The Soundness and Completeness of ACSR
(Algebra of Communicating Shared Resources)

MS-CIS-93-59
GRASP LAB 350

Patrice Brémond-Grégoire
Jin-Young Choi
Insup Lee

University of Pennsylvania
School of Engineering and Applied Science
Computer and Information Science Department
Philadelphia, PA 19104-6389

June 1993
The Soundness and Completeness of ACSR (Algebra of Communicating Share Resources) *

Patrice Brémond-Grégoire, Jin-Young Choi, Insup Lee
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104-6389
(Revised on November 1993)

Abstract

Recently, significant progress has been made in the development of timed process algebras for the specification and analysis of real-time systems. This paper describes a timed process algebra called ACSR. 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 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. Equivalence between ACSR terms is defined in terms of strong bisimulation. The paper contains a set of algebraic laws that can be used to prove equivalence of ACSR processes. The main contribution of the paper is a proof of soundness and completeness of this set of laws for finite state ACSR agents.

*This research was supported in part by ONR N00014-89-J-1131, DARPA/NSF CCR90-14621 and NSF CCR 92-09333.
1 Introduction

Reliability in real-time systems can be improved through the use of formal methods for their specification and analysis. Formal methods treat system components as mathematical objects and provide mathematical models to describe and predict the observable properties and behaviors of these objects. There are several advantages to using formal methods for the specification and analysis of real-time systems. They are, firstly, the early discovery of ambiguities, inconsistencies and incompleteness in informal requirements; secondly, the automatic or machine-assisted analysis of the correctness of specifications with respect to requirements; and thirdly, the evaluation of design alternatives without expensive prototyping.

Process algebras, such as CCS [10], CSP [5], Acceptance Trees [4] and ACP [1], have been developed to describe and analyze communicating, concurrently executing systems. They are based on the premises that the two most essential notions in understanding complex dynamic systems are concurrency and communication [10]. 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. 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.

The timing behavior of a real-time system depends not only on delays due to process synchronization, but also on the availability of shared resources. Most current real-time process algebras adequately capture delays due to process synchronization; however, they abstract out resource-specific details by assuming idealistic operating environments. On the other hand, scheduling and resource allocation algorithms used for real-time systems ignore the effect of process synchronization except for simple precedence relations between processes. Our algebra provides a formal framework that combines the areas of process algebra and real-time scheduling, and thus, can help us to reason about systems that are sensitive to deadlines, process interaction and resource availability.

The computation model of ACSR is based on the view that a real-time system consists of a set of communicating processes that use shared resources for execution and synchronize with one another. The use of shared resources is represented by timed actions and synchronization is supported by instantaneous events. The execution of a timed action is assumed to take one time unit and to consume a set of resources during that time. Idling of a process is treated as a special timed action that consumes no resources. The execution of a timed action is subject to the availability of the 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. Unlike a timed action, the execution of an event is instantaneous and never consumes any resource. Processes execute events asynchronously except when two processes synchronize through matching events. Priorities are used to direct the choice when several events are possible at the same time.

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 Sect. 6 by a proof of completeness for a syntactically characterized subset of the finite state agents. We conclude in Section 7 by discussing possible extensions to this work.

2 The Computation Model

The executions of a process are defined by a labelled transition system. For example, a process \( P_1 \) may have the following behavior:

\[
P_1 \xrightarrow{\alpha_1} P_2 \xrightarrow{\alpha_2} P_3 \xrightarrow{\alpha_3} \ldots
\]

That is, \( P_1 \) first executes \( \alpha_1 \) and evolves into \( P_2 \), which executes \( \alpha_2 \), etc. \( P_i \) represents the process’s state at the \( i \)th step of an execution, while \( \alpha_i \) represents the \( i \)th step, or action taken in the execution. This is a common – almost generic – way of describing a process’s behavior. In a process algebra, however, the states \( P_i \) are typically described by a concrete syntax, i.e., a language. Further, there is a finite set of transition rules which infer the stepwise behavior of the process.\(^1\)

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.

This dual-approach is motivated by the behavior of concurrent processes written in Ada and related languages. That is, an instantaneous event can be interpreted as an abstraction for the point-to-point handshaking that takes place between two tasks (e.g., the instant when a server accepts a select guard). After this synchronization point, a

\(^1\)The technique of a structured transition system is not limited to process algebras; e.g. see [12].
sequence of time-consuming actions is used to model the task’s internal resource requirements.

As we show in this section, the two classes of actions have separate priority orderings. The reason for this follows from the two roles that priority can play in a real-time system. First, there is the type of priority that comes “from above,” i.e., from a specification. This type is used to “break a tie” between two competing services, and is modeled in ACSR by priority on instantaneous events. (In Ada this is called preference control between guards, whereas in Occam the PRI ALT statement is provided for a similar purpose.) The other type of priority is injected “from below,” by the system’s real-time scheduler. Naturally, this is modeled by a priority relation on time-consuming actions.

**Timed Actions.** We consider a system to be composed of a finite set of serially reusable resources, denoted by \( \mathcal{R} \). An action that consumes one “tick” of time is drawn from the domain \( \mathcal{P}(\mathcal{R} \times \mathbb{N}) \), with the restriction that each resource be represented at most once. As an example, the singleton action, \( \{(r,p)\} \), denotes the use of some resource \( r \in \mathcal{R} \) running at the priority level \( p \). The action \( \emptyset \) represents idling for one time unit, since all resources are inactive.

We use \( \mathcal{D}_R \) to denote the domain of timed actions, and we let \( A, B, C \) range over \( \mathcal{D}_R \). 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 \( \mathcal{L} \cup \overline{\mathcal{L}} \cup \{\tau\} \), where if \( a \) is a given label, we say that \( \overline{a} \) is its inverse label; i.e., \( \overline{\overline{a}} = a \). As in CCS, the special identity label, \( \tau \), arises when two events with inverse labels are executed in parallel.

We use \( \mathcal{D}_E \) to denote the domain of events, and let \( e, f \) and \( g \) range over \( \mathcal{D}_E \). 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 \( \mathcal{D} = \mathcal{D}_R \cup \mathcal{D}_E \), and we let \( \alpha \) and \( \beta \) range over \( \mathcal{D} \).

### 3 The Syntax and Operational Semantics

The following grammar describes the syntax of processes:

\[
P ::= \text{NIL} \mid A.P \mid e.P \mid P + P \mid P|P \mid P \triangleleft_t (P,P,P) \mid [P]_f \mid P\setminus F \mid \text{rec } X.P \mid X
\]
NIL is a process that executes no action (i.e., it is initially deadlocked). There are two prefix operators, corresponding to the two types of actions. The first, \( A: P \), executes a resource-consuming action \( A \) at the first time unit, and proceeds to the process \( P \). On the other hand, \( e. P \), executes the instantaneous event \( e \), and proceeds to \( P \). The difference here is that we consider no time to pass during the event occurrence. 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, \( \alpha P \) stands for \( \alpha : P \) when \( \alpha \in \mathcal{D}_R \) and for \( \alpha. P \) when \( \alpha \in \mathcal{D}_E \). The Choice operator \( P + Q \) represents nondeterminism – either of the processes may be chosen to execute, subject to the constraints of the environment. The operator \( P\parallel Q \) is the parallel composition of \( P \) and \( Q \).

The Scope construct \( P \Delta^t \langle Q, R, S \rangle \) binds the process \( P \) by a temporal scope [6], 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 scope may be exited in a number of ways. First, if \( P \) successfully terminates within time \( t \) by executing an event labelled with \( a \), then control proceeds to the “success-handler” \( Q \) (here, \( a \) may be any label other than \( \tau \).) On the other hand, \( R \) is a timeout exception-handler; that is, if \( P \) fails to terminate within time \( t \), then control proceeds to \( R \). Lastly, at any time while \( P \) is executing it may be interrupted by \( S \), and the scope is then departed.

The Close operator, \( [P]_I \), produces a process \( P \) that monopolizes the resources in \( I \subseteq \mathcal{R} \). The Restriction operator, \( P \setminus F \), limits the behavior of \( P \). Here, no events with labels in \( F \) are permitted to execute. The process \( \text{rec} \ X. P \) denotes standard recursion, allowing the specification of infinite behaviors.

The semantics is defined in two steps. First, we develop the unconstrained transition system, where a transition is denoted as \( P \xrightarrow{\cdot} P' \). Within \( \xrightarrow{\cdot} \) no priority arbitration is made between actions; rather, we subsequently refine \( \xrightarrow{\cdot} \) to define our prioritized transition system, \( \xrightarrow{\pi} \).

### 3.1 The Structured Transition System

The two rules for the prefix operators are axioms; i.e., they have premises of true. There is one rule for a time-consuming action, and one for an instantaneous action.

\[
\begin{align*}
\text{ActT} & \quad \frac{}{A: P \xrightarrow{A} P} \quad \text{ActI} & \quad \frac{}{e. P \xrightarrow{\tau} P}
\end{align*}
\]

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 \( P \).
The rules for Choice are identical for both timed actions and instantaneous events (and hence we use “α” as the label).

\[
\begin{align*}
\text{Choice}_L & \quad \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \\
\text{Choice}_R & \quad \frac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'}
\end{align*}
\]

As an example, \((a, 7). P + \{(r_1, 3), (r_2, 7)\}: Q\) may choose between executing the event \((a, 7)\) or the time-consuming action \(\{(r_1, 3), (r_2, 7)\}\). The former behavior is deduced from rule \text{Act}_I, while the latter is deduced from \text{Act}_T.

The Parallel operator provides the basic constructor for concurrency and communication. The first rule, \text{Par}_T, is for two time-consuming transitions.

\[
\text{Par}_T \quad \frac{P \xrightarrow{A_1} P', Q \xrightarrow{A_2} Q'}{P || Q \xrightarrow{A_1 \cup A_2} P' || Q'} \quad (\rho(A_1) \cap \rho(A_2) = \emptyset)
\]

Note that timed transitions are truly synchronous, in that the resulting process advances only if both of the constituents take a step. The condition \(\rho(A_1) \cap \rho(A_2) = \emptyset\) mandates that each resource be truly sequential, and that only one process may use a given resource during any time step.

The next three laws are for event transitions. As opposed to timed actions, events may occur asynchronously (as in CCS and related interleaving models).

\[
\begin{align*}
\text{Par}_I& \quad \frac{P \xrightarrow{(a,n)} P'}{P || Q \xrightarrow{(a,n)} P' || Q'} \\
\text{Par}_R & \quad \frac{Q \xrightarrow{(a,n)} Q'}{P || Q \xrightarrow{(a,m)} P' || Q'} \quad (\tau, n + m)
\end{align*}
\]

The first two rules show that events may be arbitrarily interleaved. The last rule is for two synchronizing processes; that is, \(P\) executes an event with the label \(\bar{a}\), 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.

\textbf{Example 3.1} Consider the following two processes:

\[
\begin{align*}
P & \overset{\text{def}}{=} ((a,3).P_1) + \{(r_3, 8)\}:P_2 \\
Q & \overset{\text{def}}{=} ((\bar{a},5).Q_1) + \{(r_1, 7)\}:Q_2
\end{align*}
\]

The compound process \(P || Q\) admits the following four transitions:

\[
\begin{align*}
P || Q \xrightarrow{(a,3)} P_1 || Q_1 & \quad \text{[by ParIL]} \\
P || Q \xrightarrow{(\bar{a},5)} P || Q_1 & \quad \text{[by ParIR]} \\
P || Q \xrightarrow{(r,8)} P_1 || Q_1 & \quad \text{[by ParCom]} \\
P || Q \xrightarrow{(r_1,7),(r_3,8)} P_2 || Q_2 & \quad \text{[by ParT]}
\end{align*}
\]
Note than an event transition always executes before the next “tick” of the global clock. □

The construction of \textbf{ParCom} helps ensure that the relative priority ordering among events with the same labels remains consistent even after communication takes places. The following example shows how the ordering is preserved.

\textbf{Example 3.2} Consider the following two processes.

\[
P \overset{\text{def}}{=} (a, 2).P_1 + (a, 3).P_2 \\
Q \overset{\text{def}}{=} (a, 5).Q_1 + (a, 3).Q_2
\]

Thus, in \(P\) the second choice is preferred, while in \(Q\) the first choice is preferred. There are eight possible transitions for \(P \parallel Q\):

\[
\begin{align*}
P \parallel Q \xrightarrow{(a, 2)} P_1 \parallel Q_1 & \quad P \parallel Q \xrightarrow{(a, 3)} P_2 \parallel Q_2 \\
P \parallel Q \xrightarrow{(a, 5)} P_1 \parallel Q_1 & \quad P \parallel Q \xrightarrow{(a, 3)} P_2 \parallel Q_2 \\
P \parallel Q \xrightarrow{(\tau, 7)} P_1 \parallel Q_1 & \quad P \parallel Q \xrightarrow{(\tau, 5)} P_1 \parallel Q_2 \\
P \parallel Q \xrightarrow{(\tau, 8)} P_2 \parallel Q_1 & \quad P \parallel Q \xrightarrow{(\tau, 6)} P_2 \parallel Q_2
\end{align*}
\]

While there are now four possible transitions labelled with \(\tau\), the addition of priorities in \textbf{ParCom} ensures that the original relative orderings are maintained. Note that the \(\tau\)-transition with the highest priority is that associated with the derivative \(P_2 \parallel Q_1\). These transitions had the highest priorities in their original constituent processes. □

The \textbf{Scope} operator possesses a total of five transition rules, which describe the various behaviors induced by a temporal scope. The first two rules show that as long as \(t > 0\) and \(P\) fails to execute an event labelled with \(b\), the executions of \(P\) continue.

\[
\begin{align*}
\textbf{ScopeCT} & \quad \frac{P \xrightarrow{A} P'}{P \Delta^b_t(Q, R, S) \xrightarrow{A} P' \Delta^b_{t-1}(Q, R, S)} \quad (t > 0) \\
\textbf{ScopeCI} & \quad \frac{P \xrightarrow{\varepsilon} P'}{P \Delta^b_t(Q, R, S) \xrightarrow{\varepsilon} P' \Delta^b_t(Q, R, S)} \quad (l(e) \neq \bar{b}, t > 0)
\end{align*}
\]

The \textbf{ScopeE} (for “end”) shows how \(P\) can depart the temporal scope by executing an event labelled with \(\bar{b}\). Upon exit, the label \(\bar{b}\) is converted to the identity label \(\tau\) (however, the same priority is retained).

\[
\begin{align*}
\textbf{ScopeE} & \quad \frac{P \xrightarrow{(\bar{b}, n)} P'}{P \Delta^b_t(Q, R, S) \xrightarrow{(\tau, n)} Q} \quad (t > 0)
\end{align*}
\]
The next rule, **ScopeT** (for “timeout”), is applied whenever the scope times out; that is, when \( t = 0 \). At this point, control proceeds to the exception-handler \( R \).

\[
\text{ScopeT} \quad \frac{R \xrightarrow{a} R'}{P \triangle^b_t (Q, R, S) \xrightarrow{a} R'} \quad (t = 0)
\]

Finally, **ScopeI** shows that the process \( S \) may interrupt (and kill) \( P \) while the scope is still active.

\[
\text{ScopeI} \quad \frac{S \xrightarrow{a} S'}{P \triangle^b_t (Q, R, S) \xrightarrow{a} S'} \quad (t > 0)
\]

**Example 3.3** Consider the following specification: “Execute \( P \) for a maximum of 100 time units. If \( P \) executes an event labelled with \( \bar{b} \) in that time, then stop the system. However, if \( P \) fails to finish within 100 time units, then start executing \( R \). At any time during the execution of \( P \), allow interruption by an event \((c, 3)\), which will halt \( P \), and initiate the interrupt-handler \( S \).” This system may be realized by the following process: \( P \triangle^b_{100} (\text{NIL}, R, (c, 3).S) \).

The Restriction operator defines a subset of instantaneous events that are excluded from the behavior of the system. 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.

\[
\text{ResT} \quad \frac{P \xrightarrow{A} P'}{P \setminus F \xrightarrow{A} P' \setminus F}
\]

\[
\text{ResI} \quad \frac{P \xrightarrow{(a,n)} P'}{P \setminus F \xrightarrow{(a,n)} P' \setminus F} \quad (a, \bar{a} \notin F)
\]

**Example 3.4** Restriction is particularly useful in “forcing” the synchronization between concurrent processes. In Example 3.1, synchronization on \( a \) and \( \bar{a} \) is not forced, since \( P \parallel Q \) has transitions labelled with \( a \) and \( \bar{a} \). On the other hand, \((P \parallel Q) \setminus \{a\}\) has only two transitions:

\[
(P \parallel Q) \setminus \{a\} \xrightarrow{(r,8)} (P_1 \parallel Q_1) \setminus \{a\} \quad \text{and} \quad (P \parallel Q) \setminus \{a\} \xrightarrow{(r_1,7), (r_3,8)} (P_2 \parallel Q_2) \setminus \{a\}
\]

In effect, the restriction declares that \( a \) and \( \bar{a} \) define a “dedicated channel” between \( P \) and \( Q \).

While Restriction assigns dedicated channels to processes, the Close operator assigns dedicated resources. 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 either 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.)

\[
\text{CloseT} \quad \frac{P \xrightarrow{A_1} P'}{[P]_I \xrightarrow{A_1 \cup A_2} [P']_I} \quad (A_2 = \{(r, 0) \mid r \in I - \rho(A_1)\})
\]

\[
\text{CloseI} \quad \frac{P \xrightarrow{e} P'}{[P]_I \xrightarrow{e} [P']_I}
\]

The operator \( \text{rec } X.P \) denotes recursion, allowing the specification of infinite behaviors.

\[
\text{Rec} \quad \frac{P[\text{rec } X.P/X] \xrightarrow{\alpha} P'}{\text{rec } X.P \xrightarrow{\alpha} P'}
\]

where \( P[\text{rec } X.P/X] \) is the standard notation for substitution of \( \text{rec } X.P \) for each free occurrence of \( X \) in \( P \).

For instance, the process \( \text{rec } X.(A:X) \) indefinitely executes the resource-consuming action "A." By \( \text{ActT} \),

\[
A:(\text{rec } X.(A:X)) \xrightarrow{A} \text{rec } X.(A:X)
\]

, so by \( \text{Rec} \),

\[
\text{rec } X.(A:X) \xrightarrow{A} \text{rec } X.(A:X)
\]

### 3.2 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 straightforward. Let "\(<"", called the preemption relation, be a transitive, irreflexive, binary relation on actions. Then for two actions \( \alpha \) and \( \beta \), if \( \alpha \prec \beta \), we can say that "\( \alpha \) is preempted by \( \beta \)." This means that in any real-time system, if there is a choice between executing either \( \alpha \) or \( \beta \), it will never execute \( \alpha \).

**Definition 3.1 (Preemption Relation)** For two actions, \( \alpha, \beta \), we say that \( \beta \) preempts \( \alpha \) (\( \alpha \prec \beta \)), if one of the following cases hold:

1. Both \( \alpha \) and \( \beta \) are timed actions in \( \mathcal{D}_R \), where

\[
(\rho(\beta) \subseteq \rho(\alpha)) \land (\forall r \in \rho(\alpha).\pi_r(\alpha) \leq \pi_r(\beta)) \land (\exists r \in \rho(\beta).\pi_r(\alpha) < \pi_r(\beta))
\]

2. Both \( \alpha \) and \( \beta \) are events in \( \mathcal{D}_E \), where \( \pi(\alpha) < \pi(\beta) \land l(\alpha) = l(\beta) \)
(3) $\alpha \in \mathcal{D}_R$ and $\beta \in \mathcal{D}_E$, with $l(\beta) = \tau$ and $\pi(\beta) > 0$.

Case (1) shows that the two timed actions, $\alpha$ and $\beta$, compete for common resources, and in fact, the preempted action $\alpha$ may use a superset of $\beta$'s resources. However, $\beta$ uses all the resources at least the same priority level as $\alpha$ (recall that $\pi_r(B)$ is, by convention, 0 when $r$ is not in $B$). Also, $\beta$ uses at least one resource at a higher level.

Case (2) shows that an event may be preempted by another event sharing the same label, but with a higher priority.

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 $(\tau, n)$, we let the event preempt any timed action.

**Example 3.5** The following examples show some comparisons made by the preemption relation, "$\prec$.”

- a. $\{(r_1, 2), (r_2, 5)\} \prec \{(r_1, 7), (r_2, 5)\}$
- b. $\{(r_1, 2), (r_2, 5)\} \not\prec \{(r_1, 7), (r_2, 3)\}$
- c. $\{(r_1, 2), (r_2, 0)\} \prec \{(r_1, 7)\}$
- d. $\{(r_1, 2), (r_2, 1)\} \not\prec \{(r_1, 7)\}$
- e. $(\tau, 1) \prec (\tau, 2)$
- f. $(a, 1) \not\prec (b, 2)$ if $a \neq b$
- g. $(a, 2) \prec (a, 5)$
- h. $\{(r_1, 2), (r_2, 5)\} \prec (\tau, 2)$

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 $\alpha \prec \beta$.

## 4 Bisimulation and Strong Equivalence

Equivalence between two ACSR processes is based on the concept of strong bisimulation [11], which compares the computation trees of the two processes.

**Definition 4.1** For a given transition system "$\simto$", any binary relation $r$ is a strong bisimulation if, for $(P, Q) \in r$ and $\alpha \in \mathcal{D}$,
1. if $P \leadsto P'$ then, for some $Q', Q \leadsto Q'$ and $(P', Q') \in r$, and
2. if $Q \leadsto Q'$ then, for some $P', P \leadsto P'$ and $(P', Q') \in r$.

In other words, if $P$ (or $Q$) can take a step on $\alpha$, then $Q$ (or $P$) must also be able to take a step on $\alpha$ with both of the next states also bisimilar. There are some very obvious bisimulation relations; e.g. $\emptyset$ (which certainly adheres to the above rules) or syntactic identity. However, using the theory found in [7, 8, 10], 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 [3]. Similarly, “$\sim_{\pi}$” is the largest strong bisimulation over “$\rightarrow_{\pi}$,” and we call it prioritized strong equivalence.

4.1 Laws

Table 1 presents a set of equivalence-preserving laws for ACSR, $A$. In the sequel, wherever we use the equality symbol “$=$” in showing that two processes are equivalent, it means that we have used the laws $A$ 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 $\Sigma$ 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 $I = \{i_1, \ldots, i_n\}$, because of Choice(4) we are able to neglect parentheses and use the following notation:

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

and where $\sum_{i \in \emptyset} P_i \stackrel{\text{def}}{=} NIL$.

5 Soundness of the Laws

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

$$T(P) = \{(\alpha, P') \mid P \xrightarrow{\alpha} P'\} \quad \text{and} \quad \pi(P) = \{(\alpha, P') \mid P \xrightarrow{\pi} P'\}$$

and of the following two lemmas.

Lemma 5.1

$$T(P) = T(Q) \implies \pi(P) = \pi(Q) \implies P \sim_{\pi} Q$$
Table 1: The Set of ACSR Laws, \( \mathcal{A} \)
Proof: It follows from the definition of the prioritized transition system that $T_\pi(P)$ can be calculated from $T(P)$:

$$T_\pi(P) = \{ (\alpha, P') \in T(P) \mid \exists (\beta, Q). \alpha \prec \beta \}.$$ 

And therefore $T(P) = T(Q) \implies T_\pi(P) = T_\pi(Q)$.

From the definition of $\pi$, we have:

$$\forall P, Q: P \rightarrow \pi Q \implies T_\pi(P) = T_\pi(Q).$$

The identity being a bisimulation, we conclude that $P \sim_\pi Q$. 

Lemma 5.2 If $\mathcal{R}$ is a relation such that all the pairs $(P, Q) \in \mathcal{R}$ are such that

$$\forall (\alpha, P') \in T_\pi(P): \exists Q' : (\alpha, Q') \in T_\pi(Q) \land (P', Q') \in \mathcal{R}$$

and

$$\forall (\alpha, Q') \in T_\pi(Q): \exists P' : (\alpha, P') \in T_\pi(P) \land (P', Q') \in \mathcal{R}$$

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

Proof: Follows directly from the definitions of the strong bisimulation and of the functions $T$ and $T_\pi$. 

In order to prove most of the laws, we apply the appropriate formula to calculate either $T$ or $T_\pi$ for the processes in both sides and verify that the resulting sets are equal or related in a way that satisfies the lemma 5.2.

Since the behavior of a process must be derived from the rules of the operational semantics, for any process $P$ the set $T(P)$ 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
\[ T(NIL) = \emptyset \]
\[ T(\alpha P) = \{ (\alpha, P) \} \]  \hspace{1cm} \text{[ActT and ActI]} \]
\[ T(P + Q) = T(P) \cup T(Q) \]  \hspace{1cm} \text{[ChoiceL and ChoiceR]} \]
\[ T(P|Q) = \{(A \cup B, P'|Q') \mid (A, P') \in T(P) \land (B, Q') \in T(Q) \land \rho(A) \cap \rho(B) = \emptyset\} \]  \hspace{1cm} \text{[ParT]} \]
\[ \cup \{(e, P'|Q') \mid e, P' \in T(P)\} \]  \hspace{1cm} \text{[ParIL]} \]
\[ \cup \{(e, P'|Q') \mid e, Q' \in T(Q)\} \]  \hspace{1cm} \text{[ParIR]} \]
\[ \cup \{(\tau, n + m), P'|Q') \mid ((a, n), P') \in T(P) \land ((a, m), Q') \in T(Q)\} \]  \hspace{1cm} \text{[ParCom]} \]
\[ T(P_{\Delta 0}^l(Q, R, S)) = T(R) \]  \hspace{1cm} \text{[ScopeT]} \]
\[ T(P_{\Delta 0}^{\geq 1}(Q, R, S)) = \{(A, P') \Delta_{i-1}^l (Q, R, S)) \mid (A, P') \in T(P)\} \]  \hspace{1cm} \text{[ScopeCT]} \]
\[ \cup \{(e, P' \Delta_{i}^l(Q, R, S)) \mid e, P' \in T(P) \land l(e) \neq \bar{b}\} \]  \hspace{1cm} \text{[ScopeCI]} \]
\[ \cup \{(\tau, n), Q) \mid ((b, n), P') \in T(P)\} \]  \hspace{1cm} \text{[ScopeE]} \]
\[ \cup T(S) \]  \hspace{1cm} \text{[ScopeP]} \]
\[ T(P \setminus F) = \{(A, P' \setminus F) \mid (A, P') \in T(P)\} \]  \hspace{1cm} \text{[ResT]} \]
\[ \cup \{(a, n), P' \setminus F) \mid (a, n), P' \in T(P) \land a, \bar{a} \notin F\} \]  \hspace{1cm} \text{[ResI]} \]
\[ T([P]_I) = \{(A \cup B, [P']_I) \mid (A, P') \in T(P) \land B = \{(r, 0) \mid r \in I - \rho(A)\}\} \]  \hspace{1cm} \text{[CloseT]} \]
\[ \cup \{(e, [P']_I) \mid e, P' \in T(P)\} \]  \hspace{1cm} \text{[CloseI]} \]
\[ T(\text{rec } X.P) = T(P[\text{rec } X.P/X]) \]  \hspace{1cm} \text{[Rec]} \]

**Choice(1)** \[ T(P + NIL) = T(P) \cup T(NIL) = T(P) \cup \emptyset = T(P) \]

**Choice(2)** \[ T(P + P) = T(P) \cup T(P) = T(P) \]

**Choice(3)** \[ T(P + Q) = T(P) \cup T(Q) = T(Q) \cup T(P) = T(Q + P) \]

**Choice(4)** \[ T((P + Q) + R) = T(P + Q) \cup T(R) = (T(P) \cup T(Q)) \cup T(R) = T(P) \cup (T(Q) \cup T(R)) = T(P + (Q + R)) \]

**Choice(5)** \[ T(\alpha P + \beta Q) = \{(\alpha, P), (\beta, Q)\} \text{ and therefore we have } T_\pi(\alpha P + \beta Q) = \{(\beta, Q)\} = T_\pi(\beta Q) \]

**Par(1)**
\[ T(P|Q) = \{(A \cup B, P'|Q') \mid (A, P') \in T(P) \land (B, Q') \in T(Q) \land \rho(A) \cap \rho(B) = \emptyset\} \]
\[ \cup \{(e, P'|Q') \mid e, P' \in T(P)\} \]
\[ \cup \{(e, P'|Q') \mid e, Q' \in T(Q)\} \]
\[ \cup \{(\tau, n + m), P'|Q') \mid ((a, n), P') \in T(P) \land ((a, m), Q') \in T(Q)\} \]
\[ T(Q\|P) = \{ (B \cup A, Q'\|P') \mid (B, Q') \in T(Q) \land (A, P') \in T(P) \land \rho(B) \cap \rho(A) = \emptyset \} \]
\[ \cup \{ (e, Q'\|P) \mid (e, Q') \in T(Q) \} \]
\[ \cup \{ (e, Q\|P') \mid (e, P') \in T(P) \} \]
\[ \cup \{ ((\tau, n + m), Q'\|P') \mid ((a, n), Q') \in T(Q) \land ((a, m), P') \in T(P) \} \]

It follows by application of Lemma 5.2 that the relation \{(X\|Y), (Y\|X)\} is a bisimulation.

\[ \square \]

**Par(2)**

\[ T(P\|Q) = \]
\[ \{ (A \cup B, P'\|Q') \mid (A, P') \in T(P) \land (B, Q') \in T(Q) \land \rho(A) \cap \rho(B) = \emptyset \} \]
\[ \cup \{ (e, P'\|Q) \mid (e, P') \in T(P) \} \]
\[ \cup \{ (e, P\|Q') \mid (e, Q') \in T(Q) \} \]
\[ \cup \{ ((\tau, n + m), P'\|Q') \mid ((a, n), P') \in T(P) \land ((\bar{a}, m), Q') \in T(Q) \} \]
\[ \cup \{ ((\tau, n + m), Q\|P') \mid ((a, n), Q') \in T(Q) \land ((\bar{a}, m), P') \in T(P) \} \]

It follows that

\[ T((P\|Q)\|R) = \]
\[ \{ ((A \cup B) \cup C, (P'\|Q')\|R') \mid (A, P') \in T(P) \land (B, Q') \in T(Q) \land (C, R') \in T(R) \]
\[ \land \rho(A) \cap \rho(B) = \emptyset \land \rho(A \cup B) \cap \rho(C) = \emptyset \} \]
\[ \cup \{ (e, (P'\|Q)\|R) \mid (e, P') \in T(P) \} \]
\[ \cup \{ (e, (P\|Q')\|R') \mid (e, Q') \in T(Q) \} \]
\[ \cup \{ ((\tau, n + m), (P'\|Q')\|R) \mid ((a, n), P') \in T(P) \land ((\bar{a}, m), Q') \in T(Q) \} \]
\[ \cup \{ ((\tau, n + m), (P\|Q')\|R') \mid ((a, n), Q') \in T(Q) \land ((\bar{a}, m), P') \in T(P) \} \]

Similarly,

\[ T(P\|Q\|R) = \]
\[ \{ ((A \cup (B \cup C), P'\|Q'\|R') \mid (A, P') \in T(P) \land (B, Q') \in T(Q) \land (C, R') \in T(R) \]
\[ \land \rho(A) \cap (\rho(B \cup C)) = \emptyset \land \rho(B) \cap \rho(C) = \emptyset \} \]
\[ \cup \{ (e, P'(\|Q\|R)) \mid (e, P') \in T(P) \} \]
\[ \cup \{ (e, P\|Q'(\|R)) \mid (e, Q') \in T(Q) \} \]
\[ \cup \{ ((\tau, n + m), P'(\|Q\|R)) \mid ((a, n), P') \in T(P) \land ((\bar{a}, m), Q') \in T(Q) \} \]
\[ \cup \{ ((\tau, n + m), P\|Q'(\|R)) \mid ((a, n), Q') \in T(Q) \land ((\bar{a}, m), P') \in T(P) \} \]

However, we know that \( \rho(A \cup B) = \rho(A) \cup \rho(B) \). In addition, if follows from the properties of sets that \( (A \cup (B \cup C)) = ((A \cup B) \cup C) \) and that

\[ \rho(A) \cap (\rho(B) \cup \rho(C)) = \emptyset \iff \rho(A) \cap \rho(B) = \emptyset \land \rho(A) \cap \rho(C) = \emptyset \]
and similarly:

$$(\rho(A) \cup \rho(B)) \cap \rho(C) = \emptyset \iff \rho(A) \cap \rho(C) = \emptyset \land \rho(B) \cap \rho(C) = \emptyset.$$  

This proves that

$$\langle \alpha, (P'\parallel Q')\parallel R' \rangle \in T((P\parallel Q)\parallel R) \iff \langle \alpha, P'\parallel(Q'\parallel R') \rangle \in T(P\parallel(Q\parallel R)).$$

It follows by application of Lemma 5.2 that the relation $$\{((X\parallel Y)\parallel Z),(X\parallel(Y\parallel Z))\}$$ is a bisimulation.

\textbf{Par(3)}  

Let us call:

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

$$Q = \sum_{i \in J} e_i Q_j$$

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

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

We can calculate:

$$T(P + Q\parallel R + S) = \left\{ (A_i \cup B_k, P_i\parallel Q_k) \mid i \in I \land k \in K \land \rho(A_i) \cap \rho(B_k) = \emptyset \right\}$$

$$\cup \{ (e_j, Q_j\parallel(R + S)) \mid j \in J \}$$

$$\cup \{ (f_l, (P + Q)\parallel S_l) \mid l \in L \}$$

$$\cup \{ ((\tau, \pi(e_j) + \pi(f_l)), Q_j\parallel S_l) \mid j \in J \land l \in L \land l(e_j) = l(f_l) \}$$

$$\begin{align*}
T(P + Q\parallel R + S) &= \left\{ (A_i \cup B_k : (P_i\parallel R_k) \right. \\
&\quad + \sum_{j \in J} e_j : (Q_j\parallel(R + S)) \\
&\quad + \sum_{l \in L} f_l : ((P + Q)\parallel S_l) \\
&\quad + \sum_{j \in J, l(l(e_j) = l(f_l))} ((\tau, \pi(e_j) + \pi(f_l)) : (Q_j\parallel S_l))
\end{align*}$$

\textbf{Scope(1)}  

It follows from the fact that $$A:P$$ has a single transition that:

$$T(A:P \Delta^b_{t>0}(Q,R,S)) = \left\{ (A, P \Delta^b_{t=1}(Q,R,S)) \right\} \cup T(S)$$

$$= T(A:(P \Delta^b_{t=1}(Q,R,S)) + S)$$
It follows from the fact that $e.P$ has a single transition that, when $l(e) \neq \bar{b}$

$$
T(e.P \Delta_t^b (Q, R, S)) = \{ (e, P \Delta_t^b (Q, R, S)) \} \cup T(S) \\
= T(e.(P \Delta_t^b (Q, R, S)) + S)
$$

\[\square\]

It follows from the fact that $e.P$ has a single transition that, when $l(e) = \bar{b}$

$$
T(e.P \Delta_t^b (Q, R, S)) = \{ ((\tau, \pi(e)), Q) \} \cup T(S) \\
= T((\tau, \pi(e)).Q + S)
$$

\[\square\]

Follows directly from the fact that $\text{ScopeT}$ is the only transition that applies when $t = 0$:

$$
T(P_0 \Delta_0^b (Q, R, S)) = T(R)
$$

\[\square\]

We distinguish two different cases:

i) When $t = 0$

$$
T(P_1 + P_2 \Delta_0^b (Q, R, S)) = T(R) \cup T(R) \\
= T(P_1 \Delta_0^b (Q, R, S)) \cup T(P_2 \Delta_0^b (Q, R, S)) \\
= T(P_1 \Delta_0^b (Q, R, S)) + T(P_2 \Delta_0^b (Q, R, S))
$$
ii) When $\tau > 0$

$$T(P_1 + P_2 \triangle^b_\tau (Q, R, S))$$

$$= \{(A, P') \in T(P_1 + P_2) \}$$

$$\cup \{((\tau, n), Q) \in \emptyset\}$$

Scope(6) It follows from the fact that NIL has no transitions that the only non-empty set is the one corresponding to $S$.

$$T(\text{NIL} \triangle^b_\tau (Q, R, S)) = T(S)$$

Res(1)

$$T(\text{NIL} \setminus F) = \emptyset = T(\text{NIL})$$
Res(2)
\[ T((P + Q) \setminus F) = \{ (A, P') \mid (A, P') \in T(P + Q) \} \]
\[ \cup \{ ((a, n), P' \setminus F) \mid ((a, n), P') \in T(P + Q) \land a, \bar{a} \notin F \} \]
\[ = \{ (A, P' \setminus F) \mid (A, P') \in T(P) \} \]
\[ \cup \{ (A, P' \setminus F) \mid (A, P') \in T(Q) \} \]
\[ \cup \{ ((a, n), P' \setminus F) \mid ((a, n), P') \in T(P) \land a, \bar{a} \notin F \} \]
\[ \cup \{ ((a, n), P' \setminus F) \mid ((a, n), P') \in T(Q) \land a, \bar{a} \notin F \} \]
\[ = T(P \setminus F) \cup T(Q \setminus F) = T(P \setminus F + Q \setminus F) \]

\[ \square \]

Res(3)
\[ T((A : P) \setminus F) = \{ (A, P) \setminus F \} = T(A : (P \setminus F)) \]

\[ \square \]

Res(4) When \( a, \bar{a} \notin F \) we get:
\[ T(((a, n).P) \setminus F) = \{ ((a, n), P) \setminus F \} = T((a, n) \cdot (P \setminus F)) \]

\[ \square \]

Res(5)
\[ T(P \setminus E \setminus F) = \{ (A, P') \mid (A, P') \in T(P \setminus E) \} \]
\[ \cup \{ ((a, n), P' \setminus F) \mid ((a, n), P') \in T(P \setminus E) \land a, \bar{a} \notin F \} \]
\[ = \{ (A, P' \setminus E \setminus F) \mid (A, P') \in T(P) \} \]
\[ \cup \{ ((a, n), P'' \setminus E \setminus F) \mid ((a, n), P'') \in T(P) \land a, \bar{a} \notin E \land a, \bar{a} \notin F \} \]
\[ = \{ (A, P'' \setminus E \setminus F) \mid (A, P'') \in T(P) \} \]
\[ \cup \{ ((a, n), P'' \setminus E \setminus F) \mid ((a, n), P'') \in T(P) \land a, \bar{a} \notin E \cup F \} \]

However,
\[ T(P \setminus (E \cup F)) = \{ (A, P' \setminus E \cup F) \mid (A, P') \in T(P) \} \]
\[ \cup \{ ((a, n), P' \setminus E \cup F) \mid ((a, n), P') \in T(P) \land a, \bar{a} \notin E \cup F \} \]

It follows from lemma 5.2 that the relation \( \{(X \setminus E \setminus F, X \setminus E \cup F) \mid E, F \subseteq \mathcal{L} \} \) is a bisimulation.

\[ \square \]
Res(6)

\[ T(P \setminus \emptyset) = \{ \langle A, P' \setminus \emptyset \rangle \mid \langle A, P' \rangle \in T(P) \} \]
\[ \cup \{ \langle (a, n), P' \setminus \emptyset \rangle \mid \langle (a, n), P' \rangle \in T(P) \land a, \bar{a} \notin \emptyset \} \]
\[ = \{ \langle A, P' \setminus \emptyset \rangle \mid \langle A, P' \rangle \in T(P) \} \]
\[ \cup \{ \langle (a, n), P' \setminus \emptyset \rangle \mid \langle (a, n), P' \rangle \in T(P) \} \]
\[ = \{ \alpha, P' \setminus \emptyset \mid \langle \alpha, P' \rangle \in T(P) \} \]

It follows from lemma 5.2 that the relation \{ (X \setminus \emptyset, X) \} is a bisimulation. \qed

Close(1)

\[ T([\text{NIL}]_I) = \emptyset = T(\text{NIL}) \]

Close(2)

\[ T([P + Q]_I) = \]
\[ = \{ \langle A_1 \cup A_2, [P']_I \rangle \mid \langle A_1, P' \rangle \in T(P + Q) \land A_2 = \{ (r, 0) \mid r \in I - \rho(A_1) \} \}\]
\[ \cup \{ \langle e, [P']_I \rangle \mid \langle e, P' \rangle \in T(P + Q) \} \]
\[ = \{ \langle A_1 \cup A_2, [P']_I \rangle \mid \langle A_1, P' \rangle \in T(P) \land A_2 = \{ (r, 0) \mid r \in I - \rho(A_1) \} \}\]
\[ \cup \{ \langle e, [P']_I \rangle \mid \langle e, P' \rangle \in T(P) \} \]
\[ \cup \{ \langle e, [P']_I \rangle \mid \langle e, P' \rangle \in T(Q) \} \]
\[ = T([P]_I) \cup T([Q]_I) = T([P]_I + [Q]_I) \]

Close(3)  When \( A_2 = \{ (r, 0) \mid r \in I - \rho(A_1) \} \) we have:

\[ T([A_1; P]_I) = \{ A_1 \cup A_2, [P]_I \} = T((A_1 \cup A_2); [P]_I) \]

Close(4)

\[ T([e.P]_I) = \{ e, [P]_I \} = T(e.[P]_I) \]
Close(5)

\[ T([P]_I) = \]
\[ \{ \langle A' \cup A_3, [P']_I \rangle \mid \langle A', P' \rangle \in T([P]_I) \wedge A_3 = \{(r,0) \mid r \in J - \rho(A')\}\} \]
\[ \cup \{ \langle e, [P']_I \rangle \mid \langle e, P' \rangle \in T([P]_I) \} \]
\[ = \{ \langle A'' \cup A_2 \cup A_3, [P''']_I \rangle \mid \langle A'', P'' \rangle \in T(P) \wedge A_2 = \{(r,0) \mid r \in I - \rho(A'')\} \]
\[ \wedge A_3 = \{(r,0) \mid r \in J - \rho(A'' \cup A_2)\}\} \]
\[ \cup \{ \langle e, [P''']_I \rangle \mid \langle e, P'' \rangle \in T(P) \} \]
\[ = \{ \langle A'' \cup B, [P''']_I \rangle \mid \langle A'', P'' \rangle \in T(P) \wedge B = \{(r,0) \mid r \in (I \cup J) - \rho(A'')\} \}
\[ \cup \{ \langle e, [P''']_I \rangle \mid \langle e, P'' \rangle \in T(P) \} \]

However,

\[ T([P]_{I \cup J}) = \]
\[ \{ \langle A'' \cup B, [P''']_{I \cup J} \rangle \mid \langle A'', P'' \rangle \in T(P) \wedge B = \{(r,0) \mid r \in (I \cup J) - \rho(A'')\} \}
\[ \cup \{ \langle e, [P''']_{I \cup J} \rangle \mid \langle e, P'' \rangle \in T(P) \} \]

It follows from lemma 5.2 that the relation \{([X]_I]_J, [X]_{I \cup J}) \mid I, J \subseteq \mathcal{R}\} is a bisimulation.

\[ \square \]

Close(6)

\[ T([P]_{\emptyset}) = \]
\[ \{ \langle A_1 \cup A_2, [P]_{\emptyset} \rangle \mid \langle A_1, P' \rangle \in T(P) \wedge A_2 = \{(r,0) \mid r \in \emptyset - \rho(A')\} \}
\[ \cup \{ \langle e, [P]_{\emptyset} \rangle \mid \langle e, P' \rangle \in T(P) \} \]
\[ = \{ \langle A_1, [P]_{\emptyset} \rangle \mid \langle A_1, P' \rangle \in T(P) \}
\[ \cup \{ \langle e, [P]_{\emptyset} \rangle \mid \langle e, P' \rangle \in T(P) \} \]
\[ = \{ \langle \alpha, [P]_{\emptyset} \rangle \mid \langle \alpha, P' \rangle \in T(P) \} \]

It follows from lemma 5.2 that the relation \{([X]_{\emptyset}, X)\} is a bisimulation.

\[ \square \]

Close(7)

\[ T([P \setminus E]_I) = \]
\[ \{ \langle A_1 \cup A_2, [P']_I \rangle \mid \langle A_1, P' \rangle \in T(P \setminus E) \wedge A_2 = \{(r,0) \mid r \in I - \rho(A_1)\} \}
\[ \cup \{ \langle (a,n), [P']_I \rangle \mid \langle (a,n), P' \rangle \in T(P \setminus E) \} \]
\[ = \{ \langle A_1 \cup A_2, [P'' \setminus E]_I \rangle \mid \langle A_1, P'' \rangle \in T(P) \wedge A_2 = \{(r,0) \mid r \in I - \rho(A_1)\} \}
\[ \cup \{ \langle (a,n), [P'' \setminus E]_I \rangle \mid \langle (a,n), P'' \rangle \in T(P) \wedge a, \bar{a} \notin E \} \]

However,

\[ T([P]_I \setminus E) = \]
\[ \{ \langle A_1, P' \setminus E \rangle \mid \langle A_1, P' \rangle \in T([P]_I) \}
\[ \cup \{ \langle (a,n), P' \setminus E \rangle \mid \langle (a,n), P' \rangle \in T([P]_I) \wedge a, \bar{a} \notin E \} \]
\[ = \{ \langle A_1 \cup A_2, [P'']_I \setminus E \rangle \mid \langle A_1, P'' \rangle \in T(P) \wedge A_2 = \{(r,0) \mid r \in I - \rho(A_1)\} \}
\[ \cup \{ \langle (a,n), [P'']_I \setminus E \rangle \mid \langle (a,n), P'' \rangle \in T(P) \wedge a, \bar{a} \notin E \} \]

It follows from lemma 5.2 that the relation \{([X \setminus E]_I, [X]_I \setminus E) \mid I \subseteq \mathcal{R} \wedge E \subseteq \mathcal{L}\} is a bisimulation.

\[ \square \]
Rec(1) From the operational semantic rule Rec we have:

\[ T(\text{rec } X.P) = T(P[\text{rec } X.P/X]) \]

\( \square \)

Rec 2 Let \( R \overset{\text{def}}{=} \text{rec } X.Q \); by Rec(1), \( R = Q[R/X] \). We need to prove that \( P \sim X R \), 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 \( R \) 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 observation that, when \( X \) is guarded in \( Q \), the first step of \( Q[P/X] \) does not depend on the value of \( P \), more formally:

\[ Q[P/X] \xrightarrow{\alpha} Q'[P/X] \text{ if and only if } Q \xrightarrow{\alpha} Q' \]

and

\[ Q[R/X] \xrightarrow{\alpha} Q'[R/X] \text{ if and only if } Q \xrightarrow{\alpha} Q' \]

We proceed by induction on the structure of \( E \).

If \( E \) is NIL, \( T(E[P/X]) = \emptyset = T(E[Q/X]) \).

If \( E \) is \( X \), we obtain the \( E[P/X] = P = Q[P/X] \) and similarly \( E[R/X] = R = Q[R/X] \) and therefore

\[ T(E[P/X]) = \{(\alpha, Q'[P/X]) \mid Q \xrightarrow{\alpha} Q'\} \]

and

\[ T(E[R/X]) = \{(\alpha, Q'[R/X]) \mid Q \xrightarrow{\alpha} Q'\} \]

If \( E \) is \( \alpha F \) then

\[ T(E[P/X]) = \{(\alpha, F[P/X])\} \]

and

\[ T(E[Q/X]) = \{(\alpha, F[Q/X])\} \].

The other cases follow from the induction hypothesis and the fact that prioritized strong bisimulation is a congruence. \( \square \)

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

\[ f_i(P) \overset{\text{def}}{=} [P \setminus E_i]v_i \]

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

\[ f_I(P) \overset{\text{def}}{=} f_{i_1}f_{i_2} \cdots f_{i_n}(P) \]
as expected,

\[ f_{\theta}(P) \overset{\text{def}}{=} P \]

It is also easy to see that the functions \( f_I \) are associative in the sense that

\[ f_I(f_J(P)) = f_{I \cup J}(P) \]

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 \( \text{rec } X.P \), the only rule that applies is \( \text{Rec} \) and therefore any transition of \( \text{rec } X.P \) must be obtained from unrolling the recursion. The proof proceeds by successive unrollings of the recursion (i.e., applying \( \text{Rec}(1) \)) until no new behavior can be derived.

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

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

by unrolling the recursion we obtain

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

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:

\[
Q = P[Q/X] + [P[Q/X]]_U + [(Q)]_U + [Q]_U + [P[Q/X]]_V + [(Q)]_V + [Q]_V
\]

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

\[
Q = P[Q/X] + [P[Q/X]]_U + [P[Q/X]]_V \\
+ [P[Q/X]]_U + [(Q)]_U + [Q]_U \\
+ [P[Q/X]]_V + [(Q)]_V + [Q]_V \\
+ [P[Q/X]]_U + [Q]_U + [Q]_V \\
+ [P[Q/X]]_V + [Q]_V + [Q]_V \\
+ [P[Q/X]]_U + [Q]_U + [Q]_V \\
+ [P[Q/X]]_V + [Q]_V + [Q]_V \\
+ [P[Q/X]]_U + [Q]_U + [Q]_V \\
+ [Q]_U + [Q]_U + [Q]_V
\]
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 $Q = \text{rec } X.(P + \sum_{i \in I} f_i(X))$ then

$$Q = \sum_{J \subseteq I} f_J(P^{[Q/x]}) + \sum_{J \subseteq I, J \neq \emptyset} f_J(Q)$$

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

$$Q = \text{rec } X.(P + f_i(X))$$

$$= P^{[Q/x]} + f_i(Q)$$

$$= P^{[Q/x]} + f_i(P^{[Q/x]} + f_i(Q))$$

$$= P^{[Q/x]} + f_i(P^{[Q/x]}) + f_i f_i(Q)$$

$$= \sum_{J \subseteq \{i\}} f_J(P^{[Q/x]}) + \sum_{J \subseteq \{i\}, J \neq \emptyset} f_J(Q)$$

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

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

$$Q = \sum_{J \subseteq I} f_J(P'^{[Q/x]}) + \sum_{J \subseteq I, J \neq \emptyset} f_J(Q)$$

$$= \sum_{J \subseteq I} f_J(P'^{[Q/x]} + f_{i_{n+1}}(Q)) + \sum_{J \subseteq I, J \neq \emptyset} f_J(Q)$$

$$= \sum_{J \subseteq I} f_J(P'^{[Q/x]}) + \sum_{J \subseteq I} f_J f_{i_{n+1}}(Q) + \sum_{J \subseteq I, J \neq \emptyset} f_J(Q)$$

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:

$$Q = \sum_{J \subseteq I} f_J(P^{[Q/x]}) + \sum_{J \subseteq I', J \neq \emptyset} f_J(Q)$$

By unrolling the recursion one more time we obtain:

$$Q = \sum_{J \subseteq I} f_J(P^{[Q/x]}) + \sum_{J \subseteq I', J \neq \emptyset} f_J(P^{[Q/x]}) + \sum_{i \in I} f_i(Q)$$

$$= \sum_{J \subseteq I} f_J(P^{[Q/x]}) + \sum_{J \subseteq I', J \neq \emptyset} f_J(P^{[Q/x]}) + \sum_{J \subseteq I', J \neq \emptyset} f_J(\sum_{i \in I} f_i(Q))$$
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 $\text{Rec}(3)$, we define

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

and we are about to prove the equation:

$$Q \overset{?}{=} \text{rec } X.(\sum_{J \subseteq I} f_i(P))$$

From the above lemma, we have:

$$Q = \sum_{J \subseteq I} f_J(P[^Q/x]) + \sum_{J \subseteq I, J \neq \emptyset} f_J(Q)$$

Let

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

and

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

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

$$Q_2 = \sum_{J \subseteq I, J \neq \emptyset} f_J(P[^Q/x]) + \sum_{J \subseteq I, J \neq \emptyset} f_J(\sum_{J \subseteq I} f_i(Q))$$

$$= \sum_{J \subseteq I, J \neq \emptyset} f_J(P[^Q/x]) + Q_2$$

At this point 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:

$$Q = \sum_{J \subseteq I} f_J(P[^Q/x]) = (\sum_{J \subseteq I} f_J(P))[^Q/x]$$

But since $X$ is guarded in $P$, we can apply $\text{Rec}(2)$ and obtain:

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

□
6 Completeness for finite state agents

In this section, we prove that the ACSR laws are complete for some (large) subset of the finite state agents. 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 $\mathcal{A}$. We then develop the five steps of the proof of completeness which are as follows.

We prove that unguarded recursions can be eliminated. In the absence 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 and finally, we prove that those sets of equations have a unique solution up to a 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. In [9], Robin Milner 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, $X \setminus E$ 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 $\rightsquigarrow$ (without label) as the minimum relation that satisfies the following rules:

Note that this definition is validated by the soundness of the laws Res(5), Res(6), Close(5),
Based on this, we can define the notion of bisimulation that we will be using throughout this section.

**Definition 6.1** A process $P$ is bisimilar to a process $Q$, noted $P \sim_\pi Q$, if, for all $\alpha \in \mathcal{D}$, $I \subseteq \mathcal{R}$ and $E \subseteq \mathcal{L}$

1. if $P \xrightarrow{\alpha_\pi} P'$ then, for some $Q'$, $Q \xrightarrow{\alpha_\pi} Q'$ and $P' \sim_\pi Q'$, and

2. if $Q \xrightarrow{\alpha_\pi} Q'$ then, for some $P'$, $P \xrightarrow{\alpha_\pi} P'$ and $P' \sim_\pi Q'$, and

3. $P \rightarrow [X\backslash E]_I$ iff $Q \rightarrow [X\backslash E]_I$.

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 have used for finite state agents, such as [9] and [2], 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 agents can be generated even without the use of the parallel operator as is illustrated by the following example.

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

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

and

$$P \xrightarrow{A} (\text{rec } X.A:X \cong_b (\text{NIL}, \text{NIL}, B:\text{NIL})) \cong_b (\text{NIL}, \text{NIL}, B:\text{NIL})$$

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

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

$$P' \xrightarrow{B} \text{NIL} \cong_b (\text{NIL}, \text{NIL}, B:\text{NIL})$$
and

\[
P' \xrightarrow{A} (\text{rec } X.(A:X \Delta^b_{\infty} (\text{NIL}, \text{NIL}, B:\text{NIL}))) \\
\triangle^b_{\infty} (\text{NIL}, \text{NIL}, B:\text{NIL}) \triangle^b_{\infty} (\text{NIL}, \text{NIL}, B:\text{NIL})
\]

and so forth as shown in fig. 1.

![Figure 1: A non finite state agent](image)

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 through parallel nor scope. Unfortunately, this is very difficult to characterize syntactically — for example, the process \( P = \text{rec } X.(A.X||\text{NIL}) \) 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 are either FS or are provably equivalent to an FS process. — The only exceptions we have found so far are of the form \( \text{rec } X.(X \Delta^b_{\infty} (P, Q, R)) \).

We formally define FS processes by way of a recursive function \( fs \) over processes. (We assume the usual definition of the function \( \text{fv}(P) \) which yields the set of free variables of a process \( P \).)

\[
\begin{align*}
\text{fs}(\text{NIL}) &= \text{true} \\
\text{fs}(X) &= \text{true} \\
\text{fs}(A:P) &= \text{fs}(P)
\end{align*}
\]
Definition 6.2 \((FS\ \text{Process})\) A process is said to be FS if \(fs(P) = \text{true}\).

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 mean of a head normal form where all the unguarded free variables are isolated as summands.

Definition 6.3 \((Head\ \text{Normal\ Form})\) A process \(P\) is in Head Normal Form or \(\text{"HNF"}\) if it has the form

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

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 \("[X\setminus\emptyset]_g\)" using Res(6) and Close(6). Now, assume it is true for any term of depth \(\leq n\). For a term of depth \(n + 1\), we examine all the possible forms such a term can take:

case 1 \((Timed\ \text{Action\ Prefix})\): \(P = A:P'\) is in HNF.

case 2 \((Instantaneous\ \text{Event\ Prefix})\): \(P = e.P'\) is in HNF.

case 3 \((Choice)\): \(P + R\) with \(P\) and \(R\) in HNF (by induction hypothesis) has the form:

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

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

\[
\sum_{i \in I \cup I'} [X_i \setminus E_i]_{U_i} + \sum_{j \in J \cup J'} \alpha_j Q_j .
\]
case 4 (Parallel): By the induction hypothesis, and since $P$ and $Q$ are FS, $P\parallel Q$ can be written:

\[
\left( \sum_{i \in I} A_i; P_i \right) + \sum_{j \in J} e_j; Q_j \parallel R = \left( \sum_{k \in K} B_k; R_k \right) + \sum_{l \in L} f_l; S_l
\]

Using Par(3), we obtain:

\[
P\parallel Q = \sum_{i \in I, k \in K, \rho(A_i) \cap \rho(B_k) = \emptyset} (A_i \cup B_k); (P_i \parallel R_k)
\]

\[
+ \sum_{j \in J} e_j \cdot (Q_j \parallel (\sum_{k \in K} B_k; R_k + \sum_{l \in L} f_l; S_l))
\]

\[
+ \sum_{l \in L} f_l \cdot ((\sum_{i \in I} A_i; P_i + \sum_{j \in J} e_j; Q_j) \parallel S_l)
\]

\[
+ \sum_{j \in J, l \in L, \ell(e_j) = \ell(f_l)} (\tau, \pi(e_j) + \pi(f_l)) \cdot (Q_j \parallel S_l)
\]

which is in HNF.

case 5 (Scope): Let $P = Q \Delta^b_t (R, S, T)$, 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: $A:(Q_1 \Delta^b_{t-1} (R, S, T)) + T$ with $T$ is already in HNF.

When Scope(2) applies, we obtain a term of the form: $(a, m) . (Q_1 \Delta^b_t (R, S, T)) + T$, 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): $P = Q \setminus F$, by induction hypothesis, we can assume that $Q$ is in HNF. If $Q = NIL$ 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) for instantaneous events, the restriction operator can be pushed down
one level. For free variables, we have terms of the form \([X_i \setminus E]_U \setminus F\) which can be transformed into \([X_i \setminus E \cup F]_U\) by Close(7) and Res(5).

**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 restriction over every summand. Then, using Close(3) for timed action and Close(4) for instantaneous events, the restriction operator can be pushed down one level. For free variables, we have terms of the form \([X_i \setminus E]_U\) which can be transformed into \([X_i \setminus E]_U \setminus I\) by Close(5).

**case 8 (Recursion):** As we have done in the proof of Rec(3), we will use the notation:

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

and

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

Let \(P = \text{rec } X.Q\) with \(Q\) in \(HNF\). Therefore,

\[
P = \text{rec } X. \left( \sum_{i \in I} f_i(X_i) + \sum_{j \in J} \alpha_j P_j \right)
\]

Let \(I' = \{ i \in I \mid X_i \neq X \}\) and \(I'' = I - I'\), we have

\[
P = \text{rec } X. \left( \sum_{i \in I''} f_i(X_i) + \sum_{i \in I'} f_i(X_i) + \sum_{j \in J} \alpha_j P_j \right)
\]

By Rec(3) we obtain:

\[
P = \text{rec } X. \left( \sum_{K \subseteq I''} f_K \left( \sum_{i \in I'} f_i(X_i) + \sum_{j \in J} \alpha_j P_j \right) \right)
\]

\[
= \text{rec } X. \left( \sum_{K \subseteq I'} \sum_{i \in I'} f_K f_i(X_i) + \sum_{K \subseteq I'} \sum_{j \in J} f_K(\alpha_j P_j) \right)
\]

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

\[
P = \text{rec } X. \left( \sum_{K \subseteq I'} \sum_{i \in I'} f_K f_i(X_i) + \sum_{K \subseteq I''} \sum_{j \in J, \ell(\alpha_j) \in E_k} \beta_{jK} f_K(P_j) \right)
\]

where

\[
\beta_{jK} = \begin{cases} 
\alpha_j & \text{if } \alpha_j \in \mathcal{D}_E \\
\alpha_j \cup \{(r,0)\} & \text{if } \alpha_j \in \mathcal{D}_R
\end{cases}
\]
At this point, by applying $\text{Rec}(1)$ and noticing that none of the $X_i$ is $X$ we obtain the HNF:

$$P = \sum_{K \subseteq I'} \sum_{i \in I'} f_K f_i(X_i) + \sum_{K \subseteq I''} \sum_{j \in J, \ell \in E_k} \beta_{jk} f_k(P_{j[\text{rec}X,P]/X})$$

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 $P'$ is 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 $\text{rec}X.Q$. 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 $\text{Rec}(3)$. \qed

### 6.4 Standard Set of Equations

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

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

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

$$X_i = \sum_{j \in J_i} [W_j \backslash E_j] u_j + \sum_{k \in K_i} \alpha_k X_k$$

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 $\alpha X_i$ and $\beta X_j$ such that $\alpha < \beta$.

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

$$X_i = \sum_{j \in J_i} [W_j \backslash E_j] u_j + \sum_{k \in K_i} \alpha_k X_k$$

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

$$Y_i = \sum_{l \in L_i} [W_l \backslash F_l] v_l + \sum_{m \in M_i} \beta_m Y_m$$

Furthermore, we will assume that the sets of variables $\overline{W}$, $\overline{X}$ and $\overline{Y}$ 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** $(R = \text{NIL})$ $R$ satisfies the single equation $X = \text{NIL}$.

**case 2** $(R = W)$ By Res(6) and Close(6), $R$ satisfies the single equation $X = [W \setminus \emptyset]$.

**case 3** $(R = \alpha P)$ By induction hypothesis, $P$ provably satisfies $S : \bar{X} = \bar{H}$. Therefore $R$ provably satisfy 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 $S : \bar{X} = \bar{H}$ and $Q$ provably satisfies $T : \bar{Y} = \bar{G}$. Therefore $R$ provably satisfies the set of equations $\{X = H_1 + G_1\} \cup S \cup T$, with the new distinguished variable $X$.

**case 5** $(R = P \| Q)$ Note that since $R$ is FS, neither $P$ nor $Q$ have 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:

$$X_i = \sum_{k \in K_i'} A_k \cdot X_k + \sum_{k \in K_i''} e_k \cdot X_k$$

and that the equations of $T$ are written:

$$Y_i = \sum_{m \in M_i'} B_m \cdot Y_m + \sum_{m \in M_i''} f_m \cdot Y_m$$

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

$$Z_{1,1} = \sum_{k \in K_i', m \in M_i'} (A_k \cup B_m) \cdot (X_k \| Y_m)$$

$$+ \sum_{k \in K_i'} e_k \cdot (X_k \| Y_1)$$

$$+ \sum_{m \in M_i''} f_m \cdot (X_1 \| Y_m)$$

$$+ \sum_{k \in K_i'', m \in M_i''} (\tau, \pi(e_k) + \pi(f_m)) \cdot (X_k \| Y_m)$$
In the same fashion we can define a set of equations \( Z_{i,j} = H_i || G_j \); then and apply \( \text{Par}(3) \) to obtain a set of standard equations of the form:

\[
Z_{i,j} = \sum_{k \in K_i, m \in M_j} (A_k \cup B_m) ; Z_{km} + \sum_{k \in K_i} e_k . Z_{kj} + \sum_{m \in M_j} f_m . Z_{im} + \sum_{k \in K_i, m \in M_j, t(e_k) = (f_m)} (\tau, \pi(e_k) + \pi(f_m)) . Z_{km}
\]

**case 6** \((R = P \Delta_i^b(Q, S, T))\) Let us note \( Y'_1 = G'_1 \) and \( Y''_1 = G''_1 \) the distinguished equation of the sets satisfied by \( S \) and \( T \), respectively. \( R \) satisfies the non-standard equation

\[
Z_1 = H_1 \Delta_i^b (G_1, G'_1, G''_1)
\]

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

\[
Z_1 = \sum_{j \in J'_1} ((\alpha_j X_j) \Delta_i^b (G_1, G'_1, G''_1)) + \sum_{j \in J''_1} ((A_j : X_j) \Delta_i^b (G_1, G'_1, G''_1)) + \sum_{j \in J''_1} ((e_j : X_j) \Delta_i^b (G_1, G'_1, G''_1))
\]

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

\[
Z_1 = \sum_{j \in J'_1} A_j : (X_j \Delta_{t-1}^b (G_1, G'_1, G''_1)) + \sum_{j \in J''_1, l(e_j) \neq \neq b} e_j . (X_j \Delta_i^b (G_1, G'_1, G''_1)) + \sum_{j \in J''_1, l(e_j) = b} (\tau, \pi(e_j)) . G_1 + G''_1
\]
GI can be replaced by Yl. We can also take fresh variables to define a new pair of equation for each summand of H1 as follows:

\[ Z_j^i = H_j \triangle^b_i (G_1, G'_1, G''_1) \]
\[ Z_{j-1} = H_j \triangle^b_{i-1} (G_1, G'_1, G''_1) \]

and we obtain the standard equation:

\[ Z_1 = \sum_{j \in J'_1} A_j : Z_j^i + \sum_{j \in J''_1, l(e_j) \neq 5} e_j, Z_j^i + \sum_{j \in J''_1, l(e_j) = 5} (\tau, \pi(e_j)).Y_1 + G''_1. \]

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 = NIL — we obtain:

\[ Z_1 = \sum_{j \in J_1} ([W_j \backslash E_j]_{U_j} \backslash F) + \sum_{k \in K_1} (\alpha_k X_k) \backslash F \]
\[ = \sum_{j \in J_1} [W_j \backslash E_j]_{U_j} \backslash F + \sum_{k \in K'_1} (A_k : X_k) \backslash F + \sum_{k \in K''_1} (e_k \cdot X_k) \backslash F. \]

By Close(7) and Res(6) we have \([W_j \backslash E_j]_{U_j} \backslash F = [W_j \backslash E_j \cup F]_{U_j}\). Applying Res(3) and Res(4) and introducing the new equations \(Z_i = H_i \backslash F\) gives a set of standard equations of the form:

\[ Z_i = \sum_{j \in J_i} [W_j \backslash E_j \cup F]_{U_j} + \sum_{k \in K'_1} A_k : Z_k + \sum_{k \in K''_1, \ell(e_k), \ell(e_k) \not\in F} e_k \cdot Z_k. \]

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 = NIL — we obtain:

\[ Z_1 = \sum_{j \in J_1} ([W_j \backslash E_j]_{U_j})_V + \sum_{k \in K_1} [\alpha_k X_k]_V \]
\[ = \sum_{j \in J_1} ([W_j \backslash E_j]_{U_j})_V + \sum_{k \in K'_1} [A_k : X_k]_V + \sum_{k \in K''_1} [e_k \cdot X_k]_V. \]

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:

\[ Z_i = \sum_{j \in J_i} [W_j \backslash E_j]_{U_j \cup V} + \sum_{k \in K'_i} (A_k \cup \{(r, 0) | r \in V - \rho(A_k)\}) : Z_k + \sum_{k \in K''_i} e_k \cdot Z_k. \]
**Theorem 6.2** Every guarded FS process $P$ with free variables $\overline{W}$ provably satisfies a prioritized set of equations $S$ with free variables in $\overline{W}$.

**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:**

Let us consider a relation $\mathcal{R}$ such that $(i, j) \in \mathcal{R}$ iff $H_i \sim \pi G_j$. We know that such a relation exists because it defines a bisimulation between $P$ and $Q$, and $P \sim \pi Q$ by hypothesis. Note that since we are only interested here in the existence of such a relation, we do not have to provide a method for defining it.

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

$$Z_{u,v} = \sum_{j \in J_u} [W_j \backslash E_j]u_j + \sum_{(k,l,m) \in K_{uv}} \alpha_k Z_{lm}$$

With

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

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

Since $P$ satisfies $S$, there is a set of expressions $P_1, P_2, \ldots$ such that $P_1 = P$ and $\tilde{X} = \tilde{H}[\tilde{P}/\tilde{X}]$. Now take the set of processes $\tilde{R}_{i,j} = P_i$. 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, $F_{i,j}[\tilde{R}/\tilde{Z}] = P_1 = P$. 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. \qed
6.6 Unique Solution

We now have to prove that if two processes satisfy the same set of prioritized equations, they are bisimilar. Such 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 follows exactly the proof given in [10]. It proceeds by induction on the number of equations. For one equation, \( X = H \), we have the solution \( P = \text{rec } X.H \). Moreover, if there is a process \( Q \) such that \( Q = H[Q/X] \), then, by Rec(2), we have \( Q = \text{rec } X.H \).

Assume it is true for \( n \) equations. Let \( S : \overline{X} = \overline{H} \cup \{X_{n+1} = H_{n+1}\} \) be a system with \( n + 1 \) equations. Consider the system of \( n \) equations \( S' : \overline{X} = \overline{H[\text{rec } X_{n+1}.H_{n+1}/X_{n+1}]}; X_{n+1} \) does not occur free in \( S' \). Therefore, by induction hypothesis, there is a set of processes \( \overline{P} \) such that
\[
\overline{X} = \overline{H[\text{rec } X_{n+1}.H_{n+1}/X_{n+1}]}[\overline{P}/\overline{X}].
\]
If we choose \( P_{n+1} = \text{rec } X_{n+1}.H_{n+1}[\overline{P}/\overline{X}] \) we have found a solution to the system \( S \).

For the uniqueness, suppose that we have a second solution \( \overline{Q} \cup \{Q_{n+1}\} \). That is
\[
\overline{Q} = \overline{H[\text{rec } Q_{n+1}/\overline{X},X_{n+1}]} \quad Q_{n+1} = H_{n+1}[\overline{Q}_{n+1}/\overline{X},X_{n+1}] \]
The second equation can be written
\[
Q_{n+1} = H_{n+1}[\overline{Q}/\overline{X}][Q_{n+1}/X_{n+1}]
\]
and therefore, by Rec(2), we have \( Q_{n+1} = \text{rec } X_{n+1}.H_{n+1}[\overline{Q}/\overline{X}] \) which can be rewritten as \( Q_{n+1} = (\text{rec } X_{n+1}.H_{n+1})[\overline{Q}/\overline{X}] \). It follows that \( Q_{n+1} = P_{n+1} \). By induction hypothesis, it is easy to see that \( \overline{Q} = \overline{P} \).

6.7 Completeness

**Theorem 6.5** For any two FS processes \( P \) and \( Q \), if \( P \sim_n Q \) 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 \( P' = Q' \). \(\square\)
7 Conclusions

We have described a timed process algebra called ACSR that supports the notions of resources 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 complete for most finite state agents.

There are two 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. The second extension is to allow dense time so that a timed action can take an arbitrary non-zero amount of time. In addition, it should be possible 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.

References


