A Knowledge-Theoretic Semantics for Concurrent METATEM

Michael Wooldridge

Mitsubishi Electric Digital Library Group
18th Floor, Centre Point
103 New Oxford Street, London WC1 1EB
United Kingdom

mjw@dlib.com

Abstract

The aim of this paper is to develop a semantics for the multi-agent programming language Concurrent METATEM, by using the tools of *knowledge theory*. We begin by presenting an informal overview of the Concurrent METATEM language, and then formally define the notion of a Concurrent METATEM agent program, the possible states of a Concurrent METATEM agent and system, and finally, what constitutes an acceptable *run* of a Concurrent METATEM system. The various semantic constructs developed during this process are then used as the foundation upon which to construct a *temporal logic of knowledge*; we show that certain formulae of this logic correspond to properties of Concurrent METATEM systems. This correspondence can be used to systematically derive the *theory* of a Concurrent METATEM system. Reasoning about a Concurrent METATEM system then reduces to proving properties of the corresponding logical theory. We give some simple examples, to illustrate the process, and finally, speculate about future research directions. Comments on related work are also included.

1 Introduction

As the world-wide interest in multi-agent systems has intensified, so the range of software tools available for developing multi-agent systems has continued to grow [21]. Some of these tools represent developments of established areas of software technology and programming language design. For example, a number of multi-agent programming languages are available that build on the concurrent object-based programming paradigm [2]. Many of these languages can be given a semantics by using the tools of, for example, process algebra [16]; the ACTOR class of languages in particular have an elegant semantic foundation [1]. However, for many multi-agent programming languages, it is not at all clear what techniques might be used to develop a formal semantics. In brief, the purpose of this paper is to address the problem of giving a formal semantics to a multi-agent programming language called Concurrent METATEM [6, 7, 8].

A Concurrent METATEM system contains a set of concurrently executing agents, able to communicate via asynchronous broadcast message passing. Agents in Concurrent METATEM are programmed by giving them a temporal logic specification of the behaviour it is intended they should exhibit. An agent's specification is directly executed in order to generate its behaviour. Two previous attempts have been made to develop a semantics for Concurrent METATEM. In the first attempt [8], emphasis was placed on the modeling of *concurrency*, which was achieved by representing agent execution cycles as intervals over the real numbers. In the second attempt [9], a temporal logic of belief was used to axiomatize Concurrent METATEM. That is, a number of axioms were presented which, it was argued, correspond to properties of Concurrent METATEM. This axiomatization could be used to derive a theory of a Concurrent METATEM system, and properties of the system could then be demonstrated by proving properties of the theory. Both of these attempts to give a semantics to Concurrent METATEM were essentially *top down* approaches: a number of axioms were presented which, it was claimed, represent properties of Concurrent METATEM systems. No formal attempt was made in either case to justify the axioms with reference to what constitutes a *run* of a Concurrent METATEM system. For this reason, it could be argued that both attempts were in some sense *ad hoc*.

In this paper, we present a *bottom-up* semantics for Concurrent METATEM. First, we formally define Concurrent METATEM programs and systems, and then state the conditions required for a sequence of states to be considered a *run* of a Concurrent METATEM system. For any given Concurrent METATEM system, we can identify the set of all acceptable runs of that system: this set represents the formal semantics of the system. However, attempting to manipulate sets of runs directly, in order to prove properties of a system, is not practicable. For this reason, we introduce a *temporal logic of knowledge* for representing properties of sets of runs [14, 20, 10, 5]. This logic is a 'standard' temporal logic, which, in addition to the usual temporal connectives, contains an indexed set of modal connectives for referring to the information implicit within agents. We show how properties of the logic correspond to certain properties of Concurrent METATEM systems: given a Concurrent METATEM system, we can derive a temporal-knowledge theory of that system. Proving properties of the system can be done by proving properties of the corresponding temporal-knowledge theory, which is, in general, a simpler process than attempting to prove properties of runs directly. The axiomatization of Concurrent METATEM turns out to be quite intuitive, and the use of *knowledge* connectives affords us greater expressive power than the use of a temporal logic in isolation.

The remainder of this paper is structured as follows. In the next subsection, we present a summary of our notation. Then, in section 2, we give an overview of the Concurrent METATEM language (readers familiar with Concurrent METATEM may wish to skim this section). In section 3, we formally define Concurrent METATEM systems, system states, and runs. In section 4, we use these semantic structures as the foundation upon which to construct KL_n, our temporal logic of knowledge; in section 4.2, we show how certain formulae of KL_n correspond to properties of Concurrent METATEM systems, and give an example of how a KL_n theory derived using this correspondence can be used to verify simple properties of a system. In section 5, we discuss related work, and in section 6, we present some concluding remarks.

1.1 Notation

If L is a logical language, then we write $Form(L)$ for the set of (well-formed) formulae of L. We use the lowercase Greek letters φ, ψ, and χ as meta-variables ranging over formulae of the logical languages we consider. If S is a set, then by $\wp(S)$, we mean the powerset of S. If S is a set, then by S^*, we mean the set of sequences over S. Thus $(2, 4, 5) \in I\!N^*$. If s is a sequence and $u \in I\!N$, then by $s(u)$ we mean the element at position u in s, assuming that positions are indexed by the natural numbers. Thus $(2, 4, 5)(0) = 2$, and $(2, 4, 5)(2) = 5$. Throughout the paper, it is assumed that the reader is familiar with basic modal and temporal logic [12].

2 Concurrent METATEM

In this section, we present an overview of Concurrent METATEM. Note that Concurrent METATEM is a descendent of METATEM, details of which may be found in [3]; the Concurrent METATEM language itself is described in more detail in [6, 7, 8].

2.1 Agents in Concurrent METATEM

Agents in Concurrent METATEM are concurrently executing entities, able to communicate with each other through asynchronous broadcast message passing. Each Concurrent METATEM agent has two main components:

- an *interface*, which defines how the agent may interact with its environment (i.e., other agents);
- a *computational engine*, which defines how the agent will act — in Concurrent METATEM, the approach used is based on the METATEM paradigm of executable temporal logic [3].

An agent interface consists of three components:

- a unique *agent identifier* (or just agent id), which names the agent;
- a set of symbols defining which messages will be accepted by the agent — these are termed *environment propositions*; and
- a set of symbols defining messages that the agent may send — these are termed *component propositions.*

For example, the interface definition of a 'stack' agent might be:

$$\mathit{stack}(\mathit{pop}, \mathit{push})[\mathit{popped}, \mathit{full}]$$

Here, *stack* is the agent id that names the agent, $\{\mathit{pop}, \mathit{push}\}$ is the set of environment propositions, and $\{\mathit{popped}, \mathit{full}\}$ is the set of component propositions. The intuition is that, whenever a message headed by the symbol *pop* is broadcast, the *stack* agent will *accept* the message; we describe what this means below. If a message is broadcast that is not declared in the *stack* agent's interface, then *stack* ignores it. Similarly, the only messages that can be sent by the *stack* agent are headed by the symbols *popped* and *full.*

The computational engine of each agent in Concurrent METATEM is based on the METATEM paradigm of executable temporal logics [3]. The idea is to directly execute an agent specification, where this specification is given as a set of *program rules*, which are temporal logic formulae of the form:

$$\text{antecedent about past} \quad \Rightarrow \quad \text{consequent about present and future.}$$

The antecedent is a temporal logic formula referring to the past, whereas the consequent is a temporal logic formula referring to the present and future. The intuitive interpretation of such a rule is 'on the basis of the past, construct the future', which gives rise to the name of the paradigm: *declarative past and imperative future* [11]. The rules that define an agent's behaviour can be animated by directly executing the temporal specification under a suitable operational model [7].

To make the discussion more concrete, we introduce a propositional temporal logic, called Propositional METATEM Logic (PML), in which the temporal rules that are used to specify an agent's behaviour will be given. (A complete definition of PML is given in [3].) PML is essentially classical propositional logic augmented by a set of modal connectives for referring to the *temporal ordering* of events. PML is based on a model of time that is *linear* (i.e., each moment in time has a unique successor), *bounded in the past* (i.e., there was a moment that was the 'beginning of time'), and *infinite in the future* (i.e., there are an infinite number of moments in the future). The temporal connectives of PML can be divided into two categories, as follows:

1. Strict past time connectives: '●' (weak last), '◉' (strong last), '◈' (was), '■' (heretofore), '$\mathcal{S}$' (since) and '$\mathcal{Z}$' (zince, or weak since).
2. Present and future time connectives: '$\bigcirc$' (next), '$\Diamond$' (sometime), '$\Box$' (always), '$\mathcal{U}$' (until) and '$\mathcal{W}$' (unless).

The connectives $\{\circledcirc, \bullet, \blacklozenge, \blacksquare, \bigcirc, \Diamond, \Box\}$ are unary; the remainder are binary. In addition to these temporal connectives, PML contains the usual classical logic connectives. The meaning of the temporal connectives is quite straightforward, with formulae being interpreted at a particular moment in time. Let φ and ψ be formulae of PML, then: $\bigcirc\varphi$ is satisfied at the current moment in time (i.e., now) if φ is satisfied at the next moment in time; $\Diamond\varphi$ is satisfied now if φ is satisfied either now or at some future moment in time; $\Box\varphi$ is satisfied now if φ is satisfied now and at all future moments; $\varphi\,\mathcal{U}\,\psi$ is satisfied now if ψ is satisfied at some future moment, and φ is satisfied until then — $\mathcal{W}$ is a binary connective similar to $\mathcal{U}$, allowing for the possibility that the second argument might never be satisfied.

The past-time connectives have similar meanings: $\circledcirc\varphi$ and $\bullet\varphi$ are satisfied now if φ was satisfied at the previous moment in time — the difference between them is that, since the model of time underlying the logic is bounded in the past, the beginning of time is treated as a special case in that, when interpreted at the beginning of time, $\circledcirc\varphi$ cannot be satisfied whereas $\bullet\varphi$ will always be satisfied, regardless of φ; $\blacklozenge\varphi$ is satisfied now if φ was satisfied at some previous moment in time; $\blacksquare\varphi$ is satisfied now if φ was satisfied at all previous moments in time; $\varphi\,\mathcal{S}\,\psi$ is satisfied now if ψ was satisfied at some previous moment in time, and φ has been satisfied since then — $\mathcal{Z}$ is similar, but allows for the possibility that the second argument was never satisfied; finally, a nullary temporal operator can be defined, which is satisfied only at the beginning of time — this useful operator is called '**start**'.

It is convenient to identify three sub-languages of PML:

PML^{+}	—	formulae referring to the present or future
PML^{-}	—	formulae referring to the past
$\text{PML}^{\pm}$	—	formulae in the form *past* $\Rightarrow$ *future*.

Members of PML^{+} are known as *commitments*; members of PML^{-} are known as *history formulae*; and members of $\text{PML}^{\pm}$ are known as *rules*.

2.2 Agent Execution

The actual execution of an agent in Concurrent METATEM is, superficially at least, very simple to understand. Each agent obeys a cycle of trying to match the past-time antecedents of its rules against a *history*, and executing the consequents of those rules that 'fire'. More precisely, the computational engine for an agent continually executes the following cycle:

1. Update the *history* of the agent by receiving messages (i.e., environment propositions) from other agents and adding them to its history.

2. Check which rules *fire*, by comparing past-time antecedents of each rule against the current history to see which are satisfied.

3. *Jointly execute* the fired rules together with any commitments carried over from previous cycles.

   This involves first collecting together consequents of newly fired rules with old commitments — these become the *current constraints*. Now attempt to create the next state while satisfying these constraints. As the current constraints are represented by a disjunctive formula, the agent will have to choose between a number of execution possibilities.

   Note that it may not be possible to satisfy *all* the relevant commitments on the current cycle, in which case unsatisfied commitments are carried over to the next cycle.

4. Goto (1).

Clearly, step (3) is the heart of the execution process. Making the wrong choice at this step may mean that the agent specification cannot subsequently be satisfied [3].

$$
\begin{array}{l}
rp(ask1, ask2)[give1, give2] \ : \\
\quad \bullet ask1 \quad \Rightarrow \quad \Diamond give1; \\
\quad \bullet ask2 \quad \Rightarrow \quad \Diamond give2; \\
\quad\quad \mathbf{start} \quad \Rightarrow \quad \Box \neg (give1 \wedge give2). \\
rc1(give1)[ask1] \ : \\
\quad\quad \mathbf{start} \quad \Rightarrow \quad ask1; \\
\quad \bullet ask1 \quad \Rightarrow \quad ask1. \\
rc2(ask1, give2)[ask2] \ : \\
\quad \bullet (ask1 \wedge \neg ask2) \quad \Rightarrow \quad ask2.
\end{array}
$$

Figure 1: A Simple Concurrent METATEM System

Time	Agent		
	rp	*rc*1	*rc*2
0		*ask*1	
1	*ask*1	*ask*1	*ask*2
2	*ask*1, *ask*2, *give*1	*ask*1	
3	*ask*1, *give*2	*ask*1, *give*1	*ask*2
4	*ask*1, *ask*2, *give*1	*ask*1	*give*2
5	...	...	...

Figure 2: An Example Run
.

When a proposition in an agent becomes *true*, it is compared against that agent's interface (see above); if it is one of the agent's *component propositions*, then that proposition is broadcast as a message to all other agents. On receipt of a message, each agent attempts to match the proposition against the environment propositions in their interface. If there is a match, then they add the proposition to their history.

2.3 An Example Concurrent METATEM System

Figure 1 shows a simple system containing three agents: *rp*, *rc*1, and *rc*2. The agent *rp* is a 'resource producer': it can '*give*' to only one agent at a time, and will commit to eventually *give* to any agent that *ask*s. Agent *rp* will only accept messages *ask*1 and *ask*2, and can only send *give*1 and *give*2 messages. The interface of agent *rc*1 states that it will only accept *give*1 messages, and can only send *ask*1 messages. The rules for agent *rc*1 ensure that an *ask*1 message is sent on every cycle — this is because **start** is satisfied at the beginning of time, thus firing the first rule, so $\bullet ask1$ will be satisfied on the next cycle, thus firing the second rule, and so on. Thus *rc*1 asks for the resource on every cycle, using an *ask*1 message. The interface for agent *rc*2 states that it will accept both *ask*1 and *give*2 messages, and can send *ask*2 messages. The single rule for agent *rc*2 ensures that an *ask*2 message is sent on every cycle where, on its previous cycle, it did not send an *ask*2 message, but received an *ask*1 message (from agent *rc*1). Figure 2 shows a fragment of an example run of the system in Figure 1.

3 Programs, States, and Runs

The semantics of a programming language define, for any given program, what the acceptable *states* of that program are, and in addition, what represents an acceptable *run* of that program. In order to define the semantics of Concurrent METATEM, we must therefore define both the acceptable states of a Concurrent METATEM system, and the acceptable runs of a Concurrent METATEM system. The purpose of section is to do exactly this. We begin by stating the key assumptions that underpin our semantics, and then, in section 3.2, formally define what constitutes a Concurrent METATEM agent and system. In section 3.3, we define the possible states of an agent and system, and in section 3.4, we define the possible runs, or computations, of a Concurrent METATEM system. These runs are then used in section 4 as the semantic framework upon which to construct a logic for reasoning about Concurrent METATEM systems.

3.1 Some Assumptions

Synchronous execution: Perhaps the most important (and most limiting) assumption made in what follows is that of *synchronous execution.* By this, we mean it is assumed that agents execute in lock-step: when one agent completes a cycle and begins another, all other agents do likewise. This implies that we are not, in fact, modeling *concurrency* within our semantics; we avoid doing so in order to steer clear of several rather complex side-issues. One possible approach to modeling concurrency within the semantic framework would be to represent agent cycles as intervals over the real numbers, in the way proposed by Fisher in [8].

Perfect execution: By this, we mean that if it is *possible* for an agent to satisfy its specification, then it will do so. In other words, we assume that agents always make the right choices. In implemented Concurrent METATEM systems, it is not the case that an agent will always make the right choices; it may make a bad choice, and subsequently be unable to satisfy all its commitments. However, a 'perfect' Concurrent METATEM implementation would respect this assumption.

Consistent agent specifications: We do not attempt to state what happens when an agent program is unsatisfiable. An example of this situation would be an agent containing the rule **start** $\Rightarrow (\Box p) \wedge (\Diamond \neg p)$. Clearly, no execution mechanism could successfully execute this rule; in implemented Concurrent METATEM systems, the execution mechanism will simply attempt to satisfy the rule infinitely often (and not succeed).

Guaranteed message delivery: In any real communication channel, message delivery cannot be guaranteed. Many formalisms have been developed to represent this aspect of communication channels, and some interesting results have been obtained on the possibility of obtaining *common knowledge* in systems where message delivery is not guaranteed [5, pp175–222]. Nevertheless, for the purposes of this article, it seems reasonable to assume that message delivery *is* guaranteed.

3.2 Programs

We begin by assuming that a Concurrent METATEM system contains an *environment part*, e, and a set $Ag = \{1, \ldots, n\}$ of *agents*. In order to express properties of the environment e, we use a set Φ_e of primitive propositions. Similarly, we associate a set Φ_i of primitive propositions with every agent $i \in Ag$. The set Φ_i, for $i \in Ag$, contains all those propositions that can occur in i's rules, or that can be received or sent by i as messages. For reasons that will become apparent below, we shall impose two constraints on these sets:

1. The sets of propositions for talking about agents and the environment are mutually disjoint. Formally, for all $i \neq j \in \{e, 1, \ldots, n\}$, we require that $\Phi_i \cap \Phi_j = \emptyset$.

2. For every proposition p_i that occurs in Φ_i, we require that there exists a corresponding proposition p_j in Φ_j. Formally, for all $i, j \in \{e, 1, \ldots, n\}$ we require that there exists a bijection $f : \Phi_i \to \Phi_j$.

Thus each agent has a unique set of propositions that it uses for rules, messages, and so forth. However, there is a correspondence between the different sets. For example, in Figure 1, agent $rc1$ sends an $ask1$ message, which the interface to agent rp states that it will accept. The constraints we have just given state that the $ask1$ in $rc1$ is actually distinct from the $ask1$ in rp. However, we intend that the propositions correspond to each other: to make the correspondence clear, we write p_i to indicate that p is a member of Φ_i; this proposition will correspond to p_j, a member of Φ_j. Thus $ask1_{rc1} \in \Phi_{rc1}$, and $ask1_{rp} \in \Phi_{rp}$. To simplify subsequent formalisation, we will often find it convenient to ignore this constraint, and act as if all agents use a common set of propositions; the constraint is only really required in section 4.

Let $\Phi = \bigcup_i \Phi_i$ be the set of all primitive propositions. Clearly, the sets Φ_i partition Φ. An *agent i* is then a triple $(in_i, out_i, rules_i)$, where:

- $in_i \subseteq \Phi_i$ represents messages that i will *accept* (environment propositions);
- $out_i \subseteq \Phi_i$ represents messages that i will *send* (component propositions); and
- $rules_i \subseteq Form(\textsc{pml}^{\pm})$ is a set of PML rules representing i's program. We require that if $\varphi \in rules_i$, then φ contains only propositions that occur in Φ_i.

3.3 States

An agent's internal state at any moment in time is determined by two components: a *history*, and a set of *commitments*. An agent's history at time $u \in I\!N$ is a record of all the messages all the messages it has received and actions it has performed (including messages it has sent) up to and including time u. For example, the history of agent $rc2$ from Figure 1 at time 4 in the run given in Figure 2 is:

$$(\emptyset, \{ask2\}, \emptyset, \{ask2\}, \{give2\}).$$

Formally, let $H_i = (\wp(\Phi_i))^*$ be the set of all possible histories for agent $i \in Ag$. Let $H = \bigcup_i H_i$ be the set of all possible histories. We use h (with annotations: $h', h_1, \ldots$) to stand for members of H.

Suppose that h is a history, and $\varphi \in Form(\textsc{pml})$. Then we write $(h, u) \models \varphi$ if h *satisfies* φ at time u. That is, we treat h as a (partial) model for PML, and interpret φ with respect to this model. The complete formal definition of this relation is essentially a re-statement of the semantics of PML, and so, rather than attempting to give such a statement, we simply illustrate the notation by example. Suppose we have $h = (\{p\}, \{p, q\}, \{p\}, \{p\})$. Then:

$$\begin{array}{ll} (h,1) \models (\circledcirc p) \wedge q & (h,2) \models (\circledcirc\circledcirc p) \wedge (\circledcirc q) \\ (h,0) \models \Box p & (h,0) \models p\,\mathcal{U}\,q \\ (h,0) \models \Diamond p & (h,3) \models \Diamonddot\, q \end{array}$$

An agent's *commitments* represent the future-time parts of program rules that have fired, and that the agent has, as a result, committed to satisfying. Formally, let $C = \wp(Form(\textsc{pml}^{+}))$ be the set of all sets of commitments. We use c (with annotations: $c', c_1, \ldots$) to stand for members of C. If $i \in Ag$, then let $C_i \subseteq C$ be the possible commitment sets of agent i (i.e., C_i contains only commitments made up from members of Φ_i). As an example, the commitment set of agent rp in the system in Figure 1 at time 3 in the run given in Figure 2 is:

$$\{\Diamond give1, \Diamond give2, \Box\neg(give1 \wedge give2)\}.$$

The local, or internal state of an agent $i \in Ag$ at any moment in time is then a pair (h, c), where $h \in H_i$ and $c \in C_i$. Let $L_i = H_i \times C_i$ be the set of all local states for agent i, and let $L = \bigcup_i L_i$ be the set of all local states. We use l (with annotations: $l', l_1, \ldots$) to stand for members of L. If $l \in L$ is a local state, then we let $h(l)$ denote its history component, and let $c(l)$ denote its commitment set component. As an example, the state of agent $rc2$ in the system in Figure 1 at time 4 in the run given in Figure 2 is:

$$((\emptyset, \{ask2\}, \emptyset, \{ask2\}, \{give2\}), \emptyset).$$

The state of the environment at any time is simply a history over Φ_e, where each component in this history represents the set of propositions that have been sent as messages at the corresponding time. For example, the environment state of the system in Figure 1 at time 4 in the run given in Figure 2 is:

$$(\{ask1\}, \{ask1, ask2\}, \{ask1, give1\}, \{ask1, ask2, give2\}, \{ask1, give1\})$$

Formally, let $E = H_e$ be the set of all environment states.

The *global* state of a Concurrent METATEM system containing n agents $1, \ldots, n$ is an $(n+1)$-tuple $(e, l_1, \ldots, l_n)$, where $e \in E$, $l_1 \in L_1$, ..., $l_n \in L_n$. Let $G = E \times L_1 \times \cdots \times L_n$ be the set of all global states. We use g (with annotations: $g', g_1, \ldots$) to stand for members of G.

3.4 Runs

The state of an agent or system represents a *snapshot* of the agent or system at some time. However, multi-agent systems are *reactive* [17], and for this reason, we are not generally interested in the state of a system at some instant, but rather with the *behaviour* of the system over an *infinite sequence* of states. For this reason, we introduce *runs*, which are, in essence, simply infinite sequences of states. Formally, a run r is a function $r : I\!N \to G$, which assigns a global state to every natural number. Let R be the set of all runs. Following [5, p107], we refer to a pair (r, u), where $r \in R$ and $u \in I\!N$, as a *point*. Let P be the set of all points. If $r(u) = (e, l_1, \ldots, l_n)$, then we let $r_e(u)$ stand for e, and $r_i(u)$ stand for l_i.

In order to simplify subsequent formalisation, we present some auxiliary definitions. First, if $i \in Ag$, then $sent_i : R \times I\!N \to \wp(\Phi_i)$ is a function that gives the set of messages sent by i in a run at some time: $sent_i(r, u) = h(r_i(u))(u) \cap out_i$. Similarly, if $i \in Ag$, then $rcvd_i : R \times I\!N \to \wp(\Phi_i)$ is a function that gives the set of messages received by i in a run at some time: $rcvd_i(r, u) = h(r_i(u))(u) \cap in_i$. For every Concurrent METATEM system sys containing agents $\{(in_i, out_i, rules_i) \mid i \in Ag\}$, there will be some set $R^{sys} \subseteq R$ of runs such that R^{sys} contains just those members of R that represent runs of sys. The set R^{sys} will in fact represent the semantics of sys. (Note that if R^{sys} is empty, then sys has no runs, which implies that one or more of the agent programs are inconsistent.)

In order to obtain a formal semantics for Concurrent METATEM, we must define the circumstances under which an arbitrary run $r \in R$ is a member of R^{sys}. Formally, let sys be a Concurrent METATEM system containing agents $\{(in_i, out_i, rules_i) \mid i \in Ag\}$. Then a run $r \in R$ is a member of the set R^{sys} iff the following conditions obtain:

(E1) At time $u \in I\!N$, the environment is made up of all messages sent at time u. Formally, $\forall u \in I\!N$, $p_e \in r_e(u)$ iff $\exists i \in Ag$ s.t. $p_i \in sent_i(r, u)$.

(M1) A message sent by an agent is eventually received by all agents with the appropriate interface. Formally, $\forall u \in I\!N, \forall i \in Ag$, if $p_e \in r_e(u)$ and $p_i \in in_i$ then $\exists v \in I\!N$ s.t. $v > u$ and $p_i \in rcvd_i(r, v)$.

(M2) Messages can only be received once, and, moreover, if a message is received, then it must have been sent at some earlier time. Formally, $\forall i \in Ag$, $\forall u \in I\!N$, if $p_i \in rcvd_i(r, u)$, then $\exists v \in I\!N$, $v < u$, s.t. $p_e \in r_e(u)$ and $\forall w \in \{v, \ldots, w-1\}$, $p_i \notin rcvd_i(r, u)$. (Note that this condition does not imply the preceding one.)

(H1) Agents have one history state for every time step they have experienced. Formally, $\forall i \in Ag$, $\forall u \in I\!N$, $|h(r_i(u))| = u + 1$.

(H2) Agents maintain perfect histories (they never forget, and they are strictly accurate). Formally, $\forall i \in Ag, \forall u \in \mathbb{N}, \forall v \in \mathbb{N}$ s.t. $v \leq u$ if $h(r_i(u))(v) = \Gamma$, then $\forall w \in \mathbb{N}$ s.t. $w > u$, $h(r_i(w))(v) = \Gamma$.

(C1) At the start of time, an agent's commitments are the future-time parts of rules that fire against an empty history. Formally, $\forall i \in Ag, \psi \in c(r_i(0))$ iff $\exists \varphi \Rightarrow \psi \in rules_i$ s.t. $((), 0) \models \varphi$.

(C2) At time $u > 0$, an agent has a commitment ψ iff ψ is the future time part of a rule that fires at time u. Formally, $\forall i \in Ag, \forall u \in \mathbb{N}$, s.t. $u > 0$, we have $\psi \in c(r_i(u))$ iff $\exists \varphi \Rightarrow \psi \in rules_i$ s.t. $(h(r_i(u)), u) \models \varphi$.

(C3) All commitments are satisfied. Formally, $\forall i \in Ag, \forall u \in \mathbb{N}, \psi \in c(r_i(u))$ implies $(h(r_i(\omega)), u) \models \psi$. (Recall that ω is the first infinite ordinal.)

4 A Knowledge-Theoretic Semantics for Concurrent METATEM

In the preceding section, we defined the properties that must hold of any run $r \in R$ in order for r to be considered a run of a Concurrent METATEM system. However, it should be clear that attempting to prove the properties of a Concurrent METATEM system directly, using the semantic constructs presented in section 3.4, would at best be extremely awkward, and in general, would simply not be practicable. For this reason, we now introduce a logic for representing the properties of Concurrent METATEM systems. This logic, KL_n[1], is a *temporal logic of knowledge*: it is a PML-style temporal logic enriched by the addition of an indexed set of extra modal operators $\{K_i \mid i \in Ag\}$, that are used for representing the *knowledge* of agents. We show how certain formulae of KL_n correspond to properties of runs as discussed in the preceding section. This correspondence allows us to take a Concurrent METATEM system *sys*, and for this system, systematically derive a KL_n theory representing the properties of *sys*. Proving properties of *sys* then reduces to proving properties of the KL_n theory. This may be achieved either by using a suitable Hilbert-style axiom system for KL_n (such as that presented in [5, pp281–307]), or else by using an alternative proof method (such as the resolution procedure presented in [10]). We proceed by first defining the syntax and semantics of KL_n, and then by showing how certain KL_n formulae correspond to semantic properties of runs, as discussed in the proceeding section. We then give a short example, showing how some properties of the system presented in Figure 1 may be proven.

4.1 Syntax and Semantics

Syntactically, KL_n is the propositional temporal logic PML, augmented by an indexed set of modal operators K_i, one for each agent $i \in Ag$. With respect to semantics, the temporal connectives of KL_n are identical to those of PML. The semantics of the K_i operators are given in terms of *possible worlds*, in the normal modal logic fashion [4]. However, the semantics of the K_i operators are *grounded*, in that the knowledge accessibility relation for agent i is given a concrete interpretation in terms of the possible internal states of agent i [5, pp110–113]. Thus we associate an equivalence relation $\sim_i \subseteq P \times P$ with every agent $i \in Ag$, where this relation defined as follows: $(r, u) \sim_i (r', v)$ iff $r_i(u) = r'_i(v)$. If $(r, u) \sim_i (r', v)$, then we say that (r, u) is indistinguishable from (r', v) from the point of view of i, or, alternatively, that i carries exactly the same information in (r, u) as in (r', v).

A *model* for KL_n is simply a set of runs, $M \subseteq R$. (Unusually, we do not require an interpretation function in models, to tell us whether propositions are true or false at states: this is because we have all the information we require already in the model.) The semantics of KL_n are given via the usual satisfaction relation '$\models$', which holds between pairs of the form $(M, (r, u))$, (where M is a model, and (r, u) is a point in M), and formulae of KL_n. The rules defining this relation are given in Figure 3. Readers familiar with the semantics of normal modal and temporal logics will recognise that the rules

[1]The name comes from [13], where the complexity of the decision procedure for the future time fragment of the logic is discussed in detail. Related logics are discussed in [14, 20, 10], and in particular, [5].

$$
\begin{array}{llcll}
(M,(r,u)) & \models & \mathbf{true} & & \\
(M,(r,u)) & \models & p_i & \text{iff} & p_i \in h(r_i(u))(u) \quad (\text{where } p_i \in \Phi_i) \\
(M,(r,u)) & \models & p_e & \text{iff} & p_e \in r_e(u) \qquad\quad (\text{where } p_e \in \Phi_e) \\
(M,(r,u)) & \models & \neg\varphi & \text{iff} & (M,(r,u)) \not\models \varphi \\
(M,(r,u)) & \models & \varphi \vee \psi & \text{iff} & (M,(r,u)) \models \varphi \text{ or } (M,(r,u)) \models \psi \\
(M,(r,u)) & \models & K_i\varphi & \text{iff} & \forall (r',v) \in P,\ \text{if } (r,u) \sim_i (r',v) \text{ then } (M,(r',v)) \models \varphi \\
(M,(r,u)) & \models & \bigcirc\varphi & \text{iff} & (M,(r,u+1)) \models \varphi \\
(M,(r,u)) & \models & \circledcirc\varphi & \text{iff} & u > 0 \text{ and } (M,(r,u-1)) \models \varphi \\
(M,(r,u)) & \models & \varphi\,\mathcal{U}\,\psi & \text{iff} & \exists v \in I\!N \text{ s.t. } (u \leq v) \text{ and } (M,(r,v)) \models \psi \\
 & & & & \qquad\quad \text{and } \forall w \in \{u,\ldots,v-1\}, (M,(r,w)) \models \varphi \\
(M,(r,u)) & \models & \varphi\,\mathcal{S}\,\psi & \text{iff} & \exists v \in I\!N \text{ s.t. } (v < u) \text{ and } (M,(r,v)) \models \psi \\
 & & & & \qquad\quad \text{and } \forall w \in \{v+1,\ldots,u-1\} \cdot (M,(r,w)) \models \varphi
\end{array}
$$

Figure 3: Semantics of KL_n

for interpreting primitive propositions are somewhat unusual; below, we discuss the implications of defining the semantics in this way.

Derived temporal connectives: Semantic rules are only given for the temporal connectives $\bigcirc$, $\circledcirc$, $\mathcal{U}$, and $\mathcal{S}$; the remaining temporal connectives are introduced as abbreviations, as follows.

$$
\begin{array}{rcl@{\qquad}rcl}
\lozenge\varphi & \overset{\text{def}}{=} & \mathbf{true}\,\mathcal{U}\,\varphi & \blacklozenge\varphi & \overset{\text{def}}{=} & \mathbf{true}\,\mathcal{S}\,\psi \\
\square\varphi & \overset{\text{def}}{=} & \neg\lozenge\neg\varphi & \blacksquare\varphi & \overset{\text{def}}{=} & \neg\blacklozenge\neg\varphi \\
\varphi\,\mathcal{W}\,\psi & \overset{\text{def}}{=} & \varphi\,\mathcal{U}\,\psi \vee \square\varphi & \varphi\,\mathcal{Z}\,\psi & \overset{\text{def}}{=} & \varphi\,\mathcal{S}\,\psi \vee \blacksquare\varphi \\
\bullet\varphi & \overset{\text{def}}{=} & \neg\circledcirc\neg\varphi & \mathbf{start} & \overset{\text{def}}{=} & \bullet\mathbf{false}
\end{array}
$$

Validity and satisfiability: If $\varphi \in Form(KL_n)$ and $(M,(r,u)) \models \varphi$ for all points (r,u) in M, then we say that φ is valid in M, and indicate this by writing $M \models \varphi$. If φ is valid in all models, then we say that φ is valid *simpliciter*, and indicate this by writing $\models \varphi$. If $\neg\varphi$ is valid, then we say that φ is unsatisfiable.

4.2 Axiomatizing Concurrent METATEM

It should be obvious that formulae valid in the temporal logic PML, that underpins KL_n, will also be valid in KL_n. In addition, KL_n inherits as valid the KDT45 formulae that characterize normal modal logics with equivalence accessibility relations [5]:

$$
\begin{array}{ll}
\models ((K_i\varphi \Rightarrow \psi) \wedge (K_i\varphi)) \Rightarrow K_i\psi & (K) \\
\models K_i\varphi \Rightarrow \neg K_i\neg\varphi & (D) \\
\models K_i\varphi \Rightarrow \varphi & (T) \\
\models K_i\varphi \Rightarrow K_iK_i\varphi & (4) \\
\models \neg K_i\varphi \Rightarrow K_i\neg K_i\varphi & (5)
\end{array}
$$

We also have the usual *knowledge generalization rule* [5, pp50–51]: if $\models \varphi$ then $\models K_i\varphi$.

Given a Concurrent METATEM system *sys*, containing agents $\{(in_i, out_i, rules_i) \mid i \in Ag\}$, we can identify a set $R^{sys} \subseteq R$, containing just the runs of *sys*. Clearly, R^{sys} is also a model, and we shall therefore write M^{sys} for the model containing exactly the runs R^{sys}. Our aim in this section is to examine formulae φ, which have the property that $M^{sys} \models \varphi$, for all Concurrent METATEM systems *sys*. These formulae (or, more properly, these *formula schemas*) represent an *axiomatization* of Concurrent METATEM.

We begin by presenting two observations about the relationship between the truth of primitive propositions and the knowledge possessed by agents. (For the remainder of this section, it is assumed that *sys* is an arbitrary Concurrent METATEM system containing agents $\{(in_i, out_i, rules_i) \mid i \in Ag\}$, and that M^{sys} is the KL_n model corresponding to *sys*.)

Lemma 1

1. $M^{sys} \models p_i \Rightarrow K_i p_i$ *(*$p_i \in \Phi_i$*)*
2. $M^{sys} \not\models p_i \Rightarrow K_j p_i$ *(*$p_i \in \Phi_i$, $i \neq j$*)*

Proof: For (1), assume that $(M, (r, u)) \models p_i$. Then by the first semantic rule for primitive propositions, $p_i \in h(r_i(u))(u)$. We need to show that $(M, (r', v)) \models p_i$ for all (r', v) s.t. $(r, u) \sim_i (r, u)$. By the definition of $\sim_i$, we know that if $(r, u) \sim_i (r', v)$ then $r_i(u) = r'_i(v)$, and hence $h(r_i(u))(u) = h(r'_i(v))(v)$. Thus $p_i \in h(r'_i(v))(v)$ and so, by the first semantic rule for primitive propositions, $(M, (r, v)) \models p_i$, and we are done. For (2), it is easy to construct a counter example.

The first part of this lemma seems to be a very negative result indeed, since together with axiom (T), it appears to imply that the truth of a formula at some point is equivalent to knowledge of the formula at that point. If this were the case in general, then truth and knowledge would be identical. But the second part of this lemma illustrates that this is not the case: an agent *i* only knows about primitive propositions in Φ_i. An agent does not, in general, have knowledge of other agent's propositions. (This is in fact the reason for requiring agents to have different sets of primitive propositions: if we did not, then knowledge and truth would indeed be equivalent.) The first part of Lemma 1 can be generalized somewhat, as follows.

Lemma 2 $M^{sys} \models \varphi \Rightarrow K_i \varphi$ *(where* φ *is a propositional or past-time formula containing only primitive propositions from* Φ_i*).*
Proof: (Outline) By induction on the structure of φ. The first part of the base case, where $\varphi = p_i$, ($p_i \in \Phi_i$) is given by Lemma 1. For second part of the base case, where $\varphi = \neg p_i$, assume that $(M, (r, u)) \models \neg p_i$ and $p_i \in \Phi$. Then $p_i \notin h(r_i(u))(u)$. Clearly, for all $(r', v) \in P$ such that $(r', v) \sim_i (r, u)$, we will have $p_i \notin h(r'_i(v))(v)$, and so $(M, (r', v)) \models \neg p_i$. Hence $(M, (r, u)) \models K_i \neg p_i$. We leave the remainder of the proof as an exercise for the reader; the proof for past-time formulae requires the fact that agents maintain perfect histories — condition (H2), above.

The fact that agents maintain perfect histories gives us the following.

Lemma 3 $M^{sys} \models K_i \varphi \Rightarrow \bigcirc K_i \circledcirc \varphi$.
Proof: Follows from condition (H2) on runs, above. Assume that $(M, (r, u)) \models K_i \varphi$. We need to show $(M, (r, u+1)) \models K_i \circledcirc \varphi$; this follows from Lemma 2.

Agents are aware of the start of time.

Lemma 4 $M^{sys} \models$ **start** $\Rightarrow K_i$**start**.
Proof: Assume $(M, (r, u)) \models$ **start**. Then $u = 0$. We need to show that for all $(r', v) \sim_i (r, 0)$, we have $(M, (r', v)) \models$ **start**. From the from the assumption of synchronous execution, we know that if $(r', v) \sim_i (r, 0)$ then $v = 0$. The result follows easily.

Next, we give a variant of the axiom (T), above, which describes the process of message sending.

Lemma 5 *$M^{sys} \models K_i p_i \Rightarrow p_e$ (where $p_i \in out_i$).*
Proof: Follows from condition (E1) on runs, above. Assume that $(M, (r, u)) \models K_i p_i$ and $p_i \in out_i$. Then $(M, (r, u)) \models p_i$, and hence $p_i \in h(r_i(u))(u)$. Then, by definition, $p_i \in sent_i(r, u)$, and so by condition (E1) on runs, $p_e \in r_e(u)$, and hence by the second semantic rule for primitive propositions, $(M, (r, u)) \models p_e$.

Next, we have a *liveness* axiom, which states that messages sent are guaranteed to be received.

Lemma 6 *$M^{sys} \models p_e \Rightarrow \bigcirc \Diamond K_i p_i$ (where $p_i \in in_i$).*
Proof: Follows from condition (M1) on runs, above. Assume that $(M, (r, u)) \models p_e$ and $p_i \in in_i$. Then from condition (M1), $\exists v \in I\!N$, $v > u$, such that $p_i \in h(r_i(v))(v)$, hence $(M, (r, v)) \models p_i$ and thus $(M, (r, v)) \models K_i p_i$. Since $v > u$, we have $(M, (r, u)) \models \bigcirc \Diamond K_i p_i$, and we are done.

Similarly, we have a *safety* axiom, which states that if an agent receives a message, then it must have been sent at some previous time, and, moreover, that a message cannot be received more than once.

Lemma 7 *$M^{sys} \models \neg p_e \, \mathcal{Z} \, ((K_i p_i) \wedge \neg p_e) \Rightarrow \neg K_i p_i$ (where $p_i \in in_i$).*
Proof: (Outline) Follows from semantic condition (M2).

The final lemma characterizes an agent's knowledge about its program rules.

Lemma 8 *$M^{sys} \models (K_i \varphi) \Rightarrow (K_i \psi)$ (where $\varphi \Rightarrow \psi \in rules_i$).*
Proof: Assume that $(M, (r, u)) \models K_i \varphi$, and that $\varphi \Rightarrow \psi \in rules_i$. Then by condition (C2) on runs, above, we know that $\psi \in c(r_i(u))$, and thus for all $(r', v) \in P$ s.t. $(r', v) \sim_i (r, u)$, we have $\psi \in c(r'_i(v))$. From condition (C3) on runs, we know that if $\psi \in c(r'_i(v))$ then $(h(r'_i(\omega)), v) \models \psi$, and so $(M, (r', v)) \models \psi$. Hence $(M, (r, u)) \models K_i \psi$.

4.3 A Short Verification Example

In this section, we indicate how the knowledge-theoretic semantics developed in this paper can be used to verify the properties of Concurrent METATEM systems. The example we give is a simple liveness property for the system in Figure 1. Let this system be called $s1$. Then:

Theorem 1 *Eventually, agent $rp1$ will know that it has been given the resource. Formally, $M^{s1} \models$* **start** *$\Rightarrow \Diamond K_{rc1} give1$.*
Proof: See Figure 4; propositional reasoning is indicated by PR, and temporal reasoning is indicated by TR.

5 Related Work

In this section, we briefly consider related work. With respect to multi-agent development languages, many environments have been reported in the literature (a survey is presented in [21]). While Concurrent METATEM is the only multi-agent language we are aware of that is based directly on executing temporal logic, some languages with similar properties have been developed. For example, Shoham's AGENT0 language [19] allows a user to program agents in terms of *commitment rules*, which are similar to Concurrent METATEM program rules. AGENT0 rules contain no tense operators, however, and explicitly allow an agent to refer to the beliefs and commitments both of itself and other agents. The CONGOLOG

1.	$\textbf{start} \Rightarrow K_{rc1}\textbf{start}$	[Lemma 4]
2.	$K_{rc1}\textbf{start} \Rightarrow K_{rc1} ask1_{rc1}$	[Lemma 8]
3.	$\textbf{start} \Rightarrow K_{rc1} ask1_{rc1}$	[1, 2, PR]
4.	$K_{rc1} ask_{rc1} \Rightarrow ask1_e$	[Lemma 5]
5.	$ask1_e \Rightarrow \bigcirc\Diamond K_{rp} ask1_{rp}$	[Lemma 6]
6.	$\textbf{start} \Rightarrow \bigcirc\Diamond K_{rp} ask1_{rp}$	[3, 4, 5, PR]
7.	$K_{rp} ask1_{rp} \Rightarrow K_{rp}\Diamond give1_{rp}$	[Lemma 8]
8.	$\textbf{start} \Rightarrow \bigcirc\Diamond K_{rp}\Diamond give1_{rp}$	[6, 7, PR, TR]
9.	$\textbf{start} \Rightarrow \Diamond K_{rp}\Diamond give1_{rp}$	[8, TR]
10.	$K_{rp}\Diamond give1_{rp} \Rightarrow \Diamond give1_{rp}$	[9, Axiom (T)]
11.	$K_{rp}\Diamond give1_{rp} \Rightarrow \Diamond K_{rp} give1_{rp}$	[10, Lemma 1, TR, PR]
12.	$\textbf{start} \Rightarrow \Diamond K_{rp} give1_{rp}$	[9, 11, PR]
13.	$K_{rp} give1 \Rightarrow give1_e$	[Lemma 5]
14.	$\textbf{start} \Rightarrow \Diamond give1_e$	[12, 13, PR]
15.	$give1_e \Rightarrow \bigcirc\Diamond K_{rc1} give1$	[Lemma 6]
16.	$\textbf{start} \Rightarrow \Diamond K_{rc1} give1$	[14, 15, TR, PR]

Figure 4: Proof of Theorem 1

language is a multi-agent logic programming language that is also related to Concurrent METATEM [15]. Whereas the underlying program model for Concurrent METATEM is temporal logic, the corresponding program model in CONGOLOG is McCarthy's situation calculus. Neither AGENT0 nor CONGOLOG have been given a formal semantics, although the close relationship between CONGOLOG and the situation calculus makes such a semantics a reasonable proposition. Another closely related area of work is that on Belief-Desire-Intention (BDI) architectures and theories, by Rao and Georgeff [18]. They have a particular software architecture, known as the Procedural Reasoning System (PRS), and an associated family of BDI logics, developed in order to give an abstract semantics to the architecture. However, the logics cannot be used to give a semantics to the PRS in the formal sense that we have described here: the PRS does not (and was never intended to) operationalize the corresponding BDI logic.

Finally, we comment on the relationship between our work and the *knowledge-based programs* proposed by Fagin *et al* [5, pp233–269]. A knowledge-based program has the general form:

```
case of
    if $t_1 \wedge k_1$ do $a_1$
    if $t_2 \wedge k_2$ do $a_2$
    ...
end case
```

where each t_i is a propositional logic formula, and k_i is a boolean combination of formulae of the form $K_i\varphi$. To execute such a program, an agent must continually perform a cycle of evaluating the conditions, finding all those that fire, and then (non-deterministically) executing one of the corresponding actions. However, unlike temporal logic, it is not known what operational model might be used to execute knowledge formulae, hence knowledge-based programs cannot be directly executed in the way that Concurrent METATEM temporal rules are executed. Fagin *et al* comment that 'we do not have a general methodology for implementing knowledge-based programs' [5, p379]. It would be interesting to consider the extent to which the knowledge-theoretic semantics developed for Concurrent METATEM in this paper could be used to provide such a methodology.

6 Closing remarks

In this paper, we have provided a bottom-up semantics for the Concurrent METATEM multi-agent programming language. Using this semantics, we developed a temporal logic of knowledge, which we then used to axiomatize the properties of Concurrent METATEM systems. We gave a brief example, to illustrate how the logic might be used to verify properties of Concurrent METATEM systems. There are several questions that need to be addressed in future work. The most important of these is that of *completeness*. We have proved that the axiom system presented above is *sound* with respect to the set of models of Concurrent METATEM systems, but we have not addressed the issue of *completeness*. Some completeness results have been obtained for KL_n-like temporal logics of knowledge [5, pp281–307], but not for systems as complex as that for Concurrent METATEM. Another obvious area of work is proof methods for temporal logics of knowledge; we have already begun to develop such methods [20, 10].

Acknowledgments

I would like to thank Michael Fisher for his comments on this paper. This work was supported by the EPSRC under research grant GR/K57282.

References

[1] G. Agha. *ACTORS: A Model of Concurrent Computation in Distributed Systems.* The MIT Press: Cambridge, MA, 1986.

[2] G. Agha, P. Wegner, and A. Yonezawa, editors. *Research Directions in Concurrent Object-Oriented Programming.* The MIT Press: Cambridge, MA, 1993.

[3] H. Barringer, M. Fisher, D. Gabbay, G. Gough, and R. Owens. METATEM: A framework for programming in temporal logic. In *REX Workshop on Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness (LNCS Volume 430)*, pages 94–129. Springer-Verlag: Heidelberg, Germany, June 1989.

[4] B. Chellas. *Modal Logic: An Introduction.* Cambridge University Press: Cambridge, England, 1980.

[5] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning About Knowledge.* The MIT Press: Cambridge, MA, 1995.

[6] M. Fisher. A survey of Concurrent METATEM — the language and its applications. In D. M. Gabbay and H. J. Ohlbach, editors, *Temporal Logic — Proceedings of the First International Conference (LNAI Volume 827)*, pages 480–505. Springer-Verlag: Heidelberg, Germany, July 1994.

[7] M. Fisher. Representing and executing agent-based systems. In M. Wooldridge and N. R. Jennings, editors, *Intelligent Agents: Theories, Architectures, and Languages (LNAI Volume 890)*, pages 307–323. Springer-Verlag: Heidelberg, Germany, January 1995.

[8] M. Fisher. Towards a semantics for Concurrent METATEM. In M. Fisher and R. Owens, editors, *Executable Modal and Temporal Logics (LNAI Volume 897)*, pages 86–102. Springer-Verlag: Heidelberg, Germany, 1995.

[9] M. Fisher and M. Wooldridge. Specifying and verifying distributed intelligent systems. In M. Filgueiras and L. Damas, editors, *Progress in Artificial Intelligence — Sixth Portuguese Conference on Artificial Intelligence (LNAI Volume 727)*, pages 13–28. Springer-Verlag: Heidelberg, Germany, October 1993.

[10] M. Fisher, M. Wooldridge, and C. Dixon. A resolution-based proof method for temporal logics of knowledge and belief. In D. M. Gabbay and H.-J. Ohlbach, editors, *Proceedings of the First International Conference on Formal and Applied Practical Reasoning*. Springer-Verlag: Heidelberg, Germany, 1996. To appear.

[11] D. Gabbay. Declarative past and imperative future. In B. Banieqbal, H. Barringer, and A. Pnueli, editors, *Proceedings of the Colloquium on Temporal Logic in Specification (LNCS Volume 398)*, pages 402–450. Springer-Verlag: Heidelberg, Germany, 1989.

[12] R. Goldblatt. *Logics of Time and Computation (CSLI Lecture Notes Number 7)*. Center for the Study of Language and Information, Ventura Hall, Stanford, CA 94305, 1987. (Distributed by Chicago University Press).

[13] J. Y. Halpern and M. Y. Vardi. The complexity of reasoning about knowledge and time. I. Lower bounds. *Journal of Computer and System Sciences*, 38:195–237, 1989.

[14] S. Kraus and D. Lehmann. Knowledge, belief and time. *Theoretical Computer Science*, 58:155–174, 1988.

[15] Y. Lésperance, H. J. Levesque, F. Lin, D. Marcu, R. Reiter, and R. B. Scherl. Foundations of a logical approach to agent programming. In M. Wooldridge, J. P. Müller, and M. Tambe, editors, *Intelligent Agents II (LNAI 1037)*, pages 331–346. Springer-Verlag: Heidelberg, Germany, 1996.

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

[17] A. Pnueli. Specification and development of reactive systems. In *Information Processing 86*. Elsevier Science Publishers B.V.: Amsterdam, The Netherlands, 1986.

[18] A. S. Rao and M. Georgeff. BDI Agents: from theory to practice. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 312–319, San Francisco, CA, June 1995.

[19] Y. Shoham. Agent-oriented programming. *Artificial Intelligence*, 60(1):51–92, 1993.

[20] M. Wooldridge and M. Fisher. A decision procedure for a temporal belief logic. In D. M. Gabbay and H. J. Ohlbach, editors, *Temporal Logic — Proceedings of the First International Conference (LNAI Volume 827)*, pages 317–331. Springer-Verlag: Heidelberg, Germany, July 1994.

[21] M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.