\ &\qquad\cup\{\langle (a,n), P'\backslash E\rangle \mid \langle (a,n), P'\rangle \in \mathscr{T}([P]_I) \wedge a, \bar{a}\notin E\}\\ &\quad=\{\langle A_1\cup A_2, [P'']_I\backslash E\rangle \mid \langle A_1, P''\rangle \in \mathscr{T}(P) \wedge A_2=\{(r,0)\mid r\in I-\rho(A_1)\}\}\\ &\qquad\cup\{\langle (a,n), [P'']_I\backslash E\rangle \mid \langle (a,n), P''\rangle \in \mathscr{T}(P) \wedge a, \bar{a}\notin E\}.\end{aligned}$$

It follows from Lemma 5.2 that the relation $\{([X\backslash E]_I, [X]_I\backslash E] \mid X\in ACSR \wedge I\subseteq \mathscr{R} \wedge E\subseteq \mathscr{L}\}$ is a bisimulation. ▮

Rec (1). From the operational semantics rule **Rec** we know that $rec\ X.P \xrightarrow{\alpha} P'$ iff $P[rec\ X.P/X] \xrightarrow{\alpha} P'$. Hence we have

$$\begin{aligned}(rec\ X.P)&=\{\langle \alpha, P'\rangle \mid \langle \alpha, P'\rangle \in \mathscr{T}(P[rec\ X.P/X])\}\\ &=\mathscr{T}(P[rec\ X.P/X]).\quad ▮\end{aligned}$$

Rec (2). Let $R \stackrel{\text{def}}{=} rec\ X.Q$; by Rec(1), $R=Q[{}^R/_X]$. We need to prove that $P\sim_\pi R$, assuming that $P=Q[{}^P/_X]$ and X is guarded in Q. We do this by making use of Lemma 5.2 and proving that the relation $\mathscr{R}$ defined by

$$\{(E[{}^P/_X], E[{}^R/_X])\}\cup\{(E,E)\}$$

(where E ranges over the set of ACSR processes) is a prioritized strong bisimulation. The key to this proof is the fact that, when X is guarded in Q, the first step of $Q[{}^{P}/_{X}]$ does not depend on the value of P; more formally,

$$Q[{}^{P}/_{X}] \xrightarrow{\alpha}_{\pi} Q'[{}^{P}/_{X}] \qquad \text{if and only if} \quad Q \xrightarrow{\alpha}_{\pi} Q'$$

and

$$Q[{}^{R}/_{X}] \xrightarrow{\alpha}_{\pi} Q'[{}^{R}/_{X}] \qquad \text{if and only if} \quad Q \xrightarrow{\alpha}_{\pi} Q'.$$

We proceed by induction on the structure of E.

If E is NIL, $\mathscr{T}(E[{}^{P}/_{X}]) = \varnothing = (E[{}^{R}/_{X}])$.

If E is X, we obtain that $E[{}^{P}/_{X}] \equiv P = Q[{}^{P}/_{X}]$ and similarly $E[{}^{R}/_{X}] \equiv R = Q[{}^{R}/_{X}]$ and by **Rec(1)**, we know that

$$\mathscr{T}(E[{}^{P}/_{X}]) = \mathscr{T}(Q[{}^{P}/_{X}]) \text{ and } \mathscr{T}(E[{}^{R}/_{X}]) = \mathscr{T}(Q[{}^{R}/_{X}]).$$

Therefore

$$\mathscr{T}(E[{}^{P}/_{X}]) = \mathscr{T}(Q[{}^{P}/_{X}]) = \{\langle \alpha, Q'[{}^{P}/_{X}]\rangle \mid Q \xrightarrow{\alpha}_{\pi} Q'\}$$

and

$$\mathscr{T}(E[{}^{R}/_{X}]) = \mathscr{T}(Q[{}^{R}/_{X}]) = \{\langle \alpha, Q'[{}^{R}/_{X}]\rangle \mid Q \xrightarrow{\alpha}_{\pi} Q'\}.$$

If E is αF, then

$$\mathscr{T}(E[{}^{P}/_{X}]) = \{\langle \alpha, F[{}^{P}/_{X}]\rangle\}$$

and

$$\mathscr{T}(E[{}^{R}/_{X}]) = \{\langle \alpha, F[{}^{R}/_{X}]\rangle\}.$$

The other cases follow from the induction hypothesis and Lemma 4.1 that prioritized strong bisimulation is a congruence. ∎

Rec (3). To simplify the presentation, let us define

$$f_i(P) \overset{\text{def}}{=} [P \backslash E_i]_{U_i}.$$

An immediate consequence of the laws Res (6), Close (5), and Close (7) is that the functions f_i are commutative and idempotent. This justifies the following notation. Let $I = \{i_1, i_2, ..., i_n\}$,

$$f_I(P) \overset{\text{def}}{=} f_{i_1} f_{i_2} \cdots f_{i_n}(P).$$

As expected,

$$f_{\varnothing}(P) \stackrel{\text{def}}{=} P.$$

It is also easy to see the following by using the laws Res (6), Close (5) and Close (7),

$$f_I(f_J(P)) = f_{I \cup J}(P).$$

Finally, we denote by $|I|$ the cardinality of the finite set I.

The key to this proof resides in the fact that the behavior of any process must be derived from the rules of the operational semantics. In the case of a term of the form *rec X.P*, the only rule that applies is **Rec** and therefore any transition of *rec X.P* must be obtained from unrolling the recursion. The proof proceeds by successive unrollings of the recursion (i.e., applying Rec(1)) until no new behavior can be derived.

Let us see how this works by following an example. Take the process

$$Q \stackrel{\text{def}}{=} rec\ X.(P + [X]_U + [X]_V).$$

By unrolling the recursion we obtain

$$\begin{aligned} Q &= P[{}^Q/_X] + [Q]_U + [Q]_V \\ &= P[{}^Q/_X] + [rec\ X.(P + [X]_U + [X]_V)]_U + [rec\ X.(P + [X]_U + [X]_V)]_V \end{aligned}$$

If we unroll the recursion of the second and third terms, and take advantage of the idempotence and commutativity of the closure operator, we obtain

$$\begin{aligned} Q &= P[{}^Q/_X] + [P[{}^Q/_X]]_U + [[Q]_U]_U + [[Q]_V]_U \\ &\quad + [P[{}^Q/_X]]_V + [[Q]_U]_V + [[Q]_V]_V \\ &= P[{}^Q/_X] + [P[{}^Q/_X]]_U + [P[{}^Q/_X]]_V + [Q]_U + [Q]_{U \cup V} + [Q]_V \end{aligned}$$

Again by unrolling the recursion of the last three terms we obtain:

$$\begin{aligned} Q &= P[{}^Q/_X] + [P[{}^Q/_X]]_U + [P[{}^Q/_X]]_V + [P[{}^Q/_X] + [Q]_U + [Q]_V)]_U \\ &\quad + [P[{}^Q/_X] + [Q]_U + [Q]_V)]_{U \cup V} + [P[{}^Q/_X] + [Q]_U + [Q]_V)]_V \\ &= P[{}^Q/_X] + [P[{}^Q/_X]]_U + [P[{}^Q/_X]]_V + [P[{}^Q/_X]]_U + [[Q]_U]_U + [[Q]_V)]_U \\ &\quad + [P[{}^Q/_X]]_{U \cup V} + [[Q]_U]_{V \cup U} + [[Q]_V)]_{V \cup U} \\ &\quad + [P[{}^Q/_X]]_V + [[Q]_U]_V + [[Q]_V)]_V \\ &= P[{}^Q/_X] + [P[{}^Q/_X]]_U + [P[{}^Q/_X]]_V + [P[{}^Q/_X]]_{U \cup V} \\ &\quad + [Q]_U + [Q]_{U \cup V} + [Q]_V. \end{aligned}$$

From this point on, unrolling the recursion of the last three terms fails to produce any new summand. It follows that the behavior of Q is captured by the first four terms and therefore

$$Q = P[{}^{Q}/_{X}] + [P[{}^{Q}/_{X}]]_{U} + [P[{}^{Q}/_{X}]]_{V} + [P[{}^{Q}/_{X}]]_{U \cup V}.$$

We first prove the following lemma, where X may or may not be unguarded in P:

Lemma 5.3. *If $Q = rec\ X.(P + \sum_{i \in I} f_i(X))$ then*

$$Q = \sum_{J \subseteq I} f_J(P[{}^{Q}/_{X}]) + \sum_{J \subseteq I,\, J \neq \varnothing} f_J(Q).$$

Proof. We proceed by induction on the cardinality of I. When $|I| = 1$ we obtain the result by unrolling the recursion twice as follows:

$$\begin{aligned} Q &= rec\ X.(P + f_{i_1}(X)) \\ &= P[{}^{Q}/_{X}] + f_{i_1}(Q) \\ &= P[{}^{Q}/_{X}] + f_{i_1}(P[{}^{Q}/_{X}] + f_{i_1}(Q)) \\ &= P[{}^{Q}/_{X}] + f_{i_1}(P[{}^{Q}/_{X}]) + f_{i_1} f_{i_1}(Q) \\ &= \sum_{J \subseteq \{i_1\}} f_J(P[{}^{Q}/_{X}]) + \sum_{J \subseteq \{i_1\},\, J \neq \varnothing} f_J(Q). \end{aligned}$$

Now assume the result true for any set I such that $|I| = n$. Let $I' = I \cup \{i_{n+1}\}$ and $Q \overset{\text{def}}{=} rec\ X.(P + \sum_{i \in I'} f_i(X))$.

Let $P' \overset{\text{def}}{=} (P + f_{i_{n+1}}(X))$; by the induction hypothesis we have:

$$\begin{aligned} Q &= \sum_{J \subseteq I} f_J(P'[{}^{Q}/_{X}]) + \sum_{J \subseteq I,\, J \neq \varnothing} f_J(Q) \\ &= \sum_{J \subseteq I} f_J(P[{}^{Q}/_{X}] + f_{i_{n+1}}(Q)) + \sum_{J \subseteq I,\, J \neq \varnothing} f_J(Q) \\ &= \sum_{J \subseteq I} f_J(P[{}^{Q}/_{X}]) + \sum_{J \subseteq I} f_J f_{i_{n+1}}(Q) + \sum_{J \subseteq I,\, J \neq \varnothing} f_J(Q). \end{aligned}$$

We can combine the last two summations by observing that one ranges over the subsets of I' that *do* contain $\{i_{n+1}\}$ (and therefore are not empty) while the other ranges over the non-empty subsets of I' that *do not* contain $\{i_{n+1}\}$. We obtain

$$Q = \sum_{J \subseteq I} f_J(P[{}^{Q}/_{X}]) + \sum_{J \subseteq I',\, J \neq \varnothing} f_J(Q).$$

By unrolling the recursion one more time we obtain

$$\begin{aligned} Q &= \sum_{J \subseteq I} f_J(P[{}^{Q}/_{X}]) + \sum_{J \subseteq I',\, J \neq \varnothing} f_J\left(P[{}^{Q}/_{X}] + \sum_{i \in I} f_i(Q) \right) \\ &= \sum_{J \subseteq I} f_J(P[{}^{Q}/_{X}]) + \sum_{J \subseteq I',\, J \neq \varnothing} f_J(P[{}^{Q}/_{X}]) + \sum_{J \subseteq I',\, J \neq \varnothing} f_J\left(\sum_{i \in I} f_i(Q) \right). \end{aligned}$$

Notice that the first summation covers all the subsets of I' that do not contain i_{n+1} while the second covers, among others, all the subsets of I' that do contain i_{n+1}. In the third summation, we apply the idempotence of f_i to obtain the desired result. ▮

Back to the proof of Rec(3); we define

$$Q \overset{\text{def}}{=} rec\ X.\left(P + \sum_{i \in I} f_i(X) \right)$$

and we are about to prove the equation

$$Q = rec\ X.\left(\sum_{J \subseteq I} f_I(P) \right).$$

From the above lemma, we have

$$Q = \sum_{J \subseteq I} f_J(P[{}^{Q}/_{X}]) + \sum_{J \subseteq I,\, J \neq \varnothing} f_J(Q).$$

Let

$$Q_1 \overset{\text{def}}{=} \sum_{J \subseteq I} f_J(P[{}^{Q}/_{X}])$$

and

$$Q_2 \overset{\text{def}}{=} \sum_{J \subseteq I,\, J \neq \varnothing} f_J(Q) = \sum_{J \subseteq I,\, J \neq \varnothing} f_J\left(rec\ X.\left(P + \sum_{i \subseteq I} f_i(X) \right) \right).$$

Note that the behaviors induced by Q_2 must be derived from unrolling the recursion. By doing so we obtain

$$\begin{aligned} Q_2 &= \sum_{J \subseteq I,\, J \neq \varnothing} f_J(P[{}^{Q}/_{X}]) + \sum_{J \subseteq I,\, J \neq \varnothing} f_J\left(\sum_{J \subseteq I} f_I(Q) \right) \\ &= \sum_{J \subseteq I,\, J \neq \varnothing} f_J(P[{}^{Q}/_{X}]) + Q_2. \end{aligned}$$

Hence, we have the following:

$$\mathscr{T}(Q_2) = \mathscr{T}\left(\sum_{J \subseteq I,\, J \neq \varnothing} f_J(P[{}^{Q}/_{X}]) \right) \cup \mathscr{T}(Q_2).$$

It is easy to see that all behaviors of the process $\sum_{J \subseteq I, J \neq \varnothing} f_J(P[{}^{Q}/_{X}])$ are included in Q_2 that is,

$$\mathscr{T}\left(\sum_{J \subseteq I, J \neq \varnothing} f_J(P[{}^{Q}/_{X}])\right) \subseteq \mathscr{T}(Q_2).$$

Since all transitions from Q_2 must come from terms other than unguarded variables, we get the following:

$$\mathscr{T}(Q_2) \subseteq \mathscr{T}\left(\sum_{J \subseteq I, J \neq \varnothing} f_J(P[{}^{Q}/_{X}])\right).$$

Therefore, we obtain that

$$Q_2 = \sum_{J \subseteq I, J \neq \varnothing} f_J(P[{}^{Q}/_{X}]).$$

However,

$$\mathscr{T}\left(\sum_{J \subseteq I, J \neq \varnothing} f_J(P[{}^{Q}/_{X}])\right) \subseteq \mathscr{T}\left(\sum_{J \subseteq I} f_J(P[{}^{Q}/_{X}])\right).$$

Hence, it is easy to see that all the behaviors that can be induced by unrolling Q_2 are already included in Q_1, and therefore Q_2 can be ignored, resulting in

$$Q = \sum_{J \subseteq I} f_J(P[{}^{Q}/_{X}]) = \left(\sum_{J \subseteq I} f_J(P)\right)[{}^{Q}/_{X}].$$

But since X is guarded in P, we can apply Rec(2) and obtain

$$Q = rec\ X. \sum_{J \subseteq I} f_J(P).$$

6. COMPLETENESS FOR FINITE STATE PROCESSES

In this section, we prove that the ACSR laws are complete for some (large) subset of the finite state processes. The section is divided as follows. First we refine the definition of bisimulation to formally cope with free variables. We characterize a subset of ACSR processes, which we call "FS" processes, for which we prove completeness of the set of laws $\mathscr{A}$. We then develop the four steps of the proof of completeness, which are as follows. We prove that unguarded recursions can be eliminated. In the absense of unguarded recursion, any FS process satisfies a certain kind of equation set. If two processes are bisimilar, then they satisfy a common set of equations. Finally, we prove that those sets of equations have a unique solution up to bisimulation.

6.1. Refined Definition of Bisimulation

The presence of recursion will require us to have a formal treatment for free variables. In particular, we need to extend the notion of bisimulation to take the presence of free variables into account. Milner (1989a) extends the notion of bisimulation to encompass unguarded free variables. In our case, the presence of the restriction and closure operators requires more discrimination. Consider, for example, $X E$ and $[X]_I$; even though the variable X is unguarded in both cases, the two expressions are certainly not equivalent.

Let us define a relation $\rightarrow$ (without label) as the minimum relation that satisfies the following rules:

$$\frac{-}{X \rightarrow [X\backslash\emptyset]_\emptyset} \qquad \frac{P \rightarrow [X\backslash E]_I}{P \parallel Q \rightarrow [X\backslash E]_I}$$

$$\frac{P \rightarrow [X\backslash E]_I}{P+Q \rightarrow [X\backslash E]_I} \qquad \frac{Q \rightarrow [X\backslash E]_I}{P \parallel Q \rightarrow [X\backslash E]_I}$$

$$\frac{Q \rightarrow [X\backslash E]_I}{P+Q \rightarrow [X\backslash E]_I} \qquad \frac{P \rightarrow [X\backslash E]_I}{[P]_J \rightarrow [X\backslash E]_{I \cup J}}$$

$$\frac{P \rightarrow [X\backslash E]_I}{P \,\triangle_t^b\, (Q, R, S) \rightarrow [X\backslash E]_I}\, t>0 \qquad \frac{P \rightarrow [X\backslash E]_I}{P\backslash F \rightarrow [X\backslash(E \cup F)]_I}$$

$$\frac{S \rightarrow [X\backslash E]_I}{P \,\triangle_t^b\, (Q, R, S) \rightarrow [X\backslash E]_I}\, t>0 \qquad \frac{P \rightarrow [X\backslash E]_I}{rec\ Y.P \rightarrow [X\backslash E]_I}\, X \neq Y$$

$$\frac{R \rightarrow [X\backslash E]_I}{P \,\triangle_0^b\, (Q, R, S) \rightarrow [X\backslash E]_I}$$

Note that using the laws Res (5), Res (6), Res (7), Close (5), Close (6), and Close (7), a variable X with any combinations of close operators and restriction operators can be transformed into a standard form: $[X\backslash E]_I$. For instance, if $E_1 \cup E_2 = E$ and $I_1 \cup I_2 = I$, then the terms $[X\backslash E]_I$, $([X]_I)\backslash E$, $[X\backslash E_1 \cup E_2]_I$, $[X\backslash E]_{I_1 \cup I_2}$, and $[[X\backslash E_1]_{I_1}\backslash E_2]_{I_2}$ can be considered the same. Hence, if $P \rightarrow [X\backslash E]_I$, P contains a free unguarded occurrence of a term which can be transformed into $[X\backslash E]_I$.

Based on this, we can define the notion of bisimulation that we will be using throughout this section.

DEFINITION 6.1. A relation $R \subseteq ACSR \times ACSR$ is a bisimulation if for all $\alpha \in \mathscr{D}$, $I \subseteq \mathscr{R}$, and $E \subseteq \mathscr{L}$, whenever $(P, Q) \in R$,

1. if $P \xrightarrow{\alpha}_\pi P'$ then, for some Q', $Q \xrightarrow{\alpha}_\pi Q'$ and $(P', Q') \in R$, and
2. if $Q \xrightarrow{\alpha}_\pi Q'$ then, for some P', $P \xrightarrow{\alpha}_\pi P'$ and $(P', Q') \in R$, and
3. $P \rightarrow [X\backslash E]_I$ iff $Q \rightarrow [X\backslash E]_I$

If $(P, Q) \in R$ for some bisimulation R, then we say that P is bisimilar to Q, and write $P \sim_\pi Q$.

It is straightforward to see that this refined definition corresponds to our previous definition in the absence of free variables. None of the laws deal explicitly with free variables, and one can easily check that they remain sound under this new definition.

6.2. *Characterization of* FS *Processes*

It is well known that Turing machines can be coded in CCS, which is a subset of ACSR. Since the semantics of ACSR coincides with that of CCS on their common syntax, we know that there is no finite set of laws that can be used to prove the equivalence of *any* ACSR processes. Completeness has been proven in the past for a subset of CCS processes called "finite state agents." The definition that previous authors, such as (Milner, 1989a; Bolognesi and Smolka, 1987), have used for finite state agents has been *processes coded without the parallel operator*, and since the restriction operator becomes useless in this environment, it has been eliminated as well. This simple solution does not work for ACSR because non-finite-state processes can be generated even without the use of the parallel operator, as is illustrated by the following example.

Example 6.1. Consider the process $P \overset{\text{def}}{=} rec\ X.(A{:}X \,\triangle^{b}_{\infty}\, (\text{NIL}, \text{NIL}, B{:}\text{NIL}))$. It has two possible transitions:

$$P \xrightarrow{B} \text{NIL}$$

and

$$P \xrightarrow{A} (rec\ X.A{:}X \,\triangle^{b}_{\infty}\, (\text{NIL}, \text{NIL}, B{:}\text{NIL})) \,\triangle^{b}_{\infty}\, (\text{NIL}, \text{NIL}, B{:}\text{NIL})$$

call this last process P'; it has three possible transitions,

$$P' \xrightarrow{B} \text{NIL}$$
$$P' \xrightarrow{B} \text{NIL} \,\triangle^{b}_{\infty}\, (\text{NIL}, \text{NIL}, B{:}\text{NIL})$$

and

$$\begin{aligned} P' \xrightarrow{A} (&rec\ X.(A{:}X \,\triangle^{b}_{\infty}\, (\text{NIL}, \text{NIL}, B{:}\text{NIL})) \\ &\triangle^{b}_{\infty}\, (\text{NIL}, \text{NIL}, B{:}\text{NIL}) \,\triangle^{b}_{\infty}\, (\text{NIL}, \text{NIL}, B{:}\text{NIL})) \end{aligned}$$

and so forth as shown in Fig. 2.

Eliminating the Scope operator altogether would eliminate too much expressiveness of the ACSR language and render the whole exercise futile. Therefore we decided to extend the proof to the set of processes that do not have recursion

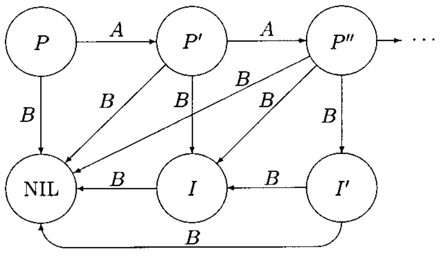

FIG. 2. An infinite state process

through parallel nor scope. Unfortunately, this is very difficult to characterize syntactically—for example, the process $P = rec\ X.(A{:}X \| \mathrm{NIL})$ is equivalent to NIL and therefore does not have recursion through parallel. Nevertheless there are obvious advantages to a syntactic characterization and therefore we limit our proof to processes that have "no free variable under parallel and no free variable in a process under a scope operator." We say that such processes are "FS." It seems that most finite state processes either are FS or are provably equivalent to an FS process. The only exceptions we have found so far are of the form $rec\ X.(X \triangle_{\infty}^{b} (P, Q, R))$.

We formally define FS processes by way of a recursive function *fs* over processes (we assume the usual definition of the function $fv(P)$ which yields the set of free variables of a process P):

$$
\begin{aligned}
fs(\mathrm{NIL}) &= true \\
fs(X) &= true \\
fs(A{:}P) &= fs(P) \\
fs((a, n).P) &= fs(P) \\
fs(P + Q) &= fs(P) \wedge fs(Q) \\
fs(P \triangle_{t}^{b} (Q, R, S)) &= (fv(P) = \varnothing) \wedge fs(P) \wedge fs(Q) \wedge fs(R) \wedge fs(S) \\
fs(P \| Q) &= (fv(P) \cup fv(Q) = \varnothing) \wedge fs(P) \wedge fs(Q) \\
fs([P]_I) &= fs(P) \\
fs(P \backslash F) &= fs(P) \\
fs(rec\ X.P) &= fs(P).
\end{aligned}
$$

DEFINITION 6.2 (FS Process). *A process is said to be* FS *if* $fs(P) = true$.

6.3. Elimination of Unguarded Variables

In this section we prove that any FS process is provably equivalent to a process where all the recursions are guarded. We do this by defining a head normal form where all the unguarded free variables are isolated as summands.

DEFINITION 6.3 (Head Normal Form). *A process P is in Head Normal Form or* "HNF" *if it has the form*

$$\sum_{i \in I} [X_i \backslash E_i]_{U_i} + \sum_{j \in J} a_j Q_j.$$

Note that we do not require that the Q_j have any particular form.

LEMMA 6.1. *For any* FS *process* P, *there exists a process* Q *such that* $P = Q$ *and* Q *is in* HNF.

Proof. We proceed by induction on the structure of P. For the base cases, NIL is in HNF, and X can be transformed into the HNF "$[X\backslash\emptyset]_{\emptyset}$" using Res (7) and Close (6). Now, assume it is true for any term of depth $\leqslant n$. For a term of depth $n+1$, we examine all the possible forms such a term can take:

Case 1 (Timed Action Prefix). $P = A{:}P'$ is in HNF.

Case 2 (Instantaneous Event Prefix). $P = e.P'$ is in HNF.

Case 3 (Choice). $P+R$ with P and R in HNF (by the induction hypothesis) has the form

$$\sum_{i\in I}[X_i\backslash E_i]_{U_i} + \sum_{j\in J}\alpha_j Q_j + \sum_{i\in I'}[X_i\backslash E_i]_{U_i} + \sum_{j\in J'}\alpha_j Q_j.$$

Using the laws Choice (3) and Choice (4) we can re-arrange the terms and obtain the normal form:

$$\sum_{i\in I\cup I'}[X_i\backslash E_i]_{U_i} + \sum_{j\in J\cup J'}\alpha_j Q_j.$$

Case 4 (Parallel). By the induction hypothesis, and since P and Q are FS, $P\parallel Q$ can be written (Note that neither P nor Q has any free variable)

$$\left(\sum_{i\in I} A_i{:}P_i + \sum_{j\in J} e_j.Q_j\right)\Bigg\|\left(\sum_{k\in K} B_k{:}R_k + \sum_{l\in L} f_l.S_l\right).$$

Using Par (3), we obtain

$$\begin{aligned} P\parallel Q = & \sum_{i\in I,\, k\in K,\, \rho(A_i)\cap\rho(B_k)=\emptyset} (A_i\cup B_k){:}(P_i\parallel R_k) \\ & + \sum_{j\in J} e_j.\left(Q_j\Bigg\|\left(\sum_{k\in K} B_k{:}R_k + \sum_{l\in L} f_l.S_l\right)\right) \\ & + \sum_{l\in L} f_l.\left(\left(\sum_{i\in I} A_i{:}P_i + \sum_{j\in J} e_j.Q_j\right)\Bigg\| S_l\right) \\ & + \sum_{j\in J,\, l\in L,\, l(e_j)=l(f_l)} (\tau, \pi(e_j)+\pi(f_l)).(Q_j\parallel S_l) \end{aligned}$$

which is in HNF.

Case 5 (Scope). Let $P = Q \bigtriangleup_t^b (R, S, T)$, by induction hypothesis we can assume that Q, R, S, and T are in HNF. Observe that if P can be transformed into a sum of terms in HNF, it can further be transformed to be in HNF by using Choice (3) and Choice (4).

We prove that P can be transformed into a sum of terms in HNF by examining all the possible forms P can take.

When $t=0$, by Scope(4) we have $P=S$, which is in HNF.

Otherwise, since Q is FS and in HNF, we can distribute Scope over the summation by using Scope (5). To each summand, we can apply one of the three laws Scope (1), Scope (2), and Scope (3).

When Scope (1) applies, we obtain a term of the form $A{:}(Q' \bigtriangleup^{b}_{t-1} (R, S, T)) + T$, with T already in HNF.

When Scope (2) applies, we obtain a term of the form $(a, m).(Q' \bigtriangleup^{b}_{t} (R, S, T)) + T$; again, T is already in HNF.

When Scope (3) applies, we obtain a term of the form: $(\tau, m).Q' + T$, with T already in HNF.

Case 6 (Restriction). $P = Q\backslash F$; by the induction hypothesis, we can assume that Q is in HNF. If $Q = \text{NIL}$ we obtain a HNF by Res (1). Otherwise, using Res (2), we can distribute the restriction over every summand. Then, using Res (3) for timed action and Res (4) or Res (5) for instantaneous events, the restriction operator can be pushed down one level, or the summand becomes NIL. For free variables, we have terms of the form $[X_i\backslash E]_U\backslash F$, which can be transformed into $[X_i\backslash E \cup F]_U$ by Close (7) and Res (6).

Case 7 (Close). $P = [Q]_I$, by induction hypothesis, we can assume that Q is in HNF. If $Q = \text{NIL}$ we obtain a HNF by Close (1). Otherwise, using Close (2) we can distribute the closure operator over every summand. Then, using Close (3) for timed action and Close (4) for instantaneous events, the closure operator can be pushed down one level. For free variables, we have terms of the form $[[X_i\backslash E]_U]_I$ which can be transformed into $[X_i\backslash E]_{U \cup I}$ by Close (5).

Case 8 (Recursion). As we have done in the proof of Rec (3), we will use the notation

$$f_i(P) \overset{\text{def}}{=} [P\backslash E_i]_{U_i}$$

and

$$f_I(P) \overset{\text{def}}{=} f_{i_1} f_{i_2} \cdots f_{i_n}(P).$$

Let $P = rec\ X.Q$ with Q in HNF. Therefore,

$$P = rec\ X.\Big(\sum_{i \in I} f_i(X_i) + \sum_{j \in J} \alpha_j P_j\Big).$$

Let $I' = \{i \in I \mid X_i \neq X\}$ and $I'' = I - I'$; we have

$$P = rec\ X.\Big(\sum_{i \in I''} f_i(X) + \sum_{i \in I'} f_i(X_i) + \sum_{j \in J} \alpha_j P_j\Big)$$

By Rec (3) we obtain:

$$\begin{aligned} P &= rec\ X.\Big(\sum_{K \subseteq I''} f_K\Big(\sum_{i \in I'} f_i(X_i) + \sum_{j \in J} \alpha_j P_j\Big)\Big) \\ &= rec\ X.\Big(\sum_{K \subseteq I''} \sum_{i \in I'} f_K f_i(X_i) + \sum_{K \subseteq I''} \sum_{j \in J} f_K(\alpha_j P_j)\Big). \end{aligned}$$

Using Close (3), Close (4), Res (3), Res (4), and Res (5) we obtain

$$P = rec\ X.\left(\sum_{K \subseteq I''} \sum_{i \in I'} f_K f_i(X_i) + \sum_{K \subseteq I''} \sum_{j \in J,\, l(\alpha) \notin E_k} \beta_{jK} f_K(P_j) \right)$$

where

$$\beta_{jK} = \begin{cases} \alpha_j & \text{if } \alpha_j \in \mathscr{D}_E \\ \alpha_j \cup \{(r, 0) \mid r \in \bigcup_{k \in K} U_k - \rho(\alpha_j)\} & \text{if } \alpha_j \in \mathscr{D}_R \end{cases}.$$

At this point, by applying Rec(1) and noticing that none of the X_i is X we obtain the HNF:

$$P = \sum_{K \subseteq I''} \sum_{i \in I'} f_K f_i(X_i) + \sum_{K \subseteq I''} \sum_{j \in J,\, l(\alpha) \notin E_k} \beta_{jK} f_K(P_j[{}^{rec\ X.P}/_X].$$

We are now ready to prove the following theorem.

THEOREM 6.1. *For every FS process P, there exists a process P' such that $P = P'$ and all the recursions in P' are guarded.*

Proof. By induction on the depth of recursion. The base case is vacuously true. Let us look at the outmost level of recursion of a process P, and assume it is of the form $rec\ X.Q$. By Lemma 6.1, Q can be put in HNF, $\hat{Q}$, where, by induction hypothesis, all the recursion are guarded. If X is unguarded in $\hat{Q}$ then it is among the X_i and it can be eliminated by applying Rec (3). ∎

6.4. Standard Set of Equations

In this section we prove that any guarded FS process provably satisfies a particular set of equations.

Let $\tilde{X} = \{X_1, X_2, \ldots X_n\}$ and $\tilde{W} = \{W_1, W_2, \ldots\}$ be disjoint sets of variables. Let $\tilde{H} = \{H_1, H_2, \ldots H_n\}$ be terms with free variables in $\tilde{X} \cup \tilde{W}$. We say that a process P provably satisfies a set of equations $S{:}\tilde{X} = \tilde{H}$ if there is a set of terms: $\tilde{P} = \{P_1, P_2, \ldots P_n\}$ such that $fv(\tilde{P}) \subseteq \tilde{W}$ and $\tilde{P} = \tilde{H}[{}^{\tilde{P}}/_{\tilde{X}}]$ and $P = P_1$.

A set of equations S is said to be standard if every equation is of the form

$$X_i = \sum_{j \in J_i} [W_j \backslash E_{ij}]_{U_{ij}} + \sum_{k \in K_i} \alpha_{ik} X_k.$$

Finally, a set of equations is said to be prioritized if it is standard and if, in any given equation, there are no two summands αX_i and βX_j such that $\alpha \prec \beta$.

In this section, we will assume that the set of standard equations satisfied by a process P is noted $S{:}\tilde{X} = \tilde{H}$, with X_1 being the distinguished variable, and that every equation has the form

$$X_i = \sum_{j \in J_i} [W_j \backslash E_{ij}]_{U_{ij}} + \sum_{k \in K_i} \alpha_{ik} X_k.$$

Similarly, we will assume that the set of standard equations satisfied by a process Q is noted $T{:}\,\tilde{Y}=\tilde{G}$, with Y_1 being the distinguished variable, and that every equation has the form

$$Y_i=\sum_{l\in L_i}[W_l\backslash F_{il}]_{V_{il}}+\sum_{m\in M_i}\beta_{im}Y_m.$$

Furthermore, we will assume that the sets of variables $\tilde{W}$, $\tilde{X}$, and $\tilde{Y}$ are all disjoint.

LEMMA 6.2. *Every guarded FS process R with free variables in $\tilde{W}$ provably satisfies a standard set of equations with free variables in $\tilde{W}$.*

Proof. By induction on the structure of R.

Case 1 ($R=\mathrm{NIL}$). R satisfies the single equation $X=\mathrm{NIL}$.

Case 2 ($R=W$). By Res (7) and Close (6), R satisfies the single equation $X=[W\backslash\varnothing]_{\varnothing}$.

Case 3 ($R=\alpha P$). By the induction hypothesis, P provably satisfies $S{:}\tilde{X}=\tilde{H}$. Therefore R provably satisfies the standard set $\{X=\alpha X_1\}\cup S$ with the new distinguished variable X.

Case 4 ($R=P+Q$). By induction hypothesis, P provably satisfies $S{:}\tilde{X}=\tilde{H}$ and Q provably satisfies $T{:}\,\tilde{Y}=\tilde{G}$. Since

$$H_1+G_1=\sum_{j\in J_1}[W_j\backslash E_{1j}]_{U_{1j}}+\sum_{k\in K_1}\alpha_{1k}X_k+\sum_{l\in L_1}[W_l\backslash F_{1l}]_{V_{1l}}+\sum_{m\in M_1}\beta_{1m}Y_m,$$

using the laws Choice(3) and Choice(4), we obtained the following equation:

$$X_n=\sum_{j\in J_1\cup L_1}[W_j\backslash E_{nj}]_{U_{nj}}+\sum_{k\in K_1\cup M_1}\alpha_{nk}X_k.$$

Therefore R provably satisfies the set of equations $\{X_n=\sum_{j\in J_1\cup L_1}[W_j\backslash E_{nj}]_{U_{nj}}+\sum_{k\in K_1\cup M_1}\alpha_{nk}X_k\}\cup S\cup T$, with the new distinguished variable X.

Case 5 ($R=P\parallel Q$). Note that since R is FS, neither P nor Q has any free variable and therefore all the sets J_i and L_i are empty. Therefore, we can assume that the equations of S are written

$$X_i=\sum_{k\in K_i'}A_{ik}{:}X_k+\sum_{k\in K_i''}e_{ik}.X_k$$

and that the equations of T are written

$$Y_i=\sum_{m\in M_i'}B_{im}{:}Y_m+\sum_{m\in M_i''}f_{im}.Y_m.$$

R satisfies the (non-standard) equation $Z_{1,1}=H_1\parallel G_1$. It follows from Par (3) that this equation can be written

$$\begin{aligned}
Z_{1,1} = & \sum_{k \in K'_1,\, m \in M'_1,\, \rho(A_{1k}) \cap \rho(B_{1m}) = \varnothing} (A_{1k} \cup B_{1m}):(X_k \parallel Y_m) \\
& + \sum_{k \in K''_1} e_{1k}.(X_k \parallel Y_1) \\
& + \sum_{m \in M''_1} f_{1m}.(X_1 \parallel Y_m) \\
& + \sum_{k \in K''_1,\, m \in M''_1,\, l(e_{1k}) = \overline{l(f_{1m})}} (\tau, \pi(e_{1k}) + \pi(f_{1m})).(X_k \parallel Y_m)
\end{aligned}$$

In the same fashion we can define a set of equations $Z_{i,j} = H_i \parallel G_j$ and then apply Par (3) to obtain a set of standard equations of the form

$$\begin{aligned}
Z_{i,j} = & \sum_{k \in K'_i,\, m \in M'_j,\, \rho(A_{ik}) \cap \rho(B_{jm}) = \varnothing} (A_{ik} \cup B_{jm}):Z_{k,m} \\
& + \sum_{k \in K''_i} e_{ik}.Z_{k,j} \\
& + \sum_{m \in M''_j} f_{jm}.Z_{i,m} \\
& + \sum_{k \in K''_i,\, m \in M''_j,\, l(e_{ik}) = \overline{l(f_{jm})}} (\tau, \pi(e_{ik}) + \pi(f_{jm})).Z_{k,m}
\end{aligned}$$

Case 6 ($R = P \triangle_t^b (Q, S, T)$). Let us denote by $Y'_1 = G'_1$ and $Y''_1 = G''_1$ the distinguished equations of the sets satisfied by S and T, respectively. R satisfies the non-standard equation

$$Z_1 = H_1 \triangle_t^b (G_1, G'_1, G''_1).$$

If $t = 0$, we can apply Scope (4) and obtain the standard equation $Z_1 = G'_1$. If $t > 0$ and $H_1 = \mathrm{NIL}$, we apply Scope (6) and obtain $Z_1 = G''_1$. Otherwise, using Scope (5), we can distribute the scope operator over each summand of H_1 and obtain an equation of the following form (remember that Q is FS and therefore P does not have any free variable):

$$\begin{aligned}
Z_1 & = \sum_{j \in J_1} ((\alpha_{1j} X_j) \triangle_t^b (G_1, G'_1, G''_1)) \\
& = \sum_{j \in J'_1} ((A_{1j}:X_j) \triangle_t^b (G_1, G'_1, G''_1)) + \sum_{j \in J''_1} ((e_{1j}.X_j) \triangle_t^b (G_1, G'_1, G''_1)).
\end{aligned}$$

We can now apply Scope (1), Scope (2), or Scope (3) to each summand and obtain an equation of the form

$$\begin{aligned}
Z_1 = & \sum_{j \in J'_1} A_{1j}:(X_j \triangle_{t-1}^b (G_1, G'_1, G''_1)) + \sum_{j \in J''_1,\, l(e_{1j}) \neq \bar{b}} e_{1j}.(X_j \triangle_t^b (G_1, G'_1, G''_1)) \\
& + \sum_{j \in J''_1,\, l(e_{1j}) = \bar{b}} (\tau, \pi(e_{1j})).G_1 + G''_1.
\end{aligned}$$

G_1 can be replaced by Y_1. We can also take fresh variables to define a new pair of equation for each summand of H_1 as follows,

$$Z_j^t = H_j \bigtriangleup_t^b (G_1, G'_1, G''_1)$$
$$Z_j^{t-1} = H_j \bigtriangleup_{t-1}^b (G_1, G'_1, G''_1)$$

and we obtain the standard equation

$$Z_1 = \sum_{j \in J'_1} A_{1j}{:}Z^{t-1}_j + \sum_{j \in J''_1,\, l(e_{1j}) \neq \bar{b}} e_{1j}.Z_j^t + \sum_{j \in J''_1,\, l(e_{1j}) = \bar{b}} (\tau, \pi(e_{1j})).Y_1 + G''_1.$$

The same process can be applied to standardize the newly defined equations. The number of equations generated by this process is limited to the number of equations in S times t, when t is finite, and to the exact number of equations in S when $t = \infty$. Therefore the process always terminates and leads to a standard set of equations.

Case 7 ($R = P \backslash F$). R satisfies the non-standard equation $Z_1 = H_1 \backslash F$. If we expand H_1 and apply Res(2)—or Res(1) if $H_1 = \mathrm{NIL}$—we obtain

$$\begin{aligned} Z_1 &= \sum_{j \in J_1} ([W_j \backslash E_j]_{U_j}) \backslash F + \sum_{k \in K_1} (\alpha_{1k} X_k) \backslash F \\ &= \sum_{j \in J_1} [W_j \backslash E_j]_{U_j} \backslash F + \sum_{k \in K'_1} (A_{1k}{:}X_k) \backslash F + \sum_{k \in K''_1} (e_{1k}.X_k) \backslash F. \end{aligned}$$

By Close (7) and Res(7) we have $[W_j \backslash E_j]_{U_j} \backslash F = [W_j \backslash E_j \cup F]_{U_j}$. Applying Res(3), Res(4) and Res(5) and introducing the new equations $Z_i = H_i \backslash F$ gives a set of standard equations of the form

$$Z_i = \sum_{j \in J_i} [W_j \backslash E_j \cup F]_{U_j} + \sum_{k \in K'_i} A_{ik}{:}Z_k + \sum_{k \in K''_i,\, l(e_{ik}),\, \overline{l(e_{ik})} \notin F} e_{ik}.Z_k.$$

Case 8 ($R = [P]_V$). R satisfies the non-standard equation $Z_1 = [H_1]_V$. If we expand H_1 and apply Close (2)—or Close (1) if $H_1 = \mathrm{NIL}$—we obtain

$$\begin{aligned} Z_1 &= \sum_{j \in J_1} [[W_j \backslash E_j]_{U_j}]_V + \sum_{k \in K_1} [\alpha_{1k} X_k]_V \\ &= \sum_{j \in J_1} [[W_j \backslash E_j]_{U_j}]_V + \sum_{k \in K'_1} [A_{1k}{:}X_k]_V + \sum_{k \in K''_1} [e_{1k}.X_k]_V. \end{aligned}$$

By Close (5) we have $[[W_j \backslash E_j]_{U_j}]_V = [W_j \backslash E_j]_{U_j \cup V}$. Applying Close (3) and Close (4) and introducing the new equations $Z_i = [H_i]_V$ gives a set of standard equations of the form

$$Z_i = \sum_{j \in J_i} [W_j \backslash E_j]_{U_j \cup V} + \sum_{k \in K'_i} (A_{ik} \cup \{(r, 0) \mid r \in V - \rho(A_{ik})\}){:}Z_k + \sum_{k \in K''_i} e_{ik}.Z_k.$$

Case 9 ($R = rec\ W.P$). Since W is guarded in P, it does not occur in H_1. Using Rec(1), we can define a new set of equations by replacing every occurence of W

by H_1. That is, $Z_i = H_i[{}^{H_1}/_W]$. This clearly eliminates W, but generates terms of the form $[H_1 \backslash E_j]_{U_j}$. However, from cases 7 and 8 above, we know that these can be standardized. ∎

THEOREM 6.2. *Every guarded FS process P with free variables $\tilde{W}$ provably satisfies a prioritized set of equations S with free variables in $\tilde{W}$.*

Proof. From the above lemma, we know that the process P satisfies a standard set of equations S. In addition, each equation can be prioritized by using the law Choice (5). ∎

6.5. *Common Set of Prioritized Standard Equations*

In this section we prove that when two processes are bisimilar, they satisfy a common set of prioritized equations.

THEOREM 6.3. *Let P and Q provably satisfy two standard sets of equations S and T. If P and Q are bisimilar, then there exists a third standard set of equations S′ satisfied by both P and Q.*

Proof. We assume that $\tilde{X} = \{X_1, X_2, ... X_n\}$, $\tilde{Y} = \{Y_1, Y_2, ... Y_m\}$, and $\tilde{W} = \{W_1, W_2, ...\}$ are disjoint sets of variables, that the given sets of equations are:

$$S: \tilde{X} = \tilde{H}$$

$$T: \tilde{Y} = \tilde{G},$$

and that there are terms $\tilde{P} = \{P_1, P_2, ... P_n\}$ and $\tilde{Q} = \{Q_1, Q_2, ... Q_m\}$ such that $fv(\tilde{P}) \cup fv(\tilde{Q}) \subseteq \tilde{W}$, and $P = P_1$, and $Q = Q_1$, so that

$$P_u = \sum_{j \in J_u} [W_j \backslash E_{uj}]_{U_{uj}} + \sum_{k \in K_u} \alpha_{uk} P_k \qquad u \leqslant n$$

$$Q_v = \sum_{l \in L_v} [W_l \backslash F_{vl}]_{V_{vl}} + \sum_{k' \in M_v} \beta_{vk'} Q_{k'} \qquad v \leqslant m.$$

Let us consider a relation $\mathscr{R}$ such that $(i, j) \in \mathscr{R}$ iff $H_i \sim_\pi G_j$. Clearly, $(1, 1) \in R$. Since P_u and Q_v must have equal transitions and equal variables when $(u, v) \in R$, the following three statements are true:

1. $\sum_{j \in J_u} [W_j \backslash E_{uj}]_{U_{uj}} = \sum_{j \in L_v} [W_l \backslash F_{vl}]_{V_{vl}}$,

2. for each $(u, v) \in R$, for each $k \in K_u$, there exists $k' \in M_v$ such that $\alpha_{uk} = \beta_{vk'}$ and $(k, k') \in R$,

3. for each $k' \in M_v$, there exists $k \in K_u$ such that $\beta_{vk'} = \alpha_{uk}$ and $(k, k') \in R$.

Let us now consider the set of equations $\tilde{Z} = \tilde{F}$, defined for all $(u, v) \in \mathscr{R}$ by

$$Z_{u,v} = \sum_{j \in J_u} [W_j \backslash E_j]_{U_j} + \sum_{(k, l, m) \in K_{uv}} \alpha_{uvk} Z_{l,m}.$$

With

$$K_{uv} = \{(k, l, m) \mid \alpha_k X_l \text{ is a summand of } H_u \text{ and } \alpha_k Y_m \text{ is a summand of } G_v \text{ and } (l, m) \in \mathscr{R}\}.$$

Note that, since $(u, v) \in \mathscr{R}$, $H_u \sim_\pi G_v$ and therefore $J_u = L_v$.

Since P satisfies S, there is a set of expressions $P_1, P_2, ...$ such that $P_1 = P$ and $\tilde{X} = \tilde{H}[{}^{\tilde{P}}/_{\tilde{X}}]$. Now take the set of processes $R_{i,j} = P_i$. It is easy to see that the terms $F_{i,j}[{}^{\tilde{R}}/_{\tilde{Z}}]$ contains the same summands as $H_i[{}^{\tilde{P}}/_{\tilde{X}}]$ with some possible duplications. In particular, $F_{1,1}[{}^{\tilde{R}}/_{\tilde{Z}}] = P_1 = P$. Hence P satisfies this new set of equations. A similar reasoning can be applied to prove that Q satisfies this set of equations as well. ▌

6.6. *Unique Solution*

We now have to prove that if two processes satisfy the same set of prioritized equations, they are bisimilar. This is the objective of the following theorem.

THEOREM 6.4. *A set of prioritized standard equations has a unique solution up to a bisimulation.*

Proof. This proof exactly follows the proof given by Milner (1989a). It proceeds by induction on the number of equations. For one equation, $X = H$, we have the solution $P = rec\ X.H$. Moreover, if there is a process Q such that $Q = H[{}^{Q}/_{X}]$ then, by Rec(2), we have $Q = rec\ X.H$.

Assume it is true for n equations. Let S: $\tilde{X} = \tilde{H} \cup \{X_{n+1} = H_{n+1}\}$ be a system with $n+1$ equations. Consider the system of n equations S': $\tilde{X} = \tilde{H}[{}^{rec\ X_{n+1}.H_{n+1}}/_{X_{n+1}}]$; X_{n+1} does not occur free in S'. Therefore, by the induction hypothesis, there is a set of processes $\tilde{P}$ such that

$$\tilde{X} = \tilde{H}[{}^{rec\ X_{n+1}.H_{n+1}}/_{X_{n+1}}][{}^{\tilde{P}}/_{\tilde{X}}].$$

If we choose $P_{n+1} = rec\ X_{n+1}.H_{n+1}[{}^{\tilde{P}}/_{\tilde{X}}]$ we have found a solution to the system S.

For the uniqueness, suppose that we have a second solution $\tilde{Q} \cup \{Q_{n+1}\}$. That is,

$$\tilde{Q} = \tilde{H}[{}^{\tilde{Q},\ Q_{n+1}}/_{\tilde{X},\ X_{n+1}}]$$

$$Q_{n+1} = H_{n+1}[{}^{\tilde{Q},\ Q_{n+1}}/_{\tilde{X},\ X_{n+1}}]$$

The second equation can be written

$$Q_{n+1} = H_{n+1}[{}^{\tilde{Q}}/_{\tilde{X}}][{}^{Q_{n+1}}/_{X_{n+1}}]$$

and therefore, by Rec(2), we have $Q_{n+1} = rec\ X_{n+1}.\ H_{n+1}[{}^{\tilde{Q}}/_{\tilde{X}}]$, which can be rewritten as $Q_{n+1} = (rec\ X_{n+1}.H_{n+1})[{}^{\tilde{Q}}/_{\tilde{X}}]$. It follows that $Q_{n+1} = P_{n+1}$. By the induction hypothesis, it is easy to see that $\tilde{Q} = \tilde{P}$. ▌

6.7. *Completeness*

THEOREM 6.5. *For any two FS processes P and Q, if $P \sim_\pi Q$ then $P = Q$.*

Proof. By Theorem 6.1, there exists two processes, P' and Q', with no unguarded recursion, such that $P = P'$ and $Q = Q'$. By Theorem 6.2 and Theorem 6.3, P' and Q' satisfy a common set of prioritized equations S. And by Theorem 6.4, $P' = Q'$. ▮

7. CONCLUSIONS

We have described a timed process algebra called ACSR that supports the notions of resources, interrupt, and priorities. ACSR employs a synchronous semantics for resource-consuming actions that take time and an asynchronous semantics for events that are instantaneous. There is a single parallel operator that can be used to express both interleaving at the event level and lock-step parallelism at the action level.

ACSR's algebraic laws are derived from a term equivalence based on prioritized strong bisimulation, which incorporates a notion of preemption based on priority, synchronization and resource utilization. These laws can be used to rewrite process terms in proving the correctness of a real-time system. The set of laws is proved sound and complete for most finite state processes.

Traditionally, researchers in process algebra have proved the soundness of algebraic axioms by using the bisimulation definition as described by Milner (1989): one must first pick up a possible derivation of a process and show such a derivation can be found in the other process. Unlike Milner's method based on the bisimulation definition, our soundness proof is based on two functions, $\mathscr{T}$ and $\mathscr{T}_\pi$, such that given a ACSR process P, $\mathscr{T}(P)$ ($\mathscr{T}_\pi(P)$) is a set of pairs $\langle a, P' \rangle$, where $P \xrightarrow{a} P'$ ($P \xrightarrow{a}_\pi P'$). Furthermore, these are complete sets of derivations of a process. The main use of these functions is as follows: given two ACSR finite processes P and Q, if $\mathscr{T}(P)$ is the same as the $\mathscr{T}(Q)$, then $\mathscr{T}_\pi(P)$ is the same as the $\mathscr{T}_\pi(Q)$, and hence, P and Q are prioritized bisimilar. Hence, the soundness of an axiom $P = Q$ can be proved by showing that $\mathscr{T}(P) = \mathscr{T}(Q)$ or, if $\mathscr{T}(P) \neq \mathscr{T}(Q)$ then showing that $\mathscr{T}_\pi(P) = \mathscr{T}_\pi(Q)$. This approach simplifies the soundness proof, and seems to increase the degree of understandability for the intuitive meaning of the soundness proof.

The completeness proof described in this paper is based on the proof in (Milner, 1989, 1989a). The completeness proof of Milner consists the following four steps: (1) prove that unguarded recursions can be eliminated; (2) prove that any finite state process without unguarded recursion satisfies a certain kind of equations; (3) prove that if two processes are bisimilar, then they satisfy a common set of equations; (4) prove that those sets of equations have a unique solution up to bisimulation. Due to ACSR's operators, such as scope and resource closure, our complete axiomatization is for a larger subset of finite state processes than those considered by Milner (1989, 1989a).

There are three areas of research that should be explored to extend the capability of ACSR. The first extension is to support dynamic priorities. ACSR supports only static priority; i.e., the priorities of actions and events cannot change during the execution of a process. Since modeling of many real-time scheduling algorithms, such as earliest deadline first, first-come-first-served, *etc.*, requires dynamic priorities, it would be useful to support dynamic priority in timed process algebras. This requires some method to capture the state information and then use that information in reassigning priorities. A preliminary result on this extension is reported in (Choi, Lee, and Xie, 1995). The second extension is to allow dense time so that a timed action can take an arbitrary non-zero amount of time. The dense time version ACSR has been developed by Brémond-Grégoire (1994). The third extension is to specify the value of time using a variable and then derive the range of time values that ensure the correct timing of a process.

ACKNOWLEDGMENTS

The authors gratefully acknowledge the comments and suggestions made by the referees and Walter Vogler on improving the quality of this paper. The work described in this paper was strongly influenced by interaction with the members of the Real-Time Systems Group at the University of Pennsylvania.

Received November 22, 1995; final manuscript received April 11, 1997

REFERENCES

Baeten, J. C. M., and Bergstra, J. A. (1991), Real time process algebra, *Formal Aspects Comput.* **3**(2), 142–188.

Baeten, J. C. M., and Bergstra, J. A. (1992), Discrete time process algebra, *in* "CONCUR 92," Lecture Notes in Computer Science, Vol. 630, pp. 401–420, Springer-Verlag, Berlin.

Baeten, J., Bergstra, J., and Klop, J. (1987), Ready-trace semantics for concrete process algebra with a priority operator, *Comput. J.* **30**(6), 498–506.

Brémond-Grégoire, P. (1994), "A Process Algebra of Communicating Shared Resources with Dense Time and Priorities," Ph.D. thesis, Department of Computer and Information Science, The University of Pennsylvania.

Bergstra, J. A., and Klop, J. W. (1985), Algebra of communicating processes with abstraction, *J. Theoret. Comput. Sci.* **37**, 77–121.

Bolognesi, T., and Smolka, S. (1987), Fundamental results for verification of observational equivalence, *in* "Protocol Specification, Testing and Verification."

Cleaveland, R., and Hennessy, M. (1990), Priorities in process algebras, *Inform. and Comput.* **87**, 58–77.

Choi, J.-Y., Lee, I., and Xie, H.-L. (1995), The specification and schedulability analysis of real-time systems using ACSR, *in* "Proceedings, 16th IEEE Real-Time Systems Symposium."

Camilleri, J., and Winskel, G. (1991), CCS with Priority Choice, *in* "Proceedings, 6th IEEE Symposium on Logic in Computer Science."

Davies, J., and Schneider, S. (1995), A brief history of timed CSP, *Theoret. Comput. Sci.* **138**, 243–271.

Gerber, R. (1991) "Communicating Shared Resources: A Model for Distributed Real-Time Systems," Ph.D. thesis, Department of Computer and Information Science, University of Pennsylvania.

Gerber, R., and Lee, I. (1994), A resource-based prioritized bisimulation for real-time systems, *Inform. and Comput.* **113**(1), 102–142.

Hennessy, M. (1988), "Algebraic theory of processes," MIT Press Series in the Foundations of Computing, MIT Press.

Hennessy, M., and Regan, T. (1990), "A Temporal Process Algebra," Technical Report 2/90, University of Sussex.

Hoare, C. (1985), "Communicating Sequential Processes," Prentice–Hall, Englewood Cliffs, NJ.

Klusener, A. S. (1991), Completeness in real time process algebra, *in* "Proceedings, CONCUR 91," Lecture Notes in Computer Science, Vol. 715, pp. 376–392, Springer-Verlag, Berlin.

Lee, I., Brémond-Grégoire, P., and Gerber, R. (1994), A process algebraic approach to the specification and analysis of resource-bound real-time systems, *Proc. IEEE*, 158–171.

Lee, I., and Gehlot, V. (1985), Language constructs for distributed real-time programming, *in* "Proceedings, 5th IEEE Real-Time Systems Symposium."

Milner, R. (1980), "A Calculus for Communicating Systems," Lecture Notes in Computer Science, Vol. 92, Springer-Verlag, Berlin.

Milner, R. (1983), Calculi for synchrony and asynchrony, *Theoret. Comput. Sci.* **25**, 267–310.

Milner, R. (1984), A complete inference system for a class of regular behaviours, *J. Comput. System Sci.* **28**, 439–466.

Milner, R. (1989), "Communication and Concurrency," Prentice–Hall, New York.

Milner, R. (1989a), A complete axiomatisation for observational congruence of finite-state behaviors, *Inform. and Comput.* **81**, 227–247.

Moller, F., and Tofts, C. (1990), A temporal calculus of communicating systems, *in* "Proceedings, CONCUR '90," Lecture Notes in Computer Science, Vol. 458, pp. 401–415, Springer-Verlag, Berlin.

Nicollin, X., and Sifakis, J. (1994), The algebra of timed processes ATP: Theory and application, *Inform. and Comput.* **114**(1), 131–178.

Park, D. (1981), Concurrency and automata on infinite sequences, *in* "Proceedings, 5th GI Conference," Lecture Notes in Computer Science, Vol. 104, Springer-Verlag, Berlin.

Yi, W. (1991), CCS + time = An interleaving model for real time systems, *in* "Proceedings, International Conference on Automata, Languages and Programming."