March 1990

CCSR: A Calculus for Communicating Shared Resources

Richard Gerber
University of Maryland

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

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-90-16.

CCSR: A Calculus for Communicating Shared Resources

Abstract

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 models 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. To bridge the gap between these two disciplines, we have developed a formalism called Communicating Shared Resources, or CSR. This paper presents the priority-based process algebra called the Calculus for Communicating Shared Resources (CCSR), which provides an equational characterization of the CSR language. The computation model of CCSR is resource-based in that multiple resources execute synchronously, while processes assigned to the same resource are interleaved according to their priorities. CCSR possesses a prioritized strong equivalence for terms based on strong bisimulation. The paper also describes a producer and consumer problem whose correct timing behavior depends on priority.

Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-90-16.

CCSR: A Calculus For Communicating Shared Resources

MS-CIS-90-16
GRASP LAB 208

Richard Gerber
Insup Lee

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

March 1990

ACKNOWLEDGEMENTS:
This research was supported in part by ONR N000014-89-J-1131.

CCSR: A Calculus for Communicating Shared Resources *

Richard Gerber† and Insup Lee‡
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104

March 12, 1990

Abstract

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 models 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. To bridge the gap between these two disciplines, we have developed a formalism called Communicating Shared Resources, or CSR. This paper presents the priority-based process algebra called the Calculus for Communicating Shared Resources (CCSR), which provides an equational characterization of the CSR language. The computation model of CCSR is resource-based in that multiple resources execute synchronously, while processes assigned to the same resource are interleaved according to their priorities. CCSR possesses a prioritized strong equivalence for terms based on strong bisimulation. The paper also describes a producer and consumer problem whose correct timing behavior depends on priority.

---

*This research was supported in part by ONR N000014-89-J-1131.

†Email: rich@linc.cis.upenn.edu

‡Email: lee@central.cis.upenn.edu

1 Introduction

The correctness of a real-time system depends not only on how concurrent processes interact, but also the time at which these interactions occur. In the tradition of untimed concurrency theory, however, formal models for time-dependent computation have treated the execution of processes abstractly, quite isolated from their operating environments. These environments often have a profound effect on the timing behavior of real-time systems, and cannot be ignored when reasoning about them.

To help bridge the gap between abstract computation models and implementation, we have developed a real-time formalism called *Communicating Shared Resources*, or CSR [4, 3]. CSR's underlying computational model is *resource-based*, where a resource may be a processor, an Ethernet link, or any other constituent device in a real-time system. At any point in time, each resource has the capacity to execute an action consisting of only a single event or particle. However, a resource may host a set of many processes, and at every instant, any number of these processes may compete for its availability. "True" parallelism may take place only *between* resources; on a single resource, the actions of multiple processes must be interleaved. To arbitrate between competing events, CSR employs a priority-ordering among them.

Our priority semantics of CSR is based on the linear-history model [2], and its extension to real-time computing [9, 7]. While this semantics adequately captures the temporal properties of prioritized resource interaction, it does not easily lend itself to an equational characterization of the CSR language. It is for this reason that we have developed the Calculus for Communicating Shared Resources, or CCSR. Strongly influenced by SCCS [11, 13], CCSR is a process algebra that uses a synchronous form of concurrency, and possesses a term equivalence based on strong bisimilarity. Syntactically, CSR is a "richer" formalism, in that it contains many real-time language constructs such as timed interrupt-handlers, temporal scopes [10], and periodic processes. However, all of the CSR constructs can be formulated in CCSR, and further, CCSR provides the ability to perform equivalence proofs by syntactic manipulation.

This paper describes our resource-based view of concurrency, including the notion of strong prioritized equivalence based on *strong bisimulation* and the equational characteristics of the CCSR terms. To illustrate the effect of priority on computation, consider the following SCCS

program fragment:

$$((a : P_1 \ + \ b : P_2) \times (\overline{a} : Q_1 \ + \ \overline{b} : Q_2))\backslash(Act - \{a \cdot \overline{b}, b \cdot \overline{a}\})$$

where a and b are particles. In unprioritized SCCS, a strongly equivalent agent is:

$$1 : (P_1 \times Q_1)\backslash(Act - \{a \cdot \overline{b}, b \cdot \overline{a}\}) \ + \ 1 : (P_2 \times Q_2)\backslash(Act - \{a \cdot \overline{b}, b \cdot \overline{a}\})$$

That is, since the calculus has no underlying priority structure, the resulting term is nondeterministic. However, with the introduction of priority, the result can be much different. For example, assume that the priority of a is greater than that of b, or informally $a > b$, and $\overline{a} > \overline{b}$. In this case we would expect the first term of the summation to emerge; that is, the resulting unit action could not be treated as having no priority.

An interesting problem arises when priorities are "circular"; that is, when $a > b$ but $\overline{b} > \overline{a}$. Is the resulting term *inaction* or nondeterministic? If we view particles as "belonging" to system resources, there is no reason why such conflicts cannot arise. The problem is complicated further in SCCS, where *actions* consist of many such particles, each having its own priority. In order to assign composite priorities to actions, we clearly require some additional structure in our calculus.

Previous research has, with varying success, treated some issues of the priority problem. There has been a spate of effort directed toward defining models for concurrency based on "maximum parallelism" [15], in which if processes are ready to communicate, they *will* communicate. Thus maximum parallelism incorporates a very limited, bi-level priority scheme, where non-idle actions always take precedence over idle actions, and contention between non-idle actions is resolved nondeterministically. A priority scheme for CCS is treated in [1], in which particles can only communicate with inverses of the *same* priority; this avoids the multiple priority problem mentioned above. Further, it can be debated whether there is justification for priority in completely asynchronous contexts such as those defined by CCS.

The remainder of this paper is organized as follows. In Section 2, we briefly overview the language of CCSR. Then, in Section 3, we describe the resource-based action domain; in particular, we stress the priority function on actions, and the partitioning of particles according to resources constraints. The semantics of CCSR terms is defined in two steps: we define the unconstrained operational semantics for closed terms in Section 4, followed by the prioritized

strong equivalence in Section 5. Section 6 presents an example of a real-time problem whose correct temporal behavior depends on priority.

2 The CCSR Language

The syntax of CCSR resembles, in large part, that of SCCS, and we wish to retain much of its flavor. The major differences occur in several places. First, the action domain contains sets of particles, does not form a monoid or a group. Second, our notion of communication is closer to that of CSP [6, 5], and is not performed with the use of inverse actions. In fact, there is no concept the inverse in our calculus, in that the basic combinator of actions is the union operation. Finally, and most importantly, our actions have priorities associated with them.

Let Σ be the set of all particles, or events, and let a, b, and c range over Σ. Let the letters A, B and C range $\mathcal{P}(\Sigma)$, and Γ range over $\mathcal{P}(\mathcal{P}(\Sigma))$. Let P range over the domain of terms, and let X range over the domain of term variables. As usual, we assume the existence of an infinite set of free term variables, FV. Also, we let the letter ϕ range over renaming functions on Σ; that is, $\phi \in \Sigma \to \Sigma$. We overload notation and extend such functions to sets in the usual way, where $\phi(A) = \{\phi(a) \mid a \in A\}$.

The following grammar defines the terms of CCSR:

$$P \;:=\; NIL \mid A : P \mid P + P \mid P_I \|_J P \mid [P]_I \mid P \backslash A \mid fix(X.P) \mid X$$

While we give formal semantics for these terms in subsequent sections, we briefly present some motivation for them here. The term NIL corresponds to $\mathbf{0}$ in SCCS – it can execute no action. The Action operator, "$A : P$", has the following behavior. At the first time unit, the action A is executed, proceeded by the term P. The Sum operator represents standard SCCS choice – either of the terms can be chosen to execute, subject to the constraints of the environment. The Conjunction operator $P_I \|_J P$ has two functions. It limits the resources that can be used by the two terms, and also forces synchronization between them. The Constrainment operator, $[P]_I$, denotes that the term P occupies *exactly* the resources represented in the index I. The Hiding operator $P \backslash A$ masks actions in P up to their priority, in that while the actions themselves are hidden, their priorities are still observable. The term $fix(X.P)$ denotes guarded recursion, allowing the specification of infinite behaviors.

The term X is a free variable that belongs to the infinite set FV. Any term in the calculus that contains a free variable is called *open*; a term that contains no free variables is considered *closed*. Staying within the terminology of CCS, we call closed terms *agents*.

3 A Resource-Based Action Domain

The considerations mentioned in section 1 have led us to construct a calculus with semantics based on resource constraints. In our execution model, we consider individual resources to be inherently sequential in nature. To put this in the parlance of SCCS, a single resource is capable of synchronously executing actions that consist, *at most*, of a single particle. Actions that consist of multiple particles must be formed by the synchronous execution of multiple resources.

This notion of execution leads to a natural partition of Σ into mutually disjoint subsets, each of which can be considered the set of particles available to a single resource. Letting $\mathcal{R}$ represent the index set of system resources, for some i in $\mathcal{R}$ we denote Σ_i as the collection of particles available to resource i. Also, since

$$\bigcup_{i \in \mathcal{R}} \Sigma_i = \Sigma$$

and

$$\forall i \in \mathcal{R}, \forall j \in \mathcal{R} \,.\, i \neq j,\ \Sigma_i \cap \Sigma_j = \emptyset,$$

the binary relation over $\mathcal{P}(\Sigma) \times \mathcal{P}(\Sigma)$ characterizing "belonging to the same set of resources" is an equivalence relation on actions.

As we have stated, a single resource is capable of executing actions that consist of at most one particle. We formalize this concept by defining, for each resource i, the domain of actions each is capable of performing:

$$\mathcal{D}_i = \{\{a\} \,|\, a \in \Sigma_i\} \cup \{\emptyset\}$$

We now can formally define the domain of actions $\mathcal{D}$:

$$\mathcal{D} = \{A \in p(\Sigma) \,|\, \forall i \in \mathcal{R},\ A \cap \Sigma_i \in \mathcal{D}_i\}$$

where $p(\Sigma)$ denotes the set of finite subsets of Σ. It is often convenient to map actions in $\mathcal{D}$ to the resource sets they inhabit. For a given action A, we use the notation $\mathcal{R}(A)$ to represent the resource set that executes particles in A:

$$\mathcal{R}(A) = \{i \in \mathcal{R} \mid \Sigma_i \cap A \neq \emptyset\}$$

It is important to briefly discuss the role of "$\emptyset$" in CCSR. What does it mean, for example, when for some $i \in \mathcal{R}$, $i \notin \mathcal{R}(A)$? It would be tempting to suggest that resource i is idling, but this is not necessarily true; resource i may not even be a member of the subsystem under observation. Thus, if $i \notin \mathcal{R}(A)$, we can only state that resource i is not contributing to the observed behavior. This notion is developed in the sequel.

3.1 Priorities

Each resource has a finite range of priorities at which its particles can execute. Letting mp_i be the maximum priority on resource i, we denote $PRI_i = [0, \ldots, mp_i] \subseteq \mathbb{N}$ as the set of priorities available to resource i.

Thus we can linearly order the particles in each Σ_i by a priority mapping $\pi_i \in \Sigma_i \rightarrow PRI_i$. Extending this ordering to $\mathcal{D}_i$, we construct a new mapping $\Pi_i \in \mathcal{D}_i \rightarrow PRI_i \cup \{\perp\}$ where for each A in $\mathcal{D}_i$,

$$\Pi_i(A) = \begin{cases} \pi_i(a) & \text{if } A = \{a\} \\ \perp & \text{otherwise} \end{cases}$$

and $\forall n \in PRI_i$, $\perp < n$. While technically unnecessary, we assign to the emptyset the undefined priority "$\perp$" to distinguish it from singleton actions.

Finally, we can define the partial ordering "$\leq_p$" that reflects our notion of priority over the domain $\mathcal{D}$. For all $A, B \in \mathcal{D}$,

$$A \leq_p B \quad \text{iff} \quad \forall i,\ \Pi_i(A \cap \Sigma_i) \leq \Pi_i(B \cap \Sigma_i)$$

3.2 Anonymous Execution Particles

In SCCS, the unit action "1" serves two distinctly different functions. One is to denote or idling, or a "busy waiting" condition. For example, the term $1 : (P \times Q)$ represents a process that idles for one time unit, and subsequently executes the term $P \times Q$. On the other hand,

the unit can denote the combined actions of two communicating partners. For example, the term

$$(a : P) \times (\overline{a} : Q)$$

is strongly equivalent to $1 : (P \times Q)$, yet in the resource-based view on concurrency, it has a distinctly different meaning.

With the introduction of priority into the calculus, we cannot make such an equivalence between terms. For example, assume that that a and $\overline{a}$ are particulate actions, and that each particle has a nonzero priority on its respective processor. To allow the equivalence above would be contradictory to our execution model, in that two actions with nonzero priority could synchronize into an idle action. Priority mandates that there be a difference between time consumed by execution, and time consumed by a busy-wait condition. This restriction leads to the following constraint placed on the calculus: *At every time unit, each active resource in a system must contribute a minimum amount of observational information – the priority of particle being executed.* Thus priorities may be considered "lights" on a resource's control panel; whenever a particle is executed by the resource, the light corresponding to its priority is illuminated.

Note that a priority function π_i naturally partitions each Σ_i into equivalence classes. That is, for some $n \in PRI_i$, a particle a is in the class $[b]_i^n$ if and only if $\pi_i(a) = \pi_i(b) = n$. In CCSR, we use the symbol "τ_i^n" to represent a "canonical" particle from each class.

Now, let $a \in \Sigma_i$ be an arbitrary particle. Then there exists some τ_i^n in Σ such that $\pi_i(a) = \pi_i(\tau_i^n)$. Further, there is a unique renaming function ϕ_π such that $\phi_\pi(a) = \tau_i^n$. This renaming is unique up to particle priority, in that if $\pi_i(a) = \pi_i(b)$, but $a \neq b$, then

$$\phi_\pi(a) = \phi_\pi(b) = \tau_i^n$$

It follows that the τ_i^n are fixed-points of priority renaming; that is, $\phi_\pi(\tau_i^n) = \tau_i^n$.

Now we turn briefly to the topic of resource idling. As stated in section 3.1, when a resource contributes *no* particle to an action, it does not imply that the resource is in an idle state. Instead, the action merely remains unspecified with respect to that resource. In our semantics, idling is an *observable* behavior, corresponding to the τ action in CCS under strong bisimulation. When a resource i idles, it executes a particle τ_i^0; in the scenario portrayed

above, the "light" corresponding to a priority of 0 is illuminated. Thus for every $i \in \mathcal{R}$, there is a τ_i^0 in Σ_i. This permits each resource to have the capacity to execute an idle particle. For a given set of resources $I \subseteq R$, the action composed of *all* of their idle particles is:

$$\mathcal{T}_I^0 = \{\tau_i^0 \mid i \in I\}$$

3.3 Synchronization

Resources synchronize through the use of *connection sets*, which can be thought of as the port connections between them. In a particulate calculus such as CCS, a particle "a" typically synchronizes with its inverse, or "$\overline{a}$," and the only *fully synchronized* action is "τ." This is a valid and even obvious approach, and certainly could be adapted for a priority-sensitive calculus. In CCSR, however, we take a more general approach. First, we wish to preserve the flavor of a fully synchronized action, without losing the ability to observe each of the action's constituent particles. (For example, when the CCS actions "a" and "$\overline{a}$" communicate, a degree of observability is lost, in that "τ" is fairly "generic.") Second, it is desirable to incorporate the SCCS expressibility of n-way communication within the structure of our prioritized action domain, $\mathcal{D}$.

Example 3.1 In CSP-type languages, the alphabet of particles is:

$$\{c_1!, c_1?, c_2!, c_2?, c_3!, c_3?, \ldots\}$$

where each c_i is considered a channel, $c_i!$ is interpreted as a write action, and $c_i?$ is interpreted as a read action. When a read and a write occur simultaneously on the same channel, the communication is considered successful. Thus, the connection sets in such languages are simply:

$$\{c_1!, c_1?\},\ \{c_2!, c_2?\},\ \{c_3!, c_3?\},\ \ldots$$

□

Akin to resources, connection sets partition Σ into mutually disjoint subsets of particles. We denote $\mathcal{C}$ as the index set of connection sets across the system, and for all $i \in \mathcal{C}$, C_i is a connection set. The connection sets must satisfy several properties:

1. The connection sets form a cover of Σ: $\bigcup_{i \in \mathcal{C}} C_i = \Sigma$.

2. The connection sets are mutually disjoint: $\forall i \in \mathcal{C} \forall j \in \mathcal{C}, i \neq j,\ C_i \cap C_j = \emptyset$.

3. The connection sets are constructed so that no particle of any Σ_i depends on synchronizing with another particle from Σ_i: $\forall i \in \mathcal{R} \forall j \in \mathcal{C},\ \Sigma_i \cap C_j \in \mathcal{D}_i$.

4. All "priority-canonical" particles belong to their own connection sets:
   $\forall a \in \Sigma\ \exists j \in \mathcal{C} \,.\, \{\phi_\pi(a)\} = C_j$.

The reason for condition 3 is apparent when we view connection sets in context of resource mapping. If the property did not hold, a processor would have to simultaneously execute two different particles for synchronization to occur. Such behavior violates our execution model, as each resource is sequential in nature.

Since every particle is a member of a *unique* connection set, we can construct a mapping from particles to the connection sets that contain them. We use the function *connections* to represent this. For any a in Σ, there is a distinct C_i such that $connections(a) = C_i$. For instance, in example 3.1, $connections(a?) = \{a?, a!\}$. We can naturally extend the *connections* function to sets of particles as follows:

$$Connections(A) = \bigcup_{a \in A} connections(a)$$

Definition 3.1 We say that a set is *fully synchronized* if it can be fully decomposed into a set of the connection sets (or it is empty). We use the predicate *sync* to represent this:

$$sync(A) \quad \text{iff} \quad Connections(A) = A$$

□

It is often convenient to be able to decompose a set $A \in \mathcal{D}$ into two parts: that which is fully synchronized, and that which is not. To do this, we make use of the following two definitions:

$$\begin{aligned} res(A) &= A \cap Connections(A) \\ unres(A) &= A - res(A) \end{aligned}$$

Finally, we need not view actions in merely two ways, as being either synchronized or unsynchronized. An action can be synchronized with respect to a resource set I. This means that for an action A, all of the connections that can be made with particles on resources in I are

made. This concept is particularly useful in defining the Conjunction operator. We let $\mathcal{D}_{\sigma(I)}$ denote the subdomain of $\mathcal{D}$ in which actions are synchronized with respect to I:

$$\mathcal{D}_{\sigma(I)} = \{A \in \mathcal{D} | \, A \; = \; Connections(A) \cap (\bigcup_{i \in I} \Sigma_i)\}$$

Of course if $sync(A)$ is true, then $A \in \mathcal{D}_{\sigma(\mathcal{R}(A))}$.

4 Semantics

In this section we give the *unconstrained* operational semantics for closed terms, in the style of [14]. By *unconstrained*, we mean that no priority structure is given to the domain. It is indeed possible to include our priority structure in a much more complicated set of transitional rules. The reason for this is that, when dealing with properties of priority, it is not sufficient to include in a rule's premise an action an agent *can* perform. In addition, it would be necessary to also include actions it *cannot* perform if higher priority actions are present. This highly complicates matters, and leads to infinite branching on finite terms. Thus we present an unconstrained version of our semantics, and we then refine our notion of priority with an equivalence relation based on it. In this we follow the path of [1] in their treatment of CCS priority, as well as [7] in their approach to maximum parallelism.

Let $\mathcal{E}$ represent the domain of closed terms. The labeled transition system $\langle \mathcal{E}, \rightarrow, \mathcal{D} \rangle$ is defined by the relation $\rightarrow \in \mathcal{E} \times A \times \mathcal{E}$, whose members are denoted: "$P \xrightarrow{A} Q$". Throughout, we use the following notation. For a given set of resources $I \subseteq \mathcal{R}$, we let Σ_I represent the set $\bigcup_{i \in I} \Sigma_i$. Table 1 presents the unconstrained transition system. The Action, Sum and Recursion rules are straightforward, and similar to their counterparts in SCCS. The other operators, however, require special treatment.

Conjunction. The four side conditions are what makes the Conjunction operation different from a more general "product" combinator, such as that found in SCCS. First, the two resource sets I and J are mutually disjoint. This, combined with the next two side conditions, places a very strong constraint on the sorts that each term can execute. Not only are the particles in both of the sorts mutually disjoint, but they are drawn from completely different resources. This corresponds to our resource- oriented view of concurrency, in which the Conjunction operator merges the operations of two *different* subsystems.

Action : $A : P \xrightarrow{A} P$

Hide : $\dfrac{P \xrightarrow{B} P'}{P\backslash A \xrightarrow{\phi_{hide(A)}(B)} P'\backslash A}$ $(sync(A \cap B))$

SumL : $\dfrac{P \xrightarrow{A} P'}{P + Q \xrightarrow{A} P'}$

Constrain : $\dfrac{P \xrightarrow{A} P'}{[P]_I \xrightarrow{A \cup (\mathcal{T}_I^0 - \mathcal{T}_{\mathcal{R}(A)}^0)} [P']_I}$ $(A \subseteq \Sigma_I)$

SumR : $\dfrac{Q \xrightarrow{A} Q'}{P + Q \xrightarrow{A} Q'}$

Recursion : $\dfrac{P \xrightarrow{A} P'}{fix(X.P) \xrightarrow{A} P'[(fix(X.P))/X]}$

Conjunction :

$$\frac{P_1 \xrightarrow{A_1} P_1',\ P_2 \xrightarrow{A_2} P_2'}{P_1\,{}_I\|_J\, P_2 \xrightarrow{A_1 \cup A_2} P_1'\,{}_I\|_J\, P_2'} \quad (I \cap J = \emptyset,\ A_1 \subseteq \Sigma_I,\ A_2 \subseteq \Sigma_J,\ \mathcal{A}_1 \cup A_2 \in \mathcal{D}_{\sigma(I \cup J)})$$

Table 1: Unconstrained Transition System

The final side condition, "$A_1 \cup A_2 \in \mathcal{D}_{\sigma(I \cup J)}$", defines the essence our synchronization model: Assume that P can execute an action $A_P \subseteq \Sigma_I$. Likewise, assume that Q can execute an action $A_Q \subseteq \Sigma_J$. Then, if A_P and A_Q are to synchronize, they must be connected in the following sense:

- If any particle in $a \in A_P$ shares a connection set with some particle $b \in \Sigma_J$, then b *must* appear in A_Q.
- If any particle in $b \in A_Q$ shares a connection set with some particle $a \in \Sigma_I$, then a *must* appear in A_P.

Hiding. In CCSR we do not allow the general use of morphisms on actions. If we did, one could use it to reallocate a particle to a resource other than the one that "owns" it. Even a more restricted use of morphism, which only permitted functions that maintained the resource structure would still be too general – the connection set structure would then be violated. We are left with a very restricted use of morphism – one which reduces fully synchronized particles to their "anonymous" priority representation (see section 3.2). We call this operator Hiding,

and denote it as $P\backslash A$.

Assume that $A \in \mathcal{D}$. We construct the function $\phi_{hide(A)}$ as follows. For all a in Σ,

$$\phi_{hide(A)}(a) = \begin{cases} \phi_\pi(a) & \text{if } a \in A \\ a & \text{otherwise} \end{cases}$$

Thus, all of the particles in A are reduced to their "canonical" priority representation.

Constrainment. The constrainment operator assigns terms to occupy *exactly* the resource set denoted by the index I. First, if the action A utilizes *more* than the resources in I, it is deleted. On the other hand, the particles in A utilize less than the set I, the action is augmented with the "idle" particles from each of the unused resources (see section 3.2).

Proposition 4.1 *All agents in $\mathcal{E}$ are well-defined, in that if $P \in \mathcal{E}$ and $P \xrightarrow{A} P'$, then $A \in \mathcal{D}$.*

The proof follows directly from the definition of the operators. □

5 Priority Equivalence

In our semantics, equivalence between processes is based on the concept of *strong bisimulation.*

Definition 5.1 *For a given transition system $\langle \mathcal{E}, \rightarrow, \mathcal{D} \rangle$, the symmetric relation $r \subseteq (\mathcal{E}, \mathcal{E})$ is a* strong bisimulation *if, for $(P,Q) \in r$ and $A \in \mathcal{D}$,*

1. *if $P \xrightarrow{A} P'$ then, for some Q', $Q \xrightarrow{A} Q'$ and $(P',Q') \in r$, and*
2. *if $Q \xrightarrow{A} Q'$ then, for some P', $P \xrightarrow{A} P'$ and $(P',Q') \in r$.* □

We let "$\sim$" denote *unconstrained strong equivalence*, or the largest such bisimulation with respect to the transition system $\langle \mathcal{E}, \rightarrow, \mathcal{D} \rangle$. As in [11, 12, 13], "$\sim$" exists, and is a congruence over the agents in $\mathcal{E}$.

In this section we define a new transitional system, $\langle \mathcal{E}, \rightarrow_\pi, \mathcal{D} \rangle$ grounded in our notion of priority. From this we derive a measure of *prioritized strong equivalence* based on strong bisimulations. Some care must be taken in this definition to ensure that it yields an equivalence with well-defined properties, properties that reflect a sound model of execution. To do this, we must find an adequate *preemption measure.* A preemption measure is a relation $\prec \in \mathcal{D} \times \mathcal{D}$

such that, for any $P \in \mathcal{E}$, $A, B \in \mathcal{D}$, when P may execute A and $B \prec A$, P will *never* execute B.

Definition 5.2 *For all $A \in \mathcal{D}$, $B \in \mathcal{D}$, $A \preceq B$ if and only if*

$$\mathcal{R}(A) = \mathcal{R}(B) \wedge unres(A) = unres(B) \wedge res(A) \leq_p res(B)$$

The relation "$\preceq$" defines a partial order over $\mathcal{D}$ and thus, we say $A \prec B$ if $A \preceq B$ and $B \not\preceq A$. □

Definition 5.3 *The labeled transition system $\langle \mathcal{E}, \rightarrow_\pi, \mathcal{D} \rangle$ is a relation $\rightarrow_\pi \in \mathcal{E} \times \mathcal{D} \times \mathcal{E}$ and is defined as follows: $(P, A, P') \in \rightarrow_\pi$ (or $P \xrightarrow{A}_\pi P'$) if:*

1. *$P \xrightarrow{A} P'$, and*
2. *For all $A' \in \mathcal{D}$, $P'' \in \mathcal{E}$ such that $P \xrightarrow{A'} P''$, $A \not\prec A'$.* □

The following result shows that "$\prec$" is progress-preserving, in that for a given transition $P \xrightarrow{A} P'$, either $P \xrightarrow{A} P'$ is itself executed under "$\rightarrow_\pi$", or some preempting transition $P \xrightarrow{A'} P''$, with $A \prec A'$, is executed under "$\rightarrow_\pi$".

Lemma 5.1 *If there is an $A \in \mathcal{D}$ and $P, P' \in \mathcal{E}$ such that $P \xrightarrow{A} P'$, then there exist $A' \in \mathcal{D}$, $P'' \in \mathcal{E}$ such that $P \xrightarrow{A'}_\pi P''$ with $A \preceq A'$.*

Proof: Assume that the conclusion is false. Then, setting $A_0 = A$ and inductively applying definition 5.3, we see that $\forall\, i \in \mathbb{N}, i > 0$, there exist $A_i \in \mathcal{D}$, $P_i' \in \mathcal{E}$ such that $P \xrightarrow{A_i} P_i'$ with $A_{i-1} \prec A_i$. So we have the infinite chain over $\mathcal{D}$: $A_0 \prec A_1 \prec A_2 \prec \ldots$, and by definition 5.2,

$$res(A_0) <_p res(A_1) <_p res(A_2) <_p \ldots$$

However, note that $\forall i, j \in \mathbb{N}$, $\mathcal{R}(A_i) = \mathcal{R}(A_j)$, and thus $\forall i, j \in \mathbb{N}$, $\mathcal{R}(res(A_i)) = \mathcal{R}(res(A_j))$. Further, since every $A \in \mathcal{D}$ is finite, $\mathcal{R}(res(A))$ is finite. Thus there are finitely many distinct priorities on sets using the resources in $\mathcal{R}(res(A))$: $\prod_{i \in \mathcal{R}(res(A))}(mp_i + 1)$ to be exact. So such infinite, strictly increasing chains cannot exist. □

We now define our notion of prioritized strong equivalence, "$\sim_\pi$".

Definition 5.4 *We denote "$\sim_\pi$" as the largest strong bisimulation over the transition system $\langle \mathcal{E}, \rightarrow_\pi, \mathcal{D} \rangle$.* □

Relying on the well-known theory found in [11, 12, 13], we state without proof that "$\sim_\pi$" exists, and that it is an equivalence relation over $\mathcal{E}$. The Appendix presents some of the equational characteristics of CCSR with respect to prioritized strong equivalence. Also, by the following theorem, we see that "$\sim_\pi$" forms a congruence over the operators.

Theorem 5.1 *Prioritized strong equivalence is a congruence over agents in $\mathcal{E}$. That is, for agents P, Q and R in $\mathcal{E}$, A in $\mathcal{D}$, such that $P \sim_\pi Q$, we have:*

(1) $A : P \sim_\pi A : Q$ (2a) $P + R \sim_\pi Q + R$ (2b) $R + P \sim_\pi R + Q$

(3a) $P_I \| _J R \sim_\pi Q_I \| _J R$ (3b) $R_I \| _J P \sim_\pi R_I \| _J Q$ (4) $P \backslash A \sim_\pi Q \backslash A$ (5) $[P]_I \sim_\pi [Q]_I$

We shall only present the proof for case (3a); the proofs for the other cases are similar. Before doing so, we require the following lemma.

Lemma 5.2 *Let $P_1, P_2, P_1', P_2' \in \mathcal{E}$, and $A \in \mathcal{D}$. If $P_1{}_I \| _J P_2 \xrightarrow{A}_\pi P_1'{}_I \| _J P_2'$, then $P_1 \xrightarrow{A \cap \Sigma_I}_\pi P_1'$ and $P_2 \xrightarrow{A \cap \Sigma_J}_\pi P_2'$.*

Proof: Denote $A_I = A \cap \Sigma_I$ and $A_J = A \cap \Sigma_J$. By definition 5.3, $P_1{}_I \| _J P_2 \xrightarrow{A} P_1'{}_I \| _J P_2'$. So by the transition rule for Conjunction, we have that $P_1 \xrightarrow{A_I} P_1'$ and $R \xrightarrow{A_J} R'$ with $A_I \cup A_J \in \mathcal{D}_{\sigma(I \cup J)}$. We now claim that $P_1 \xrightarrow{A_I}_\pi P_1'$ and $P_2 \xrightarrow{A_J}_\pi P_2'$.

To the contrary, assume it is false that $P_1 \xrightarrow{A_I}_\pi P_1'$. Then there is a $A_I' \in \mathcal{D}$ and $P_1'' \in \mathcal{E}$ such that $P_1 \xrightarrow{A_I'} P_1''$ with $A_I \prec A_I'$. By definition 5.2, $\mathcal{R}(A_I') = \mathcal{R}(A_I)$, and thus, $A_I' \subseteq \Sigma_I$. Since $unres(A_I') = unres(A_I)$ and $A_I \cup \mathcal{A}_J \in \mathcal{D}_{\sigma(I \cup J)}$, we also have that $A_I' \cup \mathcal{A}_J \in \mathcal{D}_{\sigma(I \cup J)}$. But these are exactly the side conditions required for $P_1{}_I \| _J P_2 \xrightarrow{A_I' \cup A_J} P_1''{}_I \| _J P_2''$. Now, since $res(A_I') >_p res(A_I)$, we have that

$$\begin{aligned} res(A_I \cup A_J) &= res(A_I) \cup res(A_J) \cup res(unres(A_I) \cup unres(A_J)) \\ &<_p res(A_I') \cup res(A_J) \cup res(unres(A_I) \cup unres(A_J)) \\ &= res(A_I') \cup res(A_J) \cup res(unres(A_I') \cup unres(A_J)) \\ &= res(A_I' \cup A_J) \end{aligned}$$

Also, $\mathcal{R}(A_I \cup A_J) = \mathcal{R}(A_I' \cup A_J)$ and $unres(A_I \cup A_J) = unres(A_I' \cup A_J)$, so $(A_I \cup A_J) \prec (A_I' \cup A_J)$. But this contradicts our original assumption. So, $P_1 \xrightarrow{A_I}_\pi P_1'$ and by a similar argument, $P_2 \xrightarrow{A_J}_\pi P_2'$. □

Proof of Theorem 5.1, (3a): Here we make use of the fact that "$\sim_\pi$" is the largest bisimulation with respect to $\langle \mathcal{E}, \rightarrow_\pi, \mathcal{D} \rangle$. Thus to prove that $P_I \| _J R \sim_\pi Q_I \| _J R$, it suffices to find any bisimulation r with respect to $\langle \mathcal{E}, \rightarrow_\pi, \mathcal{D} \rangle$ such that $(P_I \| _J R, Q_I \| _J R) \in r$, since $r \subseteq \sim_\pi$.

We claim that $r = \{(P_I \| _J R, Q_I \| _J R) \mid P \sim_\pi Q \wedge R \in \mathcal{E}\}$ is a strong bisimulation on $\langle \mathcal{E}, \rightarrow_\pi \mathcal{D} \rangle$. By definition, $(P_I \| _J R, Q_I \| _J R)$ is in r. To prove that r satisfies property 1 of definition 5.1, assume there exist $P', R' \in \mathcal{E}$, $A \in \mathcal{D}$ such that

$$(\dagger) \qquad P_I \| _J R \xrightarrow{A}_\pi P'_I \| _J R'.$$

It suffices to show that for some Q', $Q_I \| _J R \xrightarrow{A}_\pi Q'_I \| _J R'$ and that $P' \sim_\pi Q'$. Let $A_I = A \cap \Sigma_I$ and $A_J = A \cap \Sigma_J$. By lemma 5.2 we have that $P \xrightarrow{A_I}_\pi P'$ and $R \xrightarrow{A_J}_\pi R'$.

Now because $P \sim_\pi Q$, we have that $Q \xrightarrow{A_I}_\pi Q'$, with $P' \sim_\pi Q'$. To finish showing that r enjoys property 1 of definition 5.1, we must prove that $Q_I \| _J R \xrightarrow{A}_\pi Q'_I \| _J R'$. Obviously part 1 of definition 5.3 is satisfied, so assume part 2 is violated. That is, assume there is some $A' \in \mathcal{D}$, $Q'', R'' \in \mathcal{E}$ such that $Q_I \| _J R \xrightarrow{A'} Q''_I \| _J R''$ with $A \prec A'$. Then by lemma 5.1, we know there exist some $A'' \in \mathcal{D}, Q''', R''' \in \mathcal{E}$ such that $Q_I \| _J R \xrightarrow{A''}_\pi Q'''_I \| _J R'''$ with $A' \preceq A''$, and hence $A \prec A''$.

Letting $A''_I = A'' \cap \Sigma_I$, by lemma 5.2 have that $Q \xrightarrow{A''_I}_\pi Q'''$, and since $P \sim_\pi Q$, there is also some $P''' \in \mathcal{E}$ such that $P \xrightarrow{A''_I} P'''$. But this implies that $P_I \| _J R \xrightarrow{A''} P'''_I \| _J R'''$ with $A \prec A''$, again contradicting our assumption $(\dagger)$. So $Q_I \| _J R \xrightarrow{A}_\pi Q'_I \| _J R'$ and the proof of property 1 is complete. By a symmetric argument, r satisfies property 2 in definition 5.1, and so r is a bisimulation. □

The next theorem shows that the strong equivalence defined by "$\sim_\pi$" is coarser than that defined by "$\sim$".

Theorem 5.2 *Let $P, Q \in \mathcal{E}$ and assume P is strongly equivalent to Q under the transition system $\langle \mathcal{E}, \rightarrow, \mathcal{D} \rangle$, (that is, $P \sim Q$). Then $P \sim_\pi Q$.*

Proof: We need only show that the relation "$\sim$" is a bisimulation on the transition system $\langle \mathcal{E}, \rightarrow_\pi, \mathcal{D} \rangle$. Assume $P \sim Q$, and let $P \xrightarrow{A}_\pi P'$. By definition 5.3, $P \xrightarrow{A} P'$. Since $P \sim Q$, there is some $Q' \in \mathcal{E}$ such that $Q \xrightarrow{A} Q'$ with $P' \sim Q'$. Thus we must prove that $Q \xrightarrow{A}_\pi Q'$. If this is false, there is some $A' \in \mathcal{D}$, $Q'' \in \mathcal{E}$ such that $Q \xrightarrow{A'} Q'$ and $A \prec A'$.

But since $P \sim Q$, there is also some $P'' \in \mathcal{E}$ such that $P \xrightarrow{A'} P'$, which is a contradiction. Similarly, if $Q \xrightarrow{A}_{\pi} Q'$, then for some P', $P \xrightarrow{A}_{\pi} P'$ with $P' \sim Q'$. □

We now turn briefly to the subject of infinite terms. First we give the standard extension of "$\sim_\pi$" to terms with free variables.

Definition 5.5 *Let the set* $\{X_1, \ldots, X_n\}$ *include the free variable in the terms* P *and* Q. *Then* $P \sim_\pi Q$ *if, for all agents* $P_1, \ldots, P_n$, $P[P_1/X_1, \ldots, P_n/X_n] \sim_\pi Q[P_1/X_1, \ldots, P_n/X_n]$. □

With this definition we are able to show that "$\sim_\pi$" forms a congruence over recursive terms. For brevity we state the result without proof, which is performed by induction on transitional inference.

Theorem 5.3 *If* $P \sim_\pi Q$, *and at most* X *is free in* P *and* Q, *then* $fix(X.P) \sim_\pi fix(X.Q)$. □

6 An Example

In this section we present a simple example that illustrates the role of priority in CCSR. Our system is a time-critical Producer/Consumer problem that has two producers and one consumer; both producers possess real-time constraints that must be satisfied to ensure that they operate correctly. Our goal is to show that in some real-time applications, a system's correctness can hinge on the ability to implement priority.

First we introduce some notation that facilitates a concise specification of our system. For an action $A \in \mathcal{D}$ a term $P \in \mathcal{E}$, and a positive integer t, let "$\delta_t A : P$" be the term that *must* execute the A action within t time units, but *may* execute $\emptyset$ up to that point. That is:

$$\delta_t A : P = \begin{cases} A : P & \text{if } t = 1 \\ (A : P) \ + \ (\emptyset : \delta_{t-1} A : P) & \text{otherwise} \end{cases}$$

Also, let "$A^t : P$" be the term that executes the action A for t time units before proceeding to P:

$$A^t : P = \begin{cases} A : P & \text{if } t = 1 \\ A : A^{t-1} : P & \text{otherwise} \end{cases}$$

The System is composed of three agents: $Consumer$, $Producer_1$ and $Producer_2$. Initially, $Producer_1$ can choose either to idle or to enter its "production" phase. In this phase, it "produces" for 1 time unit by executing the action $\{p_1\}$. Then it attempts to interrupt the $Consumer$ by executing the action $\{int_1!\}$. However, if the interrupt is not accepted within 2 time units, $Producer_1$ deadlocks. If it is accepted, the agent has a latency of 2 time units before re-starting the loop:

$$Producer_1 = fix(X_{P_1}.\ (\emptyset : X_{P_1})\ +\ (\{p_1\} : \delta_2\{int_1!\} : \emptyset^2 : X_{P_1})\ )$$

$Producer_2$ is exactly like $Producer_1$ except for one fact: it gives the $Consumer$ 3 time units to accept its interrupt.

$$Producer_2 = fix(X_{P_2}.\ (\emptyset : X_{P_2})\ +\ (\{p_2\} : \delta_3\{int_2!\} : \emptyset^2 : X_{P_2})\ )$$

The $Consumer$ waits for either $Producer_1$ or $Producer_2$ to interrupt. Once either interrupt is received, there is a digestion period of 2 time units, during which the action $\{c\}$ is executed in a critical section.

$$Consumer = fix(X_C.\ (0 : X_C)\ +\ (\{int_1?\} : \{c\}^2 : X_C)\ +\ (\{int_2?\} : \{c\}^2 : X_C)\ )$$

Let $Producer_1$ be hosted on resource 1, and let $\{p_1, int_1!\} \subseteq \Sigma_1$. Further, let $\pi_1(p_1) = 0$ and $\pi_2(int_1) = 0$, which makes $Producer_1$ a "passive" agent. Similarly, let $Producer_2$ be hosted on resource 2, and $\{p_2, int_2!\} \subseteq \Sigma_2$. Let $\pi_2(p_2) = 0$ and $\pi_2(int_2) = 0$.

Let $Consumer$ be hosted on resource 3, with $\{c, int_1?, int_2?\} \subseteq \Sigma_3$; let $\pi_3(c) = 1$, $\pi_3(int_1?) = 2$, and $\pi_3(int_2?) = 1$. The only connection sets of importance are $C_1 = \{int_1!, int_1?\}$ and $C_2 = \{int_2!, int_2?\}$. All other particles are assumed to belong to their own connection sets. From this priority scheme, we have the property that if both interrupts "$int_1!$" and "$int_2!$" are raised simultaneously, the $Consumer$ will handle "$int_1!$".

The entire system is posed as follows:

$$[(Producer_{1\,\{1\}} \|_{\{2\}} Producer_2)_{\{1,2\}} \|_{\{3\}} Consumer\,]_{\{1,2,3\}}$$

We claim that this priority ordering is exactly the key to keeping the system deadlock-free. That is, it contains no proper derivatives that terminate in NIL. Assume that both interrupts "$int_1!$" and "$int_2!$" are raised simultaneously. In the priority-based system, "$int_1!$" is

handled first, and there will be a delay of exactly 3 time units that "$int_2!$" is forced to wait. But $Producer_2$ *can* wait that long, and because $Producer_1$ cannot attempt to raise another interrupt for at least 4 time units, the system will remain safe. On the other hand, if "$int_2!$" had been serviced first, the system would not have been safe, as $Producer_1$ cannot wait for 3 time units to have "$int_1!$" serviced. In a semantics without priority structure (e.g., under the "$\rightarrow$" transition system), the choice between the two interrupts would be nondeterministic. Thus, the system would not be deadlock-free.

7 Conclusion

In this paper we have presented a synchronous, priority-based process algebra called CCSR. Influenced by SCCS, this calculus gives an appropriate equational characterization of the CSR design language. The calculus, accompanied by a proof system, facilitates the syntactic manipulation of CCSR terms based on both resource configuration and priority ordering. Thus the formalism can be considered one step toward unifying abstract, real-time specifications with their resource-specific implementation environments.

We are currently incorporating a dynamic priority structure into the syntax and semantics of the CCSR model. Because deadline-driven scheduling can be formulated in terms dynamic priority, we will then be able to reason about the properties of real-time scheduling algorithms, and their efficacy in guaranteeing the deadlines of the processes with which they interact.

References

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

[2] N. Francez, D. Lehmann, and A. Pnueli. A Linear History Semantics for Distributed Programming. *Theoretical Computer Science*, 32:25–46, 1984.

[3] R. Gerber and I. Lee. Communicating Shared Resources: A Model for Distributed Real-Time Systems. In *Proc. 10th IEEE Real-Time Systems Symposium*, 1989.

[4] R. Gerber and I. Lee. The Formal Treatment of Priorities in Real-Time Computation. In *Proc. 6th IEEE Workshop on Real-Time Software and Operating Systems*, 1989.

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

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

[7] C. Huizing, R. Gerth, and W.P. de Roever. Full Abstraction of a Denotational Semantics for Real-time Concurrency. In *Proc.* 14^{th} *ACM Symposium on Principles of Programming Languages*, pages 223–237, 1987.

[8] R. Janicki and P. Lauer. On the Semantics of Priority Systems. In *Proc. of Int. Conf. on Parallel Processing*, 1988.

[9] R. Koymans, R.K. Shyamasundar, W.P. de Roever, R. Gerth, and S. Arun-Kumar. Compositional Semantics for Real-Time Distributed Computing. In *Logic of Programs Workshop '85, LNCS 193*, 1985.

[10] I. Lee and V. Gehlot. Language Constructs for Distributed Real-Time Programming. In *IEEE Real-Time Systems Symposium*, 1985.

[11] R. Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25:267–310, 1983.

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

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

[14] Gordon Plotkin. *A Structural Approach to Operational Semantics.* Technical Report DAIMI FN-19, Computer Science Dept., Aarhus University, 1981.

[15] A. Salwicki and T. Müldner. On the Algorithmic Properties of Concurrent Programs. In *Proceedings of Logic of Programs, LNCS 125*, 1981.

Appendix: Equational Characteristics of CCSR

(1) $P + NIL \sim_\pi P$

(2) $P + P \sim_\pi P$

(3) $P + Q \sim_\pi Q + P$

(4) $(P + Q) + R \sim_\pi P + (Q + R)$

(5) $(A : P) + (B : Q) \sim_\pi A : P \quad \text{if } A \prec B$

(6) $(A : P)_I \| _J (B : Q) \sim_\pi \begin{cases} (A \cup B) : (P_I \| _J Q) & \text{if } A \subseteq \Sigma_I,\ B \subseteq \Sigma_J,\ A \cup B \in \mathcal{D}_{\sigma(I \cup J)} \\ NIL & \text{otherwise} \end{cases}$

(7) $P_I \| _J NIL \sim_\pi NIL$

(8) $P_I \| _J Q \sim_\pi Q_J \| _I P$

(9) $(P_I \| _J Q)_{(I \cup J)} \| _K R \sim_\pi P_I \| _{(J \cup K)} (Q_J \| _K R)$

(10) $P_I \| _J (Q + R) \sim_\pi (P_I \| _J Q) + (P_I \| _J R)$

(11) $(A : P) \backslash B \sim_\pi \begin{cases} \phi_{hide(B)}(A) : (P \backslash B) & \text{if } sync(A \cap B) \\ NIL & \text{otherwise} \end{cases}$

(12) $(P + Q) \backslash B \ \sim_\pi \ P \backslash B + Q \backslash B$

(13) $NIL \backslash B \sim_\pi NIL$

(14) $[[P]_I]_J \sim_\pi \begin{cases} [P]_J & \text{if } I \subseteq J \\ NIL & \text{otherwise} \end{cases}$

(15) $([P]_I)_I \| _J ([Q]_J) \sim_\pi [P_I \| _J Q]_{I \cup J}$

(16) $[P]_I + [Q]_I \sim_\pi [P + Q]_I$

(17) $[A : P]_I \sim_\pi \begin{cases} (A \cup (\mathcal{T}_I^0 - \mathcal{T}_{\mathcal{R}(A)}^0)) : [P]_I & \text{if } A \subseteq \Sigma_I \\ NIL & \text{otherwise} \end{cases}$