Programming Research Group

INFINITE BEHAVIOURS IN THE TEMPORAL AGENT MODEL

Gavin Lowe

PRG-TR-3-95

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

Infinite Behaviours in the Temporal Agent Model[1]

Gavin Lowe

February 1995

Abstract

In this paper we extend the Temporal Agent Model [SZJ94] so as to be able to reason about infinite agents. We augment the syntax with an infinite iteration operator, and extend the semantic model so as to allow reasoning about non-terminating executions.

1 Introduction

The Temporal Agent Model (TAM) [SZJ94] is a wide spectrum language for modelling real-time, distributed computer systems. The TAM language includes most of the standard features of a real-time programming language, such as parallel composition, deadlines and communication. A semantic model represents agents by a predicate describing their behaviour. The language also includes a specification statement; a number of algebraic laws allow such a specification to be refined into concrete TAM via provably correct transformations.

One short-coming of the existing model is that it considers only *finite* agents, i.e. agents that are guaranteed to terminate within a finite amount of time. However, many real-time agents, such as operating systems or controllers of industrial processes, are not expected to terminate. We address this short-coming in this paper: we augment the syntax with an infinite iteration operator, and extend the semantic model so as to be able to reason about the resulting infinite executions.

The existing model uses a finite iteration operator, μ_n: the agent $\mu_n \mathcal{A}$ executes $\mathcal{A}$ n times. In previous papers, the lack of infinite agents has been handled by considering such agents for "sufficiently large" n. For example, in [Low94b] and [LZ95] we had a requirement that was supposed to hold for ever. We specified that the requirement should hold up until some time T, and then refined this specification into several parallel agents of the form $\mu_n \mathcal{A}$ (preceded by some initialization), where n is some function of T. We then argued that replacing the finite iteration by an infinite one would create an agent that would meet the requirement for ever.

[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: +44-1865-273837, e-mail: `library@comlab.ox.ac.uk`) or by anonymous ftp from `ftp.comlab.ox.ac.uk`, filename `pub/Documents/techpapers/Gavin.Lowe/infinite.ps.gz`.

The disadvantage with this approach—other than its lack of elegance—is that it makes calculations harder: one has to deal explicitly with the termination time of the agent even though we know that it will eventually be implemented as a non-terminating agent. Further, as we will see, the refinement law for parallel composition can be considerably simplified if we know that both sides of the parallel composition will not terminate.

2 The Temporal Agent Model

A TAM program is a collection of parallel agents, each with their own set of variables. Agents communicate via a form of asynchronous channel called a *shunt*; a shunt can be thought of as a location in shared memory, holding a pair: 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 (finite) TAM language is defined by the BNF expression:

$$\begin{array}{rcl} \mathcal{A} & ::= & w : \Phi \mid \mathsf{skip} \mid \Delta t \mid x := e \mid x \leftarrow s \mid e \rightarrow s \mid \\ & & (x : T)\mathcal{A} \mid (s : T)\mathcal{A} \mid [t]\mathcal{A} \mid \mathcal{A} \,;\, \mathcal{A} \mid \mathcal{A} \sqcap \mathcal{A} \mid \\ & & \bigsqcup_{t \atop i \in I} g_i \Rightarrow \mathcal{A}_i \mid \mathcal{A} \rhd^s_t \mathcal{A} \mid \mathcal{A} \mid \mathcal{A} \mid \mu_n \mathcal{A} . \end{array}$$

where x and y are variables, w is a set of variables and shunts, 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, I is a finite indexing set, g_i is a boolean expression on variables, and n is a natural number.

$w : \Phi$ is a specification statement: it specifies that only the variables and shunts in the frame w may be changed by the program, and that all executions must satisfy the predicate Φ. In [SZ92, SZJ94], refinement rules are given that allow a specification in this form to be refined into concrete TAM.

The agent skip terminates immediately, whereas Δt terminates after t time units. The agent $x := e$ evaluates e, storing the result in variable x. The agent $x \leftarrow s$ reads shunt s, storing the result in x. $e \rightarrow s$ evaluates e, and writes the result to shunt s, time-stamping it with the time of the write.

$(x : T)\mathcal{A}$ declares x to be a local variable of type T within $\mathcal{A}$: its initial value is chosen nondeterministically. $(s : T)\mathcal{A}$ declares s to be a local shunt of type $\mathit{Time} \times T$ within $\mathcal{A}$: its initial value is chosen nondeterministically, but it is initially time-stamped with the time at which it is created.

Agents may be given a specific duration using the construct $[t]\,\mathcal{A}$, which executes $\mathcal{A}$ with the proviso that the execution must last t time units. If $\mathcal{A}$ terminates before t time units then the agent idles to fill the interval; if the execution fails to meet the deadline then we consider the agent to have failed.

$\mathcal{A} \,;\, \mathcal{A}'$ represents a sequential composition, with the second agent being started at the termination time of the first. $\mathcal{A} \sqcap \mathcal{A}'$ performs a nondeterministic choice between $\mathcal{A}$ and $\mathcal{A}'$.

Alternation is modelled by the agent $\bigsqcup_{t \atop i \in I} g_i \Rightarrow \mathcal{A}_i$: each of the guards g_i is evaluated, and if some are true a corresponding $\mathcal{A}_i$ is executed; if all the g_i

evaluate to false then the agent simply terminates. Evaluation of the guards should take precisely t time units; if necessary the agent should idle to fill this interval. $\mathcal{A} \rhd^s_t \mathcal{A}'$ monitors shunt s for t time units: if a write occurs within this time, then it executes $\mathcal{A}'$; otherwise it times-out and executes $\mathcal{A}$.

The agent $\mathcal{A} \mid \mathcal{A}'$ executes $\mathcal{A}$ and $\mathcal{A}'$ in parallel, terminating when both agents have terminated. The agent $\mu_n \bullet \mathcal{A}$ represents a finite iteration, executing $\mathcal{A}$ n times; in the next section we will add an infinite iteration, written as $\mu\,\mathcal{A}$.

In [SZJ94] a semantics was given for this language, representing an agent 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. Φ may also include terms of the form $s@t$ to refer to the contents of 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.

We introduce a macro which will prove useful; it specifies that a shunt or variable x is not written to between times t and t':

$$stable(x, t, t') \;\widehat{=}\; \forall\, t'' \in [t, t'] \bullet x@t'' = x@t.$$

This definition extends to sets of variables and shunts in the obvious way.

We can define a semantic function

$$\mathcal{F} \;:\; TAM \rightarrow TAMLL,$$

such that $\mathcal{F}[\![\mathcal{A}]\!]$ gives the predicate describing the functionality of agent $\mathcal{A}$. Let W be the alphabet of the agent, and let W_1 and W_2 be the alphabets of $\mathcal{A}_1$ and $\mathcal{A}_2$.

$$\begin{aligned}
\mathcal{F}[\![w : \Phi]\!] &\;\widehat{=}\; stable(W \setminus w, t_\alpha, t_\omega) \wedge \Phi, \\
\mathcal{F}[\![\mathsf{skip}]\!] &\;\widehat{=}\; t_\alpha \leqslant t_\omega \wedge stable(W, t_\alpha, t_\omega), \\
\mathcal{F}[\![\Delta t]\!] &\;\widehat{=}\; t_\omega = t_\alpha + t \wedge stable(W, t_\alpha, t_\omega), \\
\mathcal{F}[\![x := e]\!] &\;\widehat{=}\; t_\alpha < t_\omega \wedge stable(W \setminus x, t_\alpha, t_\omega) \wedge x@t_\omega = e@t_\alpha, \\
\mathcal{F}[\![x \leftarrow s]\!] &\;\widehat{=}\; t_\alpha < t_\omega \wedge stable(W \setminus x, t_\alpha, t_\omega) \\
&\qquad \wedge \exists\, t \in [t_\alpha, t_\omega) \bullet x@t_\omega = s@t, \\
\mathcal{F}[\![e \rightarrow s]\!] &\;\widehat{=}\; t_\alpha < t_\omega \wedge stable(W \setminus s, t_\alpha, t_\omega) \\
&\qquad \wedge \exists\, t \in (t_\alpha, t_\omega] \bullet stable(s, t_\alpha, t - 1) \wedge s@t = (t, e@t_\alpha) \\
&\qquad\qquad\qquad\qquad \wedge stable(s, t, t_\omega), \\
\mathcal{F}[\![(x : T)\mathcal{A}]\!] &\;\widehat{=}\; \exists\, v : Time \rightarrow T \bullet \mathcal{F}[\![\mathcal{A}]\!][v(t)/x@t \mid t \in [t_\alpha, t_\omega]], \\
\mathcal{F}[\![(s : T)\mathcal{A}]\!] &\;\widehat{=}\; \exists\, v : Time \rightarrow Time \times t \bullet v(t_\alpha).ts = t_\alpha \\
&\qquad\qquad\qquad\qquad \wedge \mathcal{F}[\![\mathcal{A}]\!][v(t)/s@t \mid t \in [t_\alpha, t_\omega]], \\
\mathcal{F}[\![[t]\mathcal{A}]\!] &\;\widehat{=}\; \exists\, t'_\omega : Time \bullet \mathcal{F}[\![\mathcal{A}]\!][t'_\omega/t_\omega] \wedge t'_\omega \leqslant t_\alpha + t = t_\omega \\
&\qquad\qquad \wedge stable(W, t'_\omega, t_\omega),
\end{aligned}$$

$$
\begin{aligned}
\mathcal{F}[\![\mathcal{A}\,;\mathcal{A}']\!] &\mathrel{\widehat=} \exists\, t_{mid} : \mathit{Time} \bullet \mathcal{F}[\![\mathcal{A}]\!][t_{mid}/t_\omega] \wedge \mathcal{F}[\![\mathcal{A}']\!][t_{mid}/t_\alpha],\\
\mathcal{F}[\![\mathcal{A} \sqcap \mathcal{A}']\!] &\mathrel{\widehat=} \mathcal{F}[\![\mathcal{A}]\!] \vee \mathcal{F}[\![\mathcal{A}']\!],\\
\mathcal{F}[\![\textstyle\bigsqcup_{t,\, i\in I} g_i \Rightarrow \mathcal{A}_i]\!] &\mathrel{\widehat=} \begin{array}[t]{ll} \text{let} & t'_\alpha = t_\alpha + t \quad \text{and} \quad J = \{i \in I \mid g_i@t_\alpha\} \quad \text{in}\\ \text{if} & J = \{\} \quad \text{then} \quad t_\omega = t'_\alpha \wedge \mathit{stable}(W, t_\alpha, t_\omega)\\ \text{else} & \mathit{stable}(W, t_\alpha, t'_\alpha) \wedge \exists\, j \in J \bullet \mathcal{F}[\![\mathcal{A}_j]\!][t'_\alpha/t_\alpha], \end{array}\\
\mathcal{F}[\![\mathcal{A} \rhd^s_t \mathcal{A}']\!] &\mathrel{\widehat=} \begin{array}[t]{ll} \text{if} & \exists\, t' \in [t_\alpha, t_\alpha + t) \bullet s@t'.ts = t'\\ \text{then} & \text{let} \quad t'_\alpha = \inf\{t' \geqslant t_\alpha \mid s@t'.ts = t'\} + 1\\ \text{in} & \mathit{stable}(W, t_\alpha, t'_\alpha) \wedge \mathcal{F}[\![\mathcal{A}']\!][t'_\alpha/t_\alpha]\\ \text{else} & \mathit{stable}(W, t_\alpha, t_\alpha + t) \wedge \mathcal{F}[\![\mathcal{A}]\!][t_\alpha + t/t_\alpha], \end{array}\\
\mathcal{F}[\![\mathcal{A}_1 \mid \mathcal{A}_2]\!] &\mathrel{\widehat=} \exists\, t_1, t_2 : \mathit{Time} \bullet \mathcal{F}[\![\mathcal{A}_1]\!][t_1/t_\omega] \wedge \mathit{stable}(W_1, t_1, t_\omega)\\
&\qquad\qquad\qquad\quad \wedge\, \mathcal{F}[\![\mathcal{A}_2]\!][t_2/t_\omega] \wedge \mathit{stable}(W_2, t_2, t_\omega)\\
&\qquad\qquad\qquad\quad \wedge\, t_\omega = \max\{t_1, t_2\},\\
\mu_0\, \mathcal{A} &\mathrel{\widehat=} \mathsf{skip},\\
\mu_{n+1}\, \mathcal{A} &\mathrel{\widehat=} \mathcal{A}\,; \mu_n\, \mathcal{A}.
\end{aligned}
$$

Refinement can be defined by:

$$\mathcal{A} \sqsubseteq_{\mathcal{F}} \mathcal{A}' \ \ \textit{iff} \ \ \mathcal{F}[\![\mathcal{A}]\!] \Leftarrow \mathcal{F}[\![\mathcal{A}']\!].$$

$\mathcal{A}$ is refined by $\mathcal{A}'$ iff $\mathcal{A}'$ is more fully defined than $\mathcal{A}$.

3 Infinite behaviours in TAM

We extend the syntax of TAM to include an infinite loop construct:

$$
\begin{array}{rcl}
\mathcal{A} & ::= & w : \Phi \mid \mathsf{skip} \mid \Delta t \mid x := e \mid x \leftarrow s \mid e \rightarrow s \mid\\
& & (x : T)\mathcal{A} \mid (s : T)\mathcal{A} \mid [t]\mathcal{A} \mid \mathcal{A}\,; \mathcal{A} \mid \mathcal{A} \sqcap \mathcal{A} \mid\\
& & \bigsqcup_{t,\, i\in I} g_i \Rightarrow \mathcal{A}_i \mid \mathcal{A} \rhd^s_t \mathcal{A} \mid \mathcal{A} \,|\, \mathcal{A} \mid \mu_n\, \mathcal{A} \mid \mu\, \mathcal{A}\,.
\end{array}
$$

We will only allow infinite iterations for agents that are *time-guarded*, i.e. agents that cannot terminate immediately; fortunately nearly all TAM agents are time-guarded; we give a formal definition of time-guardedness below.

We will no longer be assuming that agents always terminate, and so we change the TAM Logic Language to reflect this: previously we allowed the termination time t_ω to range over the set *Time* of times, which we took to be the natural numbers; we extend this to allow t_ω to also take the value ∞, representing non-termination:

$$t_\omega \in \mathit{Time}^+ \mathrel{\widehat=} \mathit{Time} \cup \{\infty\}.$$

We define a new semantic function

$$\mathcal{I} : \mathit{TAM} \rightarrow \mathit{TAMLL},$$

giving the semantics in this new model. For the atomic agents, the semantics in the new model is a simple adaptation of the semantics in the old model, with the extra condition that the agents do terminate ($t_\omega < \infty$):

$$
\begin{aligned}
\mathcal{I}[\![w : \Phi]\!] &\mathrel{\widehat{=}} \mathit{stable}(W \setminus w, t_\alpha, t_\omega) \wedge \Phi, \\
\mathcal{I}[\![\mathsf{skip}]\!] &\mathrel{\widehat{=}} t_\alpha \leqslant t_\omega < \infty \wedge \mathit{stable}(W, t_\alpha, t_\omega), \\
\mathcal{I}[\![\Delta t]\!] &\mathrel{\widehat{=}} t_\omega = t_\alpha + t \wedge \mathit{stable}(W, t_\alpha, t_\omega), \\
\mathcal{I}[\![x := e]\!] &\mathrel{\widehat{=}} t_\alpha < t_\omega < \infty \wedge \mathit{stable}(W \setminus x, t_\alpha, t_\omega) \wedge x@t_\omega = e@t_\alpha, \\
\mathcal{I}[\![x \leftarrow s]\!] &\mathrel{\widehat{=}} t_\alpha < t_\omega < \infty \wedge \mathit{stable}(W \setminus x, t_\alpha, t_\omega) \\
&\qquad \wedge \exists\, t \in [t_\alpha, t_\omega) \bullet x@t_\omega = s@t, \\
\mathcal{I}[\![e \rightarrow s]\!] &\mathrel{\widehat{=}} t_\alpha < t_\omega < \infty \wedge \mathit{stable}(W \setminus s, t_\alpha, t_\omega) \\
&\qquad \wedge \exists\, t \in (t_\alpha, t_\omega] \bullet \mathit{stable}(s, t_\alpha, t - 1) \wedge s@t = (t, e@t_\alpha) \\
&\qquad\qquad \wedge \mathit{stable}(s, t, t_\omega).
\end{aligned}
$$

Similarly, the semantic definitions for many of the language operators are simple adaptations of the definitions in the finite model:

$$
\begin{aligned}
\mathcal{I}[\![(x : T)\mathcal{A}]\!] &\mathrel{\widehat{=}} \exists\, v : \mathit{Time} \rightarrow T \bullet \mathcal{I}[\![\mathcal{A}]\!][v(t)/x@t \mid t \in [t_\alpha, t_\omega]], \\
\mathcal{I}[\![(s : T)\mathcal{A}]\!] &\mathrel{\widehat{=}} \exists\, v : \mathit{Time} \rightarrow \mathit{Time} \times t \bullet v(t_\alpha).ts = t_\alpha \\
&\qquad\qquad \wedge \mathcal{I}[\![\mathcal{A}]\!][v(t)/s@t \mid t \in [t_\alpha, t_\omega]], \\
\mathcal{I}[\![[t]\mathcal{A}]\!] &\mathrel{\widehat{=}} \exists\, t'_\omega : \mathit{Time} \bullet \mathcal{I}[\![\mathcal{A}]\!][t'_\omega/t_\omega] \wedge t'_\omega \leqslant t_\alpha + t = t_\omega \\
&\qquad \wedge \mathit{stable}(W, t'_\omega, t_\omega), \\
\mathcal{I}[\![\mathcal{A} \sqcap \mathcal{A}']\!] &\mathrel{\widehat{=}} \mathcal{I}[\![\mathcal{A}]\!] \vee \mathcal{I}[\![\mathcal{A}']\!], \\
\mathcal{I}[\![\textstyle\bigsqcup_{t,\, i \in I} g_i \Rightarrow \mathcal{A}_i]\!] &\mathrel{\widehat{=}} \text{let} \quad t'_\alpha = t_\alpha + t \quad \text{and} \quad J = \{i \in I \mid g_i@t_\alpha\} \quad \text{in} \\
&\qquad \text{if} \quad J = \{\} \quad \text{then} \quad t_\omega = t'_\alpha \wedge \mathit{stable}(W, t_\alpha, t_\omega) \\
&\qquad \text{else} \ \ \mathit{stable}(W, t_\alpha, t'_\alpha) \wedge \exists\, j \in J \bullet \mathcal{I}[\![\mathcal{A}_j]\!][t'_\alpha/t_\alpha], \\
\mathcal{I}[\![\mathcal{A} \rhd^s_t \mathcal{A}']\!] &\mathrel{\widehat{=}} \text{if} \quad \exists\, t' \in [t_\alpha, t_\alpha + t) \bullet s@t'.ts = t' \\
&\qquad \text{then let} \quad t'_\alpha = \inf\{t' \geqslant t_\alpha \mid s@t'.ts = t'\} + 1 \\
&\qquad \text{in} \quad \mathit{stable}(W, t_\alpha, t'_\alpha) \wedge \mathcal{I}[\![\mathcal{A}']\!][t'_\alpha/t_\alpha] \\
&\qquad \text{else} \quad \mathit{stable}(W, t_\alpha, t_\alpha + t) \wedge \mathcal{I}[\![\mathcal{A}]\!][t_\alpha + t/t_\alpha].
\end{aligned}
$$

The definition for the sequential composition $\mathcal{A};\mathcal{A}'$ depends upon whether $\mathcal{A}$ terminates: if $\mathcal{A}$ terminates at time $t_{mid} < \infty$ then $\mathcal{A}'$ is started at this time; if $\mathcal{A}$ does not terminate then $\mathcal{A}'$ is never executed and the composition is non-terminating:

$$
\mathcal{I}[\![\mathcal{A}\,;\mathcal{A}']\!] \mathrel{\widehat{=}} \exists\, t_{mid} : \mathit{Time}^+ \bullet \mathcal{I}[\![\mathcal{A}]\!][t_{mid}/t_\omega] \wedge \begin{pmatrix} t_{mid} < \infty \wedge \mathcal{I}[\![\mathcal{A}']\!][t_{mid}/t_\alpha] \\ \vee\ t_{mid} = t_\omega = \infty \end{pmatrix}.
$$

Surprisingly, the definition for parallel composition in this model is very similar to the definition for the finite model:

$$
\begin{aligned}
\mathcal{I}[\![\mathcal{A}_1 \mid \mathcal{A}_2]\!] &\mathrel{\widehat{=}} \exists\, t_1, t_2 : \mathit{Time}^+ \bullet \mathcal{I}[\![\mathcal{A}_1]\!][t_1/t_\omega] \wedge \mathit{stable}(W_1, t_1, t_\omega) \\
&\qquad \wedge \mathcal{I}[\![\mathcal{A}_2]\!][t_2/t_\omega] \wedge \mathit{stable}(W_2, t_2, t_\omega) \\
&\qquad \wedge t_\omega = \max\{t_1, t_2\},
\end{aligned}
$$

If either component does not terminate, then the composition does not terminate.

The definition of finite iteration remains the same:

$$\mu_{0}\,\mathcal{A} \mathrel{\widehat=} \mathsf{skip}, \qquad\qquad \mu_{n+1}\,\mathcal{A} \mathrel{\widehat=} \mathcal{A}\,;\mu_{n}\,\mathcal{A}.$$

Note that:

$$\begin{aligned}\mathcal{I}[\![\mu_{n}\,\mathcal{A}]\!] \;=\; & \exists\, t_0,\ldots,t_n : \mathit{Time}^{+} \bullet \\ & \quad t_\alpha = t_0 \leqslant t_1 \leqslant \ldots \leqslant t_n = t_\omega \\ & \quad \wedge \forall\, i \in 0\,..\,n-1 \bullet t_i < \infty \Rightarrow \mathcal{I}[\![\mathcal{A}]\!][t_i/t_\alpha, t_{i+1}/t_\omega].\end{aligned}$$

Here the ith iteration, if it is ever reached, begins at time t_i and ends at time t_{i+1}; if $t_i = \infty$ then this represents that some previous iteration fails to terminate, and so the ith iteration is not reached.

We can adapt this to give the semantics of infinite iteration:

$$\begin{aligned}\mathcal{I}[\![\mu\,\mathcal{A}]\!] \;\mathrel{\widehat=}\; & \exists\, t_0,\ldots : \mathit{Time}^{+} \bullet \\ & \quad t_\alpha = t_0 \leqslant t_1 \leqslant \ldots \wedge t_\omega = \infty \\ & \quad \wedge \forall\, i \in \mathbb{N} \bullet t_i < \infty \Rightarrow \mathcal{I}[\![\mathcal{A}]\!][t_i/t_\alpha, t_{i+1}/t_\omega].\end{aligned}$$

This is defined only for agents $\mathcal{A}$ that are *time-guarded*:

Definition 1: Agent $\mathcal{A}$ is time-guarded iff $\mathcal{I}[\![\mathcal{A}]\!] \Rightarrow t_\alpha < t_\omega$. □

This condition ensures that the times $t_0, t_1, \ldots$, above, are strictly increasing, and so tend to infinity. This outlaws agents, such as $\mu\,\mathsf{skip}$, that can perform an infinite number of iterations in a finite amount of time, against our intuition.

Note that

$$\mu\,\mathcal{A} \;=\; \mathcal{A}\,;\mu\,\mathcal{A},$$

so infinite iteration is a fixed point of sequential composition, as required. Further,

$$\mu\,\mathcal{A}\,;\mathcal{A}' \;=\; \mu\,\mathcal{A},$$

because infinite iterations never terminate.

The semantics of an infinite *periodic* loop is somewhat simpler. Writing $\mu^{t}\,\mathcal{A}$ for the agent that performs $\mathcal{A}$ every t time-units, we have:

$$\mathcal{I}[\![\mu^{t}\,\mathcal{A}]\!] \;\mathrel{\widehat=}\; (\forall\, i \in \mathbb{N} \bullet \exists\, t_\omega \bullet \mathcal{I}[\![[t]\;\mathcal{A}]\!][t_\alpha + i.p/t_\alpha]) \wedge t_\omega = \infty.$$

The ith iteration starts at time $t_\alpha + i.p$.

We can also define an agent $\rhd^{s}\,\mathcal{A}$ that monitors shunt s until it is written to, and then acts like $\mathcal{A}$; if the shunt is never written to, then the agent does not terminate:

$$\begin{aligned}\mathcal{I}[\![\rhd^{s}\,\mathcal{A}]\!] \;\mathrel{\widehat=}\; & \text{if} \quad \exists\, t \geqslant t_\alpha \bullet s@t.ts = t \\ & \text{then let} \quad t'_\alpha = \inf\{t \geqslant t_\alpha \mid s@t.ts = t\} + 1 \\ & \text{in} \quad \mathit{stable}(W, t_\alpha, t'_\alpha) \wedge \mathcal{I}[\![\mathcal{A}']\!][t'_\alpha/t_\alpha] \\ & \text{else} \quad t_\omega = \infty \wedge \mathit{stable}(W, t_\alpha, \infty).\end{aligned}$$

We define refinement in the infinite model in the obvious way:

$$\mathcal{A} \sqsubseteq_{\mathcal{I}} \mathcal{A}' \;\; \mathit{iff} \;\; \mathcal{I}[\![\mathcal{A}]\!] \Leftarrow \mathcal{I}[\![\mathcal{A}']\!].$$

We can show that for all finite agents $\mathcal{A}$ (i.e. agents without infinite iterations or unbounded shunt monitoring), the two semantic definitions agree:

$$\mathcal{I}[\![\mathcal{A}]\!] \;\Leftrightarrow\; \mathcal{F}[\![\mathcal{A}]\!].$$

4 Conclusions

In this paper we have augmented the syntax of TAM [SZJ94] with an infinite iteration operator, and have extended the semantic model to deal with infinite executions, thus allowing us to easier deal with a larger class of systems.

Dealing with infinite executions can sometimes make calculations easier. For example, in the finite model we have the following—somewhat complicated—refinement rule for parallel composition:

Rule 1: If $w_1 \cap w_2 = \{\}$, then

$$w_1 \cup w_2 : \Phi \;\sqsubseteq_{\mathcal{F}}\; (w_1 : \Phi_1) \mid (w_2 : \Phi_2),$$

provided

$$\begin{pmatrix} \exists\, t_1, t_2 : \mathit{Time} \bullet \Phi_1[t_1/t_\omega] \wedge \mathit{stable}(w_1, t_1, t_\omega) \\ \qquad\qquad\qquad \wedge\, \Phi_2[t_2/t_\omega] \wedge \mathit{stable}(w_2, t_2, t_\omega) \wedge t_\omega = \max\{t_1, t_2\} \end{pmatrix} \;\Rightarrow\; \Phi.$$

□

However, if we know that both sides are infinite, then we have a somewhat simpler rule:

Rule 2: If $w_1 \cap w_2 = \{\}$, then

$$(w_1 \cup w_2 : \Phi_1 \wedge \Phi_2 \wedge t_\omega = \infty) \;\sqsubseteq_{\mathcal{I}}\; (w_1 : \Phi_1 \wedge t_\omega = \infty) \mid (w_2 : \Phi_2 \wedge t_\omega = \infty).$$

□

In this paper we have dealt only with infinite *iteration*: we have not dealt with the problem of infinite *recursion*. It would be useful to extend the ideas in this paper to handle infinite recursion, but this will be considerably more difficult, because we would have to develop a fixed-point theory. It might be possible to develop a theory using metric spaces along the lines of [RR87]; this will require finding a metric where the limit of agents with successively longer executions is an infinite agent.

We would like to extend the idea of infinite executions to the scheduling-oriented model of [Low94a]; we believe that this would be reasonably straightforward.

References

[Low94a] Gavin Lowe. Scheduling-oriented models for real-time systems. Submitted for publication. Also available as Oxford University Computing Laboratory Technical Report PRG-TR-10-94, 1994.

[Low94b] Gavin Lowe. Formal development of aircraft control software: A case study in the specification, design and scheduling of a real-time system. Technical Report PRG-TR-15-94, Oxford University Computing Laboratory, 1994.

[LZ95] Gavin Lowe and Hussein Zedan. Refinement of complex systems: A case study. Submitted for publication. Also available as Oxford University Computing Laboratory Technical Report PRG-TR-2-95, 1995.

[RR87] G. M. Reed and A. W. Roscoe. Metric spaces as models for real-time concurrency. In *Proceedings of the Third Workshop on the Mathematical Foundations of Programming Semantics, LNCS 298*, pages 331–343. Springer Verlag, 1987.

[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.

[SZJ94] D. Scholefield, H. Zedan, and He Jifeng. A specification oriented semantics for the refinement of real-time system. *Theoretical Computer Science*, 131(1), 1994.