January 1995

A Process Algebra of Communicating Shared Resources With Dense Time and Priorities

Patrice Brémond-Grégoire
University of Pennsylvania

Insup Lee
University of Pennsylvania, lee@cis.upenn.edu

A Process Algebra of Communicating Shared Resources With Dense Time and Priorities

Abstract

The correctness of real-time distributed systems depends not only on the function they compute but also on their timing characteristics. Furthermore, these characteristics are strongly influenced by the delays due to synchronization and resource availability. Process algebras have been used successfully to define and prove correctness of distributed systems. More recently, there has been a lot of activity to extend their application to real-time systems. The problem with most current approaches is that they ignore resource constraints and assume either maximum parallelism (i.e., unlimited resources) or pure interleaving (i.e., single resource). Algebra of Communicating Shared Resources (ACSR) is a process algebra designed for the formal specification and manipulation of distributed systems with resources and real-time constraints. A dense time domain provides a more natural way of specifying systems compared to the usual discrete time. Priorities provide a measure of urgency for each action and can be used to ensure that deadlines are met. In ACSR, processes are specified using resource bound, timed actions and instantaneous synchronization events. Processes can be combined using traditional operators such as nondeterministic choice and parallel execution. Specialized operators allow the specification of real-time behavior and constraints. The semantics of ACSR is defined as a labeled transition system. Equivalence between processes is based on the notion of strong bisimulation. A sound and complete set of algebraic laws can be used to transform almost any ACSR process into a normal form.

Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-95-08.

A Process Algebra of Communicating Shared Resources with Dense Time and Priorities

MS-CIS-95-08

Patrice Bremond-Gregoire
Insup Lee

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

March 1995

A Process Algebra of Communicating Shared Resources with Dense Time and Priorities [*]

Patrice Brémond-Grégoire, Insup Lee
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104-6389

March 8, 1994

The correctness of real-time distributed systems depends not only on the function they compute but also on their timing characteristics. Furthermore, those characteristics are strongly influenced by the delays due to synchronization and resource availability. Process algebras have been used successfully to define and prove correctness of distributed systems. More recently, there has been a lot of activity to extend their application to real-time systems. The problem with most current approaches is that they ignore resource constraints and assume either maximum parallelism (i.e., unlimited resources) or pure interleaving (i.e., single resource). Algebra of Communicating Shared Resources (ACSR) is a process algebra designed for the formal specification and manipulation of distributed systems with resources and real-time constraints. A dense time domain provides a more natural way of specifying systems compared to the usual discrete time. Priorities provide a measure of urgency for each action and can be used to ensure that deadlines are met. In ACSR, processes are specified using resource bound, timed actions and instantaneous synchronization events. Processes can be combined using traditional operators such as nondeterministic choice and parallel execution. Specialized operators allow the specification of real-time behavior and constraints. The semantics of ACSR is defined as a labeled transition system. Equivalence between processes is based on the notion of strong bisimulation. A sound and complete set of algebraic laws can be used to transform almost any ACSR process into a normal form.

[*]This research was supported in part by DARPA/NSF CCR90-14621 and NSF CCR 93-11622.

Contents

1 Introduction

Reliability in real-time systems can be improved through the use of formal methods for their specification and analysis. Formal methods treat systems 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 [45], CSP [28], and ACP [11], 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 [45]. 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 to process algebras.

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. What is needed is 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. Algebra of Communicating Shared Resources (ACSR) is an attempt at providing such a framework.

ACCR exhibits several salient features that differenciates it from other process algebras. The notion of resources is integral to ACSR, it allows a close modeling of situations where several processes compete for the same resources. A common method to arbitrate such conflits is to assign priorities to processes; this method is formally supported in ACSR. Among the new operators introduced by ACSR are the interrupt which allows to model reaction to asynchronous signals and the exception operator which now commonly used in modern programming languages. Finally, ACSR uses a dense time paradigm, which provides more flexibility than the alternative, discrete time, in the specification of real-time processes; dense time also requires the formal treatment of time intervals which, in turn, leads to smaller normalized processes.

A formal method comprises a mathematical model, a syntax and a semantics. The mathematical model is the domain in which the objects of the language take a meaning. In our case it involves the definition of a time domain, a set of resources and actions, and a structured labelled transition system. These are the subject of Section 2. The syntax defines the rules for constructing valid sentences in the language. In ACSR, this consists of a simple algebraic expressions with a small set of operators and is described in Section 3. The semantics of the phrases of the language is elaborated in two steps. Section 3.1 provides a set of unprioritized (i.e., priority-ignored) operational rules. In this section we also discuss the intuitive meaning of each operator and give some examples of their usage. Priorities are treated in Section 3.2. A prioritized semantics is derived from the unprioritized one. We define a notion of compositionality, which ensures that prioritization can be enforced in any context, and prove that ACSR has that property. Finally, we describe our motivating example to give an ACSR specification of the system.

Section 4 is dedicated to the definition of a notion of equivalence based on strong bisimilarity; we present a sound and complete set of algebraic laws. The details of the proofs of soundness and completeness of these laws can be found in the appendix. Again we re-visit our example and give a partial proof of correctness.

Section 5 contains survey of relevant research related to the formal treatment of real-time system. We look at logics based methods, models based on automata theory and other work concerned with the incorporation of time and priorities in process algebras.

The conclusion in Section 6 reflects on the strengths and weaknesses of our work and explore areas where additional research is waranted.

2 ACSR Model

An ACSR process is a term over the ACSR signature, which will be described in the next section. We note *Proc* the set of all processes and use P, Q, R and S to range over *Proc*. Furthermore, we use a set of process variables *ProcVars* and let W, X, Y and Z range over it. A process evolves by executing successive actions. We denote by *Act* the set of all actions and use the Greek letters α and β to range over *Act*. There are two kinds of actions, timed and untimed. Timed actions are used to model the passage of time and the consumption of resources. Untimed actions are used to label instants in time and to affect inter-process synchronization.

2.1 Resource Consuming and Timed Actions

We assume a finite set of serially reusable resources Res. We use r to range over Res. A resource consuming action, A, represents the usage of a subset of these resources. It is defined by the set of resources used, noted $\rho(A)$, and a total function $\pi_A(r)$: $Res \rightarrow \boldsymbol{R}^{\geq 0}$ such that $\pi_A(r)$ is the priority of the resource r in the action A, and $\pi_A(r) = 0$ when $r \notin \rho(A)$.

Our time domain is the set of real numbers plus infinity: $\boldsymbol{R}^+ \cup \{\infty\}$. We use u, v and w to denote time values.

A timed action A^u is the execution of a resource consuming action for a duration u, where u is a positive and finite real number. In addition to A, the letters B and C are used to denote resource consuming actions and correspondingly B^v and C^w are used for timed actions.

We write a resource consuming action A as a set of pairs $\{(r_1, p_1), \ldots, (r_n, p_n)\}$ such that only the resources in $\rho(A)$ appear in the set and each one appears exactly once, paired with its priority. For example, we write: $A = \{(IOP, 2), (BUS, 3)\}$ for the action that consists of using the *IOP* resource at priority 2 and the *BUS* resource at priority 3; and $A^{2.4}$ for the execution of that action during 2.4 units of time.

We define two operations on resource consuming and timed actions: synchronous composition and closure.

The synchronous composition, noted "$A|B$" creates a single action, equivalent to two actions occurring simultaneously. Synchronous composition is only defined if the two actions are using disjoint sets of resources. This enforces the serial reusability aspect of the resources. Synchronous composition over timed actions requires, in addition, that the two action have the same duration. This ensures the uniform passage of time. The composition of two actions preserves timing, resource usage and priorities; in other words, assuming that $\rho(A) \cap \rho(B) = \emptyset$ we have:

$$\begin{aligned} A^u | B^u &= (A|B)^u \\ \rho(A|B) &= \rho(A) \cup \rho(B) \\ \pi_{A|B}(r) &= \pi_A(r) + \pi_B(r) \end{aligned}$$

It follows immediately from the definition that the synchronous composition of resource consuming and timed actions is commutative and associative.

The closure operation, $[A]_U$, consists of increasing the set of resources used by the resource consuming action A to include all the resources of the set U. The priority function is not affected by this operation: the incremental resources remain at priority 0. Closure over timed action is similar and is independent of timing, that is:

$$[A^u]_U = ([A]_U)^u$$

$$\begin{aligned}\rho([A]_U) &= \rho(A) \cup U \\ \pi_{[A]_U}(r) &= \pi_A(r)\end{aligned}$$

The idea behind this operation is to be able to reserve a set of resources for a process, even though some of its actions may not be using all of them.

It follows immediately from the definition that closure over resource consuming and timed actions is idempotent and associative in the sense that:

$$[[A]_U]_V = [A]_{U \cup V}$$

moreover, closure over an empty set has no effect: $[A]_\emptyset = A$.

2.2 Instantaneous Events

Instantaneous actions, or events, provide the basic synchronization mechanism of ACSR. An instantaneous event, e has a label, $l(e)$ and a priority $\pi(e)$. The labels are drawn from a countable set $\Lambda = \mathcal{L} \cup \{\tau\}$. The priority is a non-negative real number. We assume the existence of a complement operation over $\mathcal{L}$ such that

$$\forall a \in \mathcal{L}\colon \exists \overline{a} \in \mathcal{L} \qquad \text{and} \qquad \overline{\overline{a}} = a \qquad \text{and} \qquad \overline{a} \neq a$$

We use a, b and c to range over Λ and the lowercase letters e and f to denote events.

Events are used for pair-wise synchronization which is modeled as a composition operation over events. As in CCS, the special label $\tau \notin \mathcal{L}$ denotes internal actions. It will be convenient to overload the synchronous composition symbol that we have used for actions. The composition of two events, noted "$e|f$" is defined only when the two events have complementary labels, such as a and $\overline{a}$. The priority of the resulting event is the sum of the priority of the two original events. The reason for this choice will become clearer when we discuss compositionality in Section 3.2.2. Formally, assuming that $l(e) = \overline{l(f)}$, we have:

$$\begin{aligned}l(e|f) &= \tau \\ \pi(e|f) &= \pi(e) + \pi(f)\end{aligned}$$

It follows from this definition that the composition of two events is commutative, i.e., $e|f = f|e$. For example, "$(char\_in, 5)$" denotes the event with label "$char\_in$" and priority 5. The composition: $(char\_in, 5)|(\overline{char\_in}, 2) = (\tau, 7)$.

2.3 Computation Model

The behavior of a process is given by a labelled transition system, which is a subset of $Proc \times Act \times Proc$. For example, a process P can execute an action α and turn into a process P' if (P, α, P') is in the labelled transition system. We call this an execution step and write it $P \xrightarrow{\alpha} P'$. A process evolves by executing a succession of steps as follows:

$$P \xrightarrow{\alpha} P' \xrightarrow{\alpha'} P'' \xrightarrow{\alpha''} \cdots \quad .$$

3 ACSR Syntax and Operational Semantics

The following grammar describes the syntax of processes:

$$\begin{array}{rcl} P & ::= & \mathbf{0} \mid A^u{:}P \mid e.P \mid P+P \mid P \,\|\, P \mid \\ & & P \,\triangle_v\, P \mid P \dagger P \mid [P]_U \mid P\backslash F \mid rec\ X.P \mid X \end{array}$$

The process $\mathbf{0}$ executes no action (i.e., it is initially deadlocked). There are two prefix operators, corresponding to the two types of actions. The first, $A^u{:}P$, executes a resource consuming action A for a duration u, and then proceeds to the process P. In this prefix operation, it will be convenient to let the range of u extend over the whole time domain, i.e., the set of real numbers plus infinity. However, the operational semantics is such that a null duration can be ignored, a negative duration corresponds to a deadlock and an infinite duration cannot be exhausted. For the second kind of prefix, $e.P$ 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, αP stands for $\alpha{:}P$ when α is a timed action, and for $\alpha.P$ when α is an instantaneous event. 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 $P \,\|\, Q$ is the parallel composition of P and Q. In addition to these traditional operations we define specialized operators to express real-time behaviors.

The timeout operator $P \,\triangle_v\, Q$ allows the process P to execute for exactly v time units; the execution of P is then abandoned and the execution of Q starts. The exception operator allows the specification of a process that can permanently interrupt another process. In the expression $P \dagger Q$, the execution of the process P can be abandoned at any time in favor of the execution of the process Q. The execution of Q is started in one of three ways: the resources it requires are available, it synchronizes with a parallel process, or P executes an event that synchronizes with a starting event of Q. This last behavior is useful to model the exception construct of modern programming languages; it can also be used as sequential composition. The Close operator, $[P]_U$, produces a process P that monopolizes the resources in $U \subseteq Res$. The Restriction operator, $P\backslash F$, with $F \subseteq \mathcal{L}$, limits the behavior of P. Here, no event whose label, or its complement, is in F is permitted to execute. The process $rec\ X.P$ denotes standard recursion, allowing the specification of infinite behaviors.

In order to lighten the presentation by reducing the number of parenthesis required to unambiguously parse a term, we associate with each operator a binding power. We give to the prefix operators the highest binding power and to the choice operator the lowest, the other operators being of equal binding power, in between choice and prefix. For example, the term:

$$e.A^u{:}P \dagger Q + A^v{:}R \| f.S$$

is to be interpreted as:

$$(((e.(A^u{:}P)) \dagger Q) + ((A^v{:}R) \| (f.S))$$

3.1 Unprioritized Operational Semantics

The first two rules for the prefix operators are *axioms*; i.e., they have premises of *true*. There is one for a timed action and another for an instantaneous event. The third rule states that when a timed action has been completed, the system proceeds with the next possible action or event.

$$\textbf{ActI}\ \frac{-}{e.P \xrightarrow{e} P} \qquad \textbf{ActT}\ \frac{-}{A^u{:}P \xrightarrow{A^{u'}} A^{u-u'}{:}P}\ 0 < u' \leq u \wedge u' < \infty$$

$$\textbf{ActTZ}\ \frac{P \xrightarrow{\alpha} P'}{A^0{:}P \xrightarrow{\alpha} P'}$$

As an example, the process $(a, p).P$ executes the event "(a, p)," and proceeds to P. Alternatively, the process $\{(r_1, p_1), (r_2, p_2)\}^2{:}P$ simultaneously uses resources r_1 and r_2 for a total of two time units before executing P.

Note that there is no transition labelled by A^0. Furthermore, because no operational semantics rule apply, a process of the form $A^u{:}P$ with $u < 0$ has no behavior; it is deadlocked. Finally, the process $A^\infty{:}P$ can never proceed to P since it can only execute actions of finite duration A^u.

ActT is the foundation of the dense time semantics. It states that a timed action can be split arbitrarily into any number of consecutive segments.

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

$$\textbf{ChoiceL}\ \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \qquad \textbf{ChoiceR}\ \frac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'}$$

For example, "$(a,7).P + \{(r_1,3),(r_2,7)\}^{1.2}{:}Q$" may choose between executing the event "$(a,7)$" or the timed action "$\{(r_1,3),(r_2,7)\}^{1.2}$." The first behavior is deduced from rule **ActI**, while the other is deduced from **ActT**.

The Parallel operator provides the basic constructor for concurrency and communication. The first rule, **ParT**, combines two timed transitions.

$$\textbf{ParT} \quad \frac{P \xrightarrow{A_1^u} P', Q \xrightarrow{A_2^u} Q'}{P \parallel Q \xrightarrow{(A_1|A_2)^u} P' \parallel Q'} \qquad (\rho(A_1) \cap \rho(A_2) = \emptyset)$$

Note that timed transitions are truly synchronous, in that the resulting process advances only if both constituents take a step. The condition $\rho(A_1) \cap \rho(A_2) = \emptyset$, which ensures that $(A_1|A_2)$ is defined, mandates that each resource be truly sequential, that is only one process may use a given resource at any instant.

The case where two actions have different timings is implicitly handled by this single rule. Indeed, as we will prove in Theorem 3.1, if a process can perform a transition labelled A^u, it can perform a transition $A^{u'}$ for all $0 < u' \leq u$. Therefore, all the transitions with common time values (up to the shortest duration of the two actions) will be combined by virtue of the rule **ParT**.

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

$$\textbf{ParIL} \quad \frac{P \xrightarrow{e} P'}{P \parallel Q \xrightarrow{e} P' \parallel Q} \qquad\qquad \textbf{ParIR} \quad \frac{Q \xrightarrow{e} Q'}{P \parallel Q \xrightarrow{e} P \parallel Q'}$$

$$\textbf{ParCom} \quad \frac{P \xrightarrow{(a,p)} P', Q \xrightarrow{(\bar{a},q)} Q'}{P \parallel Q \xrightarrow{(\tau,p+q)} P' \parallel Q'}$$

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 a, while Q executes an event with the inverse label $\bar{a}$. This model allows sequences of events to occur at the same instant in time. This is useful to express causality relations between events that no measurable amount of time separate.

When two events synchronize, their resulting priority is the sum of their constituent priorities. Example 3.6 illustrates why we find it useful to allow events with different priorities to synchronize together. The choice of using the sum of the synchronizing events for the resulting priority was dictated by mathematical considerations that are explained in Section. 3.2.2.

Example 3.1 Consider the following two processes:

$$\begin{aligned} P &\overset{\text{def}}{=} (a,3).P_1 \;+\; \{(r_3,8)\}^2{:}P_2 \\ Q &\overset{\text{def}}{=} (\bar{a},5).Q_1 \;+\; \{(r_1,7)\}^3{:}Q_2 \end{aligned}$$

The compound process $P \parallel Q$ admits the following transitions:

$$\begin{array}{rll} P \parallel Q & \xrightarrow{(a,3)} P_1 \parallel Q & \text{[by } \textbf{ParIL}\text{]} \\ P \parallel Q & \xrightarrow{(\bar{a},5)} P \parallel Q_1 & \text{[by } \textbf{ParIR}\text{]} \\ P \parallel Q & \xrightarrow{(\tau,8)} P_1 \parallel Q_1 & \text{[by } \textbf{ParCom}\text{]} \\ P \parallel Q & \xrightarrow{\{(r_1,7),(r_3,8)\}^u} P' \| Q' & \text{[by } \textbf{ParT}\text{]} \end{array}$$

with $P' \overset{\text{def}}{=} \{(r_3,8)\}^{2-u}{:}P_2$, $Q' \overset{\text{def}}{=} \{(r_1,7)\}^{3-u}{:}Q_2$ and $0 < u \leq 2$.

Note than an event transition, if chosen, always executes immediately, i.e., before any time elapses. □

The construction of **ParCom** helps ensure that the *relative* priority ordering among events with the same labels remains consistent even after communication takes places.

The timeout operator possesses three transition rules. The first two rules correspond to timed and untimed transitions occurring before the timeout has expired, i.e., when $v > 0$. The third rule, **TimeoutE** is applied when the timeout expires, i.e., when $v = 0$.

$$\textbf{TimeoutCT} \quad \frac{P \xrightarrow{A^u} P'}{P \bigtriangleup_v Q \xrightarrow{A^u} P' \bigtriangleup_{v-u} Q} \quad (u \leq v)$$

$$\textbf{TimeoutCI} \quad \frac{P \xrightarrow{e} P'}{P \bigtriangleup_v Q \xrightarrow{e} P' \bigtriangleup_v Q} \quad (v > 0)$$

$$\textbf{TimeoutE} \quad \frac{Q \xrightarrow{\alpha} Q'}{P \bigtriangleup_v Q \xrightarrow{\alpha} Q'} \quad (v = 0)$$

The exception operator has three transitions: **ExceptC** corresponds to the continuation of the process P; **ExceptI** is the start of an interrupt due to synchronization with another process or availability of resources; **ExceptE** applies when the process P raises an exception caught by the process Q.

$$\textbf{ExceptC} \quad \frac{P \xrightarrow{\alpha} P'}{P \dagger Q \xrightarrow{\alpha} P' \dagger Q} \qquad\qquad \textbf{ExceptI} \quad \frac{Q \xrightarrow{\alpha} Q'}{P \dagger Q \xrightarrow{\alpha} Q'}$$

$$\textbf{ExceptE} \quad \frac{P \xrightarrow{(a,n)} P',\, Q \xrightarrow{(\bar{a},m)} Q'}{P \dagger Q \xrightarrow{(\tau,n+m)} Q'}$$

Example 3.2 Consider the following specification: Send a message (denoted by the event "sendMsg") and wait until an answer is received. If a response (event "rcvAck") is received within 100 time units execute the process Q, otherwise start over. This system may be realized by the process P defined recursively.

$$P \stackrel{\text{def}}{=} ((sendMsg, 1).\emptyset^{100}:P) \dagger (rcvAck, 2).Q$$

□

The exception operator, along with the infinite execution of the empty action "$\emptyset^{\infty}$" allow us to define an indefinite delay operator δ, for which we use a prefix notation:

$$\delta P \stackrel{\text{def}}{=} \emptyset^{\infty}:\mathbf{0} \dagger P$$

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, $F \subseteq \mathcal{L}$, and deriving only those behaviors that do not involve events with those labels or their complement. Timed actions, on the other hand, remain unaffected.

$$\textbf{ResT} \quad \frac{P \xrightarrow{A^u} P'}{P\backslash F \xrightarrow{A^u} P'\backslash F} \qquad\qquad \textbf{ResI} \quad \frac{P \xrightarrow{(a,n)} P'}{P\backslash F \xrightarrow{(a,n)} P'\backslash F} \quad (a, \bar{a} \notin F)$$

Example 3.3 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 $P \,\|\, Q$ has transitions labelled with a and $\bar{a}$. On the other hand, $(P \,\|\, Q)\backslash\{a\}$ has only the transitions:

$$(P \,\|\, Q)\backslash\{a\} \xrightarrow{(\tau,8)} (P_1 \,\|\, Q_1)\backslash\{a\}$$

and

$$(P \,\|\, Q)\backslash\{a\} \xrightarrow{\{(r_1,7),(r_3,8)\}^u} (P'\|Q')\backslash\{a\}$$

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. Embedding a process P in a closed context such as $[P]_U$, ensures that there is no further sharing of the resources in U. Instantaneous events are not affected.

$$\textbf{CloseT} \quad \frac{P \xrightarrow{A^u} P'}{[P]_U \xrightarrow{[A^u]_U} [P']_U} \qquad\qquad \textbf{CloseI} \quad \frac{P \xrightarrow{(a,n)} P'}{[P]_U \xrightarrow{(a,n)} [P']_U}$$

In the context of the prioritized transition system, the Close operator is useful to force progress. A process may have a choice between progressing using some resources, or idling in case some other process requires the same resources at a higher priority. Closure ensures that no other process can compete for the closed resources and therefore those resources can be committed to the action with the highest priority. For example, as we shall see later, the actions $\{(r,5)\}$ and $\emptyset$ are not comparable under the preemption relation — because the high priority action uses more resources than the low priority one. However $[\{(r,5)\}]_{\{r\}}$ and $[\emptyset]_{\{r\}}$ are comparable and the former will preempt the latter.

The operator "$rec\ X.P$" denotes recursion, allowing the specification of infinite behaviors.

$$\textbf{Rec}\quad \frac{P\left[{}^{rec\ X.P}/_{X}\right] \xrightarrow{\alpha} P'}{rec\ X.P \xrightarrow{\alpha} P'}$$

where "$P\left[{}^{rec\ X.P}/_{X}\right]$" is the standard notation for substitution of "$rec\ X.P$" for each free occurrence of X in P.

Example 3.4 Consider the process "$rec\ X.(A^1{:}X)$," which indefinitely executes the resource consuming action A. By **ActT** and **ActTZ**,

$$A^1{:}(rec\ X.(A^1{:}X)) \xrightarrow{A^1} rec\ X.(A^1{:}X)$$

so by **Rec**,

$$rec\ X.(A^1{:}X) \xrightarrow{A^1} rec\ X.(A^1{:}X) \quad .$$

□

We are now in a position to prove the following theorem which, in essence, characterizes dense time.

Theorem 3.1 *If a process P is such that $P \xrightarrow{A^u} P'$ then, for all $0 < v \leq u$ there exists P'' such that $P \xrightarrow{A^v} P''$.*

Proof: By algebraic induction on the structure of processes. It is vacuously true for $\mathbf{0}$, it is true for prefix (from **ActT**), and it is preserved by all other operators. □

3.2 Preemption and Prioritized Transitions

Not all the actions that are ready for execution at a given point in time have the same urgency. It is often the case in real-time systems that the choice made between possible alternative directly impacts the correctness of the system.

In this section we define a relation between ACSR actions that specifies when an action must be preferred over another in a choice; we call this preemption. Based on this relation, we derive a prioritized semantics for ACSR terms in the form of a subset of the labelled transition system in which all preempted transitions have been eliminated. We call this the *prioritized labelled transition system.* Preemption should be applicable regardless of the context, splice, this property is called compositionality. Section 3.2.2 contains a formal definition of compositionality and a proof that it applies to ACSR.

3.2.1 The Preemption Relation

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 "$\prec$", called the *preemption relation*, be a transitive, irreflexive, binary relation over actions. For two actions α and β, if $\alpha \prec \beta$, we say that "α is preempted by β." This means that any real-time system that has a choice between executing either α or β will not execute 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, where

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

ii) $\forall r : \pi_\alpha(r) \leq \pi_\beta(r)$, and

iii) $\exists r : \pi_\alpha(r) < \pi_\beta(r)$

(2) Both α and β are instantaneous events, where $l(\alpha) = l(\beta)$ and $\pi(\alpha) < \pi(\beta)$

(3) α is a timed action, β is an event, with $l(\beta) = \tau$ and $\pi(\beta) > 0$. □

Case (1) applies when 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 no resource at a lower priority level than α and at least one 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 only case in which an event and a timed action are comparable under "$\prec$." That is, when $p > 0$ in an event (τ, p), we let the event preempt any timed action. This, in effect, makes synchronization take precedence over the passage of time and is similar to the notion of maximum progress found in [26, 46, 53]. The case where $p = 0$ is treated differently. It is meant to allow the specification of nondeterministic behaviors, e.g., to model an environment that can interact with a process at any time rather than at the earliest possible time.

Note that the preemption relation is independent of the duration of each action. To see why this is required, assume that the preemption relation be restricted to actions with the same duration, and that $\{(r,1)\}^1 \prec \{(r,2)\}^1$. Then the process

$$P = \{(r,1)\}^2{:}P_1 + \{(r,2)\}^1{:}\{(r,2)\}^1{:}P_2$$

can still take the transition $P \xrightarrow{\{(r,1)\}^2} P_1$. However, Theorem 3.1 tells us that this process may always take an initial "step" of $\{(r,1)\}^1$, which *should be preempted.* This leads to an inconsistency, as do other ways of associating time with preemption.

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. $\{(r_1,2),(r_2,5)\} \prec (\tau,2)$
i. $\{(r_1,2),(r_2,5)\} \not\prec (\tau,0)$
j. $\{(r_1,2),(r_2,5)\} \not\prec (a,2)$ □

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

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

i) $P \xrightarrow{\alpha} P'$ is an unprioritized transition, and

ii) There is no unprioritized transition $P \xrightarrow{\beta} P''$ such that $\alpha \prec \beta$. □

It is straightforward to see that the preemption relation defined above is transitive and irreflexive. This ensures that the prioritized transition system is well-defined. Namely, if an action α preempts an action β, then any action that would have been preempted by β will be preempted by α. In addition, no action will preempt itself.

Example 3.6 This example illustrates the use of synchronization and priorities to model a semaphore. The event label s_p represents the P operation of the semaphore and the event label s_v represents the V operation. The semaphore M is defined as follows:

$$M \overset{\text{def}}{=} \delta\,(\overline{s_p}, 0).\delta\,(\overline{s_v}, 0).M$$

To see how this works, let P_1 and P_2 be two processes that must execute a critical section using two robot arms, $CR = \{(left\_arm, 1), (right\_arm, 1)\}$ followed by a non-critical section, NCR. Assume that the process P_1 has priority 1 and the process P_2 has priority 2.

$$\begin{aligned} P_1 &\overset{\text{def}}{=} \delta\,(s_p, 1).CR{:}(s_v, 1).NCR{:} \\ P_2 &\overset{\text{def}}{=} \delta\,(s_p, 2).CR{:}(s_v, 2).NCR{:} \\ S &\overset{\text{def}}{=} (P_1 \parallel P_2 \parallel M)\backslash\{s_p, s_v\} \end{aligned}$$

Before entering the critical section, each process must execute the event s_p. By applying the rules of the operational semantics, we see that there are only three unprioritized transitions that the system S can take:

$$\begin{array}{llll} (1) & S & \xrightarrow{\emptyset^u} & S \\ (2) & S & \xrightarrow{(\tau,1)} & (CR{:}(s_v,1).NCR \ \parallel P_2 \ \parallel \ \delta\,(\bar{s_v}, 0).M)\backslash\{s_p, s_v\} \\ (3) & S & \xrightarrow{(\tau,2)} & (P_1 \ \parallel \ CR{:}(s_v,2).NCR \ \parallel \ \delta\,(\bar{s_v}, 0).M)\backslash\{s_p, s_v\} \end{array}$$

Only transition (3) remains admitted by the prioritized transition system. This allows P_2 to proceed. From this point and until P_2 executes $(s_v, 2)$, both P_1 and M will have to idle, i.e., execute $\emptyset^u$ transitions matching the duration of CR. The execution of $(s_v, 2)$ by P_2 will release the semaphore and subsequently allows P_1 to acquire it. □

The application of preemption is used to eliminate unwanted transitions from the labelled transition system. It is natural to extend this notion to processes and define that a process is preempted by another if all of its possible transitions are preempted. We will overload the symbol for the preemption relation over actions to denote preemption over processes.

Definition 3.3 (Preemption over Processes) *We say that a process P preempts a process Q, noted $P \succ Q$ if and only if*

$$\forall \beta{:}\ Q \xrightarrow{\beta} \quad \exists \alpha{:}\ P \xrightarrow{\alpha} \ \wedge \ \alpha \succ \beta$$

□

This notion will be useful in the definition of equivalence laws between processes in Section 4.2.

3.2.2 Compositionality of Preemption

It will be important in the syntactical manipulation of ACSR processes to be able to prune out preempted branches as early as possible, without regard to the context. This property is known as compositionality. Namely if two processes P and Q differ only in behaviors that are preempted, the prioritized transitions of any context will not be changed if P is replaced by Q or vice versa.

The action of pruning out transitions that are preempted can be defined using a priority operator, for which we use a prefix notation "θP," from [7], with the following operational semantics:

$$\textbf{Prty} \quad \frac{P \xrightarrow{\alpha} P', \; P \not\xrightarrow{\beta}}{\theta P \xrightarrow{\alpha} \theta P'} \quad \forall \beta \succ \alpha$$

Compositionality can be stated informally as *when the priority operator is applied to a term, the meaning of that term does not change when the priority operation is applied to any of its subterms.*

Before giving a formal definition, let us introduce a notation:

Definition 3.4 *Let "$\preceq_\theta$" be a binary relation such that $P \preceq_\theta Q$ if and only if $P \equiv Q$ or there exists a context $C[\_]$ and a term R such that $P \equiv C[R]$ and $Q \equiv C[\theta R]$.*

Let "$\cong_\theta$" be the reflexive transitive closure of "$\preceq_\theta$." □

In other words, two processes P and Q are equivalent up to θ, noted $P \cong_\theta Q$, when they are syntactically identical, or when they differ only by the introduction of θ operators.

Now for the formal definition of compositionality:

Definition 3.5 (Compositionality) *A priority operator θ is compositional relative to an operational semantics $\longrightarrow$ when, for all contexts $C[\_]$, processes P, P' and actions α:*

$$\textit{If} \;\; \theta C[P] \xrightarrow{\alpha} P' \;\; \textit{then} \;\; \exists P'' \cong_\theta P' \!: \; \theta C[\theta P] \xrightarrow{\alpha} P''$$

and conversely:

$$\textit{If} \;\; \theta C[\theta P] \xrightarrow{\alpha} P' \;\; \textit{then} \;\; \exists P'' \cong_\theta P' \!: \; \theta C[P] \xrightarrow{\alpha} P''$$

□

Of course we intent to prove that the preemption relation that we have introduced in Definition 3.1 is compositional. A direct proof is very long, tedious and does not bring much insight. There is, however a sufficient condition that can be applied to the set of operational rules, in conjunction with the preemption relation, to determine the compositionality of the operator θ.

Lemma 3.1 (Sufficient Condition for Compositionality) *Let $I = \{i_1, \ldots, i_n\}$ and $J = \{j_1, \ldots, j_m\} \subseteq I$ be two index sets. Let $\mathcal{C}(\cdots)$ be a boolean condition, $Op(\cdots)$ an ACSR term and $C[\cdots]$ an ACSR context. If the set of operational semantics rules, excluding those of the θ operator, is such that, whenever for a rule* $\mathbf{R}$ *of the form*

$$\mathbf{R} \quad \frac{\left\{ P_j \xrightarrow{\alpha_j} P'_j \mid j \in J \right\}}{Op(P_{i_1}, \ldots, P_{i_n}) \xrightarrow{f(\alpha_{j_1}, \ldots, \alpha_{j_m})} C[P_{i_1}, \ldots, P_{i_n}, P'_{j_1}, \ldots, P'_{j_m}]} \; \mathcal{C}(\alpha_{j_1}, \ldots, \alpha_{j_m})$$

the condition $\mathcal{C}$ holds (i.e. the rule fires) then for all $\alpha' \succ \alpha_{j_k}$ there is a rule $\mathbf{R}'$, possibly different from $\mathbf{R}$ but with the same premises, such that $\mathcal{C}'(\alpha_{j_1}, \ldots, \alpha', \ldots \alpha_{j_m})$ holds and

$$f'(\alpha_{j_1}, \ldots, \alpha', \ldots \alpha_{j_m}) \succ f(\alpha_{j_1}, \ldots, \alpha_{j_k}, \ldots \alpha_{j_m})$$

then the priority operator θ is compositional.

Proof: By induction on the algebraic structure of $Op(P_{i_1}, \ldots, P_{i_n})$. The details can be found in [17] □

Theorem 3.2 *Preemption based on Definition 3.1 is compositional in the ACSR operational semantics.*

Proof: We prove that the operational semantics of ACSR complies with the hypothesis of lemma 3.1. For every rule and every premise, we need to check that if the rule fires for an action α, resulting in an action α', then for any $\beta \succ \alpha$ there is a rule (often the same) that fires, and the resulting action β' preempts α'. Note that some of the ACSR rules have implicit conditions in the premises. To comply with the form of lemma 3.1 it is straightforward to rewrite these rules so that the premises use the generic action α with the appropriate side condition. For example, a premise $P \xrightarrow{(a,p)} P'$ would be replaced by $P \xrightarrow{\alpha} P'$ and the side condition $\alpha = (a, p)$.

The only non-trivial cases are the operational rules involving the application of an operation over actions, or a condition other than a pure timing condition. (Pure timing conditions satisfy the requirements by virtue of the fact that the preemption relation is independent of time.) The details can be found in [17] □

4 Strong Equivalence

There are processes that are syntactically different but have the same behavior, that is, they can execute the same first step and then become syntactically equal. Such is the case of $P+Q$ and $Q+P$. This equivalence, however is of little use because it is not a congruence — take for example $P \dagger (Q + R)$ and $P \dagger (R + Q)$, after an initial step of P they will not be syntactically equal. This problem is easily solved by requiring that the end-point of the transitions be themselves *equivalent*. Such is the notion of *strong bisimulation,* due to Park [48]. This section defines strong bisimulation as applied to ACSR and provides a sound and complete set of laws that can be used to prove bisimulation between finite state agents through syntactic manipulations.

Bisimulation is too fine for most practical purposes but it seems to be the finest congruence that equates terms that cannot be differentiated by their operational semantics. As such, it is a subset of most other equivalence and preorder relations. Consequently, any law that is sound for strong bisimulation is also sound for most other relations. Strong bisimilarity is a stepping stone towards more useful relations.

4.1 Prioritized Strong Bisimulation

Definition 4.1 *For closed terms, i.e., terms with no free variables, and for a given transition system "$\rightsquigarrow$", any binary relation $\mathcal{R}$ is a* strong bisimulation *if, for $(P,Q) \in \mathcal{R}$ and $\alpha \in Act$,*

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

In other words, if P (or Q) can execute a step α, then Q (or P) must also be able to execute a step α and the two next states are 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 [42, 43, 45], it is straightforward to show that there exists a largest such bisimulation over "$\rightarrow$," which we denote as "$\sim$," and that it is an equivalence relation.

All the operational semantics rules of ACSR, including the priority operator θ follow the format of "GSOS" [2] and "Grand" [14]. It follows from those theories that strong bisimulation is a congruence. Furthermore, [2] gives an algorithm for the development of a sound and complete set of proof rules. In general we will follow this algorithm; we will deviate in a few cases when we found a better set of rules but we will note those deviations.

We note "$\sim_\pi$" the largest bisimulation over "$\longrightarrow_\pi$,". It follows immediately from the definition of the operator θ that $P \xrightarrow{\alpha}_\pi P'$ if and only if $\theta P \xrightarrow{\alpha} \theta P'$ whence $P \sim_\pi Q$ if and only if $\theta P \sim \theta Q$. This ensures the existence and uniqueness of $\sim_\pi$. That $\sim_\pi$ is a congruence follows from the compositionality of θ; indeed, the very definition of compositionality implies that $\theta C[P] \sim \theta C[\theta P]$.

4.2 Equational Laws for Prioritized Bisimulation

In this section, we present a set of algebraic manipulation laws that preserve prioritized strong bisimulation. The idea behind these laws is to be able to transform ACSR processes into some normalized form that can be easily compared. Normalized processes are coded exclusively with prefix and choice operators.

The strategy for building this sets of laws is fairly straight forward and has been described in some details in [2]. For non recursive processes, the basic idea is to eliminate each operation (except choice and prefix) in two steps. First, operations over a summation are transformed into a summation of operations using a distribution law. Second, operations over prefixed processes are transformed into either prefix over an operation (via an action law) or a NIL process via an inaction law. Some operations, however, are not distributive over summation. In those cases we utilize auxiliary operators to effect distributivity. In ACSR, Parallel and Exeption fall in that category. The operational semantics for these operators is presented in the next subsection.

The exception operator poses a unique challenge in dense time because it denotes a choice over a continuous interval of time. We work around this difficulty by introducing a new prefix operation that embodies the concept of continuous choice and replaces the timed action prefix in normalized processes.

The complete set of ACSR laws is given in Tables 1 and 2

By induction on the depth of prefix operations on a term, it is straightforward to prove that this strategy leads to a normal form, in the absence of infinite behavior. Bisimilarity of recursive processes can always be proved via an induction principle, but this method is sometimes difficult to apply. By limiting ourselves to some specific form of process, bisimilarity is always provable with a small set of recursion laws.

We refer to the whole set of ACSR laws as $\mathcal{A}$. In the sequel, we use the equality symbol "=" to mean provable bisimilarity using $\mathcal{A}$. In other words, we use $P = Q$ as a short hand for $\mathcal{A} \vdash P \sim_\pi Q$.

4.3 Distributivity of the Parallel Operator

The parallel operator is not distributive over choice. To work around this problem, inspired by GSOS and ACP [11] we introduce auxiliary operators that are distributive. The *synchronous execution* operation $P \,|\, Q$ forces both P and Q to take a first step simultaneously, either a synchronized-events execution or a combined timed action. The *left-merge* operation "$P \lfloor\!\lfloor Q$" forces P to take the first step while the process Q remains still; the combined process is deadlocked when P can execute only timed actions.

The synchronous execution operator has two operational rules. One for timed actions, **SyncT**, which corresponds exactly to **ParT** and one for events, **SyncI** which corresponds to **ParCom**.

$$\textbf{SyncT} \quad \frac{P \xrightarrow{A^u} P', Q \xrightarrow{B^u} Q'}{P \mid Q \xrightarrow{(A|B)^u} P' \parallel Q'} \quad (\rho(A) \cap \rho(B) = \emptyset)$$

$$\textbf{SyncI} \quad \frac{P \xrightarrow{(a,p)} P', Q \xrightarrow{(\bar{a},q)} Q'}{P \mid Q \xrightarrow{(\tau,p+q)} P' \parallel Q'}$$

There is only one rule for the left merge operator: **LeftM**. If the process on the left of the operator can take an event transition, the left merge process can do the same.

$$\textbf{LeftM} \quad \frac{P \xrightarrow{e} P'}{P \mathbin{\|\!\lfloor} Q \xrightarrow{e} P' \parallel Q}$$

It is worth noting that both left-merge and synchronous execution lead to a term defined using the *parallel* operator.

Unfortunately, compositionality is *not* preserved by the synchronized execution operator. The consequence is that the prioritized strong bisimulation is not a congruence under this extended signature. To see this, take $P \stackrel{\text{def}}{=} (\tau,1).P_1 + A^u{:}P_2$ and $Q \stackrel{\text{def}}{=} B^u{:}Q'$ with $\rho(A) \cap \rho(B) = \emptyset$, and let $P' \stackrel{\text{def}}{=} (\tau,1).P_1$. Obviously $P \sim_\pi P'$, but $(P|Q) \not\sim_\pi (P'|Q)$ since $(P|Q) \xrightarrow{(A|B)^u}_\pi (P_2\|Q')$ while $(P'|Q)$ is deadlocked.

This does not invalidate completely the strategy, simply the elimination of preempted processes by application of the law Choice(5) of Table 1 cannot be used within the context of a sync operator. It is however possible to delay application of Choice(5) until all the sync and left merge operators have been eliminated. More formally, if we call ACSR^{ls} the ACSR signature augmented by left-merge and sync; let $C[\_]$ be a context and P a term over the ACSR signature; let C^{ls} and P^{ls} be a context and a process over ACSR^{ls}. Given that $\mathcal{A}$ is the set of ACSR laws, and using the usual symbols, "$\vdash$" for provability and "$\models$" for truth, we have the following:

$$\begin{aligned} \mathcal{A} \vdash P_1 \sim_\pi P_2 &\implies \models C[P_1] \sim_\pi C[P_2] \\ \mathcal{A} - \{\text{Choice}(5)\} \vdash P_1^{ls} \sim_\pi P_2^{ls} &\implies \models C^{ls}[P_1^{ls}] \sim_\pi C^{ls}[P_2^{ls}] \end{aligned}$$

Since the ACSR signature is a subset of ACSR^{ls} and the set of laws $\mathcal{A} - \{\text{Choice}(5)\}$ is a subset of $\mathcal{A}$, any valid proof over ACSR^{ls} is valid over ACSR. Starting with an ACSR term, application of Par(3) transforms it into an ACSR^{ls} term, from then on, and until all left merge and sync operators have been eliminated, the proof system $\mathcal{A} - \{\text{Choice}(5)\}$ must be used.

4.4 Distributivity of the Exception Operator

The exception operator is not distributive over its second argument, therefore we need to introduce auxiliary operators. In this case, it is a family of unary operators *Guard* indexed over the set of events. Guard allows the process to which it applies to proceed only when its first action is an event which complements the specified event. The operational semantics is as follows:

$$\textbf{Guard} \quad \frac{Q \xrightarrow{(\bar{a}, m)} Q'}{(a,n) \wr Q \xrightarrow{(\tau, n+m)} Q'}$$

In a dense time setting, the exception denotes a continuous choice, i.e., a choice that remains open during an interval of time. It is the only ACSR operator to do so. For that reason it cannot be replaced by any of the operators already defined. The problem gets even more complicated in presence of the parallel operator as illustrated by the following example.

Example 4.1 Consider the following process:

$$P \overset{\text{def}}{=} ((a,n).Q + A^2{:}R) \;\; \| \;\; (B^3{:}S \dagger (b,m).T)$$

Assume that there exists a process P' that has the same transitions as P but is written without the parallel operator. P', or one of its subterms, must have the form $P_1 \dagger P_2$. Before any time has elapsed, the exception P_2 should be:

$$(b,m).(((a,n).Q + A^2{:}R) \,\|\, T)$$

After t time units have elapsed, with $0 < t < 2$ the exception becomes:

$$(b,m).(A^{2-t}{:}R \,\|\, T)$$

After exactly 2 time units, the exception takes the form:

$$(b,m).(R \,\|\, T)$$

□

This example illustrates the need to limit the scope of an exception handling process to the execution time of a single timed action. In addition, it shows that time variables are required to express the exception itself.

Just as we have defined a unary prefix operator for each timed action, we now define a binary operator called *Interruptible Timed Prefix* or ITP, indexed over the set of all timed actions, with the following syntax:

$$A^{t \leq u}\,(P, Q)$$

where A^u is the interruptible action, t is a time variable and P and Q are processes. The operational semantics of is defined ITP such that at any time during the execution of A^u, but not before it has started, the process P can interrupt; the variable t is then bound to the actual starting time of P, relative to the start time of A^u. Therefore, t will always be positive and at most u. If P does not interrupt A^u, then the execution continues with Q. We will refer to the process P and the *interrupt* and the process Q as the *continuation*. When there is no variable to bind, the behavior of P remains constant regardless of its starting time and we write $A^u\,(P,Q)$. This is the case, for example in the following equation that will be justified by the operational semantics and can be derived from the laws ITP(1) and Except(4a).

$$(A^u{:}Q) \dagger P = P + A^u\,(P, Q \dagger P) \tag{1}$$

ITP has two transition rules. **ItpT** states that the process can execute any portion of the timed action, then choose between executing the interrupt P and carrying on with the timed action. The operational rule **ItpZ** specifies that an ITP with zero duration can be ignored. This behavior is consistent with the behavior of the original timed action prefix.

$$\textbf{ItpT} \quad \frac{-}{A^{t\leq u}\,(P,Q) \xrightarrow{A^{u'}} P\left[{}^{u'}/_{t}\right] + A^{t\leq u-u'}\left(P\left[{}^{u'+t}/_{t}\right], Q\right)} \quad 0 < u' \leq u,\ u' < \infty$$

$$\textbf{ItpZ} \quad \frac{Q \xrightarrow{\alpha} Q'}{A^{t\leq 0}\,(P,Q) \xrightarrow{\alpha} Q'}$$

Note that the condition expressed by these rules satisfies the hypothesis of lemma 3.1 and therefore the priority operator remains compositional when ACSR is extended with the ITP operator.

4.5 Normalization of ACSR processes

With ITP we can define a normal form for processes with exception and dense time:

Definition 4.2 (Head Normal Form) *A process P is in head normal form (or HNF) if it has the form:*

$$\sum_{i\in I} e_i.P_i + \sum_{j\in J} A_j{}^{t_j\leq u_j}\,(Q_j, R_j)$$

with all the $u_j > 0$. □

Table 1: The ACSR bisimulation laws

ITP(1)	$A^u : P = A^{t\leq u}(\mathbf{0}, P)$	
ITP(2)	$A^{t\leq u}(P, Q) = \mathbf{0}$	if $P \succ A{:}Q$
ITP(3)	$A^{t\leq 0}(P, Q) = Q$	
ITP(4)	$A^{t\leq \infty}(P, Q) = A^{t\leq\infty}(P, \mathbf{0})$	
ITP(5)	$A^{t\leq u}(P, Q) = \mathbf{0}$	if $u < 0$
Choice(1)	$P + \mathbf{0} = P$	
Choice(2)	$P + P = P$	
Choice(3)	$P + Q = Q + P$	
Choice(4)	$(P + Q) + R = P + (Q + R)$	
Choice(5)	$P + Q = Q$	if $P \prec Q$
Par(1)	$P \Vert Q = Q \Vert P$	
Par(2)	$(P \Vert Q) \Vert R = P \Vert (Q \Vert R)$	
Par(3)	$P \Vert Q = P \mid Q + P \mathbin{\vert\!\lfloor} Q + Q \mathbin{\vert\!\lfloor} P$	
LeftM(1)	$e.P \mathbin{\vert\!\lfloor} Q = e.(P \Vert Q)$	
LeftM(2)	$(A^{t\leq u}(P, Q)) \mathbin{\vert\!\lfloor} R = \mathbf{0}$	if $u > 0$
LeftM(3)	$(P + Q) \mathbin{\vert\!\lfloor} R = (P \mathbin{\vert\!\lfloor} R) + (Q \mathbin{\vert\!\lfloor} R)$	
LeftM(4)	$\mathbf{0} \mathbin{\vert\!\lfloor} R = \mathbf{0}$	
Sync(1)	$(a, p).P \mid (\overline{a}, q).Q = (\tau, p + q).(P \Vert Q)$	
Sync(2)	$(A^{t_1\leq u}(PI, PC)) \mid (B^{t_2\leq v}(QI, QC)) = (A \vert B)^{t\leq w}(RI, RC)$	
	if $\rho(A) \cap \rho(B) = \emptyset$ and $w = \min(u, v)$	
	and $RI = \quad PI\,[{}^{t}/_{t_1}] \mathbin{\vert\!\lfloor} (QI\,[{}^{t}/_{t_2}] + B^{t_2\leq v-t}(QI\,[{}^{t+t_2}/_{t_2}], QC))$	
	$\quad + PI\,[{}^{t}/_{t_1}] \mid (QI\,[{}^{t}/_{t_2}] + B^{t_2\leq v-t}(QI\,[{}^{t+t_2}/_{t_2}], QC))$	
	$\quad + QI\,[{}^{t}/_{t_2}] \mathbin{\vert\!\lfloor} (PI\,[{}^{t}/_{t_1}] + A^{t_1\leq u-t}(PI\,[{}^{t+t_1}/_{t_1}], PC))$	
	$\quad + QI\,[{}^{t}/_{t_2}] \mid (PI\,[{}^{t}/_{t_1}] + A^{t_1\leq u-t}(PI\,[{}^{t+t_1}/_{t_1}], PC))$	
	and $RC = \quad (PI\,[{}^{w}/_{t_1}] + A^{t_1\leq u-w}(PI\,[{}^{w+t_1}/_{t_1}], PC))$	
	$\quad \Vert\ (QI\,[{}^{w}/_{t_2}] + A^{t_2\leq v-w}(QI\,[{}^{w+t_2}/_{t_2}], QC))$	
Sync(3)	$e.P \mid f.Q = \mathbf{0}$	if $l(e) \neq \overline{l(f)}$
Sync(4)	$e.P \mid A^{t\leq u}(Q, R) = \mathbf{0}$	
Sync(5)	$A^{t_1\leq u}(P_1, Q_1) \mid B^{t_2\leq v}(P_2, Q_2) = \mathbf{0}$	
		if $u > 0 \ \wedge\ v > 0 \ \wedge\ \rho(A) \cap \rho(B) \neq \emptyset$
Sync(6)	$P \mid Q = Q \mid P$	
Sync(7)	$(P + Q) \mid R = P \mid R + Q \mid R$	
Sync(8)	$\mathbf{0} \mid R = \mathbf{0}$	
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)	$rec\ X.(P + [X\backslash E]_U) = rec\ X.(P + [P\backslash E]_U])$	

As usual, we define $\sum_{i\in\emptyset} P_i$ to be $\mathbf{0}$. The omission of parenthesis is legitimated by the laws Choice(2) to Choice(4) of Table 1. We also refer to a full normal form, or simply normal form, where all the P_i Q_i and R_i are also in normal form. Processes in normal form are coded exclusively using the prefix and summation operators.

Table 2: The ACSR bisimulation laws (cont.)

Timeout(1)	$\mathbf{0} \,\triangle_v\, Q = \mathbf{0}$	if $v > 0$
Timeout(2)	$(P_1 + P_2) \triangle_v Q = P_1 \triangle_v Q \ + \ P_2 \triangle_v Q$	
Timeout(3)	$(A^{t \leq u}(P, Q)) \triangle_v R = A^{t \leq w}(P \triangle_{v-t} R, Q \triangle_{v-w} R)$	if t is not free in R and $0 < w = \min(u, v)$
Timeout(4)	$e.P \triangle_v Q = e.(P \triangle_v Q)$	if $v > 0$
Timeout(5)	$P \triangle_0 Q = Q$	
Res(1)	$\mathbf{0} \backslash F = \mathbf{0}$	
Res(2)	$(P + Q) \backslash F = (P \backslash F) + (Q \backslash F)$	
Res(3)	$A^{t \leq u}(P, Q) \backslash F = A^{t \leq u}(P \backslash F, Q \backslash F)$	
Res(4)	$((a, n).P) \backslash F = (a, n).(P \backslash F)$	if $a, \bar{a} \notin F$
Res(5)	$((a, n).P) \backslash F = \mathbf{0}$	if $a \in F \vee \bar{a} \in F$
Res(6)	$P \backslash E \backslash F = P \backslash E \cup F$	
Res(7)	$P \backslash \emptyset = P$	
Close(1)	$[\mathbf{0}]_U = \mathbf{0}$	
Close(2)	$[P + Q]_U = [P]_U + [Q]_U$	
Close(3)	$[A^u{:}P]_U = [A]_U^u{:}[P]_U$	
Close(4)	$[e.P]_U = e.[P]_U$	
Close(5)	$[[P]_U]_V = [P]_{U \cup V}$	
Close(6)	$[P]_\emptyset = P$	
Close(7)	$[P \backslash E]_U = [P]_U \backslash E$	
Except(1)	$\mathbf{0} \dagger Q = Q$	
Except(2)	$P \dagger \mathbf{0} = P$	
Except(3)	$(P + Q) \dagger R = P \dagger R \ + \ Q \dagger R$	
Except(4)	$A^{t \leq u}(P, Q) \dagger R = R + A^{t \leq u}((P + R), Q \dagger R)$	if t is not free in R
Except(5)	$e.P \dagger Q = Q + e.(P \dagger Q) + (e \wr Q)$	
Except(6)	$(P \dagger Q) \dagger R = P \dagger (Q \dagger R)$	
Except(7)	$P \dagger Q = Q + P \dagger Q$	
Guard(1)	$(a, p) \wr (\bar{a}, q).Q = (\tau, p + q).Q$	
Guard(2)	$e \wr f.Q = \mathbf{0}$	if $l(e) \neq \overline{l(f)}$
Guard(3)	$e \wr A^{t \leq u}(P, Q) = \mathbf{0}$	
Guard(4)	$e \wr (P + Q) = (e \wr P) + (e \wr Q)$	
Guard(5)	$e \wr \mathbf{0} = \mathbf{0}$	

Example 4.2 Using the ACSR laws we can transform the term of Example 4.1 into an equivalent term in head normal form. By ITP(1) we obtain:

$$P = ((a, n).Q + A^2(\mathbf{0}, R)) \ \| \ (B^3(\mathbf{0}, S) \dagger (b, m).T)$$

Using Except(4a) gives:

$$P = ((a, n).Q + A^2(\mathbf{0}, R)) \ \| \ ((b, m).T + B^3((b, m).T, \ S \dagger (b, m).T))$$

By Par(3) and the distributivity laws LeftM(3) and Sync(7) we obtain:

$$\begin{array}{rll}
P = & (a,n).Q \mathbin{|\!\lfloor} ((b,m).T + B^3((b,m).T,\ S \dagger (b,m).T)) & (a)\\
+ & A^2(\mathbf{0},R) \mathbin{|\!\lfloor} ((b,m).T + B^3((b,m).T,\ S \dagger (b,m).T)) & (b)\\
+ & (a,n).Q \mid (b,m).T & (c)\\
+ & (a,n).Q \mid B^3((b,m).T,\ S \dagger (b,m).T) & (d)\\
+ & A^2(\mathbf{0},R) \mid (b,m).T & (e)\\
+ & A^2(\mathbf{0},R) \mid B^3((b,m).T,\ S \dagger (b,m).T) & (f)\\
+ & (b,m).T \mathbin{|\!\lfloor} ((a,n).Q + A^2(\mathbf{0},R)) & (g)\\
+ & B^3((b,m).T,\ S \dagger (b,m).T) \mathbin{|\!\lfloor} ((a,n).Q + A^2(\mathbf{0},R)) & (h)
\end{array}$$

On the terms (a) and (g) we apply LeftM(1), on the terms (b) and (h) we apply LeftM(2a); on (c) we apply Sync(3) then LeftM(4) and Sync(8); on (d) and (e) we apply Sync(4a) and on (f) we apply Sync(2a). Finally, by Choice(1) we can eliminate $\mathbf{0}$ terms to obtain the head normal form:

$$\begin{array}{ll}
P = (a,n).(Q \parallel ((b,m).T + B^3((b,m).T,\ S \dagger (b,m).T))) & (a)\\
\quad + (A|B)^{t\leq 2}((b,m).T \mathbin{|\!\lfloor} A^{2-t}(\mathbf{0},R),\ R \| ((b,m).T + B^1((b,m).T,\ S \dagger (b,m).T))) & (f)\\
\quad + (b,m).(T \parallel ((a,n).Q + A^2(\mathbf{0},R))) & (g)
\end{array}$$

Note the introduction of the variable t. It is used to capture how much of the action A^2 has been executed when the event (b,m) occurs. □

4.6 Soundness of the Bisimulation Laws

Theorem 4.1 *The ACSR laws of tables 1 and 2 are sound with respect of bisimulation equivalence.*

The traditional way of proving the soundness of a bisimulation equational law has been to identify a bisimulation that relates the two sides of the equation. A more systematic approach consists of characterizing and comparing the set of transitions (i.e., pairs label-endpoint) both sides of the equation can take. To facilitate this process we define two functions $\mathcal{T}: Proc \rightarrow \mathbb{P}(Act \times Proc)$ and $\mathcal{T}_\pi: Proc \rightarrow \mathbb{P}(Act \times Proc)$ by

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

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 that applies. This leads to the set of equations of Table 3, where the operational rule applied to calculate each term is shown in brackets.

The proof of soundness of some typical laws we can be found in Appendix A. Most of the laws are proved using the equations of Table 3 to compare the value of the $\mathcal{T}$ or $\mathcal{T}_\pi$ function and apply either of the following two lemmas.

Table 3: The function $\mathcal{T}$ yields the set of transitions of a process

$\mathcal{T}(\mathbf{0})$	$=$	$\emptyset$	
$\mathcal{T}(e.P)$	$=$	$\{\langle e, P\rangle\}$	[ActI]
$\mathcal{T}(A^{u>0}{:}P)$	$=$	$\{\langle A^{u'}, A^{u-u'}{:}P\rangle \mid 0 < u' < u\}$	[ActT]
$\mathcal{T}(A^{0}{:}P)$	$=$	$\mathcal{T}(P)$	[ActTZ]
$\mathcal{T}(P+Q)$	$=$	$\mathcal{T}(P) \cup \mathcal{T}(Q)$	[ChoiceL and ChoiceR]
$\mathcal{T}(P \Vert Q)$	$=$	$\{\langle (A \mid B)^u, P' \Vert Q'\rangle \mid \langle A^u, P'\rangle \in \mathcal{T}(P) \wedge \langle B^u, Q'\rangle \in \mathcal{T}(Q)$	
		$\wedge\ \rho(A) \cap \rho(B) = \emptyset\}$	[ParT]
		$\cup\ \{\langle e, P' \Vert Q\rangle \mid \langle e, P'\rangle \in \mathcal{T}(P)\}$	[ParIL]
		$\cup\ \{\langle e, P \Vert Q'\rangle \mid \langle e, Q'\rangle \in \mathcal{T}(Q)\}$	[ParIR]
		$\cup\ \{\langle (\tau, p+q), P' \Vert Q'\rangle \mid \langle (a,p), P'\rangle \in \mathcal{T}(P)$	
		$\wedge\ \langle (\bar{a}, q), Q'\rangle \in \mathcal{T}(Q)\}$	[ParCom]
$\mathcal{T}(P \vartriangle_0 Q)$	$=$	$\mathcal{T}(Q)$	[TimeoutE]
$\mathcal{T}(P \vartriangle_{v>0} Q)$	$=$	$\{\langle A^u, P' \vartriangle_{v-u} Q\rangle \mid \langle A^u, P'\rangle \in \mathcal{T}(P) \wedge u \leq v\}$	[TimeoutCT]
		$\cup\ \{\langle e, P' \vartriangle_v Q\rangle \mid \langle e, P'\rangle \in \mathcal{T}(P)\}$	[TimeoutCT]
$\mathcal{T}(P \dagger Q)$	$=$	$\mathcal{T}(Q)$	[ExceptI]
		$\cup\ \{\langle \alpha, P' \dagger Q\rangle \mid \langle \alpha, P'\rangle \in \mathcal{T}(P)\}$	[ExceptC]
		$\cup\ \{\langle (\tau, p+q), Q'\rangle \mid \exists a, P' : \langle (a,p), P'\rangle \in \mathcal{T}(P)$	
		$\wedge\ \langle (\bar{a}, q), Q'\rangle \in \mathcal{T}(Q)\}$	[ExceptE]
$\mathcal{T}(P \backslash E)$	$=$	$\{\langle A^u, P' \backslash E\rangle \mid \langle A^u, P'\rangle \in \mathcal{T}(P)\}$	[ResT]
		$\cup\ \{\langle (a,p), P' \backslash E\rangle \mid \langle (a,p), P'\rangle \in \mathcal{T}(P) \wedge a, \bar{a} \notin E\}$	[ResI]
$\mathcal{T}([P]_U)$	$=$	$\{\langle [A^u]_U, [P']_U\rangle \mid \langle A^u, P'\rangle \in \mathcal{T}(P)\}$	[CloseT]
		$\cup\ \{\langle e, [P']_U\rangle \mid \langle e, P'\rangle \in \mathcal{T}(P)\}$	[CloseI]
$\mathcal{T}(\mathit{rec}\ X.P)$	$=$	$\mathcal{T}(P[\mathit{rec}\ X.P/X])$	[Rec]
$\mathcal{T}(P \mid Q)$	$=$	$\{\langle (A \mid B)^u, P' \Vert Q'\rangle \mid \langle A^u, P'\rangle \in \mathcal{T}(P) \wedge \langle B^u, Q'\rangle \in \mathcal{T}(Q)$	
		$\wedge\ \rho(A) \cap \rho(B) = \emptyset\}$	[SyncT]
		$\cup\ \{\langle (\tau, n+m), P' \Vert Q'\rangle \mid \langle (a,n), P'\rangle \in \mathcal{T}(P)$	
		$\wedge\ \langle (\bar{a}, m), Q'\rangle \in \mathcal{T}(Q)\}$	[SyncI]
$\mathcal{T}(P \mathbin{\lfloor\!\lfloor} Q)$	$=$	$\{\langle e, P' \Vert Q\rangle \mid \langle e, P'\rangle \in \mathcal{T}(P)\}$	[LeftM]
$\mathcal{T}((a,p) \wr Q)$	$=$	$\{\langle (\tau, p+q), Q'\rangle \mid \langle (\bar{a}, q), Q'\rangle \in \mathcal{T}(Q)\}$	[Guard]
$\mathcal{T}(A^{t \leq u}(P,Q))$	$=$	$\{\langle A^{u'}, P\left[{}^{u'}/_{t}\right] + A^{t \leq u-u'}\left(P\left[{}^{u'+t}/_{t}\right], Q\right)\rangle \mid 0 < u' \leq u\}$	[ItpT]
		(When $u > 0 \wedge \nexists \langle \alpha, P'\rangle \in \mathcal{T}(P)$: $\alpha \succ A$)	
$\mathcal{T}(A^{t \leq u}(P,Q))$	$=$	$\emptyset$ (When $u > 0 \wedge \exists \langle \alpha, P'\rangle \in \mathcal{T}(P)$: $\alpha \succ A$)	
$A^{t \leq 0}(P,Q)$	$=$	$\{\langle \alpha, Q'\rangle \mid \langle \alpha, Q'\rangle \in \mathcal{T}(Q)\}$	[ItpZ]

Lemma 4.1

$$\mathcal{T}(P) = \mathcal{T}(Q) \implies \mathcal{T}_\pi(P) = \mathcal{T}_\pi(Q) \implies P \sim_\pi Q$$

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

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

And therefore $\mathcal{T}(P) = \mathcal{T}(Q) \implies \mathcal{T}_\pi(P) = \mathcal{T}_\pi(Q)$.

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

$$\text{if } \mathcal{T}_\pi(P) = \mathcal{T}_\pi(Q) \text{ then } \left\{ \begin{array}{l} \forall \alpha\colon\ P \xrightarrow{\alpha}_\pi P' \implies Q \xrightarrow{\alpha} P', \text{ and} \\ \forall \alpha\colon\ Q \xrightarrow{\alpha}_\pi Q' \implies P \xrightarrow{\alpha} Q' \end{array} \right.$$

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

Lemma 4.2 *If "$\sim$" is a bisimulation and $\mathcal{R}$ is a relation such that all the pairs $(P, Q) \in \mathcal{R}$ are such that*

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

and

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

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

Proof: Follows directly from the definitions of the strong bisimulation and of the functions $\mathcal{T}$ and $\mathcal{T}_\pi$ and the fact that the union of two bisimulations is a bisimulation. □

4.7 Completeness for Recursive Processes

There are two ways to handle the recursion operator. The first one is an induction principle. This says that if two processes are bisimilar in all their finite approximations, then they are bisimilar. This law is sound for ACSR but it is sometimes very hard to apply in practice.

The second approach is to limit the scope to finite state agents and use the three Rec laws of Table 1. Rec(1) is the straightforward unrolling of the recursion. Rec(2) is the unique solution to guarded equations. Finally Rec(3) allows the elimination of unguarded variables.

4.7.1 Characterization of *FS* Processes

The definition of "finite state agents." that previous authors have used, such as [44] and [15], 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 infinite state agents can be generated even without the use of the parallel operator as is illustrated by the following example.

Example 4.3 Consider the process $P \stackrel{\text{def}}{=} rec\ X.(A{:}X \dagger B{:}\mathbf{0}))$. It has two possible transitions:

$$P \xrightarrow{B} \mathbf{0}$$

and

$$P \xrightarrow{A} (rec\ X.A{:}X \dagger B{:}\mathbf{0}) \dagger B{:}\mathbf{0}$$

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

$$P' \xrightarrow{B} \mathbf{0}$$

$$P' \xrightarrow{B} \mathbf{0} \dagger B{:}\mathbf{0}$$

and

$$P' \xrightarrow{A} ((rec\ X.(A{:}X \dagger B{:}\mathbf{0})) \dagger B{:}\mathbf{0}) \dagger B{:}\mathbf{0}$$

and so forth as shown in Figure. 1. □

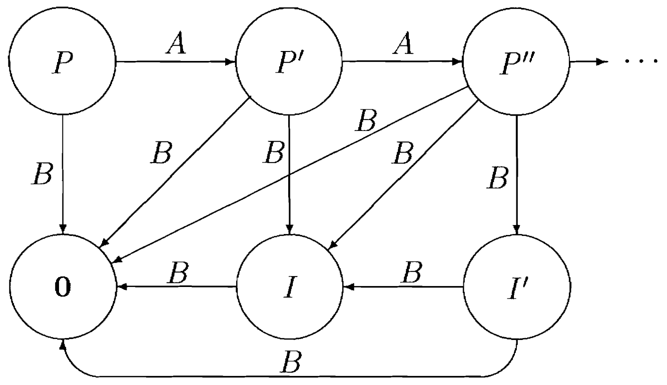

Figure 1: An infinite state agent

A way to ensure finite-state is to require that processes do not have recursion *through* parallel, timeout or exception. Unfortunately, this is very difficult to characterize syntactically — for example, the process $rec\ X.(A.X \parallel \mathbf{0})$ is equivalent to $\mathbf{0}$ and therefore does not actually have recursion through parallel, while the process $rec\ X.(e.X \parallel \mathbf{0})$ does. Nevertheless there are obvious advantages to a syntactic characterization and therefore we limit our proof to processes that have "no free variable in a process under parallel, exception or timeout operators." We say that such processes are "FS." It seems that most finite-state agents are either *FS* processes or are provably equivalent to an *FS* process.

We use an auxiliary predicate, *fs* to characterize *FS* processes. (We assume the usual definition of the function $fv(P)$ which yields the set of free variables of a process P.)

$$\begin{array}{rcl}
fs(\mathbf{0}) & = & true \\
fs(X) & = & true \\
fs(\alpha P) & = & fs(P) \\
fs(P+Q) & = & fs(P) \wedge fs(Q) \\
fs(P \bigtriangleup_t Q) & = & (fv(P) = \emptyset) \wedge fs(P) \wedge fs(Q) \\
fs(P \dagger Q) & = & (fv(P) = \emptyset) \wedge fs(P) \wedge fs(Q) \\
fs(P \parallel Q) & = & (fv(P) = fv(Q) = \emptyset) \wedge fs(P) \wedge fs(Q) \\
fs([P]_U) & = & fs(P) \\
fs(P \backslash F) & = & fs(P) \\
fs(rec\ X.P) & = & fs(P)
\end{array}$$

Definition 4.3 (*FS* **Process)** *A process is said to be FS if* $fs(P) = true$. □

In a discrete time setting, the set of *FS* processes would in fact be a very large subset of the finite state agents. In dense time, however, only processes encoded without timed action can truly be finite state, but the syntactical definition given given above remains useful to describe the set of processes for which we can prove equivalence.

4.7.2 Bisimulation and Free Variables

The presence of recursion will require us to have a formal treatment for free variables. In particular, we need a definition of bisimulation that takes the presence of free variables into account. In [44], 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, $X \backslash E$ 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 " $\longrightarrow$ " (without label) as the minimum relation that satisfies the rules in Table 4. Note that this definition is valid because of the soundness of the laws

Res(5), Res(6), Close(5), Close(6) and Close(7). Based on this, we can define the notion of bisimulation that we will be using throughout this section.

Table 4: Unguarded Variables

$\dfrac{-}{X \longrightarrow [X\backslash\emptyset]_\emptyset}$	$\dfrac{Q \longrightarrow [X\backslash E]_U}{P\dagger(Q) \longrightarrow [X\backslash E]_U}$
$\dfrac{P \longrightarrow [X\backslash E]_U}{P+Q \longrightarrow [X\backslash E]_U}$	$\dfrac{P \longrightarrow [X\backslash E]_U}{P \parallel Q \longrightarrow [X\backslash E]_U}$
$\dfrac{Q \longrightarrow [X\backslash E]_U}{P+Q \longrightarrow [X\backslash E]_U}$	$\dfrac{Q \longrightarrow [X\backslash E]_U}{P \parallel Q \longrightarrow [X\backslash E]_U}$
$\dfrac{P \longrightarrow [X\backslash E]_U}{P \,\Delta_t\, Q \longrightarrow [X\backslash E]_U} \; t > 0$	$\dfrac{P \longrightarrow [X\backslash E]_U}{[P]_V \longrightarrow [X\backslash E]_{U\cup V}}$
$\dfrac{Q \longrightarrow [X\backslash E]_U}{P \,\Delta_t\, Q \longrightarrow [X\backslash E]_U} \; t = 0$	$\dfrac{P \longrightarrow [X\backslash E]_U}{P\backslash F \longrightarrow [X\backslash(E \cup F)]_U}$
$\dfrac{P \longrightarrow [X\backslash E]_U}{P\dagger(Q) \longrightarrow [X\backslash E]_U}$	$\dfrac{P \longrightarrow [X\backslash E]_U}{rec\, Y.P \longrightarrow [X\backslash E]_U} \; X \neq Y$

Definition 4.4 *A process P is bisimilar to a process Q, noted $P \sim_\pi Q$, if, for all $\alpha \in Act$, $U \subseteq Res$ 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. *$P \longrightarrow [X\backslash E]_U$ iff $Q \longrightarrow [X\backslash E]_U$* □

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.

4.8 Completeness

Theorem 4.2 *The set of ACSR laws presented in Tables 1 and 1 are complete to prove bisimilarity of any ACSR processes in discrete time and ACSR processes coded without the exception operator in dense time.*

The proof of completeness, whose details can be sound in [17], follows the scheme described in [44]. We first prove that all the unguarded recursions can be eliminated by application of the law Rec(3). In the absence of unguarded recursion, any *FS* process proven to be the solution of a set of normalized kind of equation. 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 a bisimulation.

This proof applies equally to a discrete time setting as well as a dense time setting, in the absence of the exception operator. We conjecture that the completeness result also holds for dense time with the exception operator, but the proof is complicated by the introduction of time variables.

5 Related Research

The formal specification of real-time systems is a very active field of research. Most of the work can be classified in three main categories: timed logics, automata theory and process algebra.

In methods based on timed logics, systems are described by a set of assertions and properties are theorems. A property holds for a system if it can be logically inferred from the assertions. Such methods do not have an execution model *per se* and therefore they do not directly lead to an implementation. Temporal logic [49] views a program as a sequence of states and allows the expression of logical formulae relating those states. New quantifiers such as $\Box$ (for all states) and $\Diamond$ (for some future state) provide the capability of specifying invariance and eventuality and generally reason about time in a qualitative fashion. A quantitative notion of time can be introduced by allowing the specification of time bounds with the eventuality and invariance quantifiers [36, 35, 34, 3]. Another approach is to introduce a mechanism to access the value of a real-time clock; in [37, 1] it is read from a state variable; in [31, 32, 21] it is denoted by a predicates; and in [5, 6] it is bound by a new quantifier called "freeze."

Finite state automata have been used extensively in the specification and analysis of reactive systems. Several attempts have been made to extend their usage to real-time systems. Modechart [30] is a graphical language for the formal specification of the behavior of real-time systems. It is a hierarchical representation of finite state machines. State transitions are the consequences of event occurrences and timing constraints. The semantics of modecharts can be expressed as a set of events with their time of occurrence (timed traces) or as a Real-Time Logics formula [30]. In Timed Automata [4, 3] a set of clocks is associated to a traditional (untimed) automaton; these clocks can be tested and reset with each transition. The semantics of such automaton is the language it accepts, which is a set of timed traces. Hierarchical Multi-State Machines, or HMS [24, 22] is an extension of finite state automata where multiple states can be active at the same time, and multiple transitions can occur simultaneously. Transitions are controlled (enabled or disabled) based on temporal and state constraints. In addition, states can hold and pass tokens, and HMS machines can be organized hierarchically. Timed I/O Automata are based on *input-output automata* model [38]. An I/O automaton is defined over an alphabet of actions, by a set of states, a set of start states and a set of transitions. I/O automata are *input enabled* which means that any input is accepted in any state and will cause a transition, possibly to the same state. In [39] states are assigned a time stamp and time passage is denoted by time passing transitions. Timed automata can be composed by action transducers [52]. These offer a much wider variety of compositions than simple parallel composition.

Process algebras have been successfully used to specify untimed distributed systems. Many extensions have been introduced to extend their application to real-time environments. We distinguish between the algebras based on CSP [28] with a denotational semantics, those based on CCS [45] whose semantics is typically given operationally and ACP [10, 11] which is defined as an algebraic theory.

The algebra of Communicating Sequential Processes, or CSP [27, 28], was introduced for the formalization and mathematical treatment of concurrent systems. The syntax of CSP includes prefix operators to denote actions to be executed, external choice to allow interaction with an environment and internal choice to model nondeterminism. There is a parallel operator that also enforces synchronization. In addition, CSP provides operators for abstraction and renaming of actions. The semantics of CSP is given as an algebraic theory and there are a number of models used to provide a denotational semantics.

Real-time is introduced into CSP by means of a delay operation that can be a separate operator [50, 25] or combined with the action prefix [55]. The semantic models are usually based on timed traces, that is, a trace where each action is associated a time stamp. Timed traces do not adequately capture nondeterministic behaviors and therefore additional information is attached such as refusals [50], failures [25] or acceptances [55].

A CSP like process algebra is defined in [54] with an operator, **claim** t which denotes the exclusive usage of a processor for t time units, as opposed to the operator **delay** t which denotes the idling of a process, for t time units. The semantics of processes is given in duration calculus, an extension to interval temporal logic. Using duration calculus, several scheduling algorithms such as first-ready-first-run or fair time-slicing can be specified. The intent of this work is to answer the question, given a set of processes, what scheduling algorithm will satisfy a particular system specification.

Communicating Concurrent Processes, or CCS, is a process algebra that introduces the notion of communication through the execution of complementary actions, which are then converted into an internal action. The semantics of CCS is given by a labelled transition system, and the interpretation of the parallel operator is interleaving. Equivalence in CCS is based on the notion of bisimulation.

There has been many extensions to CCS to accommodate real-time. Most of them simply add a time passing action which is assimilated to idling. Actions, on the other hand, are instantaneous and the semantics of the parallel operator is interleaving, as in untimed CCS. In most cases, the parallel and choice operators are patient with regard to time, that is, if the two arguments of the operator can let time elapse the combined process can let time elapse without committing to a particular behavior. Another common notion is that of maximal progress [26, 46, 53] by which if two parallel processes can communicate, this communication occurs as early as possible.

Temporal CCS (or TCCS) [46] which CCS not only with time passing actions but also with a weak choice operator. Both operators are patient with regard to the passage of time, but while the choice operator (also referred to as *strong* choice) deadlocks if one of the branches refuses to wait, the weak choice operator will commit to one branch if the other is not willing to wait. The weak choice operator can be used to build more sophisticated constructs such as timeout. The operational semantics of TCCS is given via two transition systems, one for time passage and another for instantaneous actions. ℓTCCS is a subset of TCCS in which each process can be delayed for any amount of time. This allows the definition of a preorder that implements the intuitive notion of a process being faster than another. This relation turns out to be a precongruence and admit a sound proof system which is complete for the subset of the language that excludes the parallel operator.

The salient aspect of Timed CCS [18] is that time variables are explicitly included. An action is associated lower and upper bounds for its execution and a time variable which is bound to the actual time when the action occurs. Here, the choice operator is patient with regard to elapsed time as long as both processes can wait; after that, if delay is still required (e.g., the other branch cannot synchronize), then the branch that can wait is taken. This behavior is similar to the weak choice of Temporal CCS. The semantics of Timed CCS is defined in terms of a labelled transition system indexed over the time domain. That is, a transition is defined as $P \xrightarrow{\alpha}{}_t P'$ where P can perform an action α at time t. Idling is denoted by a transition without label, only the subscript denotes the elapsed time. Equivalence for Timed CCS is defined as strong or weak bisimulation. There is an alternate characterization based on modal logics. Two finite image (i.e., finite state and finitely branching) processes are equivalent if and only if they satisfy the same set of formulae.

A version of CCS with priorities is found in [19, 20]. Each action is assign one of two priority levels and only actions of the same level can synchronize; they then turn into τ-action of the same priority. Operators to change the priority of a process are also defined.

The authors show that, in order for strong bisimulation to be a congruence, only high priority τ actions can preempt low priority actions, i.e., prevent them from occurring. We have given a formal explanation for this fact in our treatment of compositionality, in Section 3.2.2.

The Algebra of Communicating Processes, ACP [10, 11] differs from CSP and CCS in some interesting ways. First, actions are considered as processes and are combined by sequential composition, instead of being used as prefix operators. This allows the definition of processes whose behavior is described as a regular expressions. For example, the behaviors of the process "$x \stackrel{\text{def}}{=} ab + axb$" is to execute any finite number of actions "a" followed by the same number of "b." Communication is defined as the result of a binary operation on processes which yields another process (a generalization of the τ action of CCS) when communication is possible, or a deadlock when the two processes cannot communicate. The semantics of ACP is given by an equational theory. Infinite behaviors are defined as the solution of process equations.

ACPρ [9] is a generalization of ACP where all actions can be assigned a time stamp. Time stamps can be absolute or relative. Absolute time stamps require the introduction of time variables in the recursive definition of processes. For example, the process "$x(t) \stackrel{\text{def}}{=} a(t).x(t+1)$" performs an action "$a$" at every time unit. Integration is also used to specify a process that can execute an action at any time within an interval: the process "$x \stackrel{\text{def}}{=} \int_{v\in[1,2]} a(v)$" can execute the action a as early as time 1, as late as time 2, or at any time in between. ACPρ, like ACP, is defined as an equational theory. However, it can be given an operational semantics where both processes and transitions are assigned a time stamp. For example, "$\langle a(2)x, 1\rangle \xrightarrow{a(2)} \langle x, 2\rangle$" denotes that a process that has the form $a(2)x$ at time 1, can execute the action a at time 2 and thus become the process x. Strong bisimulation equivalence can be defined on this transition system and the equational theory is sound and complete with respect to it. Interestingly enough, even though ACPρ is a generalization of ACP and therefore has weaker axioms, the original axioms can be recovered if all the actions take the form $\int_{v\in(0,\infty)} a(v)$.

ACP is added a priority operator θ in [7], This work differs from ours in the sense that application of priority must be explicitly expressed in the syntax, while in our case it is implicit. Nevertheless, our treatment of compositionality was inspired by it. In [8] it is shown that some equivalence relations such as ready and failure equivalence are no longer congruences when priorities are introduced.

Algebra of Timed Processes or ATP [47] is another process algebra with discrete time. The execution model is similar to ours in that processes evolve in two-phase steps; in the first phase, all instantaneous actions are executed in an asynchronous (interleaved) manner with some possible communication. When no more component can execute any instantaneous action, time passes synchronously in all the components via the execution of the timed action "$\mathcal{X}$." Unlike other algebras, $\mathcal{X}$ is not used as a prefix but is the result of a *unit-delay* operator which is similar to one time unit timeout. Other operators allow the specification of arbitrary delays and timeouts. ATP is defined by an operational semantics; it has an axiomatization which is sound and complete with regard to strong bisimulation.

RTSL (Real-Time Specification Language) [23] couples a real-time process algebra with a global priority function. The behavior of processes is specified by algebraic terms. There are constructs to specify timing constraints and deadlines. The priority function returns the set of highest priority processes at each execution time. A reachability analysis allows the detection of failure states. The separation of the priority function from process expressions makes it easy to test the effectiveness of various scheduling algorithm.

6 Conclusion

We have developed a formal, algebraic method for the specification and verification of distributed real-time systems. ACSR differs from most other process algebras in that it distinguishes between timed actions that consume resources, and instantaneous events that are used for synchronization. In addition, it features specialized operators to specify real-time behaviors, including timeout and exception constructs. Priorities are assigned to give an action and an event a measure of its urgency. The execution model of ACSR ensures that the most urgent actions are executed first. The dense time domain used in the model provides a versatile way of specifying durations without being tied to particular time base.

Preemption defines when a less urgent action can be ignored in favor of a more urgent one. It is important that preemption be compositional, that is, when an action preempts another, no ACSR context would prefer the preempted action. We have given a formal way to ensure the compositionality of a particular preemption scheme.

ACSR can adequately be used to specify fairly complex real-time systems. There are, however, some aspects of the model that could be improved upon. The first one concerns the fact that ACSR actions are monolithic, that is, once started, an action must either be executed to completion without relinquishing its resources or completely abandoned if a timeout or interrupt occurs. Points where an action may be suspended in favor of a more urgent process (such as a hardware device service interrupt) and later resumed have to be explicitly specified through a delay operator (δ). This behavior is necessary to adequately model processes that can capture their resources (by disabling interrupts for example) and non-preemptive scheduling systems. In the other cases, it is difficult in ACSR to define patient actions that can be suspended at almost any time.

We have defined ACSR with static priorities. This is a necessary step in the understanding of the formal treatment priorities. Nevertheless, many actual systems use dynamic priority schemes such as first-in-first-out or earliest-deadline-first. One way to support such schemes would be to provide a mechanism for the priority function (π) to get timing information about the current execution (such as a relative time of occurrence of certain events) and adjust its value accordingly.

ACSR is an algebraic language and as such is very terse and easy to treat formally. Its terseness, however, may not be very appealing to many practitioners. A coat of syntactic sugar should be applied to ACSR to give it the readability and intuitiveness of a high level language.

Equivalence between ACSR processes is defined as strong bisimulation. This is a very fine equivalence relation; it differentiates between terms that would often be considered equivalent in practice. There are other equivalence relations such as failure equivalence [8] and ready simulation equivalence [13] that are less discriminating. Unfortunately, as shown in [8] these relations are not congruences in the presence of priorities and therefore are not very useful. Nevertheless there is a need for less discriminating relations.

In summary, ACSR provides the theoretical foundation for a practical system to specify real-time distributed systems. The addition of higher level notions such as dynamic priorities, refinement and a more appealing syntax would improve its usefulness in practice. With adequate automation tools it can be a significant help in the design of correct distributed real-time systems.

A Selected Proofs of Soundness of ACSR Laws

These proofs are based on the application of the lemmas 4.1 and 4.2. For each law, using Table 3, we calculate the value of the $\mathcal{T}$ (or sometimes $\mathcal{T}_\pi$) function for both sides of the equation and verify that the results are equal or related in a way that satisfies the condition of lemma 4.2.

ITP(4)

$$\begin{aligned}
\mathcal{T}(A^{t\leq\infty}(P,Q)) &= \{(A^{u'}, P\left[{}^{u'}/_{t}\right] + A^{t\leq\infty}\left(P\left[{}^{u'+t}/_{t}\right], Q\right))\} \\
\mathcal{T}(A^{t\leq\infty}(P,\mathbf{0})) &= \{(A^{u'}, P\left[{}^{u'}/_{t}\right] + A^{t\leq\infty}\left(P\left[{}^{u'+t}/_{t}\right], \mathbf{0}\right))\}
\end{aligned}$$

It follows from lemma 4.2 that the relation defined by $\{(A^{t\leq\infty}(P,Q), A^{t\leq\infty}(P,\mathbf{0}))\}$ is a prioritized strong bisimulation. □

Choice(4) $\mathcal{T}((P+Q)+R) = \mathcal{T}(P+Q) \cup \mathcal{T}(R) = (\mathcal{T}(P) \cup \mathcal{T}(Q)) \cup \mathcal{T}(R)$
$= \mathcal{T}(P) \cup (\mathcal{T}(Q) \cup \mathcal{T}(R)) = \mathcal{T}(P + (Q+R))$ □

Timeout(3) We distinguish two cases.

i) When $u \leq v$ we have:

$$\begin{array}{l} \mathcal{T}((A^u{:}P) \bigtriangleup_v Q) \\ \quad = \quad \{\langle A^{u'}, (A^{u-u'}{:}P) \bigtriangleup_{v-u'} Q\rangle \mid \langle A^{u'}, A^{u-u'}{:}P\rangle \in \mathcal{T}(A^u{:}P) \wedge u' \leq u\} \\ \mathcal{T}(A^u{:}(P \bigtriangleup_{v-u} Q)) \\ \quad = \quad \{\langle A^{u'}, A^{u-u'}{:}(P \bigtriangleup_{v-u} Q)\rangle \mid \langle A^{u'}, A^{u-u'}{:}P\rangle \in \mathcal{T}(A^u{:}P) \wedge u' \leq u\} \end{array}$$

It follows from lemma 4.2 that, under the condition $u \leq v$, the relation defined by $\{(A^u{:}X \bigtriangleup_v Y, A^u{:}(X \bigtriangleup_{v-u} Y))\}$ is a prioritized strong bisimulation.

ii)When $v < u$ we have:

$$\begin{array}{l} \mathcal{T}((A^u{:}P) \bigtriangleup_v Q) \\ \quad = \quad \{\langle A^{u'}, (A^{u-u'}{:}P) \bigtriangleup_{v-u'} Q\rangle \mid \langle A^{u'}, A^{u-u'}{:}P\rangle \in \mathcal{T}(A^u{:}P) \wedge u' \leq v\} \\ \mathcal{T}(A^v{:}(P \bigtriangleup_0 Q)) \\ \quad = \quad \{\langle A^{u'}, A^{u-u'}{:}(P \bigtriangleup_0 Q)\rangle \mid \langle A^{u'}, A^{u-u'}{:}P\rangle \in \mathcal{T}(A^u{:}P) \wedge u' \leq v\} \end{array}$$

And therefore, when $v < u$ the relation defined by $\{(A^u{:}X \bigtriangleup_v Y, A^v{:}(X \bigtriangleup_0 Y))\}$ is a prioritized strong bisimulation. It follows, that in all cases, the relation defined by

$$\{(A^u{:}X \bigtriangleup_v Y, A^w{:}(X \bigtriangleup_{v-w} Y)) \mid w = \min(u, v)\}$$

is a prioritized strong bisimulation. □

Par(3)

$$\begin{array}{l} \mathcal{T}(P \| Q) = \\ \qquad \{\langle (A|B)^u, P' \parallel Q'\rangle \mid \langle A^u, P'\rangle \in \mathcal{T}(P) \wedge \langle B^u, Q'\rangle \in \mathcal{T}(Q) \wedge \rho(A) \cap \rho(B) = \emptyset\} \\ \quad \cup \{\langle e, P' \parallel Q\rangle \mid \langle e, P'\rangle \in \mathcal{T}(P)\} \\ \quad \cup \{\langle e, P \parallel Q'\rangle \mid \langle e, Q'\rangle \in \mathcal{T}(Q)\} \\ \quad \cup \{\langle (\tau, n+m), P' \parallel Q'\rangle \mid \langle (a, n), P'\rangle \in \mathcal{T}(P) \wedge \langle (\bar{a}, m), Q'\rangle \in \mathcal{T}(Q)\} \end{array}$$

$$\begin{array}{l} \mathcal{T}(P|Q + P \mathbin{\lfloor\!\lfloor} Q + Q \mathbin{\lfloor\!\lfloor} P) = \\ \qquad \{\langle (A|B)^u, P' \parallel Q'\rangle \mid \langle A^u, P'\rangle \in \mathcal{T}(P) \wedge \langle B^u, Q'\rangle \in \mathcal{T}(Q) \wedge \rho(A) \cap \rho(B) = \emptyset\} \\ \quad \cup \{\langle (\tau, n+m), P' \parallel Q'\rangle \mid \langle (a, n), P'\rangle \in \mathcal{T}(P) \wedge \langle (\bar{a}, m), Q'\rangle \in \mathcal{T}(Q)\} \\ \quad \cup \{\langle e, P' \parallel Q\rangle \mid \langle e, P'\rangle \in \mathcal{T}(P)\} \\ \quad \cup \{\langle e, Q' \parallel P\rangle \mid \langle e, Q'\rangle \in \mathcal{T}(Q)\} \end{array}$$

At this point, we need to observe that, by Par(1), $P \| Q' = Q' \| P$ It follows, from lemma 4.2 that the relation defined by $\{(X \| Y, X|Y + X \mathbin{\lfloor\!\lfloor} Y + Y \mathbin{\lfloor\!\lfloor} X)\}$ is a prioritized strong bisimulation. □

Par(4) We can calculate:

$$
\begin{aligned}
&\mathcal{T}(\widehat{P} \parallel \widehat{Q}) \\
&= \quad \{\langle (A|B)^u, P''' \parallel Q''' \rangle \mid \langle A^u, P''' \rangle \in \mathcal{T}(\widehat{P}) \wedge \langle B^u, Q''' \rangle \in \mathcal{T}(\widehat{Q}) \wedge \rho(A) \cap \rho(B) = \emptyset\} \\
&\quad \cup \{\langle e, P''' \parallel \widehat{Q} \rangle \mid \langle e, P''' \rangle \in \mathcal{T}(\widehat{P})\} \\
&\quad \cup \{\langle e, \widehat{P} \parallel Q''' \rangle \mid \langle e, Q''' \rangle \in \mathcal{T}(\widehat{Q})\} \\
&\quad \cup \{\langle (\tau, p+q), P''' \parallel Q''' \rangle \mid \langle (a,p), P''' \rangle \in \mathcal{T}(\widehat{P}) \wedge \langle (\bar{a}, q), Q''' \rangle \in \mathcal{T}(\widehat{Q})\} \\
&= \quad \{\langle (A_i|B_k)^{w_{ik}}, A_i^{u_i - w_{ik}}{:}P'_i \parallel B_k^{v_k - w_{ik}}{:}Q'_i \rangle \mid i \in I \wedge k \in K \wedge \rho(A_i) \cap \rho(B_k) = \emptyset \\
&\qquad\qquad \wedge\, w_{ik} \le \min(u_i, v_k)\} \\
&\quad \cup \{\langle (a_j, n_j), P''_j \parallel \widehat{Q} \rangle \mid j \in J\} \\
&\quad \cup \{\langle (b_l, m_l), \widehat{P} \parallel Q''_l \rangle \mid l \in L\} \\
&\quad \cup \{\langle (\tau, n_j + m_l), P''_j \parallel Q''_l \rangle \mid j \in J \wedge l \in L \wedge a_j = \overline{b_l}\}
\end{aligned}
$$

$$
= \mathcal{T}\left(
\begin{array}{ll}
 & \sum\limits_{\substack{i \in I, k \in K, w_{ik} = \min(u_i, v_k) \\ \rho(A_i) \cap \rho(B_k) = \emptyset}} (A_i|B_k)^{w_{ik}}{:}(A_i^{u_i - w_{ik}}{:}P'_i \parallel B_k^{v_k - w_{ik}}{:}Q'_k) \\
+ & \sum\limits_{j \in J} (a_j, n_j).(Q''_j \parallel \widehat{Q}) \\
+ & \sum\limits_{l \in L} (b_l, m_l).(\widehat{P} \parallel Q''_l) \\
+ & \sum\limits_{\substack{j \in J, l \in L, \\ a_j = \overline{b_l}}} (\tau, n_j + m_l).(P''_j \parallel Q''_l)
\end{array}
\right)
$$

□

Res(6)

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

However,

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

It follows from lemma 4.2 that the relation $\{(X \backslash E \backslash F, X \backslash E \cup F) \mid E, F \subseteq \mathcal{L}\}$ is a prioritized strong bisimulation. □

Close(7)

$$\begin{array}{rcl}\mathcal{T}([P\backslash E]_U) & = & \{\langle [A]^u_U, [P']_U\rangle \mid \langle A^u, P'\rangle \in \mathcal{T}(P\backslash E)\} \\ & & \cup\ \{\langle (a,n), [P']_U\rangle \mid \langle (a,n), P'\rangle \in \mathcal{T}(P\backslash E)\} \\ & = & \{\langle [A]^u_U, [P''\backslash E]_U\rangle \mid \langle A^u, P''\rangle \in \mathcal{T}(P)\} \\ & & \cup\ \{\langle (a,n), [P''\backslash E]_U\rangle \mid \langle (a,n), P''\rangle \in \mathcal{T}(P) \wedge a, \bar{a} \notin E\}\end{array}$$

However,

$$\begin{array}{rcl}\mathcal{T}([P]_U\backslash E) & = & \{\langle A^u, P'\backslash E\rangle \mid \langle A^u, P'\rangle \in \mathcal{T}([P]_U)\} \\ & & \cup\ \{\langle (a,n), P'\backslash E\rangle \mid \langle (a,n), P'\rangle \in \mathcal{T}([P]_U) \wedge a, \bar{a} \notin E\} \\ & = & \{\langle [A]^u_U, [P'']_U\backslash E\rangle \mid \langle A^u, P''\rangle \in \mathcal{T}(P)\} \\ & & \cup\ \{\langle (a,n), [P'']_U\backslash E\rangle \mid \langle (a,n), P''\rangle \in \mathcal{T}(P) \wedge a, \bar{a} \notin E\}\end{array}$$

It follows that the relation $\{([X\backslash E]_U, [X]_U\backslash E) \mid U \subseteq \mathit{Res} \wedge E \subseteq \mathcal{L}\}$ is a prioritized strong bisimulation. □

Except(6)

$$\begin{array}{ll}\mathcal{T}((P \dagger Q) \dagger R) & \\ = & \mathcal{T}(R) \\ & \cup\ \{\langle \alpha, P' \dagger R\rangle \mid \langle \alpha, P'\rangle \in \mathcal{T}(P \dagger Q)\} \\ & \cup\ \{\langle (\tau, p+r), R'\rangle \mid \exists P', a\colon\ \langle (a,p), P'\rangle \in \mathcal{T}(P \dagger Q) \wedge \langle (\bar{a}, r), R'\rangle \in \mathcal{T}(R)\} \\ = & \mathcal{T}(R) \\ & \cup\ \{\langle \alpha, (P' \dagger Q) \dagger R\rangle \mid \langle \alpha, P'\rangle \in \mathcal{T}(P)\} \\ & \cup\ \{\langle \alpha, Q' \dagger R\rangle \mid \langle \alpha, Q'\rangle \in \mathcal{T}(Q)\} \\ & \cup\ \{\langle (\tau, p+q), Q' \dagger R\rangle \mid \exists P', a\colon\ \langle (a,p), P'\rangle \in \mathcal{T}(P) \wedge \langle (\bar{a}, q), Q'\rangle \in \mathcal{T}(Q)\} \\ & \cup\ \{\langle (\tau, p+r), R'\rangle \mid \exists P', a\colon\ \langle (a,p), P'\rangle \in \mathcal{T}(P) \wedge \langle (\bar{a}, r), R'\rangle \in \mathcal{T}(R)\} \\ & \cup\ \{\langle (\tau, q+r), R'\rangle \mid \exists Q', a\colon\ \langle (a,q), Q'\rangle \in \mathcal{T}(Q) \wedge \langle (\bar{a}, r), R'\rangle \in \mathcal{T}(R)\}\end{array}$$

However,

$$\begin{array}{ll}\mathcal{T}(P \dagger (Q \dagger R)) & \\ = & \mathcal{T}(Q \dagger R) \\ & \cup\ \{\langle \alpha, P' \dagger (Q \dagger R)\rangle \mid \langle \alpha, P'\rangle \in \mathcal{T}(P)\} \\ & \cup\ \{\langle (\tau, p+q), Q'\rangle \mid \exists P', a\colon\ \langle (a,p), P'\rangle \in \mathcal{T}(P) \wedge \langle (\bar{a}, q), Q'\rangle \in \mathcal{T}(Q \dagger R)\} \\ = & \mathcal{T}(R) \\ & \cup\ \{\langle \alpha, Q' \dagger R\rangle \mid \langle \alpha, Q'\rangle \in \mathcal{T}(Q)\} \\ & \cup\ \{\langle (\tau, q+r), R'\rangle \mid \exists Q', a\colon\ \langle (a,q), Q'\rangle \in \mathcal{T}(Q) \wedge \langle (\bar{a}, r), R'\rangle \in \mathcal{T}(R)\} \\ & \cup\{\langle \alpha, P' \dagger (Q \dagger R)\rangle \mid \langle \alpha, P'\rangle \in \mathcal{T}(P)\} \\ & \cup\ \{\langle (\tau, p+q), Q' \dagger R\rangle \mid \exists P', a\colon\ \langle (a,p), P'\rangle \in \mathcal{T}(P) \wedge \langle (\bar{a}, q), Q'\rangle \in \mathcal{T}(Q)\} \\ & \cup\ \{\langle (\tau, p+r), R'\rangle \mid \exists P', a\colon\ \langle (a,p), P'\rangle \in \mathcal{T}(P) \wedge \langle (\bar{a}, r), R'\rangle \in \mathcal{T}(R)\}\end{array}$$

It follows from lemma 4.2 that the relation defined by $\{((X \dagger Y) \dagger Z, X \dagger (Y \dagger Z))\}$ is a prioritized strong bisimulation. □

Sync(3) When $\rho(A) \cap \rho(B) = \emptyset$ and $w = \min(u, v)$ we have:

$$\begin{aligned}
&\mathcal{T}(A^{t_1 \le u}(PI, PC) \mid B^{t_2 \le v}(QI, QC)) \\
&= \{\langle (A|B)^{u'}, P' \| Q' \rangle \mid \langle A^{u'}, P' \rangle \in \mathcal{T}(A^{t_1 \le u}(PI, PC)) \wedge \langle B^{u'}, Q' \rangle \in \mathcal{T}(B^{t_2 \le v}(PI, QC))\} \\
&= \left\{ \left\langle (A|B)^{u'}, R' \right\rangle \mid u' \le w \right\}
\end{aligned}$$

with:

$$\begin{aligned}
R' \;=\; & \left(PI\left[{}^{u'}/_{t_1}\right] + A^{t_1 \le u-u'}\left(PI\left[{}^{u'+t_1}/_{t_1}\right], PC\right)\right) \\
& \| \left(QI\left[{}^{u'}/_{t_2}\right] + B^{t_2 \le v-u'}\left(QI\left[{}^{u'+t_2}/_{t_2}\right], QC\right)\right)
\end{aligned}$$

After using Par(3) and LeftM(2) we obtain:

$$\begin{aligned}
R' \;=\; & A^{t_1 \le u-u'}\left(PI\left[{}^{u'+t_1}/_{t_1}\right], PC\right) \mid B^{t_2 \le v-u'}\left(QI\left[{}^{u'+t_2}/_{t_2}\right], QC\right) \\
& + PI\left[{}^{u'}/_{t_1}\right] \mathbin{\|\!\_} \left(QI\left[{}^{u'}/_{t_2}\right] + B^{t_2 \le v-u'}\left(QI\left[{}^{u'+t_2}/_{t_2}\right], QC\right)\right) \\
& + PI\left[{}^{u'}/_{t_1}\right] \mid \left(QI\left[{}^{u'}/_{t_2}\right] + B^{t_2 \le v-u'}\left(QI\left[{}^{u'+t_2}/_{t_2}\right], QC\right)\right) \\
& + QI\left[{}^{u'}/_{t_2}\right] \mathbin{\|\!\_} \left(PI\left[{}^{u'}/_{t_1}\right] + A^{t_1 \le u-u'}\left(PI\left[{}^{u'+t_1}/_{t_1}\right], PC\right)\right) \\
& + QI\left[{}^{u'}/_{t_2}\right] \mid \left(PI\left[{}^{u'}/_{t_1}\right] + A^{t_1 \le u-u'}\left(PI\left[{}^{u'+t_1}/_{t_1}\right], PC\right)\right) \\
R' \;=\; & A^{t_1 \le u-u'}\left(PI\left[{}^{u'+t_1}/_{t_1}\right], PC\right) \mid B^{t_2 \le v-u'}\left(QI\left[{}^{u'+t_2}/_{t_2}\right], QC\right) \\
& + RI\left[{}^{u'}/_{t}\right]
\end{aligned}$$

However,

$$\begin{aligned}
&\mathcal{T}((A|B)^{t \le w}(RI, RC)) \\
&= \left\{ \left\langle (A|B)^{u'}, RI\left[{}^{u'}/_{t}\right] + (A|B)^{t \le w-u'}\left(RI\left[{}^{u'+t}/_{t}\right], RC\right)\right\rangle \mid u \le w \right\}
\end{aligned}$$

□

Rec(1) From the operational semantic rule Rec we have:

$$\mathcal{T}(rec\ X.P) \;=\; \mathcal{T}\left(P\left[{}^{rec\ X.P}/_{X}\right]\right)$$

□

Rec(2) Let $R \overset{\text{def}}{=} rec\ X.Q$; by Rec(1), $R = Q\left[{}^{R}/_{X}\right]$. We need to prove that $P \sim_\pi R$, assuming that $P = Q\left[{}^{P}/_{X}\right]$ and X is guarded in Q. We do this by making use of lemma 4.2 and proving that the relation $\mathcal{R}$ defined by

$$\left\{\left(E\left[{}^{P}/_{X}\right], E\left[{}^{R}/_{X}\right]\right)\right\} \cup \{(E, E)\}$$

(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 $Q\left[{}^{P}/_{X}\right]$

does not depend on the value of P, more formally:

$$Q\left[{}^{P}/{}_{X}\right] \xrightarrow{\alpha}_{\pi} Q'\left[{}^{P}/{}_{X}\right] \textit{ if and only if } Q \xrightarrow{\alpha}_{\pi} Q'$$

and

$$Q\left[{}^{R}/{}_{X}\right] \xrightarrow{\alpha}_{\pi} Q'\left[{}^{R}/{}_{X}\right] \textit{ if and only if } Q \xrightarrow{\alpha}_{\pi} Q'$$

We proceed by induction on the structure of E.

If E is $\mathbf{0}$, $\mathcal{T}(E\left[{}^{P}/{}_{X}\right]) = \emptyset = \mathcal{T}(E\left[{}^{Q}/{}_{X}\right])$.

If E is X, we obtain the $E\left[{}^{P}/{}_{X}\right] = P = Q\left[{}^{P}/{}_{X}\right]$ and similarly $E\left[{}^{R}/{}_{X}\right] = R = Q\left[{}^{R}/{}_{X}\right]$ and therefore

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

and

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

If E is αF then

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

and

$$\mathcal{T}(E\left[{}^{Q}/{}_{X}\right]) = \left\{\langle \alpha, F\left[{}^{Q}/{}_{X}\right]\rangle\right\} \quad .$$

The other cases follow from the induction hypothesis and the fact that prioritized strong bisimulation is a congruence. □

Rec(3) The proof is by transition induction, i.e., induction on the depth of the inference tree. Let

$$Q \overset{\text{def}}{=} rec\ X.(P + [X\backslash E]_U) \qquad \text{and} \qquad R \overset{\text{def}}{=} rec\ X.(P + [P\backslash E]_U) \quad .$$

We prove that the relation $S = \left\{(G\left[{}^{P}/{}_{X}\right], G\left[{}^{Q}/{}_{X}\right])\right\}$ is a bisimulation. The proof goes by cases on the structure of G. Most cases follow directly from the fact that bisimulation is a congruence. The only interesting case is when G is X.

We have $X\left[{}^{P}/{}_{X}\right] \xrightarrow{\alpha} P'$ if and only if $P \xrightarrow{\alpha} P'$. That is $rec\ X.(P + [X\backslash E]_U) \xrightarrow{\alpha} P'$, or $P\left[{}^{Q}/{}_{X}\right] + [Q\backslash E]_U \xrightarrow{\alpha} P'$. There are two possible cases:

i) If $P\left[{}^{Q}/{}_{X}\right] \xrightarrow{\alpha} P'$ then, by induction hypothesis we have $P\left[{}^{R}/{}_{X}\right] \xrightarrow{\alpha} P''$ with $(P', P'') \in S$.

ii) If $[Q\backslash E]_U \xrightarrow{\alpha} P'$ then, by induction hypothesis we have $[R\backslash E]_U \xrightarrow{\alpha} P''$. We replace R by its definition to obtain $[rec\ X.(P + [P\backslash E]_U)\backslash E]_U \xrightarrow{\alpha} P'$. That is

$$\left[P\left[{}^{R}/{}_{X}\right] + \left[P\left[{}^{R}/{}_{X}\right]\backslash E\right]_U \backslash E\right]_U \xrightarrow{\alpha} P''$$

and using the distributivity and idempotence of both closure and restriction we obtain: $\left[P\left[{}^{R}/{}_{X}\right]\backslash E\right]_U \xrightarrow{\alpha} P''$ whence $P\left[{}^{R}/{}_{X}\right] \xrightarrow{\alpha} P''$.

The reverse case is immediate. □

References

[1] M. Abadi and L. Lamport. An Old-Fashioned Recepi for Real Time. In *Proceedings of REX Workshop on Real Time: Theory and Practice, LNCS 600.* Springer Verlag, June 1991.

[2] L. Aceto, B. Bloom, and F. Vaandrager. Turning SOS Rules into Equations. Technical Report CS-R9218, CWI, Amsterdam, The Netherlands, 1992.

[3] R. Alur. *Techniques for Automatic Verification of Real-Time Systems.* PhD thesis, Stanford University, 1991.

[4] R. Alur and D. Dill. Automata for modeling real-time systems. In *Proc. of 17th ICALP, LNCS 443*, pages 322–335. Springer Verlag, 1990.

[5] R. Alur and T. Henzinger. A Really Temporal Logics. In *Proc. of 30th IEEE FOCS*, 1989.

[6] R. Alur and T. Henzinger. Real-Time Logics: Complexity and Expressiveness. In *Proc. of IEEE Symposium on Logic in Computer Science*, 1990.

[7] J. Baeten, J. Bergstra, and J. Klop. Syntax and defining equations for an interrupt mechanism in process algebra. *Fundamenta Informaticae*, IX(2):127–168, 1986.

[8] J. Baeten, J. Bergstra, and J. Klop. Ready-Trace Semantics for Concrete Process Algebra with a Priority Operator. *Computer Journal*, 30(6):498–506, 1987.

[9] J.C.M. Baeten and J.A. Bergstra. Real Time Process Algebra. Technical Report CS-R9053, Centre for Mathematics and Computer Science, Amsterdam, the Netherlands, 1990.

[10] J. A. Bergstra and J. W. Klop. Algebra of Communicating Processes. Technical Report CS-R8420, Centre for Mathematics and Computer Science, Amsterdam, the Netherlands, 1984.

[11] J.A. Bergstra and J.W. Klop. Algebra of Communicating Processes with Abstraction. *Journal of Theoretical Computer Science*, 37:77–121, 1985.

[12] A. Bernstein and P.K. Harter Jr. Proving Real-Time Properties of Programs with Temporal Logic. In *Proc. Symposium on Operating Systems Principles*, 1981.

[13] B. Bloom. Ready, Set, Go: Structural Operational Semantics for Linear-Time Process Algebras. Technical Report TR 93-1372, Cornell Universiy, 1993.

[14] B. Bloom. Structural Operational Semantics for Weak Bisimulation. Technical Report TR 93-1373, Cornell Universiy, 1993.

[15] T. Bolognesi and S.A. Smolka. Fundamental Results for the Verification of Observational Equivalence. In *Protocol Specification, Testing and Verification.* North-Holland, 1987.

[16] P. Brémond-Grégoire. A Process Algebra of Communicating Shared Resources with Real-Time and Priorities. Dissertation Proposal, University of Pennsylvania, 1993.

[17] P. Brémond-Grégoire. *An Algebra of Communicating Shared Resources with Dense Time and Priorities.* PhD thesis, Department of Computer and Information Science, University of Pennsylvania, 1994.

[18] Liang Chen. Specification and Verification of Real-Time Systems. Unpublished report, 1992.

[19] R. Cleaveland and M. Hennessy. Priorities in Process Algebras. In *Proc. of IEEE Symposium on Logic in Computer Science*, 1988.

[20] R. Cleaveland and M. Hennessy. Priorities in Process Algebras. *Information and Computation*, 87:58–77, 1990.

[21] A.K. Mok F. Jahanian and D. Stuart. Formal Specification of Real-Time System. Technical Report TR 88-25, Department of Computer Science, University of Texas at Austin, 1988.

[22] M.K. Franklin and A. Gabrielian. A Transformational Method for Verifying Safety Properties in Real-Time Systems. In *Proc. of IEEE Real-Time Systems Symposium*, pages 112–123, December 1989.

[23] A. Fredette and R. Cleaveland. RTSL: A Formal Language for Real-Time Schedulability Analysis. Technical Report TR-93-09, North Carolina State Univesity, 1993.

[24] A. Gabrielian and M.K. Franklin. State-Based Specification of Complex Real-Time Systems. In *Proc. of IEEE Real-Time Systems Symposium*, December 1988.

[25] R. Gerth and A. Boucher. A Timed Failure Semantics for Extended Communicating Processes. Technical Report TR. 4-4(1), Department of Mathematics and Computing Science, Eindhoven University of Technology, March 1987.

[26] M. Hennessy and T. Regan. A Process Algebra for Timed Systems. Technical Report 5/91, Univ. of Sussex, UK, April 1991.

[27] C.A.R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–676, August 1978.

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

[29] F. Jahanian and A. Goyal. A Formalism for Reasoning about Real-Time Constraints at Run Time. Technical Report RC 15252, IBM Research Division, T.J. Watson Research Center, Yorktown Heights, New York, December 1989.

[30] F. Jahanian, R.S. Lee, and A. Mok. Semantics of Modechart in Real Time Logic. In *Proc. 21st Hawaii Int. Conf. on System Sciences*, Jan. 88.

[31] F. Jahanian and A.K. Mok. Safety analysis of timing properties in real-time systems. *IEEE Transactions on Software Engineering*, SE-12(9):890–904, September 1986.

[32] F. Jahanian and A.K. Mok. A Graph-Theoretic Approach for Timing Analysis and its Implementation. *IEEE Transactions on Computers*, C-36(8):961–975, August 1987.

[33] F. Jahanian and D.A. Stuart. A Method for Verifying Properties of Modechart Specifications. In *Proc. of IEEE Real-Time Systems Symposium*, pages 12–21, December 1988.

[34] R. Koymans. Specifying Message Passing and Time-Critical Systems with Temporal Logic. *Real-Time Systems*, 16(11), November 1990.

[35] R. Koymans and W.P. de Roever. Examples of real-time temporal specification. In *The Analisys of Concurrent Systems, LNCS 207*, pages 231–252. Springer Verlag, 1985.

[36] R. Koymans, J. Vytopil, and W.P. de Roever. Real-time programming and asynchronous message passing. In *2rd ACM Symposium on Principles of Distributed Computing*, pages 187–197. ACM Press, 1983.

[37] Leslie Lamport. The Temporal Logics of Actions. Technical report, DEC Systems Research Center, Palo Alto, California, 1991.

[38] N. Lynch and M. Tuttle. An Introduction to Input/Output Automata. Technical Report MIT/LCS/TM-373, Laboratory for Computer Science, Massachusetts Institute of Technology, 1988.

[39] N. Lynch and F. Vaandrager. Forward and Backward Simulations for Timing Based Systems. In *Proc. REX Workshop "Real-Time: Theory in Practice"*. LNCS 600, Springer-Verlag, 1991.

[40] G.H. MacEwen and T.A. Montgomery. The RNet Programming System: Distributed Real-Time Logic. Technical Report Report 87-3, Dept. of Computing and Information Science, Queen's University, Kingston, Ontario, November 1987.

[41] M. Merritt, F. Modungo, and M. Tuttle. Time-Constrained Automata. In *CONCUR '91*, August 1991.

[42] R. Milner. *A Calculus for Communicating Systems.* LNCS 92, Springer-Verlag, 1980.

[43] R. Milner. Calculi for Synchrony and Asynchrony. *Theoretical Computer Science*, 25:267–310, 1983.

[44] R. Milner. A Complete Axiomatisation for Observational Congruence of Finite-State Behaviors. *Information and Computation*, 81:227–247, 1989.

[45] R. Milner. *Communication and Concurrency.* Prentice-Hall, 1989.

[46] F. Moller and C. Tofts. A Temporal Calculus of Communicating Systems. In *Proc. of CONCUR '90*, pages 401–415. LNCS 458, Springer Verlag, August 1990.

[47] X. Nicollin and J. Sifakis. The Algebra of Timed Processes ATP: Theory and Application. Technical Report RT-C26, Institut National Polytechnique De Grenoble, November 1991.

[48] D. Park. Concurrency and Automata on Infinite Sequences. In *Proc. of 5th GI Conference.* LNCS 104, Springer Verlag, 1981.

[49] A. Pnueli. The Temproal Logic of Programs. In *Proc. of Foundations of Computer Science*, pages 46–57, 1977.

[50] G.M. Reed and A.W. Roscoe. A Timed Model for Communicating Sequential Processes. In *Proc. of Int. Conf. on Automata, Languages and Programming.* LNCS 226, Springer Verlag, 1986.

[51] Douglas Stuart. Implementing a Verifyer for Real-Time Systems. In *Proc. 11th IEEE Real-Time Systems Symposium*, 1990.

[52] F. Vaandrager and N. Lynch. Action Transducers and Timed Automata. In *Proc. CONCUR '92, International Conference on Concurrency Theory.* LNCS 630, Springer-Verlag, August 1992.

[53] Y. Wang. CCS + Time = An Interleaving Model for Real Time Systems. In *Proc. of Int. Conf. on Automata, Languages and Programming*, July 1991.

[54] C. Zhou, M. Hansen, A. Ravn, and H. Rischel. Duration Specification for Shared Processors. In *Proc. Formal Techniques in Real-Time and Fault-Tolerant Systems, LNCS 571*, pages 21–32. Springer-Verlag, 1992.

[55] A. Zwarico, R. Gerber, and I. Lee. A Complete Axiomatization of Real-Time Processes. Technical Report MS-CIS-88-88, University of Pennsylvania, Department of Computer and Information Science, November 1988.