International Journal of Cooperative Information Systems,

Vol. 0, No. 0 (1993) 000–000

ON THE FORMAL SPECIFICATION AND VERIFICATION OF MULTI-AGENT SYSTEMS

MICHAEL FISHER and MICHAEL WOOLDRIDGE
Department of Computing, Manchester Metropolitan University
Manchester M1 5GD, United Kingdom

Received 31 September 1995
Revised 31 May 1996
Communicated by Michael Huhns

ABSTRACT
This article describes first steps towards the formal specification and verification of multi-agent systems, through the use of temporal belief logics. The article first describes Concurrent METATEM, a multi-agent programming language, and then develops a logic that may be used to reason about Concurrent METATEM systems. The utility of this logic for specifying and verifying Concurrent METATEM systems is demonstrated through a number of examples. The article concludes with a brief discussion on the wider implications of the work, and in particular on the use of similar logics for reasoning about multi-agent systems in general.

Keywords: Formal specification and verification; multi-agent systems.

1. Introduction

Multi-agent systems technology is arguably the most vibrant and exciting research and development area in computer science today. It is now widely accepted that this emerging technology will have a key role to play in the development of twenty-first century computer systems. To date, research on multi-agent systems has focussed primarily on foundational issues, such as coordination and the coherence of cooperative activity[4]. However, as the techniques and tools of multi-agent systems research migrate from the lab to the office of the everyday computer worker, so we might expect to find greater emphasis placed on the developmental aspects of multi-agent systems. This change of emphasis is likely to be evidenced in several ways. For example, we might expect to see the emergence of software tools to support the development of multi-agent systems. (Gasser articulated the need for such tools as long ago as 1987[18].) However, in addition, we might also expect to see the adaptation of mainstream software engineering tools and techniques to multi-agent development. One important tradition in mainstream software engineering is *formal methods*: the application of mathematical techniques to the design and construction of software[20]. Our aim, in this article, is to begin to consider the use of such formal methods for multi-agent systems.

Specifically, this article presents Concurrent METATEM an expressive new programming language for multi-agent systems, and considers how one might go about formally *specifying* and *verifying* systems implemented in Concurrent METATEM. The remainder of this article is structured as follows. In the following section, we outline the background to, and motivation for, our work. In section 2, we describe

Concurrent METATEM in more detail. In section 3, we develop a Temporal Belief Logic (TBL), which is used, in section 4, to axiomatize the properties of Concurrent METATEM systems. Examples of the use of the logic for specifying Concurrent METATEM systems are presented in section 5; verification examples are presented in section 6. Some comments and conclusions are presented in section 7; in particular, we discuss the implications of our work for reasoning about multi-agent systems in general.

1.1. Background and Motivation

Our starting point is the notion of a *reactive system*:

> Reactive systems are systems that cannot adequately be described by the *relational* or *functional* view. The relational view regards programs as functions ... from an initial state to a terminal state. Typically, the main role of reactive systems is to maintain an interaction with their environment, and they must therefore be described (and specified) in terms of their on-going behaviour ... [E]very concurrent system ... must be studied by behavioural means. This is because each individual module in a concurrent system is a reactive subsystem, interacting with its own environment which consists of the other modules. [25]

Multi-agent systems are reactive, in precisely this sense:

- the applications for which a multi-agent approach seems well suited, (e.g., distributed sensing[6]), are typically non-terminating, and therefore cannot simply be described by the functional view;
- multi-agent systems are necessarily concurrent, and as Pnueli observes (above), each agent should therefore be considered as a reactive system.

Although the fact that multi-agent systems are reactive systems in the sense described above has been recognised by theorists in multi-agent systems, (notably Singh[28], and Rao-Georgeff[26]), it has, as yet, had relatively little impact on the practice of multi-agent systems. In particular, the various software tools and languages that have been proposed for multi-agent development do not incorporate any features that make them especially well suited to the development of reactive systems (see[33] for survey of such systems).

In this article, we describe Concurrent METATEM, a multi-agent programming language, in which the notion of reactivity is central. A Concurrent METATEM system contains a number of concurrently executing agents, which are able to communicate through message passing. Each agent's behaviour is generated by directly executing a *temporal logic* specification of its desired behaviour. In a 1977 paper, Pnueli proposed temporal logic as a tool for reasoning about reactive systems[24]. The justification for this approach is that when describing a reactive system, we often wish to express properties such as 'if a request is sent, then a response is eventually given'. Such properties are easily and elegantly expressed in temporal logic. As we observed above, each agent in a Concurrent METATEM system directly executes a temporal logic specification of its desired behaviour. Reactivity is therefore at the very heart of Concurrent METATEM.

Although Concurrent METATEM is itself based on temporal logic, one cannot use such a logic to directly reason about Concurrent METATEM systems. This is because each agent in a Concurrent METATEM system is a symbolic AI system in its own right: it contains a set of explicitly represented (temporal logic) formulae which it manipulates in order to decide what to do. Thus, to represent the behaviour of a Concurrent METATEM system, we require some way of representing the formulae that each agent is manipulating at each moment in time. One way of doing this would be to use a first-order temporal meta-language (as in [2]). However, meta-languages are notationally cumbersome, are prone to inconsistency, and can be confusing to use. What we propose instead is to use a multi-modal language, which contains both temporal connectives and an indexed set of modal *belief* operators, one for each agent. These belief operators will be used to describe the formulae that each agent manipulates (see[21] for a detailed exposition on the use of belief logics without a temporal component for describing AI systems).

We have now set the scene for the remainder of the article. In section 2, we describe Concurrent METATEM in more detail. We then develop a temporal belief logic, and show how it can be used to reason about Concurrent METATEM systems.

1.2. 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. We use a VDM-style notation for manipulating functions[20]. In particular, if f is a function, then by $f \dagger \{x \mapsto y\}$, we mean the function that is the same as f except that it maps x to y. If S is a set, then by $\wp(S)$, we mean the powerset of S.

2. Concurrent METATEM

In this section, we informally introduce the Concurrent METATEM language. We begin by considering the basic components of agents, then describe the temporal logic used for defining agent behaviour and the basic execution mechanism for such descriptions, and finally give a simple example of a multi-agent system in Concurrent METATEM.

Note that Concurrent METATEM is a descendent of METATEM, details of both the philosophy and execution mechanism underlying which can be found in[1,12]. The core Concurrent METATEM language is described in[8], with a survey of the language and its applications provided in[9].

2.1. Agent Components

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 — although in principle, any computational engine could be provided for an agent, as long as the engine is consistent with the interface, in Concurrent METATEM the approach used is based on the METATEM paradigm of executable temporal logic.

2.1.1. Agent Interfaces

A Concurrent METATEM agent interface consists of three components:

- a unique *agent identifier* (or just agent id), which names the agent;
- a set of symbols defining what messages will be accepted by the agent — these are termed *environment predicates*; and

- a set of symbols defining messages that the agent may send — these are termed *component predicates*.

For example, the interface definition of a 'stack' agent[8] might be:

$$stack(pop, push)[popped, full]$$

Here, *stack* is the agent id that names the agent, $\{pop, push\}$ is the set of environment predicates, and $\{popped, full\}$ is the set of component predicates. The intuition behind this definition 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 $full$.

2.2. Specifying Agent Behaviour

The computational engine of each agent in Concurrent METATEM, which defines how the agent will act, is defined by a temporal logic specification given as a set of temporal 'rules'. This approach is based on the METATEM paradigm of executable temporal logics[1]. The idea which informs this approach is that of directly executing a declarative agent specification, where this specification is given as a set of *program rules*, which are temporal logic formulae of the form:

antecedent about past and present $\Rightarrow$ 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*[16]. The rules that define an agent's behaviour can be animated by directly executing the temporal specification under a suitable operational model[10].

To make the discussion more concrete, we will now introduce a quantified temporal logic, called First-order METATEM Logic (FML), in which the individual temporal rules that are used to specify an agent's behaviour will be given. (A complete definition of FML is given in[1].)

FML is essentially classical first-order predicate logic augmented by a set of modal connectives for referring to the *temporal ordering* of events. FML 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 FML 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 { ◉, ●, ◈, ■, $\bigcirc$, $\Diamond$, $\Box$} are unary; the remainder are binary. In addition to these temporal connectives, FML 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 FML, 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:

- $\odot\varphi$ 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, $\odot\varphi$ can not 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**'.

2.2.1. Example Formulae

Before proceeding, we present some simple examples of FML formulae Note that these example formulae are not in Concurrent MetateM rule form, but are arbitrary FML formulae. However, the separation results of Gabbay tell us that the Concurrent MetateM rules are in fact equal in expressive power to arbitrary temporal formulae: any FML formula may be rewritten into Concurrent MetateM rules. First, the following formula expresses the fact that, 'while Concurrent MetateM is not currently famous, it will be at some time in the future':

$$\neg famous(Concurrent\ MetateM) \land \Diamond famous(Concurrent\ MetateM).$$

The second example expresses the fact that 'sometime in the past, PROLOG was famous':

$$\blacklozenge\, famous(prolog).$$

We might want to state that 'if something is famous then, at some time in the future, it will cease to be famous' (i.e., that fame is not permanent):

$$\forall T \cdot\ famous(T)\ \Rightarrow\ \Diamond\,\Box\neg famous(T).$$

The final example expresses a statement that frequently occurs in human negotiation, namely 'we are not friends until you apologise':

$$(\neg friends(me, you))\,\mathcal{U}\,apologise(you).$$

2.3. 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'.* This execution mechanism is considered in more detail below.

We call any formula of FML that refers to the present or past a *history formula*, and any formula referring to the present or future a *commitment formula*. Formulae of the form

$$\textit{history formula} \quad \Rightarrow \quad \textit{commitment formula}$$

are called *rules*. It is sets of such rules that comprise agent specifications. Formally, an agent program in Concurrent METATEM is a formula

$$\Box \bigwedge_{i=1}^{n} \forall \bar{x}_i \cdot \; P_i(\bar{x}_i) \;\Rightarrow\; \exists \bar{z}_i \cdot \; F_i(\bar{x}_i, \bar{z}_i)$$

where $\bar{x}_i$ and $\bar{z}_i$ are vectors of variables, P_i is a (non-strict) past-time temporal formula and F_i is a (non-strict) future-time formula, for all $i \in \{1, \ldots, n\}$.

In order to execute such rules, the computational engine for an agent continually executes the following cycle:

1. Update the *history* of the agent by receiving messages (i.e., environment predicates) 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 $\Diamond$-formulae on the current cycle, in which case unsatisfied $\Diamond$-formulae are carried over to the next cycle as commitments.

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 (see[1,12]). If a contradiction is generated, the agent is allowed a limited form of backtracking in order to explore alternative choices[8]. The basic limitation of this activity is that externally recognised actions can not be undone. Thus, backtracking is essentially used to explore possibilities before committing to some course of action by broadcasting a message.

2.3.1. Communication

A natural question to ask is: how do agents send messages? When a predicate in an agent becomes *true*, it is compared against that agent's interface (see above);

*There are obvious similarities between the execution cycle of an agent and production systems[27] — but there are also significant differences. The reader is cautioned against taking the analogy too seriously.

$rp(ask1, ask2)[give1, give2]$:
1. $\circledcirc ask1 \Rightarrow \Diamond give1;$
2. $\circledcirc ask2 \Rightarrow \Diamond give2;$
3. $\textbf{start} \Rightarrow \Box\neg(give1 \wedge give2).$

$rc1(give1)[ask1]$:
1. $\textbf{start} \Rightarrow ask1;$
2. $\circledcirc ask1 \Rightarrow ask1.$

$rc2(ask1, give2)[ask2]$:
1. $\circledcirc(ask1 \wedge \neg ask2) \Rightarrow ask2.$

Figure 1: A Simple Concurrent METATEM System

if it is one of the agent's *component predicates*, then that predicate is broadcast as a message to all other agents. On receipt of a message, each agent attempts to match the predicate against the environment predicates in their interface. If there is a match then they add the predicate to their history, prefixed by a '$\circledcirc$' operator, indicating that the message has just been received.

The reader should note that although the use of only broadcast message-passing may seem restrictive, standard point-to-point message-passing can easily be simulated by adding an extra 'destination' argument to each message. Also, the use of broadcast message-passing as the communication mechanism gives us the ability to define more adaptable and flexible systems. We will not develop this argument further; the interested reader is urged to either consult our earlier work on Concurrent METATEM[8,13,14,10], or relevant work showing the utility of broadcast and multi-cast mechanisms[5,3,23].

2.4. An Example Concurrent METATEM *System*

To illustrate Concurrent METATEM in more detail, we present in Figure 1 a simple example system (outlined originally in[1])[†] The system contains three agents: rp, $rc1$, and $rc2$. The agent rp is a 'resource producer': it can '*give*' to only one agent at a time (rule 3), and will commit to eventually *give* to any agent that *ask*s (rules 1 and 2). Agent rp will only accept messages $ask1$ and $ask2$, and can only send $give1$ and $give2$ messages.

Agent $rc1$ has an interface which states that it will only accept $give1$ messages, and can only send $ask1$ messages. The rules for agent $rc1$ ensure that an $ask1$ message is sent on every cycle — this is because **start** is satisfied at the beginning of time, thus firing rule 1, while $\circledcirc ask1$ will then be satisfied on the next cycle, thus firing rule 2, and so on. Thus, $rc1$ asks for the resource on every cycle, using an $ask1$ message.

The interface for agent $rc2$ states that it will accept both $ask1$ and $give2$ messages, and can send $ask2$ messages. The single rule for agent $rc2$ ensures that an $ask2$ message is sent on every cycle where, on its previous cycle, it did not send an $ask2$ message, but received an $ask1$ message (from agent $rc1$).

Having specified the network of agents as in Figure 1, we can directly execute the agent specification in order to animate the system. We can observe that the system has certain properties, for example:

1. agents $rc1$ and $rc2$ will ask rp for the resource infinitely often;

[†]Note that in the interests of readability, rule numbers have been introduced to facilitate reference and only the propositional version of this example has been provided.

2. every time rp is 'asked', it must eventually 'give' to the corresponding asker.

Given these observations regarding the agent specifications, we can also deduce that

3. agent rp will give the resource to both $rc1$ and $rc2$ infinitely often.

In section 6, we formally verify this behaviour.

3. The Temporal Belief Logic (TBL)

In this section, we define the Temporal Belief Logic (TBL) that we subsequently use to axiomatize the properties of Concurrent METATEM systems. Formally, this logic is a linear discrete first-order temporal logic, enriched by the addition of an indexed set of unary modal *belief* connectives. We begin by defining the syntax of TBL in section 3.1, and its semantics in section 3.2. Proof theoretic aspects are briefly considered in section 3.3, and in section 4, we show how TBL may be used to axiomatize the properties of Concurrent METATEM systems. Note that since TBL is based on FML, details of which may be found in[1,7], our presentation, though complete, will be somewhat terse.

3.1. Syntax

Definition 1 *The alphabet of TBL contains the following symbols:*

1. *The truth constant,* '**true**'*;*
2. *A denumerable set* $Pred$*, of* predicate symbols*;*
3. *A denumerable set* Fun*, of* function symbols*;*
4. *A denumerable set* Var*, of* variable symbols*;*
5. *A denumerable set* $Ag = \{1, \ldots, n\}$ *of* agent identifiers*;*
6. *The binary temporal connectives '*$\mathcal{U}$*' (until) and '*$\mathcal{S}$*' (since), and unary temporal connectives '*$\bigcirc$*' (next) and '*$\odot$*' (last);*
7. *The binary classical connective '*$\vee$*' (or), and unary classical connective '*$\neg$*' (not);*
8. *The equality symbol, '*$=$*';*
9. *The universal quantifier, '*$\forall$*';*
10. *The square brackets, '*$]$*' and '*$[$*', parentheses '*$)$*' and '*$($*', and raised point '*$\cdot$*'.*

For convenience, we shall generally write predicates as lowercase strings of roman letters. Function symbols and variables will be written as strings starting with an uppercase roman letter. The syntax of TBL is defined by the grammar in Figure 2. The reader will note that the main difference between TBL and FML is the belief modality, which allows us to construct formulae of the form $[i]\varphi$, where φ is an FML formula, and i is an agent identifier. A TBL formula of the form $[i]\varphi$ should be read 'φ is in i's current state'. Thus, if φ is a history formula, this would say that φ was in i's history; if φ was a rule, this would say that φ was one of i's rules, and if φ were a commitment, it would say that i was committed to φ.

Associated with every predicate and function symbol is a natural number, known as its *arity*, which indicates how many arguments it takes. We assume that arity is defined by a function $arity : Pred \cup Var \rightarrow I\!N$, and that predicate and function symbols are only applied to the appropriate number of arguments. Predicate symbols of arity 0 are known as *proposition symbols*, and function symbols of arity 0 are known as *constants*.

$$
\begin{array}{rcl}
\langle \mathit{fun\text{-}sym} \rangle & ::= & \text{any element of } \mathit{Fun} \\
\langle \mathit{fun\text{-}term} \rangle & ::= & \langle \mathit{fun\text{-}sym} \rangle \\
 & | & \langle \mathit{fun\text{-}sym} \rangle(\langle \mathit{term} \rangle, \ldots, \langle \mathit{term} \rangle) \\
\langle \mathit{var\text{-}term} \rangle & ::= & \text{any element of } \mathit{Var} \\
\langle \mathit{term} \rangle & ::= & \langle \mathit{fun\text{-}term} \rangle \\
 & | & \langle \mathit{var\text{-}term} \rangle \\
\langle \mathit{pred\text{-}sym} \rangle & ::= & \text{any element of } \mathit{Pred} \\
\langle \mathit{atom} \rangle & ::= & \langle \mathit{pred\text{-}sym} \rangle \\
 & | & \langle \mathit{pred\text{-}sym} \rangle(\langle \mathit{term} \rangle, \ldots, \langle \mathit{term} \rangle) \\
\langle \mathit{ag\text{-}id} \rangle & ::= & \text{any element of } \mathit{Ag} \\
\langle \mathit{wff} \rangle & ::= & \mathbf{true} \\
 & | & \langle \mathit{atom} \rangle \\
 & | & (\langle \mathit{term} \rangle = \langle \mathit{term} \rangle) \\
 & | & \neg \langle \mathit{wff} \rangle \\
 & | & \langle \mathit{wff} \rangle \vee \langle \mathit{wff} \rangle \\
 & | & \forall \langle \mathit{var\text{-}term} \rangle \cdot \langle \mathit{wff} \rangle \\
 & | & [\langle \mathit{ag\text{-}id} \rangle] \langle \mathit{wff} \rangle \\
 & | & \bigcirc \langle \mathit{wff} \rangle \\
 & | & \circledcirc \langle \mathit{wff} \rangle \\
 & | & \langle \mathit{wff} \rangle \, \mathcal{U} \, \langle \mathit{wff} \rangle \\
 & | & \langle \mathit{wff} \rangle \, \mathcal{S} \, \langle \mathit{wff} \rangle
\end{array}
$$

Figure 2: Syntax of TBL

Definition 2 *The set $Term$, of all* terms *is defined as follows:*

1. *if $X \in Var$, then $X \in Term$;*
2. *if $T \in Fun$, $arity(T) = n$, and $\{\tau_1, \ldots, \tau_n\} \subseteq Term$, then $T(\tau_1, \ldots, \tau_n) \in Term$.*

We use τ (with decorations: $\tau', \tau_1, \ldots$) to stand for arbitrary members of $Term$.

It is sometimes convenient to deal just with constants.

Definition 3 *Let $Const = \{T \mid T \in Fun \text{ and } arity(T) = 0\}$ be the set of constants.*

3.2. Semantics

We present an overview of the key semantic features of TBL, before formally defining its semantics. First, we recall that the semantics of TBL represent a generalisation of FML semantics[1], which in turn represent a generalisation of the semantics of classical first-order logic (see, e.g.,[17]). Rather than give a detailed, first-principles discussion on the semantics of TBL, we pre-suppose familiarity with classical first-order semantics, and focus on the less standard aspects of TBL: its *temporal model* and the representation of *belief.*

Since, as we just noted, TBL is a generalisation of FML, the temporal model that underpins it is $(\mathbb{N}, <)$, i.e., the natural numbers ordered by the usual 'less than' relation. This model, although by no means universally accepted, is nevertheless widely used in mainstream computer science for representing the semantics of concurrent and distributed systems (see, e.g.,[7]).

With respect to *belief*, we eschew the 'standard' approach of possible worlds semantics[19], in favour of a much simpler model based on the scheme proposed by Konolige[21]. Recall that the state of a Concurrent METATEM agent at any time may be characterised as a set of FML formulae. We require a belief modality in order to represent these formulae; the TBL formula $[i]\varphi$ will be used to represent the fact that agent i has formula φ in its current state. We therefore adopt a simple model of belief, in which the semantics of belief modalities are given in terms of *belief sets*. At each moment in time, each agent is simply assigned a *belief set* of FML formulae representing its internal state at that moment. A formula $[i]\varphi$ is satisfied at some time $u \in \mathbb{N}$ if, and only if, φ is a member of the belief set assigned to agent i at time u. This simple scheme is certainly not adequate as a model of *human* belief. However, as we shall demonstrate in the remainder of this article, it is sufficiently rich to represent the internal state of Concurrent METATEM agents.

Definition 4 *A* domain, *D, is a non empty set. If D is a domain and $u \in \mathbb{N}$, then by D^u we mean the set of u-tuples over D.*

In order to interpret TBL, we need various functions that associate symbols of the language with semantic objects. The first of these is an *interpretation for predicates*.

Definition 5 *A* predicate interpretation, *Φ, is a function*

$$\Phi : Pred \times \mathbb{N} \to \wp(\bigcup_{u \in \mathbb{N}} D^u)$$

such that $\forall q \in Pred, \forall n, u \in \mathbb{N}$, if $arity(q) = n$ then $\Phi(q, u) \subseteq D^n$ (i.e., predicate interpretations preserve arity).

Definition 6 *An* interpretation for functions, *F, is a second-order function*

$$F : Fun \to (\bigcup_{u \in \mathbb{N}} D^u \to D)$$

For reasons that will become clear below, we shall make the assumption that every element of the domain has exactly one constant (function of arity 0) that names it. These constants may be thought of as *standard names* for the corresponding domain elements. A variable assignment simply associates variables with elements of the domain.

Definition 7 *A* variable assignment, *V, is a function $V : Var \to D$.*

We now introduce a derived function $[\![\ldots]\!]_{V,F}$, which gives the *denotation* of an arbitrary term with respect to a particular interpretation for functions and variable assignment.

Definition 8 *If V is a variable assignment and F is a function interpretation, then by $[\![\ldots]\!]_{V,F}$, we mean the function $[\![\ldots]\!]_{V,F} : Term \to D$, which interprets arbitrary terms relative to V and F:*

$$[\![\tau]\!]_{V,F} \stackrel{\text{def}}{=} \begin{cases} F(f)([\![\tau_1]\!]_{V,F}, \ldots, [\![\tau_n]\!]_{V,F}) & \textit{where } \tau \textit{ is } f(\tau_1, \ldots, \tau_n) \\ V(\tau) & \textit{otherwise.} \end{cases}$$

Since V and F will generally be clear from context, reference to them will often be suppressed. We can now define models for TBL.

Definition 9 *A* model, *M, for TBL, is a structure*

$$M = (D, F, \Phi, BS)$$

where:

- *D is a domain;*

$$
\begin{array}{lll}
\langle M, V, u \rangle \models \mathbf{true} & & \\
\langle M, V, u \rangle \models q(\tau_1, \ldots, \tau_n) & \text{iff} & \langle [\![\tau_1]\!], \ldots, [\![\tau_n]\!] \rangle \in \Phi(q, u) \\
\langle M, V, u \rangle \models (\tau = \tau') & \text{iff} & [\![\tau]\!] = [\![\tau']\!] \\
\langle M, V, u \rangle \models \neg\varphi & \text{iff} & \langle M, V, u \rangle \not\models \varphi \\
\langle M, V, u \rangle \models \varphi \vee \psi & \text{iff} & \langle M, V, u \rangle \models \varphi \;\text{ or }\; \langle M, V, u \rangle \models \psi \\
\langle M, V, u \rangle \models \forall x \cdot \varphi & \text{iff} & \langle M, V \dagger \{x \mapsto d\}, u \rangle \models \varphi \text{ for all } d \in D \\
\langle M, V, u \rangle \models [i]\varphi & \text{iff} & \varphi^{\eta_{V,F}} \in BS(u, i) \\
\langle M, V, u \rangle \models \bigcirc\varphi & \text{iff} & \langle M, V, u+1 \rangle \models \varphi \\
\langle M, V, u \rangle \models \odot\varphi & \text{iff} & u > 0 \;\text{ and }\; \langle M, V, u-1 \rangle \models \varphi \\
\langle M, V, u \rangle \models \varphi\,\mathcal{U}\,\psi & \text{iff} & \exists v \in \mathbb{N} \text{ s.t. } (v \geq u) \;\text{ and }\; \langle M, V, v \rangle \models \psi, \\
 & & \text{and } \forall w \in \mathbb{N}, \;\text{ if }\; (u \leq w < v) \;\text{ then }\; \langle M, V, w \rangle \models \varphi \\
\langle M, V, u \rangle \models \varphi\,\mathcal{S}\,\psi & \text{iff} & \exists v \in \mathbb{N} \text{ s.t. } (v < u) \;\text{ and }\; \langle M, V, v \rangle \models \psi, \\
 & & \text{and } \forall w \in \mathbb{N}, \;\text{ if }\; (v < w < u) \;\text{ then }\; \langle M, V, w \rangle \models \varphi
\end{array}
$$

Figure 3: Semantics of TBL

- $F : Fun \to (\bigcup_{n \in \mathbb{N}} D^n \to D)$ *interprets functions;*
- $\Phi : Pred \times \mathbb{N} \to \wp(\bigcup_{n \in \mathbb{N}} D^n)$ *interprets predicates; and*
- $BS : \mathbb{N} \times Ag \to \wp(Form(TBL))$ *assigns every agent a belief set at every time point.*

Before we can present the formal semantics of the language, we require two further definitions, in order to deal with the semantics of belief modalities, and in particular, with the semantics of *quantifying-in* to modal belief contexts.

Definition 10 *If F is an interpretation for functions and V is a variable assignment, then let $\eta_{V,F} : D \to Const$ be the function that gives the standard name of every element of the domain: $\eta_{V,F}(d) = T$ iff $T \in Const$ and $[\![T]\!]_{V,F} = d$.*

Note that this function will be well defined, since we required that F allocate exactly one constant to every element of the domain. The use of the naming function η is similar to the technique developed by Konolige, in his deduction model of belief[21].

Definition 11 *If φ is a formula of TBL, F is an interpretation for functions, and V is a variable assignment, then by $\varphi^{\eta_{V,F}}$, we mean the formula obtained from φ by systematically substituting $\eta_{V,F}([\![X]\!]_{V,F})$ for every free variable X that occurs in φ.*

As usual, we define the formal semantics of the language via the satisfaction relation, '$\models$'. For TBL, this relation holds between triples of the form $\langle M, V, u \rangle$, (where M is a model, V is a variable assignment, and $u \in \mathbb{N}$ is a temporal index into M), and TBL-formulae. The rules defining the satisfaction relation are given in Figure 3. Satisfiability and validity for TBL are defined in the standard way.

The remaining temporal connectives of TBL are introduced as abbreviations:

$$
\begin{array}{rcl@{\qquad}rcl}
\Diamond\varphi & \stackrel{\text{def}}{=} & \mathbf{true}\,\mathcal{U}\,\varphi & \blacklozenge\,\varphi & \stackrel{\text{def}}{=} & \mathbf{true}\,\mathcal{S}\,\varphi \\
\Box\varphi & \stackrel{\text{def}}{=} & \neg\Diamond\neg\varphi & \blacksquare\varphi & \stackrel{\text{def}}{=} & \neg\blacklozenge\,\neg\varphi \\
\varphi\,\mathcal{W}\,\psi & \stackrel{\text{def}}{=} & \varphi\,\mathcal{U}\,\psi \vee \Box\varphi & \varphi\,\mathcal{Z}\,\psi & \stackrel{\text{def}}{=} & \psi\,\mathcal{S}\,\psi \vee \blacksquare\varphi.
\end{array}
$$

3.3. Proof Theory

The proof theory of FML has been examined exhaustively elsewhere (see, for example,[7]). Here, we simply identify some axioms and inference rules that are later used in our proofs.

$$\vdash \Box(\varphi \Rightarrow \psi) \Rightarrow (\Box\varphi \Rightarrow \Box\psi) \tag{1}$$
$$\vdash \Box(\varphi \Rightarrow \psi) \Rightarrow (\Diamond\varphi \Rightarrow \Diamond\psi) \tag{2}$$
$$\vdash \bigcirc(\varphi \Rightarrow \psi) \Rightarrow (\bigcirc\varphi \Rightarrow \bigcirc\psi) \tag{3}$$
$$\vdash \Diamond\Diamond\varphi \Leftrightarrow \Diamond\varphi \tag{4}$$
$$\vdash (\mathbf{start} \Rightarrow \Box\varphi) \Rightarrow \Box\varphi \tag{5}$$
$$\vdash \Box(\varphi \Rightarrow \bigcirc\varphi) \Rightarrow (\varphi \Rightarrow \Box\varphi) \tag{6}$$
$$\vdash \Box\varphi \;\Rightarrow\; \varphi \tag{7}$$
$$\vdash (\bigcirc \odot \varphi \;\Rightarrow\; \psi) \;\Leftrightarrow\; (\varphi \Rightarrow \psi) \tag{8}$$
$$\vdash (\odot \bigcirc \varphi \;\Rightarrow\; \psi) \;\Rightarrow\; (\varphi \Rightarrow \psi) \tag{9}$$
$$\text{From } \vdash \varphi \text{ infer } \vdash \Box\varphi \tag{10}$$
$$\text{From } \vdash \varphi \text{ infer } \vdash \Diamond\varphi \tag{11}$$
$$\text{From } \vdash \varphi \text{ infer } \vdash \bigcirc\varphi \tag{12}$$

We claim that these axioms and inference rules are sound, (i.e., that $\vdash \varphi$ implies that φ is valid, and that the inference rules preserve validity). However, we do not claim completeness.

4. Axiomatizing Concurrent MetateM

In this section, we show how TBL may be used to *axiomatize* the properties of Concurrent MetateM systems. This involves extending the basic TBL proof system outlined above to account for the particular properties of the Concurrent MetateM systems that we intend to verify properties of.

The first axiom we add describes the conditions under which an agent will send a message: if a predicate becomes 'true' inside an agent, and the predicate symbol appears in the agent's component predicate list within its interface, then that predicate is broadcast to all other agents. (Note the use of ordinary FML predicates to describe messages.) So if P is one of i's component predicates, then the following axiom holds.

$$\vdash ([i]P) \Rightarrow \Diamond P \tag{13}$$

The second axiom deals with how agents *receive* messages: if a message is broadcast, and the predicate symbol of that message appears in an agent's environment predicate list, then that message is accepted by the agent, which subsequently 'believes' that the predicate was true. So, if P is one of i's environment predicates, then the following axiom holds.

$$\vdash \; P \Rightarrow \Diamond[i]\odot P \tag{14}$$

Notice the implicit assumption that messages are guaranteed to be delivered. Internal commitments will also be eventually achieved.

$$\vdash \; [i]\Diamond P \Rightarrow \Diamond[i]P \tag{15}$$

The next axiom states that agents maintain accurate histories.

$$\vdash \; ([i]\varphi) \Rightarrow [i]\bigcirc \odot \varphi \tag{16}$$

To simplify the proofs, we will assume that all agents in a Concurrent MetateM system execute synchronously, i.e., the 'execution steps' of each agent match. This simplification allows us to add the following *synchronisation axioms*. (Note that, without this simplification, each agent would be executing under a distinct local clock, and so proving properties of such a system becomes *much* more difficult, though possible[8,11].)

$$\vdash (\bigcirc[i](\varphi \Rightarrow \psi)) \Leftrightarrow ([i]\bigcirc(\varphi \Rightarrow \psi)) \tag{17}$$
$$\vdash (\circledcirc[i](\varphi \Rightarrow \psi)) \Leftrightarrow ([i]\circledcirc(\varphi \Rightarrow \psi)) \tag{18}$$

Now, for every rule, R, in an agent, i, we add the following axiom showing that once the agent has started executing, the rule is always applicable.

$$\vdash [i]\mathbf{start} \Rightarrow \square[i]R \tag{19}$$

Since there is no direct interaction between the temporal connectives and the '$[i]$' operators in the basic TBL system, we add the following.

$$\vdash [i](P \Rightarrow F) \Rightarrow (([i]P) \Rightarrow ([i]F)) \tag{20}$$

This characterises the fact that the '$\Rightarrow$' operator in Concurrent METATEM follows the same logical rules as standard implication.

Finally, to simplify the proofs still further, we will assume that all agents commence execution at the same moment, denoted by the global '**start**' operator. Thus, for every agent, i, we add the following axiom.

$$\vdash \mathbf{start} \Rightarrow [i]\mathbf{start} \tag{21}$$

5. Specification Examples

In this section, we consider some example multi-agent systems, and present descriptions of the behaviour of agents in terms of Concurrent METATEM. In section 6, we consider the formal verification of certain properties, within our framework, for each of these example systems.

5.1. Resource Controller

The first example system we consider is that presented in section 2.4 and defined in Figure 1, namely the simple 'resource controller' system. To recap, this system consists of three agents: 'rp', which is a 'resource producer' that guarantees to (eventually) give a resource to any agent that asks for it, but will only allocate one resource at a time; '$rc1$', which continually asks for a resource for itself; and '$rc2$', which asks for a resource if it sees $rc1$ asking for a resource, but has not asked for one itself in the previous cycle.

We consider the properties that we might wish to verify of this system in section 6.1.

5.2. An Abstract Distributed Problem Solving System

A common form of multi-agent system is based upon the idea of distributed problem solving[29]. Here, we consider a simple abstract distributed problem solving system, in which a single agent, called *top*, broadcasts a problem to a group of problem solvers. Some of these problem solvers can solve the problem completely, and some will reply with a solution. We define such a Concurrent METATEM system in Figure 4. Here, $solvera$ can solve a different problem from the one *top* poses, while $solverb$ can solve the desired problem, but does not announce the fact (as $solution1$ is not a component predicate for $solverb$); $solverc$ can solve the problem posed by *top*, and will *eventually* reply with the solution.

$$\boxed{\begin{array}{l} top(solution1)[problem1, solved1]: \\ \quad 1. \quad \mathbf{start} \Rightarrow \Diamond problem1; \\ \quad 2. \quad \odot\, solution1 \Rightarrow solved1. \end{array}}$$

$$\boxed{\begin{array}{l} solvera(problem2)[solution2]: \\ \quad 1. \quad \odot\, problem2 \Rightarrow solution2. \end{array}}$$

$$\boxed{\begin{array}{l} solverb(problem1)[solution2]: \\ \quad 1. \quad \odot\, problem1 \Rightarrow \Diamond solution1. \end{array}}$$

$$\boxed{\begin{array}{l} solverc(problem1)[solution1]: \\ \quad 1. \quad \odot\, problem1 \Rightarrow \Diamond solution1. \end{array}}$$

Figure 4: A Distributed Problem Solving System

Again, we will verify some properties of the above system in section . We will also consider the refinement of individual agents, (e.g., a single problem-solver), into groups of agents with the same properties.

5.3. *The Contract Net*

Finally, we look at a more complex multi-agent system in more detail. This system contains a group of agents cooperating via a contract net-like protocol. Below, we consider the specification (and implementation, once executed) of this multi-agent system using Concurrent METATEM. Throughout, we assume familiarity with the contract net[29].

Recall that a *manager* agent announces a particular task (or set of tasks) that it requires undertaking. The other agents in the system each have a specific set of capabilities and can, based upon these, bid for the contract to undertake all, or part of, a particular task. We first describe the notions of *tasks* and *capabilities* that are used throughout this specification. These will be represented by *internal predicates.*

5.3.1. Internal Predicates

An individual capability is simply represented as a constant. For example, if an agent is able to move, speak and jump, the capabilities of the agent would be represented by the *capabilities* predicate within the agent's definition:

$$capabilities(Agent, [Move, Speak, Jump])$$

A task is represented simply as the function *task* applied to certain arguments:

$$task(Name, Description, Requirements, Originator)$$

where

- *Name* is the name of the task;
- *Description* is the general description of the task (we will not provide any further details regarding this);
- *Requirements* is the list of capabilities required of an agent for it to be able to carry out the task;

Predicate	Meaning
$announce(Task)$	announces that a particular task is available for bids
$bid(Task, Bidder)$	a bid for a particular task
$award(Task, Awardee)$	awards the contract for a particular task
$completed(Task, By, Result)$	signals the completion of a particular task

Table 1: Message Predicates

- *Originator* is the agent who announced the task.

In particular, we will define the predicates *competent*, *busy*, *bidded*, and *most-preferable* as follows. (We assume that each agent awarding contracts has an internal selection procedure which is characterised by the predicate *preferable*.)

$$\begin{array}{rcl} capabilities(A, Cap) \wedge (Cap \cap Req \neq \emptyset) & \Rightarrow & competent(A, task(T, D, Req, O)) \\ (\neg completed(T, \mathsf{self}, R))\, \mathcal{S}\, award(T, \mathsf{self}) & \Rightarrow & busy(\mathsf{self}) \\ (\neg award(T, A))\, \mathcal{S}\, announce(T) \wedge bid(T, X) & \Rightarrow & bidded(T, X) \\ \neg \exists Y \cdot preferable(Y, X) \wedge bidded(T, Y) & \Leftrightarrow & most\text{-}preferable(T, X) \end{array}$$

Note that 'self' refers to the id of the agent in which the predicates occur.

5.3.2. Messages

In addition to the above internal predicates, the system utilises a set of basic message predicates; these are summarised in Table 1.

In general, the interface to an individual agent within this system is defined as

$$agent(announce, bid, award, completed)[announce, bid, award, completed].$$

Thus, every agent is capable of being both a manager and a contractor. If we wish to introduce agent types for manager and contractor, then we can define their interfaces as follows:

$$manager(bid, completed)[announce, award]$$

$$contractor(announce, award)[bid, completed]$$

In the remainder of this section, we outline the Concurrent METATEM rules that can be used to describe the behaviour of a simple agent taking part in our system. The behaviours of the agent will be split into categories relating to *task announcement*, *bidding*, the *award* of contracts, and the *completion* of contracts.

5.3.3. Task Announcement

Initially, a prospective manager agent just announces its first task, using the following rule.

$$\mathbf{start} \Rightarrow announce(task(Name, Desc, Req, \mathsf{self})) \qquad (A1)$$

If an agent has been contracted to carry out a task, yet is unable to complete it, then it must sub-contract part of the task. The rule used in this case utilises '*split*', a predicate that splits a task appropriately, given the agent's capabilities (i.e., a task is split into two tasks, the first of which the agent is able to complete, the second of which it must attempt to subcontract).

$$\circledcirc \begin{pmatrix} award(task(N, D, Req, O), \mathsf{self}) \\ \wedge\ capabilities(\mathsf{self}, Cap) \\ \wedge\ (Req - Cap \neq \emptyset) \end{pmatrix} \Rightarrow \begin{pmatrix} split(task(N, D, Req, O), T1, T2) \\ \wedge\ announce(T2) \\ \wedge\ \Diamond \exists R \cdot\ result(T1, R) \end{pmatrix} \qquad (A2)$$

5.3.4. Bidding

The first rule in the bidding process states that an agent should only define a *possible* task as one that has been announced (and not yet awarded) and which the agent has the capabilities to undertake (at least partially).

$$((\neg award(T, A))\, \mathcal{S}\, announce(T) \wedge competent(A, T)) \quad \Leftrightarrow \quad possible(A, T) \qquad (B1)$$

Given this rule, another basic property of bidding agents is that they should not bid for tasks that are not possible.

$$\neg possible(\mathsf{self}, T) \quad \Rightarrow \quad \neg bid(T, \mathsf{self}) \qquad (B2)$$

We can then add a variety of rules depending upon the behaviour required for the agent. For example, the following rules (B3) and (B4) can be used in order to ensure that each agent only bids for one task at a time.

$$possible(\mathsf{self}, T) \Rightarrow \exists Y \cdot\ bid(Y, \mathsf{self}) \qquad (B3)$$

Note that rule (B3) should be read in conjunction with (B2), which together with (B1) ensures that an agent will not bid for a task that has not been announced, or that it is not competent for.

$$(bid(X, \mathsf{self}) \wedge bid(Y, \mathsf{self})) \Rightarrow X = Y \qquad (B4)$$

The following rule is needed if we restrict the agent's behaviour so it cannot bid while it is actively undertaking a task.

$$busy(\mathsf{self}) \quad \Rightarrow \quad \neg bid(X, \mathsf{self}) \qquad (B5)$$

Finally, if we require that an agent is able to bid for every task, at any time, we would replace rules $(B3)$, $(B4)$, and $(B5)$ by the following rule.

$$possible(\mathsf{self}, T) \quad \Rightarrow \quad bid(T, \mathsf{self}) \qquad (B6)$$

5.3.5. Awarding Contracts

Given that a manager agent has announced a task then, after a certain time, it must decide which bidding agent to award the contract to. To achieve this, we simply use the following rule.

$$\circledcirc\, (bidded(T, Y) \wedge most\text{-}preferable(T, Y)) \quad \Leftrightarrow \quad award(T, Y) \qquad (W1)$$

Thus, the choice amongst those agents that have bid for the contract is made by consulting the manager's internal list of preferences, and no award is made if no bids have been received.

5.3.6. Task Completion

There are two rules relating to the completion of a task, the first for tasks solely carried out within the agent, the second for tasks that were partially sub-contracted.

$$\left(\begin{array}{c}(\neg completed(T, \mathsf{self}, X))\,\mathcal{S}\,award(T, \mathsf{self}) \\ \wedge \quad \circledcirc result(T, R)\end{array}\right) \Rightarrow completed(T, \mathsf{self}, R) \qquad (C1)$$

$$\left(\begin{array}{c}(\neg completed(T, \mathsf{self}, X))\,\mathcal{S}\,award(T, \mathsf{self}) \\ \wedge \ \blacklozenge\, split(T, T1, T2) \\ \wedge \ \blacklozenge\, result(T1, R1) \\ \wedge \ \blacklozenge\, completed(T2, By, R2)\end{array}\right) \Rightarrow completed(T, \mathsf{self}, R1 \cup R2) \qquad (C2)$$

Thus, in the first case, once the agent has produced a result, the completion of the task is announced, while in the second case completion is only announced once the agent has completed its portion of the task and the sub-contractor reports completion of the remainder.

This more complex specification indicates the type of multi-agent system that may readily be represented using Concurrent METATEM. Again, we wish to verify that certain properties hold of this specification — it is this issue that we consider in section .

6. Verification Examples

In this section, we show how TBL can be used to reason about Concurrent METATEM systems, in particular the examples provided in section 5. In addition to checking the static properties of agent specifications, there are a variety of temporal properties that might be important. These properties can be broadly categorised as follows[22]:

- *safety properties*, which intuitively state that nothing *bad* can happen;
- *liveness properties*, which intuitively state that something *will* happen; and
- *fairness properties*, which intuitively state that, if a choice is reached an infinite number of times, then it is discharged *fairly*.[‡]

In additional to this broad categorisation, there is a more structural classification of the properties that need to be established, based upon whether they can be established just within a single agent specification, or whether the whole system must be considered:

- *local properties* are those that can be verified from a single agent specification;
- *global properties* require not only the specifications of several agents to be examined, but also the axioms defining constraints upon communication and relative execution.

Thus, global properties are usually required when we wish to establish that coordination or cooperation between agents occurs, whereas local properties relate more to the attributes of individual agents.

In the proofs that follow, we use the notation $\{\mathrm{S}\} \vdash \varphi$ to represent the statement '*system S satisfies property* φ'. Also, as the majority of the proof steps involve applications of the *Modus Ponens* inference rule, we will omit reference to this rule.

6.1. Resource Controller

[‡]For a discussion of fairness, see[15].

1.	$[rc1]\textbf{start} \;\Rightarrow\; \square[rc1](\textbf{start} \Rightarrow ask1)$	(rule 1 in $rc1$)
2.	$\textbf{start} \;\Rightarrow\; \square[rc1](\textbf{start} \Rightarrow ask1)$	(axiom 19, 1)
3.	$\square[rc1](\textbf{start} \Rightarrow ask1)$	(axiom 5, 2)
4.	$[rc1](\textbf{start} \Rightarrow ask1)$	(axiom 7, 3)
5.	$[rc1]\textbf{start} \;\Rightarrow\; [rc1]ask1$	(axiom 20, 4)
6.	$\textbf{start} \;\Rightarrow\; [rc1]ask1$	(axiom 19, 5)
7.	$[rc1]\textbf{start} \;\Rightarrow\; \square[rc1](\circledcirc ask1 \Rightarrow ask1)$	(rule 2 in $rc1$)
8.	$\textbf{start} \;\Rightarrow\; \square[rc1](\circledcirc ask1 \Rightarrow ask1)$	(axiom 19, 7)
9.	$\square[rc1](\circledcirc ask1 \Rightarrow ask1)$	(axiom 5, 8)
10.	$[rc1](\circledcirc ask1 \Rightarrow ask1)$	(axiom 7, 9)
11.	$[rc1]\circledcirc ask1 \;\Rightarrow\; [rc1]ask1$	(axiom 20, 10)
12.	$\bigcirc([rc1]\circledcirc ask1 \;\Rightarrow\; [rc1]ask1)$	(inf. rule 12, 11)
13.	$\bigcirc[rc1]\circledcirc ask1 \;\Rightarrow\; \bigcirc[rc1]ask1$	(axiom 3, 12)
14.	$\bigcirc\circledcirc[rc1]ask1 \;\Rightarrow\; \bigcirc[rc1]ask1$	(axiom 18, 13)
15.	$[rc1]ask1 \;\Rightarrow\; \bigcirc[rc1]ask1$	(axiom 8, 14)
16.	$\square([rc1]ask1 \;\Rightarrow\; \bigcirc[rc1]ask1)$	(inf. rule 10, 15)
17.	$[rc1]ask1 \;\Rightarrow\; \square[rc1]ask1$	(axiom 6, 16)
18.	$\textbf{start} \;\Rightarrow\; \square[rc1]ask1$	(6, 17)

Figure 5: Proof of Lemma 1

We begin by proving some properties of the simple resource controller outlined in section 2.4 and presented in Figure 1. This multi-agent system, which we shall refer to as S1, consists of three agents: a resource producer (rp), and two resource consumers ($rc1$ and $rc2$).

The first property we prove is that the agent $rc1$, once it has commenced execution, satisfies the commitment $ask1$ on every cycle.

Lemma 1 $\{\mathrm{S1}\} \vdash \textbf{start} \;\Rightarrow\; \square[rc1]ask1.$

(The proof of this lemma is given in Figure 5; we shall omit all other proofs from this section, due to space restrictions.) Using this result, it is not difficult to establish that the message $ask1$ is then sent infinitely often.

Lemma 2 $\{\mathrm{S1}\} \vdash \square\lozenge ask1.$

Similarly, we can show that any agent that is *listening* for $ask1$ messages, in particular rp, will receive them infinitely often.

Lemma 3 $\{\mathrm{S1}\} \vdash \textbf{start} \;\Rightarrow\; \square\lozenge[rp]\circledcirc ask1.$

Now, since we know that $ask1$ is one of rp's environment predicates, then we can show that once both rp and $rc1$ have started, the resource will be given to $rc1$ infinitely often.

Lemma 4 $\{\mathrm{S1}\} \vdash \textbf{start} \;\Rightarrow\; \square\lozenge give1.$

Similar properties can be shown for $rc2$. Note, however, that we require knowledge about $rc1$'s behaviour in order to reason about $rc2$'s behaviour.

Lemma 5 $\{\mathrm{S1}\} \vdash \textbf{start} \;\Rightarrow\; \square\lozenge[rp]\circledcirc ask2.$

Given this, we can derive the following result.

Lemma 6 $\{\mathrm{S1}\} \vdash \textbf{start} \;\Rightarrow\; \square\lozenge give2.$

1.	$[top]\mathbf{start} \;\Rightarrow\; \Box[top](\mathbf{start} \Rightarrow \Diamond problem1)$	(rule 1 in *top*)
2.	$\mathbf{start} \;\Rightarrow\; \Box[top](\mathbf{start} \Rightarrow \Diamond problem1)$	(axiom 19, 1)
3.	$\Box[top](\mathbf{start} \Rightarrow \Diamond problem1)$	(axiom 5, 2)
4.	$[top](\mathbf{start} \Rightarrow \Diamond problem1)$	(axiom 7, 3)
5.	$[top]\mathbf{start} \;\Rightarrow\; [top]\Diamond problem1$	(axiom 20, 4)
6.	$\mathbf{start} \;\Rightarrow\; [top]\Diamond problem1$	(axiom 19, 5)
7.	$\mathbf{start} \;\Rightarrow\; \Diamond[top]problem1$	(axiom 15, 6)
8.	$\mathbf{start} \;\Rightarrow\; \Diamond\Diamond problem1$	(axiom 13, 7)
9.	$\mathbf{start} \;\Rightarrow\; \Diamond problem1$	(axiom 4, 8)
10.	$problem1 \;\Rightarrow\; \Diamond[solverc]\odot problem1$	(axiom 14)
11.	$\Diamond problem1 \;\Rightarrow\; \Diamond\Diamond[solverc]\odot problem1$	(axiom 2, 10)
12.	$\mathbf{start} \;\Rightarrow\; \Diamond\Diamond[solverc]\odot problem1$	(9, 11)
13.	$[solverc]\mathbf{start} \;\Rightarrow\; \Box[solverc](\odot problem1 \Rightarrow \Diamond solution1)$	(rule 1 in *solverc*)
14.	$\mathbf{start} \;\Rightarrow\; \Box[solverc](\odot problem1 \Rightarrow \Diamond solution1)$	(axiom 19, 13)
15.	$\Box[solverc](\odot problem1 \;\Rightarrow\; \Diamond solution1)$	(axiom 5, 14)
16.	$[solverc](\odot problem1 \;\Rightarrow\; \Diamond solution1)$	(axion 7, 15)
17.	$[solverc]\odot problem1 \;\Rightarrow\; [solverc]\Diamond solution1)$	(axiom 20, 16)
18.	$\Diamond[solverc]\odot problem1 \;\Rightarrow\; \Diamond[solverc]\Diamond solution1$	(axiom 2, 17)
19.	$\mathbf{start} \;\Rightarrow\; \Diamond[solverc]\odot problem1$	(axiom 4, 12)
20.	$\mathbf{start} \;\Rightarrow\; \Diamond[solverc]\Diamond solution1$	(18, 19)
21.	$\mathbf{start} \;\Rightarrow\; \Diamond\Diamond[solverc]solution1$	(axiom 15, 20)
22.	$\mathbf{start} \;\Rightarrow\; \Diamond[solverc]solution1$	(axiom 4, 21)
23.	$\mathbf{start} \;\Rightarrow\; \Diamond\Diamond solution1$	(axiom 13, 22)
24.	$\mathbf{start} \;\Rightarrow\; \Diamond solution1$	(axiom 4, 23)

Figure 6: Proof of Lemma 7

Finally, we can show the desired behaviour of the system; compare this to result (3) that we informally deduced in section 2.4.

Theorem 1 $\{\mathrm{S1}\} \vdash \mathbf{start} \;\Rightarrow\; (\Box\Diamond give1 \;\wedge\; \Box\Diamond give2).$

6.2. An Abstract Distributed Problem Solving System

We now consider properties of the simple distributed problem-solving system presented in section . If we call this system S2, then we can prove the following.

Lemma 7 $\{\mathrm{S2}\} \vdash \mathbf{start} \;\Rightarrow\; \Diamond solution1.$

(See Figure 6 for a proof of this.) We can then use this result to prove that the system solves the required problem:

Theorem 2 $\{\mathrm{S2}\} \vdash \mathbf{start} \;\Rightarrow\; \Diamond solved1.$

We briefly consider a refinement of the above system where *solverc* is replaced by two agents who together can solve *problem*1, but cannot manage this individually. These agents, called *solverd* and *solvere* are defined in Figure 7.

Thus, when *solverd* receives the problem it cannot do anything until it has heard from *solvere*. When *solvere* receives the problem, it broadcasts the fact that it can solve part of the problem (i.e., it broadcasts *solution*1.2). When *solverd* sees

$solverd(problem1, solution1.2)[solution1]$:
1. $(\odot solution1.2 \ \wedge \ \blacklozenge problem1) \Rightarrow \Diamond solution1.$

$solvere(problem1)[solution1.2]$:
1. $\odot problem1 \Rightarrow \Diamond solution1.2.$

Figure 7: Refined Problem Solving Agents

this, it knows it can solve the other part of the problem and broadcasts the whole solution. Thus, given these new agents we can prove the following (the system is now called S3).

Theorem 3 $\quad \{\mathrm{S3}\} \vdash \ \mathbf{start} \ \Rightarrow \ \Diamond solved1.$

6.3. The Contract Net

We now give an outline of how various properties of the simple Contract Net system presented in section may be established. Rather than giving detailed proofs, we present a precis of the proof process for a selection of properties that the system should exhibit. Throughout, we will refer to this system as $S4$.

Theorem 4 *If at least one agent bids for a task, then the contract will eventually be awarded to one of the bidders.*

As this is a global property of the system, not restricted to a particular agent, then it can be represented logically as follows.

$$\{\mathrm{S4}\} \vdash \ \forall T \cdot \ \exists A \cdot \ bid(T, A) \ \Rightarrow \ \exists B \cdot \ \Diamond award(T, B)$$

In order to prove this statement, we start with the assumption that an agent a has a task t for which it bids:

$$[a]bid(t, a).$$

Now, from the axioms governing communication between agents, we know that, if a particular predicate is a component predicate then it will eventually be broadcast (i.e., axiom (13) in section 4). This, together with the above, ensures that

$$\Diamond bid(t, a). \qquad (CN1)$$

Now, we know that, once broadcast, such a message will eventually reach all agents who wish to receive messages of this form (from axiom (14)). Thus, we can deduce that

$$bid(t, a) \Rightarrow \Diamond [m] bid(t, a)$$

where m is the manager agent for this particular task. Similarly, we can derive

$$bid(t, a) \Rightarrow \Diamond [m] bidded(t, a).$$

By the definition of contract allocation given by axiom (W1), we know that for some bidding agent p (the 'most preferable'), then the manager will eventually award the contract to p:

$$bid(t, a) \Rightarrow \Diamond [m] award(t, p).$$

Using this, together with (CN1), above, and axioms (4) and (2), we can derive

$$\diamondsuit[m]award(t,p).$$

Finally, as this information is broadcast, we can derive the global statement that, given $[a]bid(t,a)$,

$$\exists B \cdot \ \diamondsuit award(t,B)$$

thus establishing the theorem.

Theorem 5 *Agents do not bid for tasks that they cannot contribute to.*

In logical terms, this is simply

$$\{\mathrm{S4}\} \vdash \ \forall T \cdot \ \forall A \cdot \ (announce(T) \wedge (\neg competent(A,T))) \ \Rightarrow \neg bid(T,A)$$

If we know that, for some task t and agent a, where the task t has been announced, yet the agent a is not competent to perform the task, then we know by rule $(B1)$ that

$$[a]\neg possible(a,t).$$

Then, by rule $(B2)$, we can derive the fact that agent a will not bid for the task, i.e.,

$$[a]\neg bid(t,a).$$

Theorem 6 *Agents do not bid unless they believe there has been a task announcement.*

Again, this can be formalised as

$$\{\mathrm{S4}\} \vdash \ \forall T \cdot \ \forall A \cdot \ bid(T,A) \ \Rightarrow \ \blacklozenge\, announce(T).$$

In order to prove this statement, we simply show that for there to have been a bid, a particular agent a must have considered the task, t, possible:

$$[a]possible(t,a)$$

and, for this to occur, then by $(B1)$

$$[a](\neg award(t,B))\,\mathcal{S}\,announce(t)$$

which in turn implies

$$[a]\blacklozenge\, announce(t).$$

As *announce* is an environment predicate for agent a then it must be the case that the appropriate message was broadcast at some time in the past:

$$\blacklozenge\, announce(t).$$

Theorem 7 *Managers award the contract for a particular task to at most one agent.*

This can be simply represented by

$$award(T,A) \ \wedge \ award(T,B) \ \Rightarrow \ A = B.$$

The proof of this follows simply from $(W1)$ which states that the 'most preferable' bidder is chosen. The definition of 'most preferable' in turn utilises the linear ordering provided by the *preferable* predicate.

7. Concluding Remarks

In this article, we have described Concurrent METATEM, a multi-agent programming language, and developed a Temporal Belief Logic for reasoning about Concurrent METATEM systems. In effect, we used the logic to develop a semantics for the language; this approach did not leave us with a complete proof system for Concurrent METATEM, (since we made the assumption of synchronous action). However, the approach has the advantage of simplicity when compared to other methods for defining the semantics of the language (such as those based on dense temporal logic[8], or first-order temporal meta-languages[2]).

More generally, logics similar to that developed herein can be used to reason about a wide class of multi-agent systems: those in which agents have a classic 'symbolic AI' architecture. Such systems typically employ explicit symbolic representations, which are manipulated in order to plan and execute actions. A belief logic such as that described in this article seems appropriate for describing these representations (see also[21]). A temporal component to the logic seems to be suitable for describing reactive systems, of which multi-agent systems are an example. In other work, we have developed a family of temporal belief logics, augmented by modalities for describing the actions and messages of individual agents, and demonstrated how these logics can be used to specify a wide range of cooperative structures[30,31].

With respect to future work, there are a number of issues that require investigation. First, and most obviously, the Concurrent METATEM language needs further development and evaluation. At the time of writing, a C++ implementation of Concurrent METATEM has been developed, which supports only limited quantification, does not allow full backtracking, and does not have equality. This prototype is currently undergoing further development. With respect to formal aspects, an obvious problem is the automation of proof methods for TBL; automatic proof methods are required if verification is to become a realistic possibility. Elsewhere, we describe one approach to proof for a TBL-like logic, based on semantic tableaux[32]. This proof method has been implemented, but this implementation is not sufficiently powerful to make it usable for everyday work. Finally, we need to look at the *completeness* of Concurrent METATEM semantics.

7.1. Acknowledgments

The authors were supported by the EPSRC under grant GR/K57282.

This article incorporates work presented at the Twelfth International Workshop on Distributed Artificial Intelligence (Hidden Valley, Pennsylvania, May 1993), the 1993 Portuguese Conference on Artificial Intelligence (Porto, October 1993) and the Second International Working Conference on Cooperating Knowledge-Based Systems (Keele, June 1994).

1. H. Barringer, M. Fisher, D. Gabbay, G. Gough, and R. Owens. METATEM: A Framework for Programming in Temporal Logic. In *Proceedings of REX Workshop on Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness*, Mook, Netherlands, June 1989. (Published in *Lecture Notes in Computer Science*, volume 430, Springer Verlag).
2. H. Barringer, M. Fisher, D. Gabbay, and A. Hunter. Meta-Reasoning in Executable Temporal Logic. In J. Allen, R. Fikes, and E. Sandewall, editors, *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR)*, Cambridge, Massachusetts, April 1991. Morgan Kaufmann.
3. K. Birman. The Process Group Approach to Reliable Distributed Computing. Techanical Report TR91-1216, Department of Computer Science, Cornell University, USA, July 1991.

4. A. H. Bond and L. Gasser, editors. *Readings in Distributed Artificial Intelligence*. Morgan Kaufmann, 1988.
5. A. Borg, J. Baumbach, and S. Glazer. A Message System Supporting Fault Tolerance. In *Proceedings of the Ninth ACM Symposium on Operating System Principles*, pages 90–99, New Hampshire, October 1983. ACM. (In ACM Operating Systems Review, vol. 17, no. 5).
6. E. Durfee. *Coordination of Distributed Problem Solvers*. Kluwer Academic Press, 1988.
7. E. A. Emerson. Temporal and Modal Logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 996–1072. Elsevier, 1990.
8. M. Fisher. Concurrent METATEM — A Language for Modeling Reactive Systems. In *Parallel Architectures and Languages, Europe (PARLE)*, Munich, Germany, June 1993. Springer-Verlag.
9. 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.
10. 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.
11. M. Fisher. Towards a Semantics for Concurrent METATEM. In M. Fisher and R. Owens, editors, *Executable Modal and Temporal Logics*. Springer-Verlag, 1995.
12. M. Fisher and R. Owens. From the Past to the Future: Executing Temporal Logic Programs. In *Proceedings of Logic Programming and Automated Reasoning (LPAR)*, St. Petersberg, Russia, July 1992. (Published in *Lecture Notes in Computer Science*, volume 624, Springer Verlag).
13. M. Fisher and M. Wooldridge. Executable Temporal Logic for Distributed A.I. In *Proceedings of the Twelfth International Workshop on Distributed Artificial Intelligence*, Hidden Valley, Pennsylvania, May 1993.
14. M. Fisher and M. Wooldridge. A logical approach to simulating societies. In N. Gilbert and R. Conte, editors, *Artificial Societies: The Computer Simulation of Social Life*, pages 268–284. UCL Press: London, 1995.
15. N. Francez. *Fairness*. Springer-Verlag: Heidelberg, Germany, 1986.
16. D. Gabbay. Declarative Past and Imperative Future: Executable Temporal Logic for Interactive Systems. In B. Banieqbal, H. Barringer, and A. Pnueli, editors, *Proceedings of Colloquium on Temporal Logic in Specification*, pages 402–450, Altrincham, U.K., 1987. (Published in *Lecture Notes in Computer Science*, volume 398, Springer Verlag).
17. J. Gallier. *Logic for Computer Science: Foundations of Automatic Theorem proving*. John Wiley and Sons, 1987.
18. L. Gasser, C. Braganza, and N. Hermann. MACE: A Flexible Testbed for Distributed AI Research. In M. Huhns, editor, *Distributed Artificial Intelligence*. Pitman/Morgan Kaufmann, 1987.
19. J. Y. Halpern and Y. Moses. A guide to completeness and complexity for modal logics of knowledge and belief. *Artificial Intelligence*, 54:319–379, 1992.
20. C. B. Jones. *Systematic Software Development using VDM (second edition)*. Prentice Hall, 1990.
21. K. Konolige. *A Deduction Model of Belief*. Pitman/Morgan Kaufmann, 1986.
22. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag: Heidelberg, Germany, 1992.

23. T. Maruichi, M. Ichikawa, and M. Tokoro. Modelling Autonomous Agents and their Groups. In Y. Demazeau and J. P. Muller, editors, *Decentralized AI – Proceedings of the First European Workshop on Modelling Autonomous Agents and Multi-Agent Worlds (MAAMAW)*. Elsevier/North Holland, 1990.
24. A. Pnueli. The Temporal Logic of Programs. In *Proceedings of the Eighteenth Symposium on the Foundations of Computer Science*, 1977.
25. A. Pnueli. Specification and Development of Reactive Systems. In *Information Processing '86*. Elsevier/North Holland, 1986.
26. A. S. Rao and M. P. Georgeff. Modeling rational agents within a BDI-architecture. In R. Fikes and E. Sandewall, editors, *Proceedings of Knowledge Representation and Reasoning (KR&R-91)*, pages 473–484. Morgan Kaufmann Publishers: San Mateo, CA, April 1991.
27. P. S. Rosenbloom, J. E. Laird, A. Newell, and R. McCarl. A preliminary analysis of SOAR as a basis for general intelligence. *Artificial Intelligence*, 47:289–326, 1991.
28. M. P. Singh. *Multiagent Systems: A Theoretical Framework for Intentions, Know-How, and Communications (LNAI Volume 799)*. Springer-Verlag: Heidelberg, Germany, 1994.
29. R. G. Smith. *A Framework for Distributed Problem Solving*. UMI Research Press, 1980.
30. M. Wooldridge. *The Logical Modelling of Computational Multi-Agent Systems*. PhD thesis, Department of Computation, UMIST, Manchester, UK, 1992.
31. M. Wooldridge and M. Fisher. A First-Order Branching Time Logic of Multi-Agent Systems. In *Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI '92)*, Vienna, Austria, August 1992. Wiley and Sons.
32. 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.
33. M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.