A Formal Definition of Priority in CSP

C. J. FIDGE
The University of Queensland

The process models of Ada and occam™ are formally based on the CSP process algebra. However, for fine-tuning real-time performance, they include 'prioritized' constructs that have no counterparts in CSP. These constructs therefore lack any formal definition, a situation that leaves room for misunderstandings. We extend CSP with a formal definition of the notion of priority. The definition is then used to assess the transputer implementation of priority in occam and the definition of priority in Ada.

Categories and Subject Descriptors: D.3.1 [**Programming Languages**]: Formal Definitions and Theory—*semantics*; D.3.3 [**Programming Languages**]: Language Constructs and Features—*concurrent programming structures*; F.3.2 [**Logics and Meanings of Programs**]: Semantics of Programming Languages—*operational semantics*

General Terms: Languages, Theory

Additional Key Words and Phrases: Ada, occam, Communicating Sequential Processes, Priority, real-time programming

1. INTRODUCTION

Ada owes its interprocess communication construct, the rendezvous, to Hoare's Communicating Sequential Process specification technique. The occam programming language is even more closely based on CSP.

However, in recognition of the pragmatic needs of real-time programming, Ada and occam also include "prioritized" (or "asymmetric") features lacking in CSP. These help the programmer fine-tune performance once the logical program structure is complete. Unfortunately, because these operators have no counterparts in CSP, their meaning is open to misinterpretation, particularly in uncommon examples.

Herein we give a formal definition of asymmetric operators in CSP. It is then shown how this formal definition assists our understanding of certain code segments and helps us to assess the correctness of priority as defined for

™ 'occam' is a trademark of the INMOS Group of Companies.

This research was supported by an Australian Postdoctoral Research Fellowship. An earlier, shorter version of this paper was presented at the Fifteenth Australian Computer Science Conference, Hobart, Tasmania, January 1992.
Author's address: Key Centre for Software Technology, Department of Computer Science, The University of Queensland, Queensland 4072, Australia.

both Ada and occam. In the interests of brevity some familiarity with Ada [17], occam [15], and the 1985 definition of CSP [14] is assumed. A glossary of special symbols is included.

2. MOTIVATION

2.1 Priority in occam

occam allows both its parallelism, `PAR`, and alternative, `ALT`, statements to be preceded by the keyword `PRI` to express the intention that preference is to be given to earlier operands over later ones. The concept is intuitively simple, but it has subtle consequences for program equivalence. Although priority is only an optimization and "does not affect the logical behavior of a program" [15], a real-time programmer would be unwise to assume that `PAR` is equivalent to `PRI PAR`. As noted by Jones and Goldsmith [19], consideration of performance destroys the illusion of equality between "logically" equivalent programs. Unfortunately a formal definition for equivalence of prioritized code segments is lacking.

The occam language definition allows a degree of freedom in the implementation of `PRI`. For instance, an implementation of occam in which *all* parallel and alternative constructs are prioritized is consistent with the description in the language definition [19]. Without a formal definition how are we to assess a particular implementation? In particular we are interested in occam on its native hardware platform, the transputer.

The examples in Figure 1 illustrate some of our concerns. Often we can rely on our intuitive notion of priority to understand the meaning of prioritized statements. For instance, example `1` clearly expresses the intention that processes Q and R both execute at the same priority level, below that of P, but above that of S. However, what is the relationship between P and Q in example `2`, or Q and R in example `3`? Can we tell in advance whether communication will take place on channel `c` or `d` in examples `4` and `5`?

2.2 Priority in Ada

Ada allows priority to be expressed by including the pragma `PRIORITY` in the specification of a task. This statically associates a priority value with instances of this task definition—higher values indicate greater urgency. When scheduling decisions must be made at run-time, preference is given to eligible tasks with higher priority over those with lower values. Because the Ada "extended" rendezvous may involve the execution of a significant amount of code, a simple form of dynamic priority inheritance is allowed—when two tasks are engaged in a rendezvous the rendezvous code is executed at the higher of the two priorities (assuming both tasks have priority pragmas).

Like occam, the *Ada Language Reference Manual* allows the implementation of priority a certain amount of flexibility. Scheduling of two eligible tasks with the same priority is implementation specific. Also, a particular Ada

```
-- example 1
PRI PAR
  P
  PAR
    Q
    R
  S
```

```
-- example 2
PAR
  P
  PRI PAR
    Q
    R
```

```
-- example 3
PRI PAR
  PRI PAR
    P
    Q
  R
```

```
-- example 4
PAR
  PRI PAR
    SEQ
      c ! x
      P
    SEQ
      d ! y
      Q
  PRI ALT
    d ? v
      R
    c ? u
      S
```

```
-- example 5
PRI PAR
  PAR
    SEQ
      c ! x
      P
    SEQ
      d ! y
      Q
  PRI ALT
    d ? v
      R
    c ? u
      S
```

Fig. 1. Examples of prioritization in occam.

compiler need not support more than one level of priority, thus making priority pragmas meaningless on some systems.

Figure 2 shows some code fragments illustrating Ada priority usage. Example 6 gives the specification of three tasks, two of which are explicitly prioritized. Real-time programming normally demands predictable, deterministic behavior—can we tell in advance how actions performed by these three tasks will be scheduled? In example 7 a rendezvous may take place on entry C or D, but the programmer has attempted to give the call to C higher priority. Does the definition of priority in Ada respect the programmer's intention? Finally, example 8 shows a server, A, attempting to rendezvous with either of two clients, C and D, but can we be certain that the rendezvous will take place with the client with highest priority?

3. DEFINITION OF PRIORITY IN CSP

Our intention here, by formally defining priority, is to develop a sound foundation for addressing uncertainties such as those described above. To achieve this we introduce *prioritized* operators to CSP that favor their

```
-- example 6
task P;

task Q is
  pragma PRIORITY(5);
end Q;

task R is
  pragma PRIORITY(1);
end R;
```

```
-- example 7
task A is
  pragma PRIORITY(1);
  entry c;
  entry d;
end A;

task C is
  pragma PRIORITY(10);
end C;

task D is
  pragma PRIORITY(5);
end D;

task body A is
begin
  select
    accept c;
  or
    accept d;
  end select;
end A;

task body C is
begin
  A.c;
end C;

task body D is
begin
  A.d;
end D;
```

```
-- example 8
task A is
  entry b;
  entry c;
  entry d;
end A;

task C is
  pragma PRIORITY(10);
end C;

task D is
  pragma PRIORITY(5);
end D;

task body A is
begin
  accept b;
  select
    accept c;
  or
    accept d;
  end select;
end A;

task body C is
begin
  A.b;
  A.c;
end C;

task body D is
begin
  A.b;
  A.d;
end D;
```

Fig. 2. Examples of prioritization in Ada.

left-hand operands. These are asymmetric choice, denoted $\vec{\Box}$, asymmetric concurrency, denoted $\vec{\|}$, and asymmetric interleaving,[1] denoted with $\vec{|||}$.

3.1 Traces

Traces are central to the definition of CSP. The following laws define the traces produced by each of the asymmetric operators. The traces for all other CSP operators are unchanged.

Asymmetric concurrency. The simplest definition is that for asymmetric concurrency—its traces are identical to those of conventional concurrency [14]:

$$traces\left(P \vec{\|} Q\right) = traces(P \| Q) \qquad \text{T1}$$

To understand this surprising result consider the following example. Let $\alpha P = \{a, c\}$ and $\alpha Q = \{b, c\}$. Then,

$$traces\left(a \rightarrow P \vec{\|} b \rightarrow Q\right) = \{\langle\rangle, \langle a\rangle, \langle b\rangle, \ldots\}.$$

(Asymmetric operators are assumed to have the same precedence as their symmetric counterparts.) In an environment prepared to engage in either a or b, we naturally expect a to occur due to the asymmetry. However, in an environment prepared to perform only b the first process is blocked and the second proceeds. Thus traces beginning with both a or b are possible, exactly as for the symmetric concurrency operator. (Events in the alphabet of both processes must be performed by both at the same time, again just as for the standard concurrency operator.) Thus, *in an appropriate environment*, asymmetric concurrency is capable of performing any trace possible for conventional concurrency.

Asymmetric choice. Whereas traces of the conventional choice operator are simply the union of those defined by each of its operands [14], the asymmetric choice operator must exclude any traces in which both the high and low priority operands can perform the same event and the low priority one is selected. Thus,

$$\begin{aligned} traces\left(P \vec{\Box} Q\right) = {} & traces(P) \cup (traces(Q) \\ & - \{t \mid t \in traces(Q) \wedge \exists u : traces(P) \cdot \\ & \quad t_0 = u_0 \wedge t_0 \notin \cup\, refusals(P)\}). \end{aligned} \qquad \text{T2}$$

For example,

$$traces\left(a \rightarrow P \vec{\Box} b \rightarrow Q\right) = \{\langle\rangle, \langle a\rangle, \langle b\rangle, \ldots\}.$$

[1] Hoare's choice of the word "interleaving" to describe the '|||' operator in CSP is unfortunate. In fact this operator models the special case of concurrent composition in which the operands do not interact. The events are "interleaved" only due to the interleaving operational semantics normally used to define CSP.

Even though the first operand has priority when both a and b are possible, in an environment prepared to perform only event b, the second operand will still be selected. Either operand is thus possible in an appropriate environment.

However consider the following case,

$$traces\big(c \to a \to P \,\vec{\Box}\, c \to b \to Q\big) = \{\langle\rangle, \langle c\rangle, \langle c, a\rangle, \ldots\}.$$

The trace $\langle c, b\rangle$ is *not* included above. Where both operands are prepared to perform the same event, the first is always chosen.

The need for condition '$t_0 \notin \bigcup refusals(P)$' in T2 is rather subtle. Remember that $refusals(P)$ is the set of sets of events that P is capable of at its first step but may arbitrarily refuse to perform due to pure nondeterminism [14]. This clause thus asserts that the first event of trace t must not be one that P may arbitrarily refuse. Consider the following example,

$$\begin{aligned} &traces\big((c \to a \to P \sqcap d \to Q)\,\vec{\Box}\,(c \to b \to R)\big) \\ &\quad = \{\langle\rangle, \langle c\rangle, \langle c, a\rangle, \langle c, b\rangle, \langle d\rangle, \ldots\}. \end{aligned}$$

Imagine that the environment offers event c but the nondeterministic choice randomly decides to perform event d. In this case the low priority process will proceed even though it offers an initial event that *may* have been performed by the high priority process.

Asymmetric interleaving. The traces of asymmetric interleaving are identical to those of conventional interleaving [14] except that any trace in which both operands could have performed the same initial event and the second was chosen must be excluded.

$$\begin{aligned} &traces\big(P \,\vec{|||}\, Q\big) \\ &\quad = \{\langle\rangle\} \cup \Big\{\langle c\rangle \hat{}\, t \,\Big|\, \big(\langle c\rangle \in traces(P) \wedge t \in traces\big(P/\langle c\rangle \,\vec{|||}\, Q\big)\big) \\ &\qquad\qquad \vee \big(\langle c\rangle \in traces(Q) \wedge t \in traces\big(P \,\vec{|||}\, Q/\langle c\rangle\big) \\ &\qquad\qquad\quad \wedge (\langle c\rangle \notin traces(P) \vee c \in \bigcup refusals(P))\big)\Big\}. \end{aligned} \tag{T3}$$

In other words, the traces of asymmetric interleaving include interleavings beginning with an initial event of P, or an initial event of Q as long as P either cannot perform, or may refuse to perform, the same event. For example,

$$traces\big(a \to P \,\vec{|||}\, b \to Q\big) = \{\langle\rangle, \langle a\rangle, \langle b\rangle, \ldots\}$$

since either choice is possible in an appropriate environment. By contrast,

$$\begin{aligned} &traces\big(c \to a \to P \,\vec{|||}\, c \to b \to Q\big) \\ &\quad = \{\langle\rangle, \langle c\rangle, \langle c, a\rangle, \langle c, c\rangle, \langle c, c, a\rangle, \langle c, c, b\rangle, \langle c, c, a, b\rangle, \langle c, c, b, a\rangle, \ldots\}. \end{aligned}$$

The trace $\langle c, b \rangle$ is not included because the first operand will always be chosen over the second when c is offered.

Like asymmetric choice, the definition of asymmetric interleaving allows for the possibility that the higher priority operand may arbitrarily refuse to perform an event also offered by the low priority one. Thus, $\langle c, b \rangle$ does appear below,

$$\begin{aligned} &traces\big((c \rightarrow a \rightarrow P \sqcap d \rightarrow Q) \vec{|||} (c \rightarrow b \rightarrow R)\big) \\ &\quad = \{\langle\rangle, \langle c\rangle, \langle d\rangle, \langle c, a\rangle, \langle c, b\rangle, \langle c, c\rangle, \langle c, d\rangle, \langle d, c\rangle, \langle c, a, c\rangle, \\ &\quad\quad \langle c, b, d\rangle, \langle c, c, a\rangle, \langle c, c, b\rangle, \langle c, d, b\rangle, \langle d, c, b\rangle, \\ &\quad\quad \langle c, a, c, b\rangle, \langle c, c, a, b\rangle, \langle c, c, b, a\rangle, \ldots\}. \end{aligned}$$

3.2 Preferences Function

Rule T1 illustrates an important point: the set of traces allowed by asymmetric operators is sometimes insufficient to characterize the notion of priority. This is the same dilemma faced by Hoare when attempting to distinguish between pure nondeterminism and general choice [14], and we adopt a similar solution.

Hoare defines a function on processes, *refusals*, that returns the set of sets of events that a nondeterministic process may refuse at its first step—this function distinguishes pure nondeterminism and general choice even where their traces do not. In this section we define a function on processes, *preferences*, that returns a set of relations on the events that the argument process may perform *at its first step*. This function will distinguish between, for instance, symmetric and asymmetric concurrency, even though their traces are identical.

We use the notation for relations from the specification language Z [10]. Each relation on a set of events $\{a, b, c, \ldots\}$ is represented by a set of ordered pairs $\{a \mapsto b, d \mapsto c, \ldots\}$ and can thus be manipulated via conventional set operators.

The intended interpretation of the set of relations returned by $preferences(P)$ is that, depending upon purely nondeterministic choices made by P, the priorities among those events that can occur at the first step are expressed by one of the relations in the set. Each relation is a reflexive partial ordering on those events that P may perform on its first step. If P and its environment are both prepared to perform two events a and b, $a \mapsto b$ is in the relation and $b \mapsto a$ is not, then a must be performed in preference to b to respect the priorities. (But note that even when an event has low priority, it may still be performed next if the environment precludes selection of all other events with higher priority.) Otherwise either event is allowed.

The following rules formally define the *preferences* function for the basic CSP operators. Examples can be found in Sections 3.3 and 4.

The *STOP* process can never perform any event, and therefore expresses no preferences:

$$preferences(STOP) = \{\{\}\} \qquad \text{P1}$$

The generalized prefixing definition allows one of a number of distinct events to be performed, but expresses no particular preferences among them:

$$preferences(x{:}A \rightarrow P(x)) = \{\{a \mapsto a | a \in A\}\}, \quad \text{where } A \subseteq \alpha P \qquad \text{P2}$$

Pure nondeterminism makes it impossible to predict in advance which events are possible at the first step. Therefore more than one relation may be returned:

$$preferences(P \sqcap Q) = preferences(P) \cup preferences(Q) \qquad \text{P3}$$

Depending on the random choice the process will offer either the preferences of process P or Q.

The preferences of the general choice operator are similar except that the events offered by each operand are simultaneously available:

$$preferences(P \,\Box\, Q) = \{p \cup q | p \in preferences(P) \wedge q \in preferences(Q)\} \qquad \text{P4}$$

Since *preferences* only considers the events that may occur at the first step it does not distinguish between the alternatives offered by choice or interleaving:

$$preferences(P \,|||\, Q) = preferences(P \,\Box\, Q) \qquad \text{P5}$$

Keep in mind, however, that the *traces* of general choice and interleaving still distinguish them.

We define an operator, denoted $\ominus$, for removing all mention of a set of events from a relation as follows. Given a relation R over a set of events E, then for some $S \subseteq E$,

$$R \ominus S = \{a \mapsto b | a \notin S \wedge b \notin S \wedge aRb\}$$

The preferences expressed by concurrency are similar to those of general choice except that care must be taken to exclude any events that are in the alphabets of both processes, but not simultaneously offered by both:

$$\begin{aligned} preferences(P \| Q) = \{&(p \cup q) \ominus (\alpha P \cap \alpha Q - \mathrm{dom}(p) \cap \mathrm{dom}(q)) | \\ &p \in preferences(P) \wedge q \in preferences(Q)\} \end{aligned} \qquad \text{P6}$$

The expression '$\alpha P \cap \alpha Q$' yields all events requiring simultaneous participation of both operands. Expression '$\mathrm{dom}(p) \cap \mathrm{dom}(q)$' gives those events that both are capable of performing at the next step. (Since the relation returned by *preferences* is reflexive its domain contains all possible next events, even those that are not preferred.) The difference of these sets thus represents those events that both processes must do simultaneously but both are not prepared to perform at the next step. These events are therefore not possible and are removed via $\ominus$. (The set of *traces* will never contain a trace starting with any of these 'impossible' events.)

We define *high* as a function which, given some relation R, returns the set of all events appearing in R with 'highest priority,' i.e., those that are not 'preceded' by any other event,

$$high(R) = \{a | a \in \mathrm{dom}(R) \wedge \forall b{:}\mathrm{dom}(R) \cdot bRa \Rightarrow aRb\}.$$

The last conventional CSP operator that we consider here is concealment:

$$\begin{aligned} &preferences(P \setminus C) \\ &\quad = \{p \ominus C \mid p \in preferences(P) \wedge ((high(p) - C) \neq \{\} \vee p = \{\})\} \quad \text{P7} \\ &\qquad \cup \{p \mid p \in preferences((P/\langle c \rangle) \setminus C) \wedge c \in C \wedge \langle c \rangle \in traces(P)\} \end{aligned}$$

The first of these two sets captures all *preferences* of P with any reference to the hidden events removed. It covers those cases where a visible event is performed ($(high(p) - C) \neq \{\}$ if p includes at least one 'preferred' visible event) or no action is possible ($p = \{\}$ if P behaves like $STOP$). The second set handles the cases where a hidden event occurs. The preferences in this case are those of P after it has executed the hidden event. Definition P7, like P3, may increase the number of relations in the set. This is to be expected because concealment introduces nondeterminism.

We now turn to the asymmetric operators. Firstly, we define a 'biased' union of two relations, denoted $\oplus$. Given two relations R and T, it returns a relation consisting of their union less any pairs from T that contradict preferences expressed by R:

$$R \oplus T = R \cup (T - R^{-1})$$

Thus any preferences expressed by the first relation take precedence over those of the second.

Further, given two relations R and T, let the $\oslash$ operator return the relation formed by the cartesian product of all events in R and those events appearing in T but not R:

$$R \oslash T = \mathrm{dom}(R) \times (\mathrm{dom}(T) - \mathrm{dom}(R)).$$

Thus all events in R will have preference over any other events found in T.

In defining the preferences of asymmetric choice, we must capture the notion that any events possible for the first operand have preference over those for the second. Less obvious is that any preferences expressed by the first operand cannot be contradicted by the preferences of the second.

$$\begin{aligned} &preferences\left(P \mathbin{\vec{\Box}} Q\right) \\ &\quad = \{(p \oplus (p \oslash q)) \oplus q \mid p \in preferences(P) \wedge q \in preferences(Q)\}. \quad \text{P8} \end{aligned}$$

There are three components. The preferences of P are given the highest importance. Next, any events possible for P have preference over any *other* events possible for Q. Least important are the preferences expressed by Q. For instance, if P states that event a is preferred over event b, and Q states that b has priority over a, the view of P will prevail due to its higher status. (For conventional choice such contradictory preferences are both allowed and cancel one another.)

The preferences of asymmetric concurrency are almost identical to those of asymmetric choice except that, like conventional concurrency, care must be taken to exclude events in both alphabets that cannot be performed by both

processes on their first step:

$$\begin{aligned} &preferences\left(P \vec{|||} Q\right) \\ &\quad = \{((p \oplus (p \oslash q)) \oplus q) \ominus (\alpha P \cap \alpha Q - \mathrm{dom}(p) \cap \mathrm{dom}(q)) \mid \\ &\qquad p \in preferences(P) \wedge q \in preferences(Q)\} \end{aligned} \tag{P9}$$

Finally, the preferences of asymmetric interleaving are identical to those of asymmetric choice (see P5):

$$preferences\left(P \vec{|||} Q\right) = preferences\left(P \vec{\Box} Q\right) \tag{P10}$$

3.3 Laws

For completeness this section presents prominent equivalence laws for the asymmetric operators in the style of Hoare [14]. Readers interested in applications of the above definitions may prefer to proceed directly to Section 4.

It is assumed that the concept of process equality is extended to require equality of *preferences*, as well as *traces*, *refusals*, etc. The conventional laws [14] are unaffected by this—any equivalent nonprioritized processes have the same set of initial events, and define no particular preferences. The *refusals* sets for each of the asymmetric operators are the same as those of their symmetric counterparts.

Asymmetric choice. The asymmetric choice operator is idempotent,

$$P \vec{\Box} P = P \tag{L1}$$

but, unlike general choice, asymmetric(!). For instance,

$$preferences\left(a \to P \vec{\Box} b \to Q\right) = \{\{a \mapsto a, b \mapsto b, a \mapsto b\}\}$$

but

$$preferences\left(b \to Q \vec{\Box} a \to P\right) = \{\{a \mapsto a, b \mapsto b, b \mapsto a\}\}$$

although their *traces* are identical.

Asymmetric choice is associative,

$$P \vec{\Box} \left(Q \vec{\Box} R\right) = \left(P \vec{\Box} Q\right) \vec{\Box} R. \tag{L2}$$

For instance,

$$\begin{aligned} &preferences\left(a \to P \vec{\Box} \left(b \to Q \vec{\Box} c \to R\right)\right) \\ &\quad = preferences\left(\left(a \to P \vec{\Box} b \to Q\right) \vec{\Box} c \to R\right) \\ &\quad = \{\{a \mapsto a, b \mapsto b, c \mapsto c, a \mapsto b, b \mapsto c, a \mapsto c\}\} \end{aligned}$$

In both cases there is a total ordering among the three initial events—a is preferred to b and b is preferred to c.

Choosing between a process and inaction has no effect,

$$P \vec{\Box} STOP = STOP \vec{\Box} P = P \qquad \text{L3}$$

It is interesting to compare the following law, which formalizes the definition of asymmetric choice, with the corresponding law for general choice [14]:

$$\begin{aligned}&(x{:}A \to P(x)) \vec{\Box} (y{:}B \to Q(y)) \\ &\quad = (z{:}(A \cup B) \to (\textbf{if } z \in A \textbf{ then } P(z) \textbf{ else } Q(z)))\end{aligned} \qquad \text{L4}$$

The law is simpler than that for general choice because it has eliminated nondeterminism. Whereas general choice degenerates to pure nondeterminism when both operands offer the same initial event, asymmetric choice always favors the first operand. (Significantly, Burns and Wellings note that the purpose of prioritization in real-time programming languages is to improve predictability by reducing nondeterminism [4].)

Asymmetric choice distributes through nondeterminism,

$$P \vec{\Box} (Q \sqcap R) = \left(P \vec{\Box} Q\right) \sqcap (P \vec{\Box} R) \qquad \text{L5A}$$

$$(Q \sqcap R) \vec{\Box} P = \left(Q \vec{\Box} P\right) \sqcap (R \vec{\Box} P) \qquad \text{L5B}$$

For instance,

$$\begin{aligned}&\mathit{preferences}\left(a \to P \vec{\Box} (b \to Q \sqcap c \to R)\right) \\ &\quad = \mathit{preferences}\left(\left(a \to P \vec{\Box} b \to Q\right) \sqcap (a \to P \vec{\Box} c \to R)\right) \\ &\quad = \{\{a \mapsto a, b \mapsto b, a \mapsto b\}, \{a \mapsto a, c \mapsto c, a \mapsto c\}\}\end{aligned}$$

This example demonstrates how nondeterminism may cause *preferences* to return more than one relation. It tells us that, dependent upon the random choice made by the pure nondeterminism operator, we either have a process that can execute a or b (with preference given to the former), or a process that can execute a or c (again with priority to a).

However, pure nondeterminism does not distribute through asymmetric choice. For example,

$$\mathit{preferences}\left(a \to P \sqcap \left(b \to Q \vec{\Box} c \to R\right)\right) = \{\{a \mapsto a\}, \{b \mapsto b, c \mapsto c, b \mapsto c\}\}$$

but

$$\begin{aligned}&\mathit{preferences}\left((a \to P \sqcap b \to Q) \vec{\Box} (a \to P \sqcap c \to R)\right) \\ &\quad = \{\{a \mapsto a\}, \{a \mapsto a, c \mapsto c, a \mapsto c\}, \\ &\qquad \{a \mapsto a, b \mapsto b, b \mapsto a\}, \{b \mapsto b, c \mapsto c, b \mapsto c\}\}\end{aligned}$$

An interesting feature of the example above is that the right-hand 'P' *may* be performed if the first argument to asymmetric choice, i.e., $(a \to P \sqcap b \to Q)$, elects to behave like $b \to Q$. The reader may confirm this by calculating the *traces* of such a process. Remember that $a \in \bigcup \mathit{refusals}(a \to P \sqcap b \to Q)$.

Concealment does not distribute backwards through asymmetric choice (see [14]). For example,

$$preferences\Big(\big(a \to b \to P \mathbin{\vec{\Box}} c \to Q\big) \setminus \{a\}\Big) = \{\{b \mapsto b\}\}$$

but

$$preferences\Big(((a \to b \to P) \setminus \{a\}) \mathbin{\vec{\Box}} ((c \to Q) \setminus \{a\})\Big) = \{\{b \mapsto b, c \mapsto c, b \mapsto c\}\}$$

The first example shows that a concealed high priority event is always selected immediately.

To illustrate the way that concealment may introduce nondeterminism consider that

$$preferences(a \to b \to P \mathbin{\Box} c \to Q) = \{\{a \mapsto a, c \mapsto c\}\}$$

but that

$$preferences((a \to b \to P \mathbin{\Box} c \to Q) \setminus \{a\}) = \{\{b \mapsto b\}, \{c \mapsto c\}\}$$

The two relations in the second example cover the cases where the hidden event does and does not spontaneously occur, respectively. By contrast asymmetric choice precluded nondeterminism when it was used in place of general choice above.

Interleaving does not distribute through asymmetric choice (see [14]). For example,

$$preferences\Big(a \to P \mid\mid\mid \big(b \to Q \mathbin{\vec{\Box}} c \to R\big)\Big) = \{\{a \mapsto a, b \mapsto b, c \mapsto c, b \mapsto c\}\}$$

but

$$preferences\Big(\big(a \to P \mathbin{\vec{\Box}} b \to Q\big) \mid\mid\mid (a \to P \mathbin{\vec{\Box}} c \to R)\Big) = \{\{a \mapsto a, b \mapsto b, c \mapsto c, a \mapsto b, a \mapsto c\}\}$$

The second example states that, in an environment prepared to perform both a and c, a must always occur. Although not determinable from the *preferences*, an examination of the *traces* for such a process shows that the left-hand '$a \to P$' is always selected in this situation. In an environment prepared to perform only b and c, no particular preference is expressed.

Finally we compare asymmetric choice with its symmetric counterpart. Asymmetric choice distributes through general choice,

$$P \mathbin{\vec{\Box}} (Q \mathbin{\Box} R) = \big(P \mathbin{\vec{\Box}} Q\big) \mathbin{\Box} (P \mathbin{\vec{\Box}} R) \qquad \text{L6A}$$

$$(Q \mathbin{\Box} R) \mathbin{\vec{\Box}} P = \big(Q \mathbin{\vec{\Box}} P\big) \mathbin{\Box} (R \mathbin{\vec{\Box}} P) \qquad \text{L6B}$$

but the converse is not true, e.g.,

$$preferences\Big(a \to P \mathbin{\Box} \big(b \to Q \mathbin{\vec{\Box}} c \to R\big)\Big) = \{\{a \mapsto a, b \mapsto b, c \mapsto c, b \mapsto c\}\}$$

whereas

$$preferences\big((a \to P \,\Box\, b \to Q)\,\vec{\Box}\,(a \to P \,\Box\, c \to R)\big)$$
$$= \{\{a \mapsto a, b \mapsto b, c \mapsto c, a \mapsto c, b \mapsto c\}\}.$$

It is interesting to note in the second example that preference $b \mapsto a$ does not appear even though '$b \to Q$' appears on the left of asymmetric choice and '$a \to P$' on the right. This is because, in an environment prepared to perform a, the left-hand '$a \to P$' has priority over the right-hand one and is always selected (again this is confirmed by *traces*).

Asymmetric concurrency. The asymmetric concurrency operator is associative:

$$P\,\vec{\|}\big(Q\,\vec{\|}R\big) = \big(P\,\vec{\|}Q\big)\,\vec{\|}R \qquad \text{L1}$$

The following laws adapted from standard concurrency [14] hold straightforwardly for asymmetric concurrency. Subscripts indicate process alphabets.

$$P\,\vec{\|}STOP_{\alpha P} = STOP_{\alpha P}\,\vec{\|}P = STOP_{\alpha P} \qquad \text{L2}$$

Let $\alpha P = \{a, c, d\}$ and $\alpha Q = \{b, c, d\}$. Then

$$(c \to P)\,\vec{\|}(c \to Q) = c \to \big(P\,\vec{\|}Q\big) \qquad \text{L3A}$$

$$(c \to P)\,\vec{\|}(d \to Q) = STOP \qquad \text{L3B}$$

$$(a \to P)\,\vec{\|}(c \to Q) = a \to \big(P\,\vec{\|}(c \to Q)\big) \qquad \text{L4A}$$

(Notice that

$$preferences\big(a \to P\,\vec{\|}c \to Q\big) = \{\{a \mapsto a\}\}$$

because event c cannot proceed without the cooperation of the first operand.)

$$(c \to P)\,\vec{\|}(b \to Q) = b \to \big((c \to P)\,\vec{\|}Q\big) \qquad \text{L4B}$$

$$(a \to P)\,\vec{\|}(b \to Q) = \big(a \to \big(P\,\vec{\|}(b \to Q)\big)\big)\,\vec{\Box}\,\big(b \to \big((a \to P)\,\vec{\|}Q\big)\big) \qquad \text{L5}$$

Law L5 preserves the prioritization by defining the behavior of asymmetric concurrency in terms of asymmetric choice. Similarly, where Hoare gives a generalization of these laws using deterministic choice [14], the generalized law for asymmetric concurrency uses asymmetric choice:

Let $P = (a{:}A \to P(a))$, $Q = (b{:}B \to Q(b))$.

Then,

$$\big(P\,\vec{\|}Q\big) = \Big(\big(x{:}(A \cap B) \to \big(P(x)\,\vec{\|}Q(x)\big)\big)$$
$$\Box\big(y{:}(A - \alpha Q) \to \big(P(y)\,\vec{\|}Q\big)\big)\Big) \qquad \text{L6}$$
$$\vec{\Box}\big(z{:}(B - \alpha P) \to \big(P\,\vec{\|}Q(z)\big)\big)$$

There are three distinct cases. Either the event performed belongs to the alphabets of both processes, only to the alphabet of the first process, or only to the alphabet of the second. In the first two cases the event (x or y) has high priority since it is performed by the first operand, but in the third case it only belongs to the second operand and hence has low priority (z).

Like standard concurrency [14], asymmetric concurrency is distributive:

$$P \vec{\|} (Q \sqcap R) = (P \vec{\|} Q) \sqcap (P \vec{\|} R) \quad \text{L7}$$

$$(P \sqcap Q) \vec{\|} R = (P \vec{\|} R) \sqcap (Q \vec{\|} R) \quad \text{L8}$$

Asymmetric concurrency distributes through general choice,

$$P \vec{\|} (Q \,\Box\, R) = (P \vec{\|} Q) \,\Box\, (P \vec{\|} R) \quad \text{L9A}$$

$$(Q \,\Box\, R) \vec{\|} P = (Q \vec{\|} P) \,\Box\, (R \vec{\|} P) \quad \text{L9B}$$

However, the converse is not true. For instance, assuming $\alpha P = \{a\}$, $\alpha Q = \{b\}$ and $\alpha R = \{c\}$,

$$\mathit{preferences}(a \to P \,\Box\, (b \to Q \vec{\|} c \to R)) = \{\{a \mapsto a, b \mapsto b, c \mapsto c, b \mapsto c\}\}$$

but

$$\mathit{preferences}((a \to P \,\Box\, b \to Q) \vec{\|} (a \to P \,\Box\, c \to R)) = \{\{a \mapsto a, b \mapsto b, c \mapsto c, a \mapsto c, b \mapsto c\}\}$$

In the second example the preference for a over c refers to the left-hand process '$a \to P$' (the *traces* reveal that the right-hand one can never execute).

Asymmetric concurrency distributes forward through asymmetric choice,

$$P \vec{\|} (Q \,\vec{\Box}\, R) = (P \vec{\|} Q) \,\vec{\Box}\, (P \vec{\|} R) \quad \text{L10}$$

but not backward. For example,

$$\mathit{preferences}((b \to Q \,\vec{\Box}\, c \to R) \vec{\|} a \to P) = \{\{a \mapsto a, b \mapsto b, c \mapsto c, b \mapsto c, b \mapsto a, c \mapsto a\}\}$$

whereas

$$\mathit{preferences}((b \to Q \vec{\|} a \to P) \,\vec{\Box}\, (c \to R \vec{\|} a \to P)) = \{\{a \mapsto a, b \mapsto b, c \mapsto c, b \mapsto a, b \mapsto c, a \mapsto c\}\}$$

In an environment prepared to perform a and c the first example will give priority to c but the second will give priority to (the left-hand) a.

Asymmetric interleaving. Asymmetric interleaving is asymmetric, associative,

$$(P \vec{|||} Q) \vec{|||} R = P \vec{|||} (Q \vec{|||} R) \quad \text{L1}$$

and distributes through nondeterminism,

$$P \vec{|||} (Q \sqcap R) = \left(P \vec{|||} Q\right) \sqcap (P \vec{|||} R) \qquad \text{L2A}$$

$$(Q \sqcap R) \vec{|||} P = \left(Q \vec{|||} P\right) \sqcap (R \vec{|||} P) \qquad \text{L2B}$$

Also, like conventional interleaving [14],

$$P \vec{|||} STOP = STOP \vec{|||} P = P \qquad \text{L3}$$

It can be defined in terms of asymmetric choice,

$$(x \to P) \vec{|||} (y \to Q) = \left(x \to \left(P \vec{|||} (y \to Q)\right)\right) \vec{\Box} \left(y \to \left((x \to P) \vec{|||} Q\right)\right) \qquad \text{L4}$$

or more generally

$$\text{Let } P = (a{:}A \to P(a)),\ Q = (b{:}B \to Q(b)).$$

Then

$$P \vec{|||} Q = \left(x{:}A \to \left(P(x) \vec{|||} Q\right)\right) \vec{\Box} \left(y{:}(B - A) \to \left(P \vec{|||} Q(y)\right)\right) \qquad \text{L5}$$

4. ASSESSMENT OF PRIORITY IMPLEMENTATIONS

In the following examples let $P = (p \to P)$, $Q = (q \to Q)$, and so on, where $\alpha P = \{p\}$, $\alpha Q = \{q\}$, etc. The definitions in Section 3 successfully distinguish prioritized from nonprioritized behavior. Although the following traces of two differently prioritized processes suggest that they are logically equivalent,

$$\begin{aligned} &traces(P \| Q) \\ &\quad = traces\left(P \vec{\|} Q\right) \\ &\quad = \{\langle\rangle, \langle p\rangle, \langle q\rangle, \langle p, q\rangle, \langle q, p\rangle, \ldots\} \end{aligned}$$

they differ when their 'priority-based' behavior is considered,

$$\begin{aligned} &preferences(P \| Q) \\ &\quad = \{\{p \mapsto p, q \mapsto q\}\} \\ &\quad \neq \{\{p \mapsto p, q \mapsto q, p \mapsto q\}\} = preferences\left(P \vec{\|} Q\right). \end{aligned}$$

We now use the formal definition as a base agent against which to assess the implementations of priority in occam and Ada.

4.1 occam

The definitions are entirely consistent with the description of the concept of priority in the occam language definition. `PRI PAR` states that "if there are two processes that are not blocked, the textually earlier one must be executed; if an earlier process becomes unblocked while a later one is executing, the implementation must rapidly switch to executing the higher priority process" [19]. The relations defined for asymmetric concurrency and interleaving, by rules P9 and P10, ensure that all events possible for the first

operand alone are 'preferred' over those possible for the second operand. As long as an implementation respects these preferences, i.e., does not perform an event belonging to the second operand when one belonging to the first process is possible, then its behavior matches that of PRI PAR. Furthermore, when asymmetric interleaving cannot distinguish between the initial events definition T3 ensures that the higher priority operand always has precedence.

PRI ALT "waits until some of its branches are ready and then selects one for execution." Where two or more branches may be executed it "favors its textually earlier branches over the later ones" [19]. Relations resulting from P8 always express a preference for distinct events from the first operand over the second. Moreover, where the 'enabled' initial events offered cannot be uniquely distinguished, definition T2 guarantees that the first operand is always selected. In combination these two definitions capture the intention of PRI ALT in occam.

The situation is less clear when we consider the actual implementation of these constructs on the transputer. For each example in Figure 1 we give its *preferences* and then contrast these with the actual behavior defined by the transputer implementation of occam [16].

The partial ordering defined by example 1 confirms our intuitions,

$$\begin{aligned} &\mathit{preferences}\left(\left(P \vec{\|} (Q \| R)\right) \vec{\|} S\right) \\ &\quad = \{\{p \mapsto p, q \mapsto q, r \mapsto r, s \mapsto s, p \mapsto q, p \mapsto r, q \mapsto s, r \mapsto s, p \mapsto s\}\} \end{aligned} \tag{1a}$$

in that no preference is expressed between Q and R. On the transputer only two priority levels are available, high and, the default, low [11]. A PRI PAR causes its first argument (P in this case) to execute at high priority and all subsequent arguments at low priority. The effect thus achieved for example 1 is equivalent to

$$\begin{aligned} &\{\{p \mapsto p, q \mapsto q, r \mapsto r, s \mapsto s, p \mapsto q, p \mapsto r, p \mapsto s\}\} \\ &\quad = \mathit{preferences}\left(P \vec{\|} ((Q \| R) \| S)\right) \end{aligned}$$

However this limitation of the transputer is well-documented and we do not intend this comparison as a criticism.

More significantly, example 2 defines the following preferences,

$$\mathit{preferences}\left(P \| \left(Q \vec{\|} R\right)\right) = \{\{p \mapsto p, q \mapsto q, r \mapsto r, q \mapsto r\}\} \tag{2a}$$

P and Q, and P and R, are incomparable in the partial ordering. On the transputer, however, Q will be given a higher priority than either P or R resulting in

$$\{\{p \mapsto p, q \mapsto q, r \mapsto r, q \mapsto p, q \mapsto r\}\} = \mathit{preferences}\left(Q \vec{\|} (P \| R)\right)$$

In this case the actual behavior differs subtly from that of the formal definition, even though this is identified as a correct use of prioritization [16].

Example 3 results in a total ordering:

$$preferences\left(\left(P\vec{\|}Q\right)\vec{\|}R\right) = \{\{p \mapsto p, q \mapsto q, r \mapsto r, p \mapsto q, q \mapsto r, p \mapsto r\}\} \quad 3a$$

In this case the transputer compiler writer's guide [16] acknowledges that the limitations on the transputer priority levels would lead to a disparity with this ordering, i.e.,

$$\{\{p \mapsto p, q \mapsto q, r \mapsto r, p \mapsto q, p \mapsto r\}\} = preferences\left(P\vec{\|}(Q\|R)\right)$$

and nesting of PRI PAR constructs is explicitly outlawed, even though it is syntactically valid in occam [15].

In the formal definition of example 4, i.e.,

$$\begin{aligned} &preferences\left(\left(c \mapsto P\vec{\|}d \to Q\right)\|(d \to R\,\vec{\Box}\,c \to S)\right) \\ &\quad = \{\{c \mapsto c, d \mapsto d, c \mapsto d, d \mapsto c\}\} \end{aligned} \quad 4a$$

the contradictory requirements of the asymmetric concurrency and choice operators cancel one another: either event is allowed. (By definition, we assume that two distinct events pointing to one another in the partial ordering is equivalent to the events being unordered.) There are two possible scenarios since the PAR does not specify which process to execute first. If the first PAR operand begins execution before the second, the enclosed high priority process will become active and communication will take place on channel c. If the second PAR operand begins execution first it will always elect to communicate on channel d. Unfortunately a single transputer implementation fails to conform with this—the high priority process attempting to send on channel c will be given preference over the PRI ALT statement and communication will always occur on channel c, in effect,

$$\{\{c \mapsto c, d \mapsto d, c \mapsto d\}\} = preferences\left(c \to P\vec{\|}\left(d \to Q\|(d \to R\,\vec{\Box}\,c \to S)\right)\right).$$

According to our formal definitions example 5 expresses a preference for event d over c:

$$\begin{aligned} &preferences\left((c \to P\|d \to Q)\vec{\|}(d \to R\,\vec{\Box}\,c \to S)\right) \\ &\quad = \{\{c \mapsto c, d \mapsto d, d \mapsto c\}\}. \end{aligned} \quad 5a$$

The high priority process (consisting of two outputs executing concurrently) expresses no particular preference, whereas the low priority process (consisting of an asymmetric choice) prefers d to c. Since the wishes of the low priority process do not contradict those of the high priority process (see the definition of $\oplus$) they are still respected. In this case a single transputer implementation concurs—the first two processes execute until they block on their output statements, then the low priority process is awoken and always decides to receive the available value on channel d.

It should be stressed that the behavior of these examples may differ if the processes are distributed among geographically separated processors. For

instance, in example 4, if there is an inordinately long communication delay between the first and third processes then communication may take place on channel d. This is to be expected, however, because the occam PRI PAR statement, like the priority pragma in Ada, is not meaningful across processor boundaries [23].

4.2 Ada

The priority pragma in Ada is used to express a preference between two tasks such that "if two tasks with different priorities are both eligible for execution... using the same physical processors... then it cannot be the case that the task with the lower priority is executing while the task with higher priority is not" [17]. Our definitions of asymmetric concurrency and asymmetric interleaving clearly express the intention that events possible for the high priority operand should be executed in preference to those of the low priority operand. Furthermore, if we model an Ada rendezvous as a single action shared among the communicating process (tasks), then the priority inheritance that takes place during a rendezvous is implicit in the fact that the shared action must be in the alphabet of the high priority operand.

To formally assess the definition of priority in Ada we now consider the examples in Figure 2.

Example 6 expresses the intention that task Q has precedence over task R but the relationship with task P is left unstated. Expressing this code in CSP yields

$$preferences\left(P \vec{\|} \left(Q \vec{\|} R\right)\right) = \{\{p \mapsto p, q \mapsto q, r \mapsto r, q \mapsto r\}\}. \tag{6a}$$

However the Ada language definition states that "for tasks without explicit priority the scheduling rules are not defined" [17]. We therefore cannot assign any default priority value to P and its behavior may range anywhere between lowest

$$\{\{p \mapsto p, q \mapsto q, r \mapsto r, q \mapsto r, q \mapsto p, r \mapsto p\}\} = preferences\left(\left(Q \vec{\|} R\right) \vec{\|} P\right)$$

and highest

$$\{\{p \mapsto p, q \mapsto q, r \mapsto r, q \mapsto r, p \mapsto q, p \mapsto r\}\} = preferences\left(P \vec{\|} \left(Q \vec{\|} R\right)\right)$$

priority [3].

In Example 7, low priority task A may rendezvous with either of two tasks C, which has high priority, or D, which has medium priority:

$$\begin{aligned} &preferences\left((c \rightarrow STOP \vec{|||}\, d \rightarrow STOP) \vec{\|} (c \rightarrow STOP \,\Box\, d \rightarrow STOP)\right) \\ &\quad = \{\{c \mapsto c, d \mapsto d, c \mapsto d\}\}. \end{aligned} \tag{7a}$$

Clearly the intention is that the rendezvous should take place with the highest priority task. Unfortunately the Ada select offers no guarantee that

the task priorities will influence the choice made by A [3], and therefore either rendezvous may take place:

$$\{\{c \mapsto c, d \mapsto d\}\}$$
$$= preferences((c \rightarrow STOP \,|||\, d \rightarrow STOP) \| (c \rightarrow STOP \,\Box\, d \rightarrow STOP))$$

Example 8 is similar in that task A is prepared to rendezvous with high priority task C or low priority task D. In this case, however, both "clients" initially call the same entry b. In our equivalent CSP definition the shared events are indistinguishable—to see which tasks actually synchronize we will consider the set of possible traces, since the event following b makes the choice visible:

$$\begin{aligned} traces\big((b \rightarrow (c \rightarrow STOP \,\Box\, d \rightarrow STOP)) \| \\ (b \rightarrow c \rightarrow STOP \,\vec{|||}\, b \rightarrow d \rightarrow STOP)\big) \\ = \{\langle\rangle, \langle b\rangle, \langle b, c\rangle\}. \end{aligned} \tag{8a}$$

Definition T3 excludes trace $\langle b, d\rangle$ in favor of the higher priority alternative. Unfortunately task entry queues in Ada are strictly FIFO. This means that, if tasks C and D were executed on different physical processors, the entry call from D could be queued first:

$$\{\langle\rangle, \langle b\rangle, \langle b, c\rangle, \langle b, d\rangle\}$$
$$= traces((b \rightarrow (c \rightarrow STOP \,\Box\, d \rightarrow STOP)) \|$$
$$(b \rightarrow c \rightarrow STOP \,|||\, b \rightarrow d \rightarrow STOP))$$

The priority of tasks in entry queues is not taken into account [4].

5. DISCUSSION

5.1 Effect on Logical Behavior

In discussing Ada, Hoare praises the separation of logical from 'real-time' issues such as priority [14] and this advantage of the approach to prioritzation in occam and Ada is frequently cited in the literature. Our definition of priority in CSP, however, identifies one situation in which prioritization seemingly affects the logical behavior of a process. Where there is a prioritized choice between two instances of the same event, then the higher priority one is *always* chosen. For example, although,

$$traces(a \rightarrow b \rightarrow STOP \,\Box\, a \rightarrow c \rightarrow STOP) = \{\langle\rangle, \langle a\rangle, \langle a, b\rangle, \langle a, c\rangle\}$$

in the prioritized case trace $\langle a, c\rangle$ is not possible,

$$traces(a \rightarrow b \rightarrow STOP \,\vec{\Box}\, a \rightarrow c \rightarrow STOP) = \{\langle\rangle, \langle a\rangle, \langle a, b\rangle\}.$$

Thus prioritization has *restricted* the possible logical behaviors. This would occur in occam with a prioritized choice between two permanently 'open' guards. We therefore disagree with the usual claim that prioritization "does not affect the logical behavior of a program" [15], albeit in rather unusual code segments.

5.2 Process versus Alternative Priorities

Example 4, suggested by Burns and Wellings [4], illustrates a significant feature of our formal model. The priorities expressed by the asymmetric concurrency and choice operators are treated uniformly and may interact (see 4a). When discussing the needs of prioritization in Ada, Burns and Wellings argue that it may be preferable to keep the priorities expressed over processes and alternatives distinct, with the process priorities having precedence when such a conflict arises. This is justified mainly in the simplification it provides for the run-time scheduler which then needs to concern itself with process priorities only. We do not dispute such practical advantages, but merely note that our concern herein has been with formality and the unified approach seems to be more elegant.

5.3 Lack of Preference

Another interesting issue is the treatment of an *absence* of preference between events, as illustrated by example `5a`. In this particular example it appears that the wishes of the low priority process (which expresses a preference for d) hold sway over those of the high priority process (which does not express any preference). The question is one of intent: by using a `PAR`, rather than `PRI PAR`, in defining the high-priority process in example `5` are we to assume that the programmer (a) does not care which channel, `c` or `d`, is selected, or (b) wishes the selection to be "fair?" In other words, by *not* expressing a preference is the programmer passively accepting the wishes of other parallel processes, or actively asserting that no bias should be allowed? By letting the low priority process impose its view we have elected to assume alternative (a), keeping in mind that CSP does not guarantee fairness [14] and that scheduling of `PAR` processes in occam is implementation dependent.

6. FUTURE WORK

The problems with priority in Ada highlighted in Section 4.2 are well known. The literature also identifies other practical difficulties such as the inability of a task to change its priority and the fact that all tasks of the same type must have the same priority [3]. In light of these criticisms the next revision of the Ada language, provisionally called Ada 9X, will substantially improve the definition of priority.

This new standard is still in a state of flux at the time of writing. One of the suggested additions is a '`priority select`' statement equivalent to '`PRI ALT`' [5]. This is, of course, already covered in our model by asymmetric choice. Other proposed changes include a formally defined default priority and priority-based queuing of entry calls [20].

Presently, priorities in Ada are statically associated with tasks (apart from the special case of priority inheritance during rendezvous). Ada 9X will better differentiate between the static "base" priority of a task and its run-time "system" (or "active") priority [20, 22]. In particular it seems certain that Ada 9X will include a '`set_priority`' statement with which a task can explicitly change its priority at run-time. Such a capability is considered essential in

real-time systems for handling so-called "mode changes," i.e., a refocusing of available computational power, forced upon the system by changes in its external environment.

Our formal definition of priority in CSP is essentially static. Nevertheless mode changes can be modeled. For instance, consider a reactive system described by two processes P and Q. Upon detection of some drastic change in the system environment, represented by x, the actions undertaken by Q must be given priority over all others. We can define this using the CSP 'interrupt' operator which abandons execution of its left-hand operand upon the first occurrence of an event from the right-hand operand:

$$(P \| Q)^{\wedge} \left(x \rightarrow \left(Q \vec{\|} P \right) \right)$$

Although adequate, such a model is admittedly much less direct than the Ada 9X approach in which task Q simply gives itself a high priority using `set_priority`. Further study is required to determine whether the priority model herein can fully cater for Ada 9X, or whether fundamental extensions are needed.

7. RELATED WORK

Introduction of priority into the process algebraic specification languages is by no means a new goal. Jeffrey [18] gives a through review of recent work in this field. He observes that three distinct approaches have been used:

—*Prioritize the choice operator.* For instance, Camilleri [9] defined a prioritized choice operator for CCS (no attempt was made to define prioritized concurrency). We find Camilleri's definitions unsatisfying, however, since they are based on the events in the refusal set of the *environment* itself. Our definitions avoid any direct reference to the environment since we do not consider it acceptable to attempt to influence, or even query, the potential behaviors offered by the "outside world." Moreover Camilleri uses a complement for the special invisible event τ, a concept specifically outlawed in CCS [21]. More fundamentally, however, we believe that priority should ultimately be associated with actions, not the operators used to compose them. (To this end we defined priority as a relation over events, not processes. This corresponds to the programmer's intuitions—it is the *actions* performed by the processes that have some degree of relative "importance," not the processes themselves.)

—*Statically associate priority with actions.* In this case a function is defined from the alphabet of actions to priority values. Jeffrey takes this approach in defining a prioritized language akin to CCS and CSP. Silent (hidden) actions, one for each priority value, are then used to represent the decision to accept a high priority action over a low priority action in choices [18]. Gerber and Lee [12] also use a static priority mapping. The expressive power of this approach is limited, however, because every occurrence of an action must have the same priority. In our model each appearance of an

action in the specification may have a distinct priority (as illustrated by the mode change example in Section 6). (The work of Jeffrey also illustrates the close relationship between prioritized and "timed" process algebras—whereas a prioritized algebra can be used to state that some important action must occur next, a timed algebra can correspondingly express the intention that an urgent action must occur before any time passes [18].)

—*Locally associate priority with actions.* Jeffrey notes a suggestion, attributed to Wang, for associating priorities with actions at each appearance. The suggestion is to assign an absolute priority value to each action concealed using the CCS hiding operator, by subscripting the operator with such a value. This is very close to the approach developed herein. However, we avoid the need to use *absolute* priority values, preferring instead to indicate the *relative* priority of actions. This is a better approach when modularity and compositional behavior are considered because there is no universally accepted base or scale for expressing priority values [23].

Modeling and assessing the priority models of Ada and occam has also received prior attention:

—Camilleri's work discussed above was based on an earlier definition of `PRI ALT` in occam [8].

—Barrett [1] gives a definition of occam's priority mechanism that considers not only `PRI ALT` and `PRI PAR` but a notion of "fairness" as well. However the definition is expressed using SOS style, rather than Hoare's notation. It uses a low-level, operational approach, based on sets of "ready guards" to model the possible inputs. This approach is programming language specific and, like the work discussed above, makes an unfortunate appeal to the "initiator" of events, i.e., the process or its environment. Burns and Wellings also use a low-level, operational model to specify the behavior of Ada's priority mechanism in Z [7]; they explicitly record sets of "wanted communications" [6]. However the work has the advantage of being suitable for proving properties of the Ada mechanism, e.g., the ability of a high priority process to indefinitely block others.

—Best and Koutny [2] define a model of priority using Petri nets, albeit with static priorities only. They discuss the possible application of this model to occam's `PRI ALT` and, to a lesser extent, `PRI PAR`.

—Narayana [22] recently developed a model of prioritization for a subset of the Ada 9X standard. The concern here was with making the scheduling points, which may cause tasks to be preempted, observable, and the approach is very different from ours. A small subset language was described and a linear-history style semantics developed. Narayana notes that reasoning about priorities in the resultant model has a similar complexity to reasoning about shared variables. Unlike our work this model does not account for nested concurrency, but the subset language does include shared variables and can explicitly model "limited parallelism,"

i.e., where the number of physical processors is less than the number of tasks. Even if we assume that each CSP action is an indivisible (nonpreemptable) segment of Ada code, our model is weak in this regard due to the inability of CSP to distinguish between 'true' and interleaved concurrency.

Although not defining a model, Ho's work on improved implementations of `PRI ALT` is also of interest because it points the way towards practical implementations of occam that may better reflect the language semantics [13]. References to other works on priority are given by Barrett [1], Jeffrey [18], and Best and Koutny [2].

8. CONCLUSION

In this paper we have formally defined the concept of priority in CSP. The resulting definition was then shown to give a sound basis for assessment and criticism of occam as implemented on the transputer and the definition of the priority pragma in Ada. A number of discrepancies between the formal model and these implementations were highlighted, not all of which are clearly identified in the literature.

GLOSSARY

CSP

$\langle a, b, c, \ldots \rangle$	Trace consisting of events a, b, c, etc.
t_0	The first event in trace t.
$s \hat{} t$	Traces s and t concatenated.
αP	Alphabet of process P, i.e., those events it *may* perform.
$a \rightarrow P$	Prefix: event a followed by process P.
$(x{:}\ A \rightarrow P(x))$	Choice of a distinct event x from A, followed by $P(x)$.
$P \sqcap Q$	Pure nondeterminism: process P or process Q.
$P \Box Q$	General choice between processes P and Q.
$P \| Q$	Processes P and Q performed concurrently.
$P \mid\mid\mid Q$	Noncommunicating processes performed concurrently.
$P \setminus C$	Process P with all events in set C hidden.
P/t	Process P after performing trace t.
$P \hat{} Q$	Process P until interrupted by Q.

CSP Extensions

$P \vec{\Box} Q$	Asymmetric choice.
$P \vec{\|} Q$	Asymmetric concurrency.
$P \vec{\mid\mid\mid} Q$	Asymmetric noncommunicating concurrency.

Relations

$a \mapsto b$	The ordered pair (a, b).
$\{a \mapsto b, \ldots\}$	A relation in which a is related to b, etc.

$a\ R\ b$	Relation R as infix operator, i.e., $(a \mapsto b) \in R$.
R^{-1}	Inverse of relation R, i.e., all pairs in R reversed.
$\mathrm{dom}(R)$	Domain of relation R, i.e., $\{a \mid \exists b \cdot a\ R\ b\}$.

ACKNOWLEDGMENTS

I wish to thank Olivier de Vel for pointing out errors in an earlier version of this paper, Michael Pilling for numerous valuable discussions, and the anonymous referees for helpful suggestions.

REFERENCES

1. BARRETT, G. The semantics of priority and fairness in occam In *Mathematical Foundations of Programming Semantics, 442*, Lecture Notes in Computer Science, M. Main, A. Melton, M. Mislove, and D. Schmidt, Eds., Springer-Verlag, New York, 1989, 194–208.
2. BEST, E., AND KOUTNY, M. Petri net semantics of priority systems. *Theoret. Comput. Sci. 96*, 1 (Apr. 1992), 175–215.
3. BURNS, A., LISTER, A. M., AND WELLINGS, A. J. *A Review of Ada Tasking, 262, Lecture Notes in Computer Science*. Springer-Verlag, New York, 1987.
4. BURNS, A. AND WELLINGS, A. J. The notion of priority in real-time programming languages *Comput. Lang. 15*, 3 (1990), 153–162.
5. BURNS, A., AND WELLINGS, A. J. Usability of the Ada tasking model. *Ada Lett. X*, 4 (1990), 49–56.
6. BURNS, A., AND WELLINGS, A. J. Priority inheritance and message passing communication: A formal treatment. *J. Real-Time Syst. 3* (1991), 19–44.
7. BURNS, A , AND WELLINGS, A. J. Specifying an Ada tasking run-time support system. *Ada User 12* (1991), 160–186.
8. CAMILLERI, J. An operational semantics for occam (extended version). Tech. Rep. 144, Univ of Cambridge Computer Lab., Aug. 1988.
9. CAMILLERI, J. Introducing a priority operator to CCS. Tech. Rep. 157, Univ. of Cambridge Computer Lab., Jan. 1989.
10. DILLER, A. *Z: An Introduction to Formal Methods*. Wiley, New York, 1990.
11. GALLETLY, J. *occam 2*, Pitman, 1990.
12. GERBER, R., AND LEE, I. CCSR: A calculus for communicating shared resources. Tech. Rep. MS-CIS-90-16, Univ. of Pennsylvania, Mar. 1990.
13. HO, D. N. M. Variations of ALT implementation on transputer. In *Proceedings of the 3rd Transputer/Occam International Conference* (Tokyo, May 1990), 195–207.
14. HOARE, C. A. R. *Communicating Sequential Processes*. Prentice-Hall, Englewood Cliffs, N.J., 1985.
15. INMOS LTD. *occam 2 Reference Manual*. Prentice-Hall, Englewood Cliffs, N.J., 1988.
16. INMOS LTD. *Transputer Instruction Set*. Prentice-Hall, Englewood Cliffs, N.J., 1988.
17. AMERICAN NATIONAL STANDARDS INSTITUTE. *The Programming Language Ada Reference Manual, vol. 155 of Lecture Notes in Computer Science*. Springer-Verlag, New York, 1983.
18. JEFFREY, A. Translating timed process algebra into prioritized process algebra. In *Formal Techniques in Real-Time and Fault-Tolerant Systems, 571, Lecture Notes in Computer Science*, J. Vytopil, Springer-Verlag, New York, 1992, 493–506.
19. JONES, G., AND GOLDSMITH, M. *Programming in occam 2*. Prentice-Hall, Englewood Cliffs, N.J., 1988.
20. MAYMIR-DUCHARME, F. A. Dynamic priorities, priority scheduling and priority inheritance. *Ada Lett. X*, 9 (Fall 1990), 39–45.

21. MILNER, R. *Communication and Concurrency*. Prentice-Hall, Englewood Cliffs, N.J., 1989.
22. NARAYANA, K. T. Observing task preemption in Ada 9X. In *Formal Techniques in Real-Time and Fault-Tolerant Systems, Vol. 571 of Lecture Notes in Computer Science*, J. Vytopil, Ed. Springer-Verlag, New York, 1992, 107–129.
23. PILLING, M. J. Dangers of priority as a structuring principle for real-time languages. *Australian Comput. Sci. Commun. 13*, 1, (Feb. 1991), 18.1–18.10.

Received December 1992; revised November 1992; accepted November 1992.