Programming Research Group

SCHEDULING-ORIENTED MODELS FOR REAL-TIME SYSTEMS

Gavin Lowe

PRG-TR-10-94

Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford OX1 3QD

Scheduling-Oriented Models for Real-Time Systems[1]

Gavin Lowe

June 1994

Abstract

In this paper we define models for reasoning about resource allocation and scheduling in real-time systems. We begin by producing a model that allows us to argue about the resource requirements of real-time distributed systems, and so identify conflicts upon resources; we can use the model to define and reason about schedulers that map jobs onto resources according to some pre-defined schedule. We then extend our model to encompass dynamic schedulers, where the scheduling decisions may depend upon the results of the computation so far. The models will aid in the important transformation of mapping an initial design for a system onto an actual implementation with jobs scheduled on processors.

1 Introduction

Many formalisms exist for designing and verifying real-time systems, for example, TAM [SZ92, SZJ93, SZ93], Timed CSP [RR86, DS92, DRRS94] and Timed CCS [MT90, Yi90, HR90]. However, none of these formalisms have dealt with the important problems of resource allocation and scheduling. The formalisms have adopted the *unbounded parallelism hypothesis*, i.e. the assumption that sufficient resources are always available. However, implementations of real-time systems rarely achieve unbounded parallelism, because fluctuations in work load mean that having sufficient resources at peak times would result in considerable spare capacity at low load times, reducing cost effectiveness. Thus when designing real systems one has to consider the problem of allocation of resources to agents, or—equivalently—the scheduling of jobs on resources.

In this paper we develop semantic models that allows us to argue about resource allocation and scheduling. The models are based upon the language TAM, a wide-spectrum development language for real-time systems, with constructs for such features as communication, deadline and parallelism.

Our models will enable us to define and reason about schedulers which allocate resources to tasks. Our model will recognize that if two agents both try to access the same resource

[1] Further copies of this Technical Report may be obtained from the Librarian (Oxford University Computing Laboratory, Programming Research Group, Wolfson Building, Parks Road, Oxford OX1 3QD, England, telephone: +44-865-273837, e-mail: `library@comlab.ox.ac.uk`) or by anonymous ftp from `ftp.comlab.ox.ac.uk` in `pub/Documents/techpapers/Gavin.Lowe/scheduling.ps.gz`.

simultaneously, then the system will fail. Our model will also tell us whether the resources available are sufficient to ensure that all deadlines are met.

In [SZ93] a refinement calculus is given for TAM. This allows an abstract specification to be refined into an initial design via a number of provably correct transformations. We would like to take this one step further: we would like to transform the initial design into an implementation with components scheduled on resources in a correct manner.

We believe that issues of scheduling should be dealt with *after* the initial design, because if the available resources change, we do not want to have to redo too much of the derivation: the functionality-oriented refinement to the initial design should remain the same; only the scheduling calculations will change. Also, program derivation can be seen as a top-down activity, whereas scheduling calculations are more bottom-up, considering first the resource requirements of atomic commands, and then combining these to derive the resource requirements of the whole program.

Thus the refinement scheme is as illustrated by figure 1.

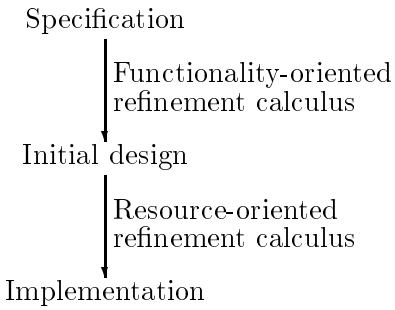

Figure 1: The two stage implementation process

Gerber and Lee [GL89, GL90, GL92] have produced a language, the *Calculus for Communicating Shared Resources*. Their model is operational, and similar to that of SCCS [Mil89], extended to model priorities (following the approach of [CH88]), and such that transitions are allowed only if they do not involve a conflict on resources. Their approach differs from ours in that we use a compositional denotational model which allows us to break down requirements on systems into requirements on the subcomponents. Also, we define a hierarchy of models, allowing us to separate the issues of functionality and scheduling.

In the next section we describe the language TAM, and present the semantics given in [SZJ93]: this represents an agent as a predicate describing its behaviour. In section 3 we extend the model so as to be able to discuss the resources available to, and used by, an agent; resources are allocated to jobs according to a fixed scheduler, i.e. a scheduler where the scheduling decisions cannot depend upon the behaviour of other parts of the system; we also model the fact that an agent may fail, either because insufficient resources are available, or because a deadline is not met. We extend this in section 4 to be able to discuss dynamic schedulers, where scheduling decisions may be based upon the checkpoints encountered so far in the execution. Thus we will define a hierarchy of models, as in figure 2.

Note that the models in this paper do not deal explicitly with the important issue of priorities, although priorities can be coded up in our dynamic-scheduling model: we hope to extend our models in the future to make it easier to deal with priorities.

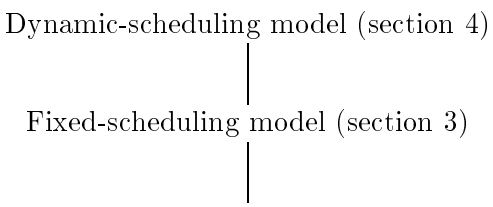

Figure 2: The hierarchy of models.

2 A functional semantics for TAM

The Temporal Agent Model (TAM) is a wide spectrum language for modelling real-time distributed computer systems. A TAM system is a collection of parallel agents, each with their own set of variables. Agents communicate via a form of asynchronous channel called a *shunt*, which can be thought of as a location in shared memory holding a pair consisting of a value and a time-stamp giving the last time at which the shunt was updated. Each shunt is owned by a single agent that can write values to it; all agents can then read this value.

The TAM language is defined by the BNF expression:

$$\begin{aligned}\mathcal{A} ::= \; & w : \Phi \mid \mathsf{skip} \mid \Delta t \mid x := E \mid (x, y) \leftarrow s \mid x \rightarrow s \mid (x : T)\mathcal{A} \mid (s : T)\mathcal{A} \mid \\ & [t]\mathcal{A} \mid \mathcal{A} \,;\, \mathcal{A} \mid \mathcal{A} \sqcap \mathcal{A}' \mid \bigsqcup_{i \in I}{}_t \; g_i \Rightarrow \mathcal{A}_i \mid \mathcal{A} \rhd^s_t \mathcal{A} \mid \mathcal{A} \mid A \mid \mu X_n \bullet F(X) \,,\end{aligned}$$

where x and y are variables, w is a set of variables, Φ is a specification predicate, s is a shunt, t is a time (which we take to be a natural number), E is an expression on variables, T is a type name, I is a finite indexing set, g_i is a boolean expression, and $F(X)$ is an agent expression (possibly containing the free variable X).

In [SZJ93] a semantics was given for this language, representing an agent by a predicate upon the start and termination times, the initial and final values of the variables, and the values held in shunts. Below we give informal descriptions of each of the language constructs, and give the predicate semantics. Note that our semantics differs a little from the semantics of [SZJ93]: we have made a few minor changes so as to suit our needs.

An agent is represented by a predicate Φ in the *TAM Logic Language* (*TAMLL*): a first-order logic with simple extensions to reason about time and the values held in variables and shunts. The predicate Φ may contain two distinguished free variables, t_α and t_ω, representing the start and termination times of an execution. We also allow terms of the form $s@t$ to refer to the value held in shunt s at time t, and similarly $x@t$ to refer to the value held in variable x. We write $s@t.ts$ for the time-stamp of s at t, and $s@t.v$ for the value. We will also write $E@t$ to represent the value of expression E where all the variables are taken at time t.

All shunts obey the following law:

Law 2.1: $\forall\, t : \mathit{Time} \bullet s@t.ts \leqslant t \wedge (s@t.ts < t \Rightarrow s@t = s@(t - 1))$. $\triangle$

The time-stamps of shunts represent the last time at which they were written to, so: (1) they will always be no more than the current time; and (2) if the time-stamp is strictly less than the current time, then the shunt has not been written to at this time step, and so its value must be the same as at the previous time step.

We will define a semantic function

$$\mathcal{P} : TAM \to TAMLL,$$

such that $\mathcal{P}[\![\mathcal{A}]\!]$ gives the predicate describing the functionality of agent $\mathcal{A}$.

We will need a predicate that asserts that the value of a variable or shunt does not change during an execution. For shunts, this means that they must hold the same value throughout the interval:

$$stable(s, t_\alpha, t_\omega) \mathrel{\widehat{=}} \forall\, t \in (t_\alpha, t_\omega] \bullet s@t = s@t_\alpha.$$

We consider variables to be private to a particular agent, and so we are only able to see their initial and final values:

$$stable(x, t_\alpha, t_\omega) \mathrel{\widehat{=}} x@t_\alpha = x@t_\omega.$$

These definitions extend to sets of variables and shunts in the obvious way.

The agent $w : \Phi$ is a specification which says that only variables and shunts in the frame w may be changed, and the execution must satisfy the predicate Φ.

$$\mathcal{P}[\![w : \Phi]\!] \mathrel{\widehat{=}} stable(W \setminus w, t_\alpha, t_\omega) \wedge \Phi.$$

Here and subsequently we take W to be the alphabet of the agent in question. The refinement rules of [SZ92, SZJ93] allow specifications of this form to be refined into an initial design in "concrete TAM".

The agent skip terminates immediately without changing the values of any variables or shunts:

$$\mathcal{P}[\![\mathsf{skip}]\!] \mathrel{\widehat{=}} t_\omega = t_\alpha.$$

The agent Δt terminates after t seconds without changing the values of any variables or shunts:

$$\mathcal{P}[\![\Delta t]\!] \mathrel{\widehat{=}} t_\omega = t_\alpha + t \wedge stable(W, t_\alpha, t_\omega).$$

The agent $x := E$ evaluates the expression E, using the *initial* values of variables, and assigns this value to x:

$$\mathcal{P}[\![x := E]\!] \mathrel{\widehat{=}} t_\omega > t_\alpha \wedge stable(W \setminus x, t_\alpha, t_\omega) \wedge x@t_\omega = E@t_\alpha.$$

The agent $(x, y) \leftarrow s$ reads shunt s and stores the time-stamp in x and the value in y; the shunt read occurs at some time t during the execution, although we do not know precisely when:

$$\begin{aligned}\mathcal{P}[\![(x, y) \leftarrow s]\!] \mathrel{\widehat{=}} {} & t_\omega > t_\alpha \wedge stable(W \setminus \{x, y\}, t_\alpha, t_\omega) \\ & \wedge \exists\, t \in (t_\alpha, t_\omega] \bullet x@t_\omega = s@t.ts \wedge y@t_\omega = s@t.v.\end{aligned}$$

The agent $x \to s$ writes the value of variable x into shunt s, time-stamping it at the time of the write:

$$\begin{aligned}\mathcal{P}[\![x \to s]\!] \mathrel{\widehat{=}} {} & t_\omega > t_\alpha \wedge stable(W \setminus s, t_\alpha, t_\omega) \\ & \wedge \exists\, t \in (t_\alpha, t_\omega] \bullet stable(s, t_\alpha, t - 1) \wedge s@t = (t, x@t_\alpha) \wedge stable(s, t, t_\omega).\end{aligned}$$

The agent $(x : T)\mathcal{A}$ declares x to be a local variable of type T whose value is not externally available. This is represented by an existential quantification over the initial and final values of the variable.

$$\mathcal{P}[\![(x : T)\mathcal{A}]\!] \mathrel{\widehat{=}} \exists\, v, v' : T \bullet \mathcal{P}[\![\mathcal{A}]\!][v/x@t_\alpha\ ,\ v'/x@t_\omega].$$

Similarly, the agent $(s : T)\mathcal{A}$ declares s to be a new shunt of type T. This is represented by an existential quantification over the values held in the shunt at all times during the execution, which we represent by a function f from times to time-stamp, value pairs. We insist that the shunt is initially time-stamped with its start time.

$$\mathcal{P}[\![(s : T)\mathcal{A}]\!] \mathrel{\widehat{=}} \exists f : Time \nrightarrow (Time \times T) \bullet f(t_\alpha).ts = t_\alpha \wedge \mathcal{P}[\![\mathcal{A}]\!][f(t)/s@t \mid t \in [t_\alpha, t_\omega]].$$

The agent $[t]\mathcal{A}$ executes $\mathcal{A}$ but gives it a deadline of t: we are only interested in those executions of $\mathcal{A}$ that terminate within t seconds:

$$\mathcal{P}[\![[t]\mathcal{A}]\!] \mathrel{\widehat{=}} \mathcal{P}[\![\mathcal{A}]\!] \wedge t_\omega - t_\alpha \leqslant t.$$

$\mathcal{A};\mathcal{A}'$ forms a sequential composition of $\mathcal{A}$ and $\mathcal{A}'$; note that we have to hide the intermediate values of the variables:

$$\mathcal{P}[\![\mathcal{A}\,;\mathcal{A}']\!] \mathrel{\widehat{=}} \exists\, t \in [t_\alpha, t_\omega]\ ;\ W' \in V \bullet \mathcal{P}[\![\mathcal{A}]\!][t/t_\omega\ ,\ W'/W@t_\omega] \wedge \mathcal{P}[\![\mathcal{A}']\!][t/t_\alpha\ ,\ W'/W@t_\alpha]$$

where $W : V$ is the set of variables that are owned by the agent.

The agent $\mathcal{A} \sqcap \mathcal{A}'$ nondeterministically acts like $\mathcal{A}$ or $\mathcal{A}'$:

$$\mathcal{P}[\![\mathcal{A} \sqcap \mathcal{A}']\!] \mathrel{\widehat{=}} \mathcal{P}[\![\mathcal{A}]\!] \vee \mathcal{P}[\![\mathcal{A}']\!]$$

The agent $\bigsqcup_{i \in I}{}_t\, g_i \Rightarrow \mathcal{A}_i$ evaluates each of the guards g_i for $i \in I$; it then executes one of the $\mathcal{A}_i$ such that the corresponding g_i is *true*; if all the g_is evaluate to *false* then the agent simply terminates. It is assumed that the guards can be evaluated within time t, and that if necessary the agent idles to pad-out this interval.

$$\begin{array}{rll} \mathcal{P}[\![\bigsqcup_{i \in I}{}_t\, g_i \Rightarrow \mathcal{A}_i]\!] \mathrel{\widehat{=}} & \text{let} & t'_\alpha = t_\alpha + t \quad \text{and} \quad J = \{\, i \in I \mid g_i@t_\alpha \,\} \quad \text{in} \\ & \text{if} & J = \emptyset \quad \text{then} \quad t_\omega = t'_\alpha \wedge \mathit{stable}(W, t_\alpha, t_\omega) \\ & \text{else} & \mathit{stable}(W, t_\alpha, t'_\alpha) \wedge \exists\, j \in J \bullet \mathcal{P}[\![\mathcal{A}_j]\!][t'_\alpha/t_\alpha]. \end{array}$$

The agent $\mathcal{A} \rhd^s_t \mathcal{A}'$ monitors the shunt s for t seconds: if the shunt is written to within this time, then $\mathcal{A}'$ is initiated at the time of this write; otherwise the agent times-out and $\mathcal{A}$ is initiated:

$$\begin{array}{rll} \mathcal{P}[\![\mathcal{A} \rhd^s_t \mathcal{A}']\!] \mathrel{\widehat{=}} & \text{let} & t'_\alpha = \inf\{t' > t_\alpha \mid s@t'.ts = t'\} \sqcap t_\alpha + t \quad \text{in} \\ & \text{if} & \exists\, t' \in (t_\alpha, t_\alpha + t] \bullet s@t'.ts = t' \\ & \text{then} & \mathit{stable}(W, t_\alpha, t'_\alpha) \wedge \mathcal{P}[\![\mathcal{A}']\!][t'_\alpha/t_\alpha] \\ & \text{else} & \mathit{stable}(W, t_\alpha, t'_\alpha) \wedge \mathcal{P}[\![\mathcal{A}]\!][t'_\alpha/t_\alpha]. \end{array}$$

Here, t'_α is either the time of the first write or the time of the timeout. We use the convention $\inf \emptyset = \infty$.

$\mathcal{A} \mid \mathcal{A}'$ executes $\mathcal{A}$ and $\mathcal{A}'$ in parallel; we assume that the composition terminates as soon as both agents have terminated:

$$\mathcal{P}[\![\mathcal{A}_1 \mid \mathcal{A}_2]\!] \mathrel{\widehat{=}} \exists\, t_1, t_2 \bullet \mathcal{P}[\![\mathcal{A}_1]\!][t_1/t_\omega] \wedge \mathcal{P}[\![\mathcal{A}_2]\!][t_2/t_\omega] \wedge t_\omega = t_1 \sqcup t_2.$$

The agent $\mu\, X_n \bullet F(X)$ represents a *finite* recursion, recursing at most $n-1$ times:

$$\begin{aligned} \mu\, X_0 \bullet F(X) &\mathrel{\widehat{=}} \mathsf{skip} \\ \mu\, X_{n+1} \bullet F(X) &\mathrel{\widehat{=}} F(\mu\, X_n \bullet F(X)). \end{aligned}$$

We can define a refinement relation between TAM agents as follows:

Definition 2.2: $A \sqsubseteq_{\mathcal{P}} \mathcal{A}'$ *iff* $\mathcal{P}[\![\mathcal{A}]\!] \Leftarrow \mathcal{P}[\![\mathcal{A}']\!]$. $\diamondsuit$

$\mathcal{A}$ is refined by $\mathcal{A}'$ if the behaviour of $\mathcal{A}'$ is more fully defined than that of $\mathcal{A}$. We will write $\sqsubseteq$ for this relation where this will not cause confusion.

Example

In this section we illustrate the TAM refinement calculus of [SZ92, SZJ93] with an example. We specify a system, then use the refinement laws to calculate an initial design. Each of the refinement laws can be proved sound with respect to the above semantics. In section 3 we will consider how to schedule our program.

Consider the following situation. We are given two input shunts, *in* and *in'*, and two output shunts, *out* and *out'*; a natural number is stored in each input shunt; we are required to design a system that takes the value written to *in* and outputs its factorial on *out*, and that takes the value written to *in'* and outputs its square on *out'*; these operations are to be completed within 6 seconds.

These requirements are met by the following TAM specification statement:

$$\begin{aligned} \{out, out'\} : \Phi \quad \text{where} \quad \Phi \mathrel{\widehat{=}} & t_\omega \leqslant t_\alpha + 6 \\ & \wedge\, \exists\, t \in (t_\alpha, t_\omega) \bullet out@t_\omega.v = (in@t.v)! \\ & \wedge\, \exists\, t \in (t_\alpha, t_\omega) \bullet out'@t_\omega.v = (in'@t.v)^2. \end{aligned}$$

We can refine this to

$$\begin{aligned} & \{out\} : \Phi_1 \mid \{out'\} : \Phi_2 \\ & \text{where} \quad \Phi_1 \mathrel{\widehat{=}} t_\omega \leqslant t_\alpha + 6 \wedge \exists\, t \in (t_\alpha, t_\omega) \bullet out@t_\omega.v = (in@t.v)! \\ & \qquad\quad \Phi_2 \mathrel{\widehat{=}} t_\omega \leqslant t_\alpha + 6 \wedge \exists\, t \in (t_\alpha, t_\omega) \bullet out'@t_\omega.v = (in'@t.v)^2 \end{aligned}$$

Continuing with the left hand side:

$$
\begin{array}{ll}
& \{out\} : \Phi_1 \\
\sqsubseteq & \Big\langle \text{definition of deadline; strengthening predicate} \Big\rangle \\
& [6]\{out\} : \exists\, x, x' : Time\ ;\ y, y', z, z' : \mathbb{N} \bullet \exists\, t \in (t_\alpha, t_\omega) \bullet (x', y') = in@t \\
& \qquad\qquad \wedge\ z' = (y')! \wedge out@t_\omega.v = z' \\
\sqsubseteq & \Big\langle \text{definition of local variables} \Big\rangle \\
& [6](x : Time)(y : \mathbb{N})(z : \mathbb{N})\{out, x, y, z\} : \exists\, t \in (t_\alpha, t_\omega) \bullet (x@t_\omega, y@t_\omega) = in@t \\
& \qquad\qquad \wedge\ z@t_\omega = (y@t_\omega)! \\
& \qquad\qquad \wedge\ out@t_\omega.v = z@t_\omega \\
\sqsubseteq & \Big\langle \text{definition of sequential composition} \Big\rangle \\
& [6](x : Time)(y : \mathbb{N})(z : \mathbb{N}) \left(\begin{array}{lll} \{x, y\} & : & \exists\, t \in (t_\alpha, t_\omega] \bullet (x@t_\omega, y@t_\omega) = in@t\ ; \\ \{z\} & : & z@t_\omega = (y@t_\alpha)!\ ; \\ \{out\} & : & out@t_\omega.v = z@t_\omega \end{array} \right) \\
\sqsubseteq & \Big\langle \text{definition of shunt read, assignment, shunt write} \Big\rangle \\
& [6](x : Time)(y : \mathbb{N})(z : \mathbb{N})\Big((x, y) \leftarrow in\ ;\ z := y!\ ;\ z \rightarrow out\Big)
\end{array}
$$

A similar refinement process can be applied to $\{out'\} : \Phi_2$. Combining the results, we achieve the following program:

$$
\begin{array}{l}
[6](x : Time)(y : \mathbb{N})(z : \mathbb{N})\Big((x, y) \leftarrow in\ ;\ z := y!\ ;\ z \rightarrow out\Big) \\
\mid \\
[6](x : Time)(y : \mathbb{N})(z : \mathbb{N})\Big((x, y) \leftarrow in'\ ;\ z := y^2\ ;\ z \rightarrow out'\Big)
\end{array}
$$

3 A model with fixed scheduling

In this section we extend the model of TAM so as to be able to argue about issues of resource allocation and fixed (i.e. compile time) scheduling.

We will represent the usage of resource r at time t by the pair (t, r). Formally we define the space *Resource* of resources by

$$Resource \mathrel{\widehat=} Time \times RName,$$

where *RName* is the space of resource names. We define

$$Resources \mathrel{\widehat=} \mathbb{P}(Resource),$$

and write R for a typical element of *Resources*. We will need a few pieces of notation for dealing with resource sets: $R \uparrow T$ restricts the resource set R to the time interval T; $R \restriction r$ restricts R to resources with resource name r; $\mathrm{Times}\, R$ returns the time domain of R; and $\mathrm{end}\, R$ returns the time of the last resource in R:

$$
\begin{array}{rcl@{\qquad\qquad}rcl}
R \uparrow T & \mathrel{\widehat=} & \{(t, r) \in R \mid t \in T\} & R \restriction r & \mathrel{\widehat=} & \{(t, r') \in R \mid r' = r\} \\
\mathrm{Times}\, R & \mathrel{\widehat=} & \{t \mid \exists\, r \bullet (t, r) \in R\} & \mathrm{end}\, R & \mathrel{\widehat=} & \sup(\mathrm{Times}\, R).
\end{array}
$$

In order to define schedulers, we will insist that all constructs requiring a scheduling decision are labelled. We can identify two sorts of resources for which scheduling decisions will be necessary:

Fixed duration resources: Operations such as assignments, shunt reads, shunt writes, and delays will require a resource (or maybe several resources) such as a CPU for a set period of time, this time being calculable at compile time; we can therefore represent the resource requirements of such an agent by an element of type *Resources*.

Variable duration resources: The operations of local variable and local shunt will require a new variable or shunt throughout the execution of the corresponding agent; it may not be convenient—or even possible—to calculate at compile time for how long this resource will be needed. However, we can define a function $\rho : Time \pfun RName$ such that $\rho(t)$ gives the resource on which the variable or shunt should be held at time t; note that this approach allows the variable or shunt to be held in different locations at different times, for example if some memory is swapped into secondary store. We also include in this class the resources monitoring shunts in the timeout construct, because we can not tell at compile time for how long the shunt will need to be monitored.

We write Lbl_{fixed} for the set of labels for fixed duration resources and $Lbl_{variable}$ for the set of labels for variable duration resources; we write Lbl for the union of these two sets, and write L for a typical label.

We define a scheduler to be a function that takes a label and the time at which the label is reached in the program: for labels corresponding to fixed duration resources, the function returns the set of resources on which the agent should be scheduled; for labels corresponding to variable duration resources, the function returns a function ρ from time to resource names, such that $\rho(t)$ gives the resource that should be used at time t:

$$\phi \in FSch \mathrel{\widehat=} (Time \times Lbl_{fixed} \to Resources) \cup (Time \times Lbl_{variable} \to Time \pfun RName).$$

For example, if the labelled assignment $L :: x := 3$ is encountered at time 7, with scheduler given by ϕ, and $\phi(7, L) = \{(9, r), (11, r)\}$, then the assignment will be scheduled on resource r at times 9 and 11.

We introduce two new syntactic classes of agents that must appear labelled: we write $\mathcal{B}$ for an agent that must be labelled with a label from Lbl_{fixed} and $\mathcal{C}$ for an agent that must be labelled with a label from $Lbl_{variable}$. We will also restrict our syntax so as to consider only parallel compositions of sequential agents, so we define a new syntactic class of sequential agents:

$$\begin{array}{lcl}
\mathcal{A} & \in & SeqTAM \\
\mathcal{A} & ::= & \mathsf{skip} \mid L_{fixed} :: \mathcal{B} \mid L_{variable} :: \mathcal{C} \mid [t]\mathcal{A} \mid \mathcal{A}\,;\mathcal{A} \mid \mathcal{A} \sqcap \mathcal{A}' \mid \mu\, X_n \bullet F(X) \\
\mathcal{B} & ::= & \Delta t \mid x := E \mid (x, y) \leftarrow s \mid x \to s \mid \bigsqcup_{i \in I}{}_t\, g_i \Rightarrow \mathcal{A}_i \\
\mathcal{C} & ::= & (x : T)\mathcal{A} \mid (s : T)\mathcal{A} \mid \mathcal{A} \rhd^s_t \mathcal{A},
\end{array}$$

where L_{fixed} ranges over Lbl_{fixed} and $L_{variable}$ ranges over $Lbl_{variable}$.

We will consider a *system* to be either a specification statement, a sequential agent with a scheduler, or a parallel composition of two sequential agents with a scheduler:

$$\begin{array}{lcl}
\mathcal{S} & \in & System \\
\mathcal{S} & ::= & w : \Phi \mid \mathcal{A} \mid \phi \mid (\mathcal{A} \mid \mathcal{A}) \mid \phi.
\end{array}$$

For simplicity, we only consider parallel compositions of two agents, but this can easily be extended to arbitrary (finite) parallel composition.

We will extend the TAM Logic Language in order to reason about resources. We will allow our predicates for sequential agents to include a distinguished free variable, R_{req}, representing the resources required by the agent. We will also add a boolean-valued variable fail which will be true precisely when an execution fails: this failure could be because (1) the agent tries to use a resource for two different purposes, or (2) a deadline is not met. For specifications and parallel compositions we will add another free variable, R_{av}, representing the resources available to the system. A parallel composition may fail if (1) either component fails, (2) there are insufficient resources available, or (3) both components try to use the same resource simultaneously.

We will define a semantic function $\mathcal{F}$ giving the representation of an agent in this model:

$$\mathcal{F} : System \to TAMLL.$$

Most of the semantic equations are simple extensions of the definitions in the previous section; in the following we concentrate upon resource and scheduling issues.

Our semantic definitions will ensure that all agents satisfy the following law, namely that resources are required only during the execution:

Law 3.1: $\mathcal{F}[\![\mathcal{A} \mid \phi]\!] \Rightarrow \text{Times}\, R_{req} \subseteq (t_\alpha, t_\omega]$. △

Our specification statement remains as before, except we insist that the agent does not fail:

$$\mathcal{F}[\![w : \Phi]\!] \mathrel{\widehat{=}} stable(W \setminus w, t_\alpha, t_\omega) \wedge \Phi \wedge \neg \mathsf{fail}.$$

The agent skip uses no resources:

$$\mathcal{F}[\![\mathsf{skip} \mid \phi]\!] \mathrel{\widehat{=}} t_\omega = t_\alpha \wedge R_{req} = \emptyset \wedge \neg \mathsf{fail}.$$

The agent $L :: \Delta t$ will delay for t seconds; if it is started at time t_α with scheduler ϕ then the delay is timed using the resources given by $\phi(t_\alpha, L)$.

$$\mathcal{F}[\![L :: \Delta t \mid \phi]\!] \mathrel{\widehat{=}} t_\omega = t_\alpha + t \wedge stable(W, t_\alpha, t_\omega) \wedge R_{req} = \phi(t_\alpha, L) \wedge \neg \mathsf{fail}.$$

The assignment $L :: x := E$ is scheduled using the resources in $R_{req} = \phi(t_\alpha, L)$. The agent will terminate when it has used all the resources, i.e. at time $\text{end}\, R_{req}$.

$$\begin{aligned}\mathcal{F}[\![L :: x := E \mid \phi]\!] \mathrel{\widehat{=}} & stable(W \setminus x, t_\alpha, t_\omega) \wedge x@t_\omega = E@t_\alpha \\ & \wedge R_{req} = \phi(t_\alpha, L) \wedge t_\omega = \text{end}\, R_{req} \wedge \neg \mathsf{fail}.\end{aligned}$$

The definitions for shunt reads and shunt writes are similar:

$$\begin{aligned}\mathcal{F}[\![L :: (x, y) \leftarrow s \mid \phi]\!] \mathrel{\widehat{=}} & stable(W \setminus \{x, y\}, t_\alpha, t_\omega) \\ & \wedge \exists\, t \in (t_\alpha, t_\omega] \bullet x@t_\omega = s@t.ts \wedge y@t = s@t.v \\ & \wedge R_{req} = \phi(t_\alpha, L) \wedge t_\omega = \text{end}\, R_{req} \wedge \neg \mathsf{fail} \\ \mathcal{F}[\![L :: x \to s \mid \phi]\!] \mathrel{\widehat{=}} & stable(W \setminus s, t_\alpha, t_\omega) \\ & \wedge \exists\, t \in (t_\alpha, t_\omega] \bullet stable(s, t_\alpha, t-1) \wedge s@t = (t, x@t_\alpha) \\ & \qquad \wedge stable(s, t, t_\omega) \\ & \wedge R_{req} = \phi(t_\alpha, L) \wedge t_\omega = \text{end}\, R_{req} \wedge \neg \mathsf{fail}.\end{aligned}$$

The agent $L :: (x : T)\mathcal{A}$ introduces a new variable x, which is held on resource $\phi(t_\alpha, L)(t)$ at all times t during the execution. If it tries to use this resource for another purpose then the agent fails.

$$\begin{aligned}\mathcal{F}[\![L :: (x : T)\mathcal{A} \mid \phi]\!] \mathrel{\widehat{=}} \exists\, v, v' : T\,;\mathsf{fail}' : \mathit{Bool}\,;\, R'_{req} : \mathit{Resources} \bullet \\ \text{let}\quad R_x = \{(t, \phi(t_\alpha, L)(t)) \mid t \in (t_\alpha, t_\omega]\}\quad \text{in} \\ \mathcal{F}[\![\mathcal{A} \mid \phi]\!][v/x@t_\alpha\,,\ v'/x@t_\omega\,,\ \mathsf{fail}'/\mathsf{fail}\,,\ R'_{req}/R_{req}] \\ \wedge\, R_{req} = R'_{req} \cup R_x \wedge (\mathsf{fail} \Leftrightarrow \mathsf{fail}' \vee R_x \cap R'_{req} \neq \emptyset).\end{aligned}$$

The definition for local shunts is very similar:

$$\begin{aligned}\mathcal{F}[\![L :: (s : T)\mathcal{A} \mid \phi]\!] \mathrel{\widehat{=}} \exists f : \mathit{Time} \nrightarrow (\mathit{Time} \times T)\,;\mathsf{fail}' : \mathit{Bool}\,;\, R'_{req} : \mathit{Resources} \bullet \\ \text{let}\quad R_s = \{(t, \phi(t_\alpha, L)(t)) \mid t \in (t_\alpha, t_\omega]\}\quad \text{in} \\ \mathcal{F}[\![\mathcal{A} \mid \phi]\!][f(t)/s@t \mid t \in [t_\alpha, t_\omega]][\mathsf{fail}'/\mathsf{fail}\,,\ R'_{req}/R_{req}] \\ \wedge\, f(t_\alpha).ts = t_\alpha \wedge R_{req} = R'_{req} \cup R_s \\ \wedge\, \mathsf{fail} \Leftrightarrow \mathsf{fail}' \vee R_s \cap R'_{req} \neq \emptyset.\end{aligned}$$

The agent $[t]\mathcal{A}$ fails if $\mathcal{A}$ fails or if the deadline t is passed:

$$\mathcal{F}[\![[t]\mathcal{A} \mid \phi]\!] \mathrel{\widehat{=}} \exists\, \mathsf{fail}' : \mathit{Bool} \bullet \mathcal{F}[\![\mathcal{A} \mid \phi]\!][\mathsf{fail}'/\mathsf{fail}] \wedge (\mathsf{fail} \Leftrightarrow \mathsf{fail}' \vee t_\omega - t_\alpha > t).$$

$\mathcal{A}_1\,;\mathcal{A}_2$ fails if either $\mathcal{A}_1$ or $\mathcal{A}_2$ fails; its resource requirements are the union of the resource requirements of the two components:

$$\begin{aligned}\mathcal{F}[\![\mathcal{A}_1\,;\mathcal{A}_2 \mid \phi]\!] \mathrel{\widehat{=}} \exists\, t \in [t_\alpha, t_\omega]\,;\, W' : V\,;\mathsf{fail}_1, \mathsf{fail}_2 : \mathit{Bool}\,;\, R_{req,1}, R_{req,2} : \mathit{Resources} \bullet \\ \mathcal{F}[\![\mathcal{A}_1 \mid \phi]\!][t/t_\omega\,,\ W'/W@t_\omega\,,\ \mathsf{fail}_1/\mathsf{fail}\,,\ R_{req,1}/R_{req}] \\ \wedge\, \mathcal{F}[\![\mathcal{A}_2 \mid \phi]\!][t/t_\alpha\,,\ W'/W@t_\alpha\,,\ \mathsf{fail}_2/\mathsf{fail}\,,\ R_{req,2}/R_{req}] \\ \wedge\, R_{req} = R_{req,1} \cup R_{req,2} \wedge (\mathsf{fail} \Leftrightarrow \mathsf{fail}_1 \vee \mathsf{fail}_2).\end{aligned}$$

As normal, nondeterministic choice is represented by disjunction:

$$\mathcal{F}[\![\mathcal{A} \sqcap \mathcal{A}' \mid \phi]\!] \mathrel{\widehat{=}} \mathcal{F}[\![\mathcal{A} \mid \phi]\!] \vee \mathcal{F}[\![\mathcal{A}' \mid \phi]\!].$$

The agent $L :: \bigsqcup_{i \in I}{}_t\, g_i \Rightarrow \mathcal{A}_i$ evaluates the guards using the resources of $\phi(t_\alpha, L)$; if the evaluation does not meet the deadline t, then the agent fails.

$$\begin{aligned}\mathcal{F}[\![L :: \textstyle\bigsqcup_{i \in I}{}_t\, g_i \Rightarrow \mathcal{A}_i \mid \phi]\!] \mathrel{\widehat{=}} \\ \text{let}\quad t'_\alpha = t_\alpha + t \quad\text{and}\quad J = \{i \in I \mid g_i@t_\alpha\} \quad\text{and}\quad R_g = \phi(t_\alpha, L)\quad \text{in} \\ \text{if}\quad J = \emptyset \\ \text{then}\quad t_\omega = t'_\alpha \wedge \mathit{stable}(W, t_\alpha, t_\omega) \wedge R_{req} = R_g \wedge (\mathsf{fail} \Leftrightarrow \mathrm{end}\, R_g > t_\alpha + t) \\ \text{else}\quad \mathit{stable}(W, t_\alpha, t'_\alpha) \\ \qquad \wedge\, \exists\, j \in J\,;\mathsf{fail}' : \mathit{Bool}\,;\, R'_{req} : \mathit{Resources} \bullet \\ \qquad\quad \mathcal{F}[\![\mathcal{A}_j \mid \phi]\!][t'_\alpha/t_\alpha\,,\ \mathsf{fail}'/\mathsf{fail}\,,\ R'_{req}/R_{req}] \wedge R_{req} = R_g \cup R'_{req} \\ \qquad\quad \wedge\, \mathsf{fail} \Leftrightarrow \mathsf{fail}' \vee \mathrm{end}\, R_g > t_\alpha + t.\end{aligned}$$

$L :: \mathcal{A} \rhd_l^s \mathcal{A}'$ monitors shunt s, using resource $\phi(t_\alpha, L)(t')$ at time t', until either it is written to or the timeout occurs:

$$
\begin{array}{l}
\mathcal{F}[\![L :: \mathcal{A} \rhd_t^s \mathcal{A}' \mid \phi]\!] \mathrel{\widehat{=}} \\
\quad \begin{array}{ll}
\text{let} & t'_\alpha = \inf\{t' > t_\alpha \mid s@t'.ts = t'\} \sqcap t_\alpha + t \\
\text{and} & R = \{(t', \phi(t_\alpha, L)(t')) \mid t' \in (t_\alpha, t'_\alpha]\} \quad \text{in} \\
\text{if} & \exists\, t' \in (t_\alpha, t_\alpha + t] \bullet s@t'.ts = t' \\
\text{then} & \mathit{stable}(W, t_\alpha, t'_\alpha) \\
 & \quad \wedge \exists\, R'_{req} : \mathit{Resources} \bullet \mathcal{F}[\![\mathcal{A}' \mid \phi]\!][t'_\alpha/t_\alpha\ ,\ R'_{req}/R_{req}] \wedge R_{req} = R \cup R'_{req} \\
\text{else} & \mathit{stable}(W, t_\alpha, t'_\alpha) \\
 & \quad \wedge \exists\, R'_{req} : \mathit{Resources} \bullet \mathcal{F}[\![\mathcal{A} \mid \phi]\!][t'_\alpha/t_\alpha\ ,\ R'_{req}/R_{req}] \wedge R_{req} = R \cup R'_{req}.
\end{array}
\end{array}
$$

Consider the parallel composition $\mathcal{A} \mid \mathcal{A}'$. Suppose $\mathcal{A}_1$ requires resources $R_{req,1}$ and $\mathcal{A}_2$ requires resources $R_{req,2}$; then the parallel composition will require $R_{req,1} \cup R_{req,2}$. The composition will fail if (1) either component fails, (2) it requires resources that are not available, or (3) both components try to access the same resource simultaneously.

$$
\begin{array}{rl}
\mathcal{F}[\![\mathcal{A}_1 \mid \mathcal{A}_2 \mid \phi]\!] \mathrel{\widehat{=}} \exists\, t_1, t_2 : \mathit{Time}\ ; \mathsf{fail}_1, \mathsf{fail}_2 : \mathit{Bool}\ ; R_{req,1}, R_{req,2} : \mathit{Resources} \bullet & \\
\quad \mathcal{F}[\![\mathcal{A}_1 \mid \phi]\!][t_1/t_\omega\ ,\ \mathsf{fail}_1/\mathsf{fail}\ ,\ R_{req,1}/R_{req}] & \\
\quad \wedge\ \mathcal{F}[\![\mathcal{A}_2 \mid \phi]\!][t_2/t_\omega\ ,\ \mathsf{fail}_2/\mathsf{fail}\ ,\ R_{req,2}/R_{req}] & \\
\quad \wedge\ t_\omega = t_1 \sqcup t_2 \wedge R_{req} = R_{req,1} \cup R_{req,2} & \\
\quad \wedge\ \mathsf{fail} \Leftrightarrow \mathsf{fail}_1 \vee \mathsf{fail}_2 \vee R_{req} \not\subseteq R_{av} \vee R_{req,1} \cap R_{req,2} \neq \emptyset. &
\end{array}
$$

As before, finite recursion is defined by

$$
\begin{array}{rcl}
\mu\, X_0 \bullet F(X) & \mathrel{\widehat{=}} & \mathsf{skip} \\
\mu\, X_{n+1} \bullet F(X) & \mathrel{\widehat{=}} & F(\mu\, X_n \bullet F(X)).
\end{array}
$$

We can define the obvious refinement relation between two systems:

Definition 3.2: $\mathcal{S} \sqsubseteq_{\mathcal{F}} \mathcal{S}'$ *iff* $\mathcal{F}[\![\mathcal{S}]\!] \Leftarrow \mathcal{F}[\![\mathcal{S}']\!]$. ♢

In order to relate our scheduling model to our basic model, we need to define a relation between the two syntaxes; we write $\mathcal{A}$ *labels* $\mathcal{A}'$ if $\mathcal{A}$ can be obtained from $\mathcal{A}'$ by adding labels.

The following lemma relates the scheduling model to the functionality model. If we know how the scheduler behaves then we know more about the behaviour of the whole system; however, we must now consider the possibility of failure.

Lemma 3.3: For all agents $\mathcal{A}$ and $\mathcal{A}'$ and schedulers ϕ, if $\mathcal{A}$ *labels* $\mathcal{A}'$ then

$$\mathsf{fail} \vee \mathcal{P}[\![\mathcal{A}']\!] \Leftarrow \mathcal{F}[\![\mathcal{A} \mid \phi]\!].$$

♡

This lemma can be proved by structural induction.

The following theorem will allow us to break down the development of a system into two stages, as in figure 1: in the first stage we can use the standard TAM refinement calculus to derive an initial design; we can then use the scheduling-oriented model to derive a correct scheduler.

Theorem 3.4: Suppose we are given a set of available resources R_{av}. If the variables R_{av} and R_{req} do not appear free in the predicate Φ, and agent $\mathcal{A}'$ meets specification Φ:

$$w : \Phi \sqsubseteq_{\mathcal{P}} \mathcal{A}';$$

and agent $\mathcal{A}$ can be obtained from $\mathcal{A}'$ by adding labels:

$$\mathcal{A} \; \mathit{labels} \; \mathcal{A}';$$

and the composition of $\mathcal{A}$ with the scheduler does not fail and meets specification Φ':

$$\mathcal{F}[\![\mathcal{A} \mid \phi]\!] \Rightarrow \neg \, \mathsf{fail} \wedge \Phi';$$

then the system meets the specification $\Phi \wedge \Phi'$:

$$w : \Phi \wedge \Phi' \sqsubseteq_{\mathcal{F}} \mathcal{A} \mid \phi$$

♡

Proof: We have:

$$\begin{array}{ll} & \mathcal{F}[\![A \mid \phi]\!] \\ \Rightarrow & \Big\langle \text{from the third assumption, and lemma 3.3} \Big\rangle \\ & \neg \, \mathsf{fail} \wedge \Phi' \wedge \mathcal{P}[\![\mathcal{A}']\!] \\ \Rightarrow & \Big\langle \text{from the first assumption} \Big\rangle \\ & \neg \, \mathsf{fail} \wedge \Phi' \wedge \mathit{stable}(\, W \setminus w, t_\alpha, t_\omega \,) \wedge \Phi \\ \equiv & \Big\langle \text{definition} \Big\rangle \\ & \mathcal{F}[\![w : \Phi \wedge \Phi']\!]. \end{array}$$

Hence $w : \Phi \wedge \Phi' \sqsubseteq_{\mathcal{F}} A \mid \phi$. □

Example

Recall in section 2 we considered the case of an agent inputting integers on channels *in* and in' and respectively outputting their factorials and squares on *out* and out'. We derived the solution:

$$\begin{array}{l} [6](x : \mathit{Time})(y : \mathbb{N})(z : \mathbb{N})\Big((x, y) \leftarrow \mathit{in} \, ; z := y! \, ; z \rightarrow \mathit{out} \Big) \\ | \\ [6](x : \mathit{Time})(y : \mathbb{N})(z : \mathbb{N})\Big((x, y) \leftarrow \mathit{in}' \, ; z := y^2 \, ; z \rightarrow \mathit{out}' \Big). \end{array}$$

In order to apply our scheduling model to this program, we add labels to obtain the program $\mathcal{A} \mid \mathcal{A}'$ where

$$\begin{aligned}\mathcal{A} \;\widehat{=}\; & [6]L_x :: (x : Time)L_y :: (y : \mathbb{N})L_z :: (z : \mathbb{N}) \\ & \Big(L_r :: (x, y) \leftarrow in \,;\, L_a :: z := y! \,;\, L_w :: z \to out\Big) \\ \mathcal{A}' \;\widehat{=}\; & [6]L'_x :: (x : Time)L'_y :: (y : \mathbb{N})L'_z :: (z : \mathbb{N}) \\ & \Big(L'_r :: (x, y) \leftarrow in' \,;\, L'_a :: z := y^2 \,;\, L'_w :: z \to out'\Big).\end{aligned}$$

We want to design a scheduler so that the resulting system will not fail. We consider two solutions: the first solution uses two CPUs, one for each side of the parallel composition; the second solution uses a single fast CPU.

First solution

We assume that we have sufficient resources for all the variables, and that we also have two CPUs, r and r':

$$R_{av} \supseteq (t_\alpha, t_\alpha + 6] \times \{r_x, r_y, r_z, r'_x, r'_y, r'_z, r, r'\}.$$

We require a scheduler ϕ such that

$$\{out, out'\} : \Phi \;\sqsubseteq_F\; \mathcal{A} \mid \mathcal{A}' \mid \phi.$$

By theorem 3.4, it will be enough to show that

$$\mathcal{F}[\![\mathcal{A} \mid \mathcal{A}' \mid \phi]\!] \Rightarrow \neg\, \mathsf{fail}.$$

Consider the scheduler that maps each of the variables to its own location:

$$\begin{array}{lll}\phi(t_\alpha, L_x)(t) = r_x & \phi(t_\alpha, L_y)(t) = r_y & \phi(t_\alpha, L_z)(t) = r_z \\ \phi(t_\alpha, L'_x)(t) = r'_x & \phi(t_\alpha, L'_y)(t) = r'_y & \phi(t_\alpha, L'_z)(t) = r'_z,\end{array}$$

and assigns the operations of $\mathcal{A}$ to CPU r, and the operations of $\mathcal{A}'$ to CPU r', where the shunt reads and writes take one second each, and each assignment takes two seconds:

$$\begin{array}{ll}\phi(t_\alpha, L_r) = \{(t_\alpha + 1\,, r)\} & \phi(t_\alpha, L'_r) = \{(t_\alpha + 1\,, r')\} \\ \phi(t_\alpha, L_a) = \{(t_\alpha + 1\,, r), (t_\alpha + 2\,, r)\} & \phi(t_\alpha, L'_a) = \{(t_\alpha + 1\,, r'), (t_\alpha + 2\,, r')\} \\ \phi(t_\alpha, L_w) = \{(t_\alpha + 1\,, r)\} & \phi(t_\alpha, L'_w) = \{(t_\alpha + 1\,, r')\}.\end{array}$$

First, let us consider $\mathcal{A}$. It is easily seen that

$$\begin{aligned}\mathcal{F}[\![(L_r :: (x, y) \leftarrow in \,;\, L_a :: z := y! \,;\, L_w :: z \to out) \mid \phi]\!] \Rightarrow \\ \Big(\neg\, \mathsf{fail} \wedge t_\omega = t_\alpha + 4 \;\wedge R_{req} = (t_\alpha, t_\omega] \times \{r\}\Big),\end{aligned}$$

so

$$\mathcal{F}[\![\mathcal{A} \mid \phi]\!] \;\Rightarrow\; \Big(\neg\, \mathsf{fail} \wedge R_{req} = (t_\alpha, t_\alpha + 4] \times \{r, r_x, r_y, r_z\}\Big).$$

Similarly

$$\mathcal{F}[\![\mathcal{A}' \mid \phi]\!] \;\Rightarrow\; \Big(\neg\, \mathsf{fail} \wedge R_{req} = (t_\alpha, t_\alpha + 4] \times \{r', r'_x, r'_y, r'_z\}\Big).$$

Combining these, using the semantic definition of parallel composition and the assumption about the resources available, we see that the system does not fail:

$$\mathcal{F}[\![\mathcal{A} \mid \mathcal{A}' \mid \phi]\!] \;\Rightarrow\; \neg\, \mathsf{fail}.$$

Second solution

We now consider an alternative solution. We assume that we have access to a fast CPU, r_{fast}, which can do the assignments in only one second. Our new resource assumption is

$$R_{av} \supseteq (t_\alpha, t_\alpha + 6] \times \{r_x, r_y, r_z, r'_x, r'_y, r'_z, r_{fast}\}.$$

And we now schedule both $\mathcal{A}$ and $\mathcal{A}'$ on r_{fast}; however we arrange that each operation except for $\mathcal{A}$'s shunt read initially idles for one second before starting:

$$\begin{array}{lll} \phi(t_\alpha, L_x)(t) = r_x & \phi(t_\alpha, L_y)(t) = r_y & \phi(t_\alpha, L_z)(t) = r_z \\ \phi(t_\alpha, L'_x)(t) = r'_x & \phi(t_\alpha, L'_y)(t) = r'_y & \phi(t_\alpha, L'_z)(t) = r'_z \end{array}$$

$$\begin{array}{ll} \phi(t_\alpha, L_r) = \{(t_\alpha + 1, r_{fast})\} & \phi(t_\alpha, L'_r) = \{(t_\alpha + 2, r_{fast})\} \\ \phi(t_\alpha, L_a) = \{(t_\alpha + 2, r_{fast})\} & \phi(t_\alpha, L'_a) = \{(t_\alpha + 2, r_{fast})\} \\ \phi(t_\alpha, L_w) = \{(t_\alpha + 2, r_{fast})\} & \phi(t_\alpha, L'_w) = \{(t_\alpha + 2, r_{fast})\}. \end{array}$$

Again, we consider $\mathcal{A}$ first. It is easily seen that

$$\mathcal{F}[\![\mathcal{A} \mid \phi]\!] \;\Rightarrow\; \Big(\neg\,\mathsf{fail} \wedge R_{req} = \{t_\alpha + 1, t_\alpha + 3, t_\alpha + 5\} \times \{r_{fast}\} \cup (t_\alpha, t_\alpha + 5] \times \{r_x, r_y, r_z\}\Big).$$

Similarly

$$\mathcal{F}[\![\mathcal{A}' \mid \phi]\!] \;\Rightarrow\; \Big(\neg\,\mathsf{fail} \wedge R_{req} = \{t_\alpha + 2, t_\alpha + 4, t_\alpha + 6\} \times \{r_{fast}\} \cup (t_\alpha, t_\alpha + 6] \times \{r'_x, r'_y, r'_z\}\Big).$$

Combining these, using the semantic definition of parallel composition and the assumption about the resources available, we see that the system does not fail:

$$\mathcal{F}[\![\mathcal{A} \mid \mathcal{A}' \mid \phi]\!] \;\Rightarrow\; \neg\,\mathsf{fail}.$$

4 A model with dynamic scheduling

In this section we extend our model so as to be able to reason about dynamic scheduling. We will allow the scheduling decisions to be a function of the labels encountered so far. We therefore need to keep track of the times at which labels were encountered, and so extend our predicate language to include a free variable χ, representing the labels or checkpoints encountered:

$$\chi \in \mathit{CheckPts} \mathrel{\widehat{=}} \mathbb{P}(\mathit{Time} \times \mathit{Lbl}).$$

We define a dynamic scheduler to be a function from the check points so far encountered to a fixed scheduler:

$$\psi \in \mathit{DSch} \mathrel{\widehat{=}} \mathit{CheckPts} \to \mathit{FSch}.$$

We are only going to consider *non-clairvoyant* schedulers.

Definition 4.1: We say that a scheduler ψ is *non-clairvoyant* iff, whenever two sets of checkpoints χ_1 and χ_2 agree up until time t,

$$\{(t', L') \in \chi_1 \mid t' \leqslant t\} = \{(t', L') \in \chi_2 \mid t' \leqslant t\},$$

scheduling decisions made before time t should be the same whether they are based on χ_1 or χ_2: when $\psi(\chi_1)$ and $\psi(\chi_2)$ are applied to fixed duration resource label L at time $t' \leqslant t$ the two sets of resources returned should agree up until time t:

$$\forall\, L \in Lbl_{fixed}\,;\, t' \leqslant t \;\bullet\; \{(t'', r) \in \psi(\chi_1)(t', L) \mid t'' \leqslant t\} = \{(t'', r) \in \psi(\chi_2)(t', L) \mid t'' \leqslant t\}\;;$$

and when they are applied to a variable duration resource label, the functions returned should agree up until time t:

$$\forall\, L \in Lbl_{variable}\,;\, t', t'' \leqslant t \;\bullet\; \psi(\chi_1)(t', L)(t'') = \psi(\chi_2)(t', L)(t'').$$

$\diamondsuit$

We will consider a system to be either a specification statement, a sequential agent with a fixed scheduler, or a parallel composition of sequential agents with a dynamic scheduler; for simplicity, we only consider parallel compositions of two agents, but this can easily be extended to arbitrary (finite) parallel composition.

$$\begin{array}{lcl} \mathcal{S} & \in & System \\ \mathcal{S} & ::= & w : \Phi \;\mid\; \mathcal{A} \mid \phi \;\mid\; (\mathcal{A} \mid \mathcal{A}) \mid \psi. \end{array}$$

We can define a semantic function for dynamic scheduling:

$$\mathcal{D} : System \to TAMLL.$$

Our specification statement takes the normal form:

$$\mathcal{D}[\![w : \Phi]\!] \mathrel{\widehat{=}} stable(W \setminus w, t_\alpha, t_\omega) \wedge \Phi \wedge \neg\, \mathsf{fail}.$$

For sequential agents, the semantic equations are simple adaptations of the previous ones, extended to include a record of the checkpoints encountered. For example, there are no checkpoints in the agent skip:

$$\mathcal{D}[\![\mathsf{skip} \mid \phi]\!] \mathrel{\widehat{=}} \mathcal{F}[\![\mathsf{skip} \mid \phi]\!] \wedge \chi = \emptyset.$$

The semantic equations for the labelled atomic agents most be adapted to record the checkpoints; for example:

$$\mathcal{D}[\![L :: \Delta t \mid \phi]\!] \mathrel{\widehat{=}} \mathcal{F}[\![L :: \Delta t \mid \phi]\!] \wedge \chi = \{(t_\alpha, L)\}.$$

The checkpoints in $\mathcal{A}_1\,;\mathcal{A}_2$ are simply the union of the checkpoints in $\mathcal{A}_1$ and $\mathcal{A}_2$:

$$\begin{array}{l} \mathcal{D}[\![\mathcal{A}_1\,;\mathcal{A}_2 \mid \phi]\!] \mathrel{\widehat{=}} \\ \quad \exists\, t \in [t_\alpha, t_\omega]\,;\, X' : V\,;\, \mathsf{fail}_1, \mathsf{fail}_2 : Bool\,;\, R_{req,1}, R_{req,2} : Resources\,;\, \chi_1, \chi_2 : CheckPts \;\bullet \\ \qquad \mathcal{D}[\![\mathcal{A}_1 \mid \phi]\!][t/t_\omega\;,\; X'/X@t_\omega\;,\; \mathsf{fail}_1/\mathsf{fail}\;,\; R_{req,1}/R_{req}\;,\; \chi_1/\chi] \\ \qquad \wedge\; \mathcal{D}[\![\mathcal{A}_2 \mid \phi]\!][t/t_\alpha\;,\; X'/X@t_\alpha\;,\; \mathsf{fail}_2/\mathsf{fail}\;,\; R_{req,2}/R_{req}\;,\; \chi_2/\chi] \\ \qquad \wedge\; R_{req} = R_{req,1} \cup R_{req,2} \wedge (\mathsf{fail} \Leftrightarrow \mathsf{fail}_1 \vee \mathsf{fail}_2) \wedge \chi = \chi_1 \cup \chi_2. \end{array}$$

The definition of parallel composition is a simple adaptation of the definition in the model for fixed scheduling:

$$\begin{aligned}
&\mathcal{D}[\![(\mathcal{A}_1 \mid \mathcal{A}_2) \mid \psi]\!] \mathrel{\widehat{=}} \\
&\quad \exists\, t_1, t_2 : \mathit{Time}\,;\, \mathsf{fail}_1, \mathsf{fail}_2 : \mathit{Bool}\,;\, R_{req,1}, R_{req,2} : \mathit{Resources}\,;\, \chi_1, \chi_2 : \mathit{CheckPts} \bullet \\
&\qquad \mathcal{D}[\![\mathcal{A}_1 \mid \psi(\chi)]\!][t_1/t_\omega\,,\, \mathsf{fail}_1/\mathsf{fail}\,,\, R_{req,1}/R_{req}\,,\, \chi_1/\chi] \\
&\qquad \wedge\, \mathcal{D}[\![\mathcal{A}_2 \mid \psi(\chi)]\!][t_2/t_\omega\,,\, \mathsf{fail}_2/\mathsf{fail}\,,\, R_{req,2}/R_{req}\,,\, \chi_2/\chi] \\
&\qquad \wedge\, t_\omega = t_1 \sqcup t_2 \wedge R_{req} = R_{req,1} \cup R_{req,2} \wedge \chi = \chi_1 \cup \chi_2 \\
&\qquad \wedge\, \mathsf{fail} \Leftrightarrow \mathsf{fail}_1 \vee \mathsf{fail}_2 \vee R_{req} \not\subseteq R_{av} \vee R_{req,1} \cap R_{req,2} \neq \emptyset.
\end{aligned}$$

$\mathcal{A}_1$ and $\mathcal{A}_2$ are scheduled according to $\psi(\chi)$.

The following definition, lemma and theorem are analogous to definition 3.2, lemma 3.3 and theorem 3.4.

Definition 4.2: For all systems $\mathcal{S}$ and $\mathcal{S}'$:

$$\mathcal{S} \sqsubseteq_{\mathcal{D}} \mathcal{S}' \;\; \textit{iff} \;\; \mathcal{D}[\![\mathcal{S}]\!] \Leftarrow \mathcal{D}[\![\mathcal{S}']\!].$$

♢

Lemma 4.3: For all agents $\mathcal{A}_1 \mid \mathcal{A}_2$ and $\mathcal{A}'_1 \mid \mathcal{A}'_2$ and schedulers ψ, if $\mathcal{A}_1 \mid \mathcal{A}_2$ *labels* $\mathcal{A}'_1 \mid \mathcal{A}'_2$ then:

$$\mathsf{fail} \vee \mathcal{P}[\![\mathcal{A}'_1 \mid \mathcal{A}'_2]\!] \Leftarrow \mathcal{D}[\![\mathcal{A}_1 \mid \mathcal{A}_2 \mid \psi]\!].$$

♡

Theorem 4.4: Suppose we are given a set of available resources R_{av}. If the variables R_{av} and R_{req} do not appear free in the predicate Φ, and agent $\mathcal{A}'_1 \mid \mathcal{A}'_2$ meets specification Φ:

$$w : \Phi \sqsubseteq_{\mathcal{P}} \mathcal{A}'_1 \mid \mathcal{A}'_2;$$

and agent $\mathcal{A}_1 \mid \mathcal{A}_2$ can be obtained from $\mathcal{A}'_1 \mid \mathcal{A}'_2$ by adding labels:

$$\mathcal{A}_1 \mid \mathcal{A}_2 \textit{ labels } \mathcal{A}'_1 \mid \mathcal{A}'_2;$$

and the composition of $\mathcal{A}_1 \mid \mathcal{A}_2$ with the scheduler does not fail and meets specification Φ':

$$\mathcal{D}[\![\mathcal{A}_1 \mid \mathcal{A}_2 \mid \psi]\!] \Rightarrow \neg\, \mathit{fail} \wedge \Phi';$$

then the system meets the specification $\Phi \wedge \Phi'$:

$$w : \Phi \wedge \Phi' \sqsubseteq_{\mathcal{D}} \mathcal{A}_1 \mid \mathcal{A}_2 \mid \psi$$

♡

Example

We illustrate our model for dynamic scheduling by applying it to a slight variation of our running example. Suppose now that we are told that the environment will write to *in* within three seconds, and this value will then remain stable for the following three seconds; again the factorial of this value must be written onto *out*, but now, for reasons of freshness, the value should be output within three seconds of the write.

Our requirement can be formalized as:

$$
\begin{array}{l}
\{out, out'\} : \Phi \\
\text{where} \\
\Phi \mathrel{\widehat{=}} t_\omega \leqslant t_\alpha + 6 \\
\qquad \wedge \forall\, t \in (t_\alpha, t_\alpha + 3] \bullet \left(\begin{array}{l} in@t.ts = t \wedge stable(in, t, t+3) \Rightarrow \\ \qquad \exists\, t' \in (t, t+3] \bullet out@t'.v = (in@t.v)! \end{array} \right) \\
\qquad \wedge \exists\, t \in (t_\alpha, t_\omega) \bullet out'@t_\omega.v = (in'@t.v)^2 .
\end{array}
$$

We use the functionality model to derive an initial design. We can refine our specification to

$$
\begin{array}{l}
\{out\} : \Phi_1 \mid \{out'\} : \Phi_2 \\
\text{where} \;\; \Phi_1 \;\widehat{=}\; t_\omega \leqslant t_\alpha + 6 \\
\qquad\qquad \wedge \forall\, t \in (t_\alpha, t_\alpha + 3] \bullet \left(\begin{array}{l} in@t.ts = t \wedge stable(in, t, t+3) \Rightarrow \\ \qquad t_\omega \leqslant t + 3 \wedge out@t_\omega.v = (in@t.v)! \end{array} \right) \\
\qquad \Phi_2 \;\widehat{=}\; t_\omega \leqslant t_\alpha + 6 \wedge \exists\, t \in (t_\alpha, t_\omega) \bullet out'@t_\omega.v = (in'@t.v)^2 .
\end{array}
$$

As in section 2, we refine the right hand side to

$$
\mathcal{A}' \mathrel{\widehat{=}} [6](x : Time)(y : \mathbb{N})(z : \mathbb{N})\Big((x, y) \leftarrow in' \,;\, z := y^2 \,;\, z \rightarrow out' \Big).
$$

Continuing with the left hand side,

$$
\begin{array}{ll}
 & \{out\} : \Phi_1 \\
\sqsubseteq & \Big\langle \text{strengthen predicate} \Big\rangle \\
 & \{out\} : \left(\begin{array}{l} stable(in, t_\alpha, t_\alpha + 3) \\ \vee \\ \exists\, t \in (t_\alpha, t_\alpha + 3] \bullet \\ \qquad stable(in, t_\alpha, t - 1) \wedge in@t.ts = t \\ \qquad \wedge \Big(t_\omega \leqslant t_\alpha + 3 \wedge \exists\, t' \in (t_\alpha, t_\alpha + 3] \bullet out@t_\omega.v = (in@t_\alpha.v)! \Big)[t/t_\alpha] \end{array} \right) \\
\sqsubseteq & \Big\langle \text{definition of timeout} \Big\rangle \\
 & \mathsf{skip} \rhd_3^{in} \{out\} : \Big(t_\omega \leqslant t_\alpha + 3 \wedge \exists\, t' \in (t_\alpha, t_\alpha + 3] \bullet out@t_\omega.v = (in@t_\alpha.v)! \Big) \\
\sqsubseteq & \Big\langle \text{as in section 2} \Big\rangle \\
 & \mathsf{skip} \rhd_3^{in} \Big([3](x : Time)(y : \mathbb{N})(z : \mathbb{N})(x, y) \leftarrow in \,;\, z := y! \,;\, z \rightarrow out \Big).
\end{array}
$$

We can now use theorem 4.4. We add labels to our initial design to obtain $\mathcal{A} \mid \mathcal{A}'$ where:

$$
\mathcal{A} \;\widehat{=}\; L_s :: \left(\mathsf{skip} \rhd_3^{in} \left(\begin{array}{l} [3] L_x :: (x : Time) L_y :: (y : \mathbb{N}) L_z :: (z : \mathbb{N}) \\ \qquad L_r :: (x, y) \leftarrow in \,;\, L_a :: z := y! \,;\, L_w :: z \rightarrow out \end{array} \right) \right)
$$

$$\mathcal{A}' \;\widehat{=}\; [6]L'_x :: (x : Time)L'_y :: (y : \mathbb{N})L'_z :: (z : \mathbb{N}) \\ \Big(L'_r :: (x, y) \leftarrow in' \,;\, L'_a :: z := y^2 \,;\, L'_w :: z \rightarrow out'\Big).$$

If we can produce a dynamic scheduler χ such that $\mathcal{D}[\![\mathcal{A} \mid \mathcal{A}' \mid \chi]\!] \Rightarrow \neg\ \mathsf{fail}$, then we can deduce that $\{out, out'\} : \Phi \sqsubseteq_{\mathcal{D}} \mathcal{A} \mid \mathcal{A}' \mid \chi$.

As before, we assume that we have enough memory locations for all the variables, and we have a CPU r_{fast} capable of doing each shunt read, assignment or shunt write in one second; we also assume that we have a special resource r_s capable of monitoring the shunt *in*:

$$R_{av} \supseteq (t_\alpha, t_\alpha + 6] \times \{r_x, r_y, r_z, r'_x, r'_y, r'_z, r_{fast}, r_s\}.$$

As before, each of the variables is allocated its own location:

$$\begin{array}{lll} \psi(\chi)(t_\alpha, L_x)(t) = r_x & \psi(\chi)(t_\alpha, L_y)(t) = r_y & \psi(\chi)(t_\alpha, L_z)(t) = r_z \\ \psi(\chi)(t_\alpha, L'_x)(t) = r'_x & \psi(\chi)(t_\alpha, L'_y)(t) = r'_y & \psi(\chi)(t_\alpha, L'_z)(t) = r'_z, \end{array}$$

and we monitor the shunt *in* on r_s:

$$\psi(\chi)(t_\alpha, L_s)(t) = r_s.$$

We want to define a scheduler that will give priority to the operations labelled by L_a and L_w, in that none of the operations labelled by L'_r, L'_a or L'_w will be scheduled once the label L_a has been encountered until the shunt write labelled by L_w has been completed: if label L_a is encountered at time t_a then the operations labelled by L'_r, L'_a and L'_w are delayed until time $t_a + 4$.

$$\begin{array}{rcl} \psi(\chi)(t, L_r) & = & \{(t+1, r_{fast})\} \\ \psi(\chi)(t, L_a) & = & \{(t+1, r_{fast})\} \\ \psi(\chi)(t, L_w) & = & \{(t+1, r_{fast})\} \\ \psi(\chi)(t, L'_r) & = & \begin{cases} \{(t_a+4, r_{fast})\}, & \text{if } (t_a, L_a) \in \chi \wedge t_a \leqslant t \leqslant t_a + 3 \\ \{(t+1, r_{fast})\}, & \text{otherwise} \end{cases} \\ \psi(\chi)(t, L'_a) & = & \begin{cases} \{(t_a+4, r_{fast})\}, & \text{if } (t_a, L_a) \in \chi \wedge t_a \leqslant t \leqslant t_a + 3 \\ \{(t+1, r_{fast})\}, & \text{otherwise} \end{cases} \\ \psi(\chi)(t, L'_w) & = & \begin{cases} \{(t_a+4, r_{fast})\}, & \text{if } (t_a, L_a) \in \chi \wedge t_a \leqslant t \leqslant t_a + 3 \\ \{(t+1, r_{fast})\}, & \text{otherwise.} \end{cases} \end{array}$$

It is easily seen that if $\chi' : \mathit{CheckPts}$ restricted to the labels of $\mathcal{A}$ equals χ, then

$$\begin{array}{l} \mathcal{D}[\![\mathcal{A} \mid \psi(\chi')]\!] \Rightarrow R_{req} \subseteq (t_\alpha, t_\alpha + 6] \times \{r_{fast}, r_x, r_y, r_z, r_s\} \\ \qquad \wedge\ (\forall\, t_a \bullet (t_a, L_a) \notin \chi) \Rightarrow R_{req} \upharpoonright r_{fast} = \emptyset \\ \qquad \wedge\ (t_a, L_a) \in \chi \Rightarrow t_\alpha < t_a \leqslant t_\alpha + 3 \wedge R_{req} \upharpoonright r_{fast} = (t_a, t_a + 3] \times \{r_{fast}\} \\ \qquad \wedge\ \neg\ \mathsf{fail}, \end{array}$$

i.e., the agent uses only the resources r_{fast}, r_x, r_y, r_z and r_s during $(t_\alpha, t_\alpha + 6]$; if it doesn't encounter label L_a then it doesn't use r_{fast}; if it encounters label L_a at time t_a then this is

between t_α and $t_\alpha + 3$, and it then uses R_{req} for precisely the next three seconds; and the agent does not fail.

Similarly,

$$\begin{aligned}\mathcal{D}[\![\mathcal{A}' \mid \psi(\chi')]\!] \Rightarrow\ & R_{req} \subseteq (t_\alpha, t_\alpha + 6] \times \{r_{fast}, r'_x, r'_y, r'_z\} \\ & \wedge\ \#(R_{req} \restriction r_{fast}) = 3 \\ & \wedge\ (t_a, L_a) \in \chi' \Rightarrow R_{req} \restriction r_{fast} \uparrow (t_a, t_a + 3] = \emptyset \\ & \wedge\ \neg\, \mathsf{fail},\end{aligned}$$

i.e., the agent uses only the resources r_{fast}, r'_x, r'_y and r'_z during $(t_\alpha, t_\alpha + 6]$; it uses three seconds of r_{fast}; if label L_a is encountered at time t_a then it does not uses the resource during the next three seconds; and the agent does not fail.

From the definition of parallel composition we know that for some $t_1, t_2 : \mathit{Time}$, $\mathsf{fail}_1, \mathsf{fail}_2 : \mathit{Bool}$, $R_{req,1}, R_{req,2} : \mathit{Resources}$, $\chi_1, \chi_2 : \mathit{CheckPts}$, we have

$$\begin{aligned}& \mathcal{D}[\![\mathcal{A} \mid \psi(\chi)]\!][t_1/t_\omega\ ,\ \mathsf{fail}_1/\mathsf{fail}\ ,\ R_{req,1}/R_{req}\ ,\ \chi_1/\chi] \\ & \wedge\ \mathcal{D}[\![\mathcal{A}' \mid \psi(\chi)]\!][t_2/t_\omega\ ,\ \mathsf{fail}_2/\mathsf{fail}\ ,\ R_{req,2}/R_{req}\ ,\ \chi_2/\chi] \\ & \wedge\ t_\omega = t_1 \sqcup t_2 \wedge R_{req} = R_{req,1} \cup R_{req,2} \wedge \chi = \chi_1 \cup \chi_2 \\ & \wedge\ \mathsf{fail} \Leftrightarrow \mathsf{fail}_1 \vee \mathsf{fail}_2 \vee R_{req} \not\subseteq R_{av} \vee R_{req,1} \cap R_{req,2} \neq \emptyset.\end{aligned}$$

Now, from the equations for $\mathcal{A}$ and $\mathcal{A}'$ we know that neither agent fails, and that they only uses resources that are available. Therefore, in order to deduce that the composition does not fail, we only have to show that they do not try to access the same resource simultaneously. For resources other than r_{fast}, this is trivial. If $(t_a, L_a) \notin \chi$ for all t_a then $\mathcal{A}$ does not use r_{fast}, and the result follows. If $(t_a, L_a) \in \chi$ then $R_{req,1} \restriction r_{fast} = (t_a, t_a + 3] \times \{r_{fast}\}$ and $R_{req,2} \restriction r_{fast} \uparrow (t_a, t_a + 3] = \emptyset$, so again the result follows.

5 Conclusions

In this paper we have developed a hierarchy of models for reasoning about allocation of resources and scheduling in real-time distributed systems. We have represented agents by predicates describing their possible behaviours. In our first model the predicates only talked about the agents' functionality; in subsequent models we extended this to talk about resource allocation and scheduling. In section 3 we represented a scheduler as a function that, given a label and the time at which the label is encountered, returns the resources upon which the agent should be scheduled. In section 4 we extended this to reason about dynamic schedulers, where the scheduling decisions could be a function of the checkpoints encountered so far elsewhere in the system.

Future work

I would like to apply the model presented in this paper to a reasonably large case study, so as to see how easily it can be applied to non-trivial examples. It may be that such a case study will reveal ways in which the model should be changed slightly to make it easier to use.

The work in this paper has presented a fairly basic model for scheduling. However, the model is easily adapted to cover many other aspects of scheduling theory. In section 4 we defined a

dynamic scheduler to be a function from the checkpoints encountered to a fixed scheduler:

$$\psi \in DSch \mathrel{\widehat=} CheckPts \to FSch.$$

Alternatively, it may be possible to define a dynamic scheduler as a *relation* between checkpoints and fixed schedulers:

$$\psi \in DSch \mathrel{\widehat=} CheckPts \times FSch,$$

and for agents to be scheduled according to *some* fixed scheduler ϕ related to the checkpoints:

$$\mathcal{D}[\![(\mathcal{A}_1 \mid \mathcal{A}_2) \mid \psi]\!] \mathrel{\widehat=} \exists \phi : FSch ; \ldots \bullet \mathcal{D}[\![\mathcal{A}_1 \mid \phi]\!][\ldots] \wedge \mathcal{D}[\![\mathcal{A}_2 \mid \phi]\!][\ldots] \wedge \ldots \wedge (\chi, \phi) \in \psi.$$

This would allow us to not fully specify our schedulers, thus giving us the normal advantages of nondeterminism. It would also open up the possibility of developing a refinement calculus for schedulers, where refining a scheduler corresponds to making it more deterministic.

We have allowed our dynamic schedulers to base their scheduling decisions upon the checkpoints encountered; we could also allow scheduling decisions to depend upon the state of variables throughout the execution. For example, if we decided that we wanted our scheduling decisions to depend also upon the value of the variable $x : T$, we could extend our predicate language to include a variable $\mathsf{vals} : Values \mathrel{\widehat=} \mathbb{P}(Time \times T)$ which records the values held by x. For example

$$\begin{aligned}
\mathcal{D}[\![L :: x := E \mid \phi]\!] &\mathrel{\widehat=} \ldots \wedge \mathsf{vals} = \{(t_\alpha, x@t_\alpha), (t_\omega, x@t_\omega)\} \\
\mathcal{D}[\![\mathcal{A}_1 ; \mathcal{A}_2 \mid \phi]\!] &\mathrel{\widehat=} \exists \mathsf{vals}_1, \mathsf{vals}_2 : Values ; \ldots \bullet \\
&\qquad \mathcal{D}[\![\mathcal{A}_1 \mid \phi]\!][\mathsf{vals}_1/\mathsf{vals}, \ldots] \wedge \mathcal{D}[\![\mathcal{A}_2 \mid \phi]\!][\mathsf{vals}_1/\mathsf{vals}, \ldots] \\
&\qquad \wedge \ldots \wedge \mathsf{vals} = \mathsf{vals}_1 \cup \mathsf{vals}_2.
\end{aligned}$$

And define a dynamic scheduler to be a function from checkpoints and values to fixed schedulers:

$$\begin{aligned}
\psi \in DSch &\mathrel{\widehat=} CheckPts \times Values \to FSch \\
\mathcal{D}[\![(\mathcal{A}_1 \mid \mathcal{A}_2) \mid \psi]\!] &\mathrel{\widehat=} \exists \mathsf{vals}_1, \mathsf{vals}_2 : Values ; \ldots \bullet \mathcal{D}[\![\mathcal{A}_1 \mid \psi(\chi, \mathsf{vals})]\!][\mathsf{vals}_1/\mathsf{vals}, \ldots] \\
&\qquad \wedge \mathcal{D}[\![\mathcal{A}_2 \mid \psi(\chi, \mathsf{vals})]\!][\mathsf{vals}_1/\mathsf{vals}, \ldots] \\
&\qquad \wedge \ldots \wedge \mathsf{vals} = \mathsf{vals}_1 \cup \mathsf{vals}_2.
\end{aligned}$$

Many schedulers use the concept of priorities. It is possible to code up priorities in our model: for example, in the example of section 4 we gave $\mathcal{A}$ higher priority that $\mathcal{A}'$ by scheduling $\mathcal{A}'$ only when $\mathcal{A}$ was not trying to execute. However, it would be useful to introduce priorities at a more primitive level, and so make reasoning about such schedulers simpler.

The deadlines in this paper have all been *hard*, in the sense that failure to meet a deadline is considered catastrophic. Alternatively, we could consider *soft* deadlines, where some penalties are imposed for failure to meet a deadline; the model should tell us how great the penalty would be for a given scheduler. We could do this by allowing our predicates to include a variable, $\mathsf{penalty}$, representing the value of the penalty; for example, if we wanted our penalty to be the sum of the amounts by which deadlines were missed, the semantic equation for the deadline operator would be:

$$\begin{aligned}
\mathcal{F}[\![[t]\mathcal{A} \mid \phi]\!] \mathrel{\widehat=} \exists \mathsf{penalty}' : \mathbb{N} \bullet \mathcal{F}[\![\mathcal{A} \mid \phi]\!][\mathsf{penalty}'/\mathsf{penalty}] \\
\wedge\ \mathsf{penalty} = \mathsf{penalty}' + (t_\omega - t_\alpha - t \sqcup 0).
\end{aligned}$$

The term $t_\omega - t_\alpha - t \sqcup 0$ represents the amount by which this deadline is missed.

In order to make reasoning about scheduling easier, it would be useful to develop some sort of tool support. One possibility would be to use the FDR tool [For93, Ros94]. This tool takes two CSP processes, a specification and an implementation, and tests whether the implementation refines the specification (in the failures-divergences model). The idea would be to code up both the TAM agents and the scheduler as CSP processes; both would include a special event *fail*: an agent should perform *fail* if it fails to meet a deadline; the scheduler should perform *fail* if more than one agent tries to use the same resource simultaneously; we could then ensure that the system does not fail by testing whether it refines the most nondeterministic process that never performs *fail*.

References

[CH88] R. Cleaveland and M. Hennessy. Priorities in process algebras. In *Proc. 3rd Symposium on Logic in Computer Science, Edinburgh*, 1988.

[DRRS94] J. Davies, G. M. Reed, A. W. Roscoe, and S. A. Schneider. *Real Time CSP.* Prentice Hall, 1994. Forthcoming.

[DS92] Jim Davies and Steve Schneider. A brief history of Timed CSP. Technical Monograph 96, Oxford University Computing Labs, 1992.

[For93] Formal Systems (Europe) Ltd. *Failures Divergence Refinement—User Manual and Tutorial*, 1993. Version 1.3.

[GL89] R. Gerber and I. Lee. Communicating shared resources: A model for distributed real-time systems. In *IEEE Real-Time Systems Symposium*, 1989.

[GL90] R. Gerber and I. Lee. A calculus for communicating shared resources. In *Concur '90, LNCS 458.* Springer Verlag, 1990.

[GL92] R. Gerber and I. Lee. Specification and analysis of resource-bound real-time systems. In *Real-Time: Theory in Practice, LNCS 600*, pages 371–396. Springer Verlag, 1992.

[HR90] M. Hennessy and T. Regan. A temporal process algebra. Technical Report 2-90, University of Sussex, 1990.

[Mil89] Robin Milner. *Communication and Concurrency.* Prentice Hall International, 1989.

[MT90] F. Moller and C. Tofts. A temporal calculus of communicating systems. In *LNCS 458, Concur '90.* Springer Verlag, 1990.

[Ros94] A. W. Roscoe. Model-checking CSP. In *A Classical Mind, Essays in Honour of C. A. R. Hoare.* Prentice-Hall, 1994.

[RR86] G. M. Reed and A. W. Roscoe. A timed model for CSP. In *Theoretical Computer Science 58*, 1986.

[SZ92] D. Scholefield and H. Zedan. The refinement of real-time systems. In *Fourth European Workshop on Real-Time Systems*. Athens, Greece, IEEE Computer Society Press, June 1992.

[SZ93] D. Scholefield and H. Zedan. A proof system for real-time programs. Technical Report YCS 210, University of York, 1993.

[SZJ93] D. Scholefield, H. Zedan, and He Jifeng. A specification oriented semantics for the refinement of real-time systems. Technical Report YCS 192, University of York, 1993.

[Yi90] Wang Yi. Real-time behaviour of asynchronous agents. In *Proceedings of Concur '90, LNCS 458*. Springer Verlag, 1990.