Sādhanā, Vol. 17, Part 1, March 1992, pp. 75–93. © Printed in India.

Compositional priority specification in real-time distributed systems

R K SHYAMASUNDAR[§] and L Y LIU[‡]

[§]Computer Science Group, Tata Institute of Fundamental Research, Homi Bhabha Road, Bombay 400 005, India
[‡]IBM Programming Systems, Cary Laboratory, 11000 Regency Parkway, Cary, NC 27511, USA

Abstract. In this paper, we develop a compositional denotational semantics for prioritized real-time distributed programming languages. One of the interesting features is that it extends the existing compositional theory proposed by Koymans *et al* (1988) for prioritized real-time languages preserving the compositionality of the semantics. The language permits users to define situations in which an action has priority over another action without the requirement of preassigning priorities to actions for partially ordering the alphabet of actions. These features are part of the languages such as Ada designed specifically keeping in view the needs of real-time embedded systems. Further, the approach does not have the restriction of other approaches such as prioritized internal moves can pre-empt unprioritized actions etc. Our notion of priority in the environment is based on the intuition that a low priority action can proceed only if the high priority action cannot proceed due to lack of the handshaking partner at that point of execution. In other words, if some action is possible corresponding to that environment at some point of execution then the action takes place without unnecessary waiting. The proposed semantic theory provides a clear distinction between the semantic model and the execution model – this has enabled us to fully ensure that there is no unnecessary waiting.

Keywords. Compositional specification; real-time distributed systems; priority specification; message passing models.

1. Introduction

Many approaches have been proposed for the modelling of communicating agents (Milner 1980), reactive systems (Pnueli & Harel 1988, pp. 84–98), and real-time distributed systems (Roscoe 1984; Koymans *et al* 1985, 1988). Most of the above studies have ignored the notion of *priority*. This is not satisfactory as priority is very important in the development of predictable systems (Stankovic 1988). Priority specification is required when one or more events happen at the same time and some events have greater importance than others. Typical examples of actions which require special treatment include interrupts in hardware systems and timeouts in communication protocols. Ada and Occam are two examples of programming languages that allow specification of priority. One can broadly distinguish priority specification into the following categories:

(1) Partial order on the alphabet of actions/events.
(2) Priority specification through evaluating expressions dynamically and assigning priorities to actions/statements.
(3) Priority of actions depending on the environment.

Perhaps the first formal study of priorities has been done in the context of process algebra in Baeten *et al* (1985). In this study, prioritization operators have been added and the consistency of the set of equations have been studied. In other words, the study falls into type (1) described above. That is, it assumes a global partial ordering of the actions (of the transition system). From a behavioural equivalence point of view, a study has been made in Cleaveland & Hennessy (1990) through the study of priority operators of type (1) in the context of calculus of communicating systems (CCS). Congruence has been obtained through the notion of *patient processes* by placing essentially the restriction that only prioritized internal actions have the pre-emptive power. The desirability of overcoming this restriction follows from the examples discussed in Cleaveland & Hennessy (1990).

A study of the second type of priority has been made in Hooman (1989) in the context of the study of distributed multi-processing. Here, priorities are attached to statements and the priorities of statements on different processors are incomparable. In other words, the relative ordering of priorities on a single processor determines the execution order; for example, if two synchronized actions have different priorities, then the priority for the synchronized action is given by the minimum of the two. Another way of overcoming this problem is to use *one-way* priority, that is, either the input or output action can be assigned priority but not both.

Let us look at the specification of reactive systems. Reactive systems maintain a continuous interaction with their environment at a speed determined by the environment rather than the program itself. In other words, the outputs may affect future inputs due to feedback. One of the primary goals of the study of reactive (real-time) systems is to develop predictable systems (cf. Stankovic 1988). Thus, it becomes essential to predict the action of each component in the context of nondeterministic interacting environment. For this purpose, priority plays a vital role. Hence, priority of type (3) discussed above plays an important role in distributed reactive systems. Let us look at the priority specification characteristics in Ada which has been designed keeping in view the design of real-time embedded systems. In Ada each task may (but need not) have a priority. We quote below relevant aspects from the revised Ada manual (cf. 9·8) given in Gehani (1983):

- The specification of priority is an indication given to assist the implementation in the allocation of processing resources to parallel tasks when there are more tasks eligible for execution than can be supported simultaneously by the available processing resources. The effect of priorities on scheduling is defined by the following rule:

  If two tasks with different priorities are both eligible for execution and could sensibly be executed using the same physical processors and the same other processing *resources, then it cannot be the case that the task with the lower priority is executing while the task with the higher priority is not.*

- The above rule essentially corresponds to the *pragma* feature. The most important aspect related to priority specification in the context of a rendezvous is cited below:
  For tasks of the same priority, the scheduling is not defined in the language.... The priority of the task is static and therefore fixed. However, the priority during a rendezvous is not necessarily static since it also depends on the priority of the task calling the entry.

It can be seen from the above that priority specification in Ada corresponds to that of type (3) described above. In appendix A, we illustrate through an example priority specification in Ada by considering the problem of minimizing head movement during a disc access. Priority specifications corresponding to type (3) given above play an important role in the specification of process-control systems.

A limited treatment of priority of type (3) has been reported in Pitassi *et al* (1986). The study in Pitassi *et al* (1986) corresponds to a special case of the prioritized alternative construct of Occam (cf. Occam 1984). The limitation can be understood by the informal interpretation of the alternative construct given below: Upon entering a prioritized alternate (ALT) statement, a linear sequence of all the open guards of the prioritized statement is constructed; if one or more of the open guards is successful upon entry, then the first successful guard in that sequence is selected and the corresponding statement is executed. If none of the open guards is successful upon entry, then the prioritized ALT construct is treated as if it were an ordinary ALT construct.

So far in the literature, there is no indication as to what approach would be realistic and useful. In this paper, we provide a formal semantics for priority in the context of real-time distributed programming languages that have the feature of specifying priority in an environment rather than providing a global partial order of actions in the system. The main contributions of the paper can be summarized as follows.

- An understanding of the notion of priority in the context of environment (i.e., real-time distributed concurrency) is provided. Our approach does not have the restriction (as in Cleaveland & Hennessy 1990) that only prioritized internal moves can pre-empt unprioritized actions. Our notion of priority in the environment is based on the intuition that a low priority action can proceed only if the high priority action cannot proceed due to lack of the handshaking partner at that point of execution. In other words, if some action is possible corresponding to that environment at some point of execution then the action takes place without unnecessarily waiting. It must be clear that such an approach clearly satisfies the requirement given in Lamport (1985) that realistic priority specification should not involve unnecessary waiting.
- A compositional semantic characterization of real-time distributed languages with priority is presented and, thus, forms a basis for the compositional proof theories of languages such as Ada and Occam.

The rest of the paper is organized as follows: Section 2 describes an abstraction of prioritized real-time distributed concurrency in terms of an extension of real-time communicating sequential processes (CSP-R) (Koymans *et al* 1988) and is referred to as CSP-R$_p$; further, the language syntax and informal interpretation of CSP-R$_p$ is discussed. Section 3 describes the real-time model and the execution model, and § 4 describes the semantic domain and semantic equations. Section 5 discusses parallel composition both informally and formally; towards, the end of § 5, we discuss the impact of various parameters including those that affect maximal parallelism on the prioritized semantics proposed. The paper concludes with a discussion of the results and the ongoing work.

2. Language syntax

For ease of presentation, we use language CSP-R$_p$ (prioritized real-time CSP) instead of Ada. It may be noted that CSP-R$_p$ is an extension of CSP-R described in Koymans *et al* (1988). In Koymans *et al* (1988), it has been shown that Ada tasking and real-time features can be simulated by CSP-R. Further, CSP-R$_p$ has the priority features corresponding to that of type (3) discussed above. In CSP-R$_p$ processes communicate via unidirectional channels, and a channel connects exactly two processes.

D, W – stand for channel variables;
x, u – stand for program variables;
e – stands for expressions;
b – stands for boolean expressions;
p – stands for priority expressions.

Program: $P ::= S \mid N$
Statement: $S ::= x := e \mid \mathrm{skip} \mid g \mid \mathrm{wait}\, d \mid S_1 ; S_2 \mid A \mid {*A} \mid [N]$
Guard: $g ::= b \mid D?x \mid W!e \mid \mathrm{wait}\, d \mid b; D?x\ \mathbf{by}\ \mathbf{p} \mid b; W!e\ \mathbf{by}\ \mathbf{p} \mid b; \mathrm{wait}\ d$
Alternative: $A ::= [\square_{i=1}^{n} g_i \rightarrow S_i]$
Network: $N ::= S_1 \parallel S_2$

The informal semantics of the language follows on the lines of the semantics of CSP-R given in Koymans *et al* (1988) and Huizing *et al* (1987). It may be observed that in our language, priority can be assigned to only I/O guards and not for pure boolean or delay guards since the local priorities can be manipulated through the boolean expressions. Further, we assume that pure boolean guards have priority over the I/O guards. Again, note that the priority for local action/communication can always be manipulated through the boolean parts of the expressions. For ease of understanding, we provide an informal interpretation of those commands that are quite different from those of CSP described in Hoare (1978).

Interpretation of *alternative command*: if none of the guards is open, then execution aborts; otherwise, check whether there is at least one open pure boolean guard and select one of them nondeterministically. In case there is no open boolean guard but there is at least one other open guard the execution proceeds in the following way: Compute the *waitvalue* which is defined to be infinite if there are no open wait guards; otherwise, it is defined to be the maximum of 1 and the minimum of the values of the durations of the open wait guards. Let us denote the waitvalue by d. As soon as there is a possible semantic match for the open I/O command, the communication action takes place over the channel that has the highest priority (the selection of the semantically matching commands of equal priority is nondeterministic). However, if no semantic match takes place within d units, then one of the open wait guards with waitvalue equal to d is selected nondeterministically. It may be observed that we have assumed a priority[1] for the boolean guards over the I/O guards. Note that the I/O guards can be assigned any positive priority; we assume a default priority of 1 in case there is no explicit specification. No explicit priority can be assigned to delay guards; in other words, the delay guards are assumed to have priority 0. This is consistent with the semantics of Ada.

For example, consider a system consisting of three robots: P_1, P_2, and P_3. Robots P_1 and P_2 compete for some resource service from P_3 such that P_1 or P_2 have to wait only linearly (i.e. one can wait till the other gets the service) for service. The controller for P_3 can be abstracted through the following control program in CSP-R$_p$:

$$P_3 ::i := 2; * [D?x\,\mathbf{by}\,i \rightarrow i := 1; \text{serve-}P_1 \,[]\, W?x\,\mathbf{by}\,3\text{-}i \rightarrow i := 2; \text{serve-}P_2].$$

Informally, in the program the priority dynamically switches from communication over D to that of communication over channel W. It must be noted that this does not mean that the communications over D and W alternate. What this means is that if this program is placed in an environment wherein the communications over both D and W are available, then the net result will be that of alternating communications over D and W. However, in the context of environments wherein the communications over D and W are not always ready, the behaviour differs. In other words, one of the important points to be noted is that programs with priority clause in general cannot be transformed to a program that does not use any priority clause unless the environment is a priori given directly or indirectly. Note that if several requests having the same priority, arrive at the same time, then the choice is nondeterministic. To be more specific, we consider the possible set of actions in the following example.

$$\begin{aligned} P1 ::= [&D1?x\,\mathbf{by}\,1 \rightarrow S1 \\ []&D2?y\,\mathbf{by}\,2 \rightarrow S2 \\ []&D3?z\,\mathbf{by}\,3 \rightarrow S3] \end{aligned}$$

Let t_0 be the time of arrival at this select statement. Let us assume that some communication takes place over some channel at time $t_1 (t_1 \geqslant t_0)$. If we assume that there is no unnecessary waiting, then the general condition is that t_1 was the earliest time at which the communication could take place. The possible interpretations are given by:

1. Communication over $D3$ takes place at t_1; this does not require any other condition as $D3$ is the channel with the highest priority.
2. Communication over $D2$ takes place at t_1; this implies that communication over $D3$ was impossible since t_0.
3. Communication over $D1$ takes place at t_1; this implies that communications over $D2$ and $D3$ were impossible since t_0.

It must however be noted that how to determine the possibility of the communication or not is an implementation issue (i.e. implementation of the synchronous communication) and is not explicit in CSP-R$_p$. Note however, in Ada it is possible to determine such a possibility through the entry queues.

[1] One of the reasons for this assumption is that in our semantics for the sake of simplicity we have assumed that expression evaluation takes no time. This restriction can be removed very easily. In fact, no generality will be lost with such an assumption.

Hiding, $[N]$, has no effect on the execution of N but changes what can be observed about such executions. In other words, communications along channels in N are internalized and cannot be observed any more.

3. Model of real-time and execution model

The language CSP-R$_p$ is a synchronous language. We assume that time proceeds in discrete time steps. This is consistent with the argument given in Pnueli & Harel (1988, pp. 84–98) that integer time domain will be appropriate for synchronous programming languages, since all the processes refer to the same global clock and operate only at certain points of time. Thus, our time domain is the set of natural numbers. For the sake of simplicity, we further assume that all primitive actions (such as assignment and communication) take one unit of time. Parameterization with respect to transmission time of the network and range of values for actions (Koymans *et al* 1988) are ignored for the sake of simplicity. In other words, real-time is modelled by relating the ith element of a history with the ith tick of a conceptual global clock. However, it should be noted that we do not either imply the existence of a global clock or assume the tightness of synchronization of the processor clocks.

A real-time execution model is useful only if we can make some assumptions about the progress each process is due to make. In this paper, we use the *maximal parallelism* model described in Koymans *et al* (1988) (we use MAXPAR as an abbreviation). In the MAXPAR model, at any instant of time all actions that can be started without violating synchronization constraints will be initiated. In other words, we assume the existence of a processor for every process. Such an assumption removes the need of considering resource scheduling. That is, a process is allowed to be idle only if all communication partners are unwilling to communicate and no local actions are possible at that point. Towards, the end of the paper, we discuss other aspects of real-time models.

4. Denotational semantics of CSP-R$_p$

Semantic domain

The domain consists of non-empty prefix-closed set of pairs: each pair consisting of a state and a finite history leading to this state. Infinite behaviours are modelled by their sets of finite approximations. In order to enforce maximal parallelism, we have to record whether the processes are suspended, and if so, on which communication the process is suspended etc. To enforce consistency of priority the semantics has to encode the priority information in a suitable manner so that the semantics remains compositional. These aspects are discussed formally in the following.

The domain is, $D_{\text{dom}} = P(\bar{S} \times H)$ where

- $\bar{S} = S \cup \{\perp\}$, S being the *set of proper states* (i.e. partial functions from *Id* (set of identifiers) to $\mathscr{V}$ (set of expression values), and
- H = set of sequences of records of the form: $\langle A, G \rangle$, where A is a set of *communication assumption records* referred to as the *Action set* and G provides the partial order

information with reference to the action set necessary for checking priority consistency.

The communication assumption records (CAR for short) are of the following types.

(1) *Communication records* of the form (D, v) where D is a channel name and $v \in \text{VAL}$ (domain of values). If the ith element is of the form (D, v), it can be interpreted as sending or receiving the value v over channel D at the ith tick of the conceptual global clock.
(2) *Ready records* of the form $R(A)$ where A is a subset of channel names. If the ith element of a history is of the form $R(A)$, it can be interpreted as the willingness of the process to communicate over the channels in A at the ith tick of the conceptual global clock, and the impossibility of communication since one of the partners is not able to communicate.
(3) *Internal moves* (denoted $\square$). If the ith element is $\square$, it can be interpreted as a local action at the ith tick of the conceptual global clock.

In other words, the observable actions are: (a) communication actions with the associated priority, (b) the time of the observable actions, and (c) the state of the time of termination.

In the a priori semantics, we keep information about the priority of the various actions in terms of triples $\langle H, D, L\rangle$ with the following interpretation:

- D is the channel over which communication is assumed to have taken place.
- H is the set of channels that have higher priority over D.
- L is the set of channels having priority less than or equal to that of D.

Note. Informally, $\langle H, D, L\rangle$ has the following interpretation.

- Processes are ready to communicate over the channels in $H \cup \{D\} \cup L$.
- Communication takes over D; that is, D is the channel that has a partner and there is no other channel that has a priority higher than D having a ready partner; H is the set of channels that have higher priority over that of D and D has a higher priority over those of L.

Obviously, sets H, $\{D\}$, and L are mutually disjoint.

Before describing the semantic domain and the equations, we formalise the priority triples.

DEFINITION 1

Consider the triple $\langle H, D, L\rangle$ where H and L are subsets of channel names and D is a channel name. Then, the triple $\langle H, D, L\rangle$ denotes the relation $\{(a, D) \mid a \in H\} \cup \{(D, b) \mid b \in L\}$.

Note. (1) By the underlying graph of $\langle H, D, L\rangle$, we mean a directed graph (V, E) where $V = H \cup \{D\} \cup L$ and E is the set of all directed edges (a, b) corresponding to (a, b) in the underlying relation. We refer to these graphs are *priority graphs*.
(2) Priority (or precedence) graphs are said to be inconsistent if they are not acyclic. We use $\perp$ to denote inconsistent priority graphs.

DEFINITION 2

Let G_1 and G_2 be priority graphs. Then,

$$join(G_1, G_2) = \begin{cases} \perp, & \text{if } G_1 \cup G_2, \text{ is not acyclic,} \\ G_1 \cup G_2, & \text{otherwise.} \end{cases}$$

In the following, we describe formally the domain.
Now,

$$\bar{S} \times H = \{\langle \sigma, h \rangle | \sigma \in \bar{S}, h \in H \text{ and } |h| < \infty\}.$$

A set $X \in$ STATE $\times$ HISTORY is said to be prefix-closed iff $\forall \langle \sigma, h \rangle \in X$, if $h' \leqslant h$ then $\langle \perp, h' \rangle \in X$.

The prefix-closure of X, denoted PFC (X), is defined as

$$X \cup \{\langle \perp, \lambda \rangle\} \cup \{\langle \perp, h' \rangle | \exists \sigma \exists h (\langle \sigma, h \rangle \in X \wedge h' \leqslant h)\}.$$

The domain consists of all nonempty prefix-closed elements of D_{dom}. Note that the domain forms a complete lattice with set-inclusion ($\subseteq$); the lub is obtained by $\cup$ (set-union) and the least element is $\{\langle \perp, \lambda \rangle\}$.

The meaning function is of the form, $M[\![\text{Statement}]\!]: \bar{S} \rightarrow D_{\text{dom}}$ defining the meaning of statements from $\bar{S}$ to D_{dom}. For defining the meaning of alternative command compositionally, we define an auxiliary function $G[\![g, A]\!]$ from $\bar{S}$ to D_{dom} which gives the meaning of the guard g in the context of a set A of alternative guards. Denoting the set of alternative guards by A, we get:

$$G[\![g]\!]: A \rightarrow \bar{S} \rightarrow D_{\text{dom}}.$$

Notation. (1) Let $h = \langle A1, G1 \rangle \circ \langle A2, G2 \rangle \circ \cdots \circ \langle An, Gn \rangle$ be a finite history of length n. Then, we use $h^1 = A1 \circ A2 \circ \cdots \circ An$ to denote the projection of the history to the first component while we use $h^2 = G1 \circ G2 \circ \cdots \circ Gn$ to denote the projection of the history to the second component; the length of h is denoted by $|h|$, and the kth element of h is denoted by $h[k]$.
(2) The length of the history (trace) denotes the time taken for arriving at the point; the empty set is denoted by $\square$. Note that $R(\phi)$ also denotes $\square$ as well as $\langle \phi, v \rangle$ in our notation.
(3) For convenience, we use $\langle \sigma, A \rangle$ to denote, $\langle \sigma, \langle A, G \rangle \rangle$ when G is an empty graph.
(4) If B is a set of histories, then we use $\{\langle \sigma, B \rangle\}$ to denote $\{\langle \sigma, h \rangle | h \in B\}$.
(5) We represent singleton action sets of the form $\{\alpha\}$, where α is some communication assumption record (i.e., $\langle D, v \rangle$, $R(A)$, or $\square$), by α itself; we also omit the set symbol for ready records. For example, $\{\{\square\}\{\square, R(\{D, W\})\}\{\langle D, v \rangle\}\}$ is denoted by $\square\{\square, R(D, W)\}\langle D, v \rangle$. We omit the concatenation operator ($\circ$) whenever it is clear from the context.
(6) Whenever it is clear, we do not enclose the elements of the trace within the angular brackets.

The semantic equations for the language constructs are formally defined in appendix B; in the following, we informally discuss features of the parallel composition.

5. Parallel composition

Informal approach

The semantics is based on the real-time semantics discussed in Koymans *et al* (1988). The semantic domain consists of state-history pairs. The history is nothing but the traces of Koymans *et al* (1988) enriched with the priority information. In the following, we informally discuss how priority consistency is ensured in the parallel composition. Our semantics has two stages.

- *A priori semantics* – Here, we consider the semantics of each process in an isolated way.
- *Binding of the processes* – Here, the meaning of the program is obtained by considering the meaning of the component processes.

While composing the processes, we check for the mutual consistency of the isolated assumptions made in each of the processes. As already mentioned, the semantics of each process is in the domain of state-history pairs. For example, let us consider two state history pairs, $\langle s_1, h_1 \rangle$ and $\langle s_2, h_2 \rangle$ in processes P_1 and P_2, respectively. The binding of the states should be understood easily as processes do not share any common variables. Let us look at the merging of the histories. The two histories are said to be consistent iff:

(1) *Communication compatible* – That is, for every communication assumption over some channel, say D, in some process P_1 there is a corresponding matching communication partner in some process P_2 at the same time.
(2) *MAXPAR consistent* (*no unnecessary waiting*) – Check that there is no unnecessary waiting, that is, histories do not indicate a situation where both the processes are waiting for a communication that the other process can provide. This can be verified by checking that there is no common ready-record between any two histories at the same time.
(3) *Priority consistent* – Check that the histories are priority consistent, that is, histories do not indicate a situation wherein a lower priority request has been accepted in spite of the possibility of a higher priority request.

In the following, we informally show how priority consistency is ensured; communication compatibility as well as MAXPAR consistent are essentially the same as in Koymans *et al* (1988); the formal equations are given in appendix B.

For the understanding of priority consistency, let us consider the priority assumptions $\langle H, D, L \rangle$ and $\langle H', D', L' \rangle$ in the histories of any two processes at some time t. If the sets H, H', $\{D\}$, $\{D'\}$, L, L' are mutually disjoint then priority consistency follows trivially. Further, it must be noted that if the two CAR under consideration are neither communication compatible nor MAXPAR compatible then there is no need of a separate check for priority consistency. The basic idea for establishing priority consistency is to derive the underlying graph and check whether there are inconsistent (circular) precedences. The important aspect of the graph construction is that it is done incrementally and the existing graph is augmented only if it is not inconsistent after augmentation. In the following, we analyse the relations among these sets and show how inconsistent histories can be removed; a separate priority check is resorted to only if the inconsistency does not follow either from communication or MAXPAR incompatibility. For the sake of informal reasoning, we do a case analysis.

The important cases are:

(1) $H \cap H' \neq \phi$:
- In such a situation, clearly two processes are waiting unnecessarily, since D has a lower priority than those of the channels in $H \cap H'$ and D' has a lower priority than those of the channels in $H \cap H'$. In other words, this is not MAXPAR consistent. Thus, if we can ensure that the histories are MAXPAR consistent, then there is no need for checking again for priority consistency.

(2) $D' \in H$
- The situation corresponds to the situation of no partner for communication over D' – corresponding to communication incompability; hence, there is no need for a separate check for priority consistency.

(3) $D' \in L$
- This case again can be ruled out on the same lines as case (2).

(4) $D \in H'$
- This case again can be ruled out on the same lines as case (2).

(5) $D \in L'$
- This case again can be ruled out on the same lines as case (2).

(6) $L \cap L' \neq \phi$
- There is no need for checking priority consistency since by definition, we assume that communication does not take place over low priority channels.

(7) $H \cap L' \neq \phi \wedge D = D'$
- Consider the precedence graph shown in figure 1 (in the graph $\rightarrow$ denotes higher priority than) for this case. It can be easily seen that the priorities are assigned inversely in the two processes on at least one common channel – hence, inconsistent (the inconsistency can be seen due to the cycle in the graph).

(8) $L \cap H' \neq \phi \wedge D = D'$
- Inconsistency in this case follows from the previous case.

(9) $H \cap L' \neq \phi \wedge H' \cap L = \phi$
- From $H \cap L' \neq \phi$ it follows that the priority of D' is greater than or equal to that of the common channels of H and L'. By considering the second conjunct, we can conclude that D must have a higher priority than that of D'. In other words, there is a cycle and hence, inconsistent (see the priority graph shown in figure 2):

(10) $L \cap H' \neq \phi \wedge H \cap L'$
- The consistency can be ensured in the same way as the previous case.

In the above analysis, we have considered only the important situations; the other situations can be considered in a similar way. The exact way of keeping track of the information will be clear from the formal set of semantic equations. The equation for

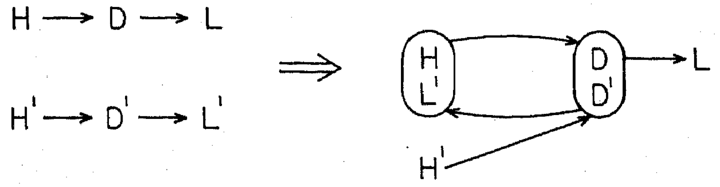

Figure 1. Priority graph for case (7).

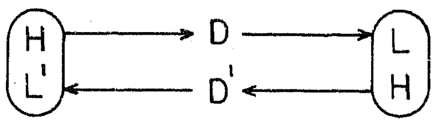

Figure 2. Priority graph for case (9).

parallel composition is given below:

$$M[\![S_1 \| S_2]\!]\sigma =$$
$$\text{PFC}(\{\langle\sigma_1 \times \sigma_2, h_1^1 \# h_2^1, join(h_1^2, h_2^2)\rangle \mid \forall i \in \{1,2\}: \langle\sigma_i, h_i\rangle \in M[\![S_i]\!]\sigma \wedge consistent(\sigma_1, h_1, \sigma_2, h_2)\}),$$

where

$$\sigma_1 \times \sigma_2(x) = \sigma_i(x) \textbf{ if } \sigma_1, \sigma_2 \neq \perp \wedge x \in var(\sigma_i)$$
$$= \sigma(x) \textbf{ if } \sigma_1, \sigma_2 \neq \perp \wedge x \neq var(\sigma_i) \wedge x \in var(\sigma)$$
$$= \perp \textbf{ otherwise.}$$

The pointwise merging of the histories $h_1 \# h_2$ and the predicate *consistent* are defined below: Let *cset* be the set of common channels in S_1 and S_2. Then, the predicate consistent is given by,

$$consistent(h_1, h_2, cset) \underline{\Delta}\ Comm(h_1, h_2, cset) \wedge NW(h_1, h_2, cset) \wedge Pri(h_1, h_2, cset)$$

That is, h_1 and h_2 are said to be consistent iff they are communication compatible (checked by predicate *Comm*), there is no unnecessary waiting (checked by predicate *NW*) and priority consistent (checked by predicate *Pri*). Note that the consistency is checked with reference to the joint channels. Each of these predicates is defined below:

- $Comm(h_1, h_2, cset) \underline{\Delta}$

  $$\forall 1 \leqslant j \leqslant max(|h_1|, |h_2|), v \in Val, D \in cset: \langle D, v\rangle \in h_1^1[j] \text{ iff } \langle D, v\rangle \in h_2^1[j].$$

  That is, for every communication on the joint channel, there is a reciprocal communication in the other process at that point of time with respect to some priority assumptions.
- $NW(h_1, h_2, cset) \underline{\Delta}\ \forall 1 \leqslant j \leqslant min(|h_1|, |h_2|)$: $R(A) \in h_1^1[j] \wedge R(B) \in h_2^1[j] \rightarrow A \cap B = \phi$. That is, processes are not unnecessarily waiting for each other.
- $Pri(h_1, h_2, cset) \underline{\Delta}$
  $\forall 1 \leqslant j \leqslant min(|h_1|, |h_2|), G_1 \in h_1^2[j] \wedge G_2 \in h_2^2[j]$:
  join(G_1, G_2) is an acyclic (precedence) graph.

Pointwise merging of histories: Let h_1 and h_2 be consistent with reference to the set of joint channels *cset*. Then, $h_1 \# h_2$ is defined by the *j*th pointwise union as follows:

$$h_1 \# h_2[j] = \langle A, G\rangle, \text{ where:}$$

(1) $A = \{\langle D, v\rangle \mid j \leqslant |h_1| \vee j \leqslant |h_2| \wedge \langle D, v\rangle \in h_1^1[j] \vee \langle D, v\rangle \in h_2^1[j]\}$.

- If the communication is over a channel not belonging to *cset* (the length of the two traces may not be equal), then the record is kept without any change; otherwise, both traces must contain the same communication record (D, v) at the time point j (as required by *Comm*).

  $$\cup \{R(\{D \mid j \leqslant |h_1| \vee j \leqslant |h_2| \wedge (R(D) \in h_1^1[j] \vee R(D) \in h_2^1[j])\})\}.$$

- Obtain the new ready set of channels by taking the union of all the channels over which the processes are waiting; note that there is no need to check whether the channel is in *cset* or not as the consistency test has already assured (by NW) that there is not unnecessary waiting; internalizing the channel in *cset* is handled by the hiding rule.

(2) $G = join(h_1^2, h_2^2)$.

Before defining the equation for hiding, we define *hide: graphs* $\times$ *channels* $\rightarrow \{\perp\} \cup$ *graphs*.

DEFINITION 3

Let G_1 and G_2 be any two priority graphs that are consistent (i.e., precedence) and *cset* be some non-empty finite set of channels. Then, we define,

$$join_{cset}(G_1, G_2) = hide(join(G_1, G_2), cset).$$

DEFINITION 4

Let G be some priority consistent graph and *cset* be some finite non-empty set of channels. Then,

(1) $hide(\perp, cset) = \perp$.
(2) $hide(G, \phi) = G$.
(3) $\forall D \in cset : hide(G, cset) = hide(G', cset - \{D\})$ where

$$G' = \{(a,b) \in G \mid D \text{ is not in } \{a,b\}\} \cup \{(a,b) \mid (a,D) \in G \wedge (D,b) \in G\}.$$

Hiding Let *cset* be the set of internal channels of S. Then,

$$M[\![[S]]\!]\sigma = \mathrm{PFC}(\{\langle \sigma, \langle A, G\rangle\rangle \mid \exists \langle A', G'\rangle : \langle \sigma, \langle A', G'\rangle\rangle \in M[\![S]\!]\sigma \wedge A = A' \uparrow cset \wedge G = hide(G', cset)\})$$

where $A' \uparrow cset$ is the history obtained after removing all the communications and readies on channels *cset* from A'. Note that the empty set is represented by $\square$ and hence the time points are preserved.

In the following, we sketch proofs of theorems for establishing the soundness (priority consistency) and the compositionality of the semantics. The following theorem establishes the priority consistency.

Theorem 1. *Consider an alternative command* $[\,[\!]_{i=1}^{n} g_i \rightarrow S_i]$. *Let the process be enabled on channels* $d_1, d_2, \ldots, d_m (m \leqslant n)$ *with increasing order of priority at time* t_1. *Then, communication started over* d_j *started at* $t_2 (t_2 \geqslant t_1)$ *implies that communication over* $d_{j+1}, \ldots, d_m$ *was impossible during* t_1 *to* t_2 *and communication over* d_j *was impossible during* t_1 *to* $t_2 - 1$.

Proof. The process was enabled on channels $d_1, d_2, \ldots, d_m (m \leqslant n)$ at time t_1 implies that the traces have the structure $\alpha R_{d_1, d_2, \ldots, d_m}^{t_2 - t_1 - 1} \beta$ where α and β are some traces upto t_1 and after $t_2 - 1$ respectively. It must be noted that β would have at least one non-wait action in its first component. Thus, by the predicates NW and Pri it follows that this was the earliest possible action. Note that the condition also holds when we consider hiding. Hence the theorem.

Theorem 2. *Parallel composition is associative.*

Proof. Associativity of the parallel composition ignoring the priority information follows on the lines of the proof given for CSP-R (Koymans *et al* 1988). What we need to show is that the composition of the priority information also satisfies the property of associativity. In other words, we have to prove that $join(join(G_1, G_2), G_3) = join(G_1, join(G_2, G_3))$. This follows from the fact that the union[2] of relations is associative.

Theorem 3. *The semantics is priority consistent and compositional.*

Proof. Proof follows from theorem 1 and theorem 2.

Illustrative example. In the following, we consider a simple example that illustrates the possibility of a deadlock in a network consisting of n processes, $P_1, P_2, \ldots, P_n$, for some given priority. For the sake of brevity, we ignore the values sent and received in the example. In the example, we consider only two processes P_1, and P_2 which have $\{\beta, \gamma\}$ as the common set of channels. The interesting feature of the example is that it depicts how the two processes get into deadlock for the given assignment of priorities irrespective of the behaviour of the other processes.

Example program

```
P1::= [α? by 1 →          P2::= [ω! by 1 →
       [] β? by 2 →              [] γ? by 2 →
       [] γ! by 3 →]             [] β! by 3 →]
```

$M[\![P1]\!]\sigma = \text{PFC}(\{\langle\sigma', H_1\}\})$ where H_1 consists of

$$\{\{R(\gamma,\beta,\alpha)\}^* \circ \langle\{R(\gamma,\beta), \langle\alpha,?\rangle\}, \langle\{\gamma,\beta\},\alpha,\phi\rangle\rangle,$$
$$\{R(\gamma,\beta,\alpha)\}^* \circ \langle\{R(\gamma), \langle\beta,?\rangle\}, G3\rangle,$$
$$\{R(\gamma,\beta,\alpha)\}^* \circ \langle\langle\beta,?\rangle, G4\rangle\},$$

where $G3 = \langle\{\gamma\}, \beta, \{\alpha\}\rangle$ and $G4 = \langle\phi, \gamma, \{\beta, \alpha\}\rangle$ and "?" denotes that the value part is ignored.

$M[\![P2]\!]\sigma = \text{PFC}(\{\langle\sigma', H_2\rangle\})$ where H_2 consists of,

$$\{\{R(\beta,\gamma,\omega)\}^* \circ \langle\{R(\beta,\gamma), \langle\omega,?\rangle\}, \langle\{\beta,\gamma\},\omega,\phi\rangle\rangle,$$
$$\{R(\beta,\gamma,\omega)\}^* \circ \langle\{R(\beta), \langle\gamma,?\rangle\}, G7\rangle,$$
$$\{R(\beta,\gamma,\omega)\}^* \circ \langle\langle\beta,?\rangle, G8\rangle\},$$

where $G7 = \langle\{\beta\}, \gamma, \{\omega\}\rangle$ and $G8 = \langle\phi, \beta, \{\gamma, \omega\}\rangle$.

The parallel composition is given by

$$M[\![P1 \parallel P2]\!]\sigma = \{\langle\bot, \lambda\rangle\}.$$

Explanation. For the sake of brevity, we consider only those histories that become inconsistent due to inconsistency of priority. The common set of channels of $P1$ and

[2]Note that the operation *join* is associative.

$P2$ is given by $cset = \{\beta, \gamma\}$. Most of the histories become inconsistent due to predicates *Comm* and *NW*. The following pairs of histories get eliminated due to inconsistent priorities, that is, both $join(G3, G8)$ and $join(G4, G7)$ are $\perp$. Let us take a closer look at how these two pairs of histories become inconsistent.

$$h_3 \equiv \langle\{R(\gamma), \langle\beta, ?\rangle\}, \langle\{\gamma\}, \beta, \{\alpha\}\rangle\rangle; h_8 \equiv \langle\langle\beta, ?\rangle, \langle\phi, \beta, \{\gamma, \omega\}\rangle\rangle.$$

Obviously, in $G3$, γ has priority over β, while in $G8$, β has priority over γ. Thus, there is an inconsistent partial ordering. Now, consider the pair of histories,

$$h_4 \equiv \langle\langle\beta, ?\rangle, \langle\phi, \gamma, \{\beta, \alpha\}\rangle\rangle; h_7 \equiv \langle\{R(\beta), \langle\gamma, ?\rangle\}, \langle\{\beta\}, \gamma, \{\omega\}\rangle\rangle.$$

Again here, γ has priority over β (in $G3$), while in $G7$, β has priority over γ -- that is, the ordering is inconsistent.

In other words, the two processes get deadlocked in the beginning itself despite the behaviour of other processes in the network. Thus, the two processes do not do anything.

Real-time execution models

In the previous sections, we have discussed semantic specification of priority using the maximal parallelism model. In this section, we briefly discuss the effect of various parameters including those that affect maximal parallelism.

- It easily follows that ignoring priority leads essentially to the same semantics as in Koymans *et al* (1988).
- In Koymans *et al* (1988) a spectrum of models ranging from interleaving to maximal parallelism has been given accounting for the communication media and a range in timings for actions. The semantics described here can also be augmented on the same lines to account for the various parameters. However, it may be noted that in the case of the interleaving model the semantics no longer ensures Lamport's requirement (cf. Lamport 1985) *that there should not be any unnecessary waiting in realistic priority specification*; this is in conformity with Lamport's conjecture.
- Though the principle of one processor to one logical process is quite feasible, there are many situations which force resource restrictions either due to logical design (for example, recursive processes) or due to physical constraints of space. The need of resource restrictions leads to scheduling requirements. Thus, the semantics should be able to handle interrupts of statements with higher priority. One posible solution is to combine the approaches of this paper with that of Hooman (1989).
- Another aspect that one comes across in realistic situations is that of assumptions about bus-arbitration or in general fairness issues. Perhaps one can handle some of these issues in a limited way similar to that of scheduling; a thorough investigation is needed to tackle the issue of fairness in the context of compositional semantics.

6. Discussion

In this paper, we have developed a compositional denotational semantics for prioritized real-time distributed programming languages. One of the interesting features is that it extends the compositional theory proposed in Koymans *et al* (1988) for prioritized real-time languages preserving the compositionality of the semantics.

As mentioned already, the language permits users to define situations in which an action has priority over another action without the requirement of preassigning priorities to actions for partially ordering the alphabet of actions. These features are part of the languages such as Ada designed specifically keeping the needs of real-time embedded systems.

Our approach does not have the restriction (as in Cleaveland & Hennessy 1990) that only prioritized internal moves can pre-empt unprioritized actions. Our notion of priority in the environment is based on the intuition that a low priority action can proceed only if the high priority action cannot proceed due to lack of the handshaking partner at that point of execution. In other words, if some action is possible corresponding to that environment at some point of execution then the action takes place without unnecessary waiting. It must be clear that such an approach clearly satisfies Lamport's requirement (Lamport 1985) that realistic priority specification should not involve unnecessary waiting. The condition of "no unnecessary waiting" itself provides a sort of priority for unprioritized actions[3] and thus, provides a natural model for the tasking features of Ada. In the semantic theory we have proposed there is a clear distinction between the semantic model and the execution model – this has enabled us to fully ensure that there is no unnecessary waiting. It is of interest to note that the real-time semantics proposed satisfies Lamport's conditions in a natural way. Also, our work is the first formal semantics for treating priority that is state-based – thus, having advantages over algebraic approaches for reasoning about reactive systems.

We believe that the proposed compositional theory provides a sound basis for the languages for programming reactive systems (see Liu & Shyamasundar 1989). It may be observed that we have developed the semantics by considering the priorities of events in each process in an isolated manner. Thus, it is only the partial ordering of the events that is important rather than the priority number associated with the events. We have used prefix-closed sets as our semantic domain in order to treat any general reactive system.

In our semantics, we have given priority for pure boolean guards so as to be consistent with the semantics defined in Koymans *et al* (1988). This restriction can be removed very easily by appropriately changing the a priori semantics of the alternative command. The theory can also be extended to include priorities of types (1) and (2) discussed earlier. Furthermore, the semantics can be tailored to terminating systems by considering complete traces instead of prefix-closed sets. The semantic equations can be easily extended for this case (the main difference will be that we would be using greatest fix point rather than the least fix point for iteration). It may be noted that from the real-time semantics, one can obtain a temporal logic proof system on the lines of Hooman & Widom (1988). A static analysis of CSP-R_p programs on the lines of the characterization in Liu & Shyamasundar (1988, pp. 134–138, 1990) can be used for deriving tools for the specification and verification of prioritized finite state systems. Currently, we are investigating the applicability of the theory to the verification of communication protocols including complex protocols such as carrier sense protocols discussed in Pnueli & Harel (1988, pp. 84–98).

[3]This should not be misunderstood as the priority model proposed; this only shows that the MAXPAR execution model has some additional advantages in the context of priority.

Appendix A

Consider the problem of minimizing the head movement in disc access which requires a different scheduling rather than the usual FIFO discipline of Ada. We briefly discuss the solution described in Gehani (1983) using the strategy of families of entries. Let us assume that the requests for service are classified into three categories declared as

type REQUEST-LEVEL **is** (URGENT, NORMAL, LOW).

Urgent requests are accepted before any other kind of requests. Normal requests are accepted only if there are no urgent requests pending. Finally requests in the low category are accepted only if there are no urgent or normal priority requests pending. Within each category requests are accepted in FIFO order.

This scheme is implemented by a task SERVICE that contains the declarations of an entry family REQUEST:

```
task SERVICE is
  entry REQUEST (REQUEST-LEVEL) (D: in out DATA);
end SERVICE;
```

Each member of REQUEST handles one request category. The body of task SERVICE is given below:

```
task SERVICE is
begin
loop
  select
       accept REQUEST (URGENT) (D: in out DATA) do
         ...processtherequest
       end REQUEST;
       or when REQUEST (URGENT)'COUNT = 0 =>
       – the number of tasks waiting at an entry is
       – given by the COUNT attribute
       accept REQUEST (NORMAL) (D: in out DATA) do
         ...process the request
       end REQUEST;
       or when REQUEST (URGENT)'COUNT = 0 ∧
                REQUEST (NORMAL)'COUNT = 0 =>
       – the number of tasks waiting at an entry is
       – given by the COUNT attribute
       accept REQUEST (LOW) (D: in out DATA) do
         ...process the request
       end REQUEST;
       :
  end select;
end loop;
end SERVICE
```

Appendix B

Before describing the semantic equations, we will define the auxiliary functions required.

Let ϕ be a function from S to D_{dom}. Then ϕ' is the function from $\bar{S}$ to D_{dom} defined by $\phi'(\sigma) = \mathbf{if}\, \sigma \in S$ then $\phi(\sigma)$ **else** $\mathrm{PFC}(\{\langle\sigma, \lambda\rangle\})$.

Further, ϕ^* is the function from D_{dom} to D_{dom} defined by,

$$\phi^*(X) = \{\langle\sigma', h \circ h'\rangle \mid \langle\sigma, h\rangle \in X \text{ and } \langle\sigma', h'\rangle \in \phi'(\sigma)\}.$$

The function G is defined using the following two auxiliary functions.

$$\Gamma(\{b_1; \bar{g}_1, \ldots, b_n; \bar{g}_n\}, \sigma) = \{R(D) \mid \exists i\colon W[\![b_i]\!]\sigma = tt \wedge (\bar{g}_i = D!e \vee \bar{g}_i = D?x)\},$$

$$waitvalue(b; \bar{g}, \sigma) = \begin{cases} 0, & \mathbf{if}\ \bar{g} = \lambda \wedge W[\![b]\!]\sigma = tt, \\ max(n, 1), & \mathbf{if}\ \bar{g} = wait\ n \wedge W[\![b]\!]\sigma = tt, \\ \infty, & \mathbf{otherwise}. \end{cases}$$

A priori semantics

$M[\![S]\!]\bot = \{\langle\bot, \lambda\rangle\}$ for any S; λ is the empty sequence.

$M[\![skip]\!]\sigma = \mathrm{PFC}(\{\langle\sigma, \{\Box\}\rangle\})$.

$M[\![x := e]\!]\sigma = \mathrm{PFC}(\{\langle\sigma[\mathcal{V}[\![e]\!]\sigma/x], \{\Box\}\rangle\})$

- where $\mathcal{V}$ is the semantic function (assumed to be given) for evaluating the arithmetic expressions.

I/O statements

$$M[\![D?x]\!]\sigma = \mathrm{PFC}(\{\langle\sigma[v/x], \langle R(D)^t \circ \langle D, v\rangle\rangle\rangle \mid v \in Val, t \geqslant 0\}).$$

The semantics corresponds to indefinite waiting till the communication succeeds.

$$M[\![D!e]\!]\sigma = \mathrm{PFC}(\{\langle\sigma, \langle R(D)^t \circ \langle D, \mathcal{V}[\![e]\!]\sigma\rangle\rangle\rangle \mid t \geqslant 0\}).$$

The semantics of guards is defined in terms of an environment of boolean, I/O and wait guards. We do not give the semantics of the wait statement as it follows from the semantics of the wait guard (assuming empty environment).

$M[\![g]\!]\sigma = G[\![g, \phi]\!]\sigma$, where g is an I/O command,

$M[\![S_1; S_2]\!]\sigma = M^*[\![S_2]\!](M[\![S_1]\!]\sigma)$,

$G[\![b, A]\!]\sigma = \mathbf{if}\ W[\![b]\!]\sigma\ \mathbf{then}\ \mathrm{PFC}\{\langle\sigma, \lambda\rangle\}\ \mathbf{else}\ \{\langle\bot, \lambda\rangle\}\ \mathbf{fi}$,

- where W is the semantic function (assumed to be given) for evaluating the boolean expressions.

$G[\![wait\ d, A]\!]\sigma =$

$$\mathrm{PFC}(\{\langle\sigma, \Gamma(A, \sigma)^t\rangle \mid max\{\mathcal{V}[\![d]\!]\sigma, 1\} = minwait(A \cup \{wait\, d\}, \sigma) \Delta\ t\}),$$

- where *minwait* (A, σ) gives the minimum of the waiting periods corresponding to elements of A.

$G[\![D?x, A]\!]\sigma =$

$$\mathrm{PFC}(\{\langle\sigma[v/x], \Gamma(GRDS, \sigma)^t \circ \langle\{R(HI(D, A)), \langle D, v\rangle\}, BPG(D, GRDS)\rangle\rangle \mid v \in V, 0 \leqslant t < minwait(A, \sigma)\}),$$

- where $GRDS = A \cup \{D?x \textbf{ by p}\}$, $HI(D,A)$ returns the set of channels in A which have higher priority than channel D, $EL(D, GRDS)$ returns the set of channels in $GRDS$ which have equal or lower priority than channel D, and

$BPG(D; GRDS) = \langle HI(D, GRDS), D, EL(D, GRDS)\rangle$.

$$G[\![D!e, A]\!]\sigma =$$

$$\text{PFC}(\{\langle\sigma, \Gamma(GRDS,\sigma)^t \circ \langle\{R(HI(D,A)), \langle D, \mathcal{V}[\![e]\!]\sigma\rangle\}, BPG(D, GRDS)\rangle\rangle$$
$$| 0 \leqslant t < minwait(A,\sigma)\}),$$

- where $GRDS = A \cup \{D!e \textbf{ by p}\}$. The other interpretations remain the same as in the case of input guard.

$$G[\![b; g, A]\!]\sigma = G[\![g, A]\!]^*(G[\![b, A]\!]\sigma), \text{ where } g \equiv D?x \textbf{ by p} \textit{ or } D!e \textbf{ by p} \textit{ or wait } d.$$

$$M[\![[\,\|_{j=1}^{n} g_j \rightarrow S_j]\,]\!]\sigma =$$
$$\textbf{if } \vee_{j=1}^{n} W[\![g_j^b]\!]\sigma (\text{where } g_j^b \text{ is the boolean part of } g_j) \textbf{ then}$$
$$\cup_{j=1}^{n} M[\![S_j]\!]^*(G[\![g_j, \{g_k | 1 \leqslant k \leqslant n, k \neq j\}]\!]\sigma)$$
$$\textbf{else } \text{PFC}(\{\langle\bot, \lambda\rangle\}) \textbf{ fi}$$

$$M[\![*A]\!]\sigma = \mu\phi.\lambda\sigma.\textbf{if } \exists i\text{: } W[\![b_i]\!]\sigma) = tt \textbf{ then } \phi^*(M[\![A]\!]\sigma) \textbf{ else } \text{PFC}(\{\langle\sigma,\lambda\rangle\})\textbf{fi}.$$

Partial support by the Indo-French Centre for the Promotion of Advanced Research/ Centre Franco-Indien Pour la Promotion de la Recherche Avancee as part of the project "Formal Specification and Development of Real-Time Reactive Programs" is gratefully acknowledged.

References

Baeten J C M, Bergstra J A, Klop J W 1985 Syntax and defining equations for an interrupt mechanism in process algebra, Report CS-R8503, Center for Mathematics and Computer Science, Amsterdam

Cleaveland R, Hennessy M 1988 Priorities in process algebras. *Inf. Comput.* 87: 58–77

Gehani N 1983 *Ada: An advanced introduction including reference manual for the Ada Programming Language* (Englewood Cliffs, NJ: Prentice Hall)

Hoare C A R 1978 Communicating sequential processes. *Commun. ACM* 21: 666–677

Hooman J 1989 A real-time semantics for multiprogramming, manuscript, REX-Concurrency Day

Hooman J, Widom J 1988 A temporal-logic based compositional proof system for real-time message passing, TR 88–919, Cornell University

Huizing C, Gerth R, de Roever W P 1987 Full abstraction of a real-time denotational semantics for an Occam-like language. ACM *Symposium on Principles of Programming Language*, (New York: ACM Press)

Koymans R, Shyamasundar R K, Gerth R, de Roever W P, Arun Kumar S 1985 Compositional semantics for real-time distributed computing. *Proc. Logics of Programs. Lecture Notes in Computer Science, vol. 197* (Berlin: Springer-Verlag)

Koymans R, Shyamasundar R K, Gerth R, de Roever W P, Arun Kumar S 1988 Compositional semantics for real-time distributed computing. *Inf. Comput.* 79: 210–256

Lamport L 1985 What it means for a concurrent program to satisfy a specification: Why no one has specified priority. *ACM Symposium on Principles of Programming Languages* (New York: ACM Press)

Liu L Y, Shyamasundar R K 1988 Static analysis of real-time distributed systems. *Proc. Formal Techniques in Real-Time and Fault-Tolerant Systems. Lectures Notes in Computer Science, vol. 331* (Berlin: Springer-Verlag) pp. 373–388

Liu L Y, Shyamasundar R K 1989 An operational semantics of real-time design language RT-CDL. *Fifth International Workshop on Software Specification and Design* (New York: IEEE Press)

Liu L Y, Shyamasundar R K 1990 Static analysis of real-time distributed systems. *IEEE Trans. Software Eng.* SE-16: 373–388

Milner R 1980 *A calculus of communicating systems. Lecture Notes in Computer Science, vol. 92* (Berlin: Springer-Verlag)

Occam 1984 *Occam Programming Manual*, Inmos Limited (London: Prentice-Hall International)

Pitassi T, Narayana K T, Shyamasundar R K 1986 A compositional semantics for Occam, TR CS-86-19, Computer Science Department, Pennsylvania State University, Pa 16802

Pnueli A, Harel E 1988 Applications of temporal logic to the specification of real-time systems. *Proc. Formal Techniques in Real-Time and Fault-Tolerant Systems. Lecture Notes in Computer Science, vol. 331* (Berlin: Springer-Verlag) pp. 84–98

Roscoe A W 1984 *Denotational semantics for Occam. Lecture Notes in Computer Science, vol. 197* (Berlin: Springer-Verlag)

Stankovic A 1988 Real-time computing systems: The next generations, COINS TR, University of Massachusetts