Verifying bounds on deliberation time in multi-agent systems

Natasha Alechina [a] Brian Logan [a]

[a] *University of Nottingham, Nottingham, NG8 1BB, UK*

Abstract

Rule-based agents (for example, agents reasoning using ontology rules) are increasingly being employed in the implementation of web services and other situations in which the time taken to generate a response is critical. To be able to provide a response time guarantee for such systems, it is important to know how long the agent's reasoning is going to take. In this paper, we describe an approach to establishing an upper bound on deliberation time of a system of rule-based agents. We propose a formal model of a system of rule-based agents which associates explicit costs with each rule application. This formal model can serve as an input to a model-checker, allowing upper bounds on deliberation time to be automatically verified.

1 Introduction

In this paper, we consider resource-bounded rule-based agents. Our starting point is an acknowledgement that an agent's deliberation takes time—even if the agent's program is capable of finding a suitable response to some event or query, this response will take time to compute. We are interested in verifying temporal properties of such agents, such as 'every time the agent receives a certain query, it will produce a response within 5 time steps'.

In general, it is impossible to predict arbitrary temporal properties of an arbitrary rule-based system. However some properties *are* decidable. The purpose of this paper is to investigate such properties and the complexity of model-checking an agent system for those properties. We show that, given some reasonable assumptions, namely that agents have bounded memory, that their state variables can take one of finitely many values, and the same holds for the environment, we can represent any multi-agent system consisting of finitely may rule-based agents as a state transition system, and specify its properties in a temporal logic. In particular, we can express bounded response properties and termination properties. We could have relaxed the finiteness assumptions, giving an infinite-state verification problem (where states are parametrised by values of some variables), however in general model-checking problem for such systems is undecidable, and in this paper we restrict ourselves to finite-state systems.

We believe that the novelty of our approach is in bringing together two established areas of research, which as far as we know have not interacted before: namely research on the space and time complexity and response times of various rule-based systems, and research on verification of real-time systems.

The rest of the paper is organised as follows. In the next section we briefly sketch the kinds of multi-agent system we want to formalise. In section 3 we sketch how to allocate time costs to the application of an agent's rules. In section 4 we show how to represent an agent-environment model as a state transition system, where transitions between the agent's states take time dependent on the cost of the rule(s) used to compute the next state. In section 5 we use known results on complexity of model-checking bounded temporal formulas (for example, [11]) to define a set of temporal properties of resource-bounded rule-based agents which admit feasible verification (linear in the size of the transition system). In section 6 we discuss related work, and in section 7 we conclude and identify areas for future work.

2 Model of a MAS

We begin with a simple MAS consisting of one or more agents and their environment. The agents' *environment* is that part of the physical world or computational system "inhabited" by the agents. The environment contains objects (or more generally state) and processes which can be perceived and acted upon the agents.

The agents execute asynchronously. We assume that each agent repeatedly executes a fixed *sense-think-act* cycle. At each cycle, the agent updates its beliefs based on its observations of the environment at that cycle, chooses an action or actions to perform and then executes the action(s). We assume that the environment is partially observable and that both sensing and action are noisy: the agent's sensors may return incorrect data and actions may fail in a variety of ways.

Our main interest here is with the *think* part of the cycle. We wish to model the execution of the steps in the agent's program—the computations that the agent performs to update it's state and ultimately select an action. In this paper we focus on rule-based agents. In a rule-based agent, the agent's program consists of a collection of rules which are executed by a rule interpreter. Compared to lower level procedural formalisms, such as Java, programming an agent in terms of rules allows a greater degree of abstraction in the specification of the agent's behaviours. A wide range of rule-based agent architectures and toolkits have been developed, e.g., [14, 21], and rule-based programming extensions are increasingly being offered as add-ons to existing, lower-level, agent toolkits, e.g., JADE [6] and FIPA-OS [19].

A rule-based agent consists of a working memory and one or more sets of condition-action rules of the form $P_1, \ldots, P_m \rightarrow Q_1, \ldots, Q_n$, where $P_1, \ldots, P_m$ are the conditions and $Q_1, \ldots, Q_n$ are the actions. The conditions may contain unbound variables which are bound when the condition is matched against the contents of working memory. The working memory constitutes the agent's state, and the rules form the agent's program.

At each execution cycle, the agent senses its environment and information obtained by sensing is added to the previously derived facts and any *a priori* knowledge in the agent's working memory. The agent then performs one or more inference cycles in which it evaluates the condition-action rules forming its program. At each inference cycle, the conditions of each rule are matched against the contents of the agent's working memory and a subset of the rules are fired. This typically adds or deletes one or more facts from working memory and/or results in some external actions being performed in the agent's environment. In general, the conditions of a rule can be consistently matched against the items in working memory in more than one way, giving rise to a number of distinct *rule instances*. Following standard rule based system terminology we call the set of rule instances the *conflict set* and the process of deciding which subset of rule instances are to be fired at any given inference cycle *conflict resolution*.

The number of inference cycles performed at each execution cycle is determined by the agent's *rule application* strategy. Agents can adopt a wide range of rule application and conflict resolution strategies. For example, they can order the conflict set and fire only the first instance in the ordering at each execution cycle, or they can fire all rule instances in the conflict set on each execution cycle, or they can repeatedly compute the conflict set and fire all the rule instances it contains until no new facts can be derived at the current execution cycle. We call these three strategies *single rule at each (execution) cycle*, *all rules at each (execution) cycle*, and *all rules to quiescence* respectively.

3 Time bounds on rule execution

We are interested in verifying properties of the form 'if the agent knows or observes $P_1, \ldots, P_m$, it will conclude (or perform the action) Q in less than t timesteps' or conversely 'if the agent knows or observes $P_1, \ldots, P_m$, it not will conclude (or perform the action) Q for at least t timesteps'. It is in general impractical to run and time the system for all possible interactions between the agents and the environment to establish such properties. It is also undesirable to specify all the minute details of the matching algorithm and conflict resolution algorithm as an input to a model checker (besides, model-checking the resulting specification will mostly likely be unfeasible). In this section, we outline our approach to generating an abstract model of a system of rule-based agents. States of the abstract model are partial descriptions of the actual contents of the agent's working memory. Transitions between states correspond to firing rules or sets of rules, and are annotated with lower and upper bounds on the time required by the rule-based system to compute the new state.

In this section we first show how to compute the upper and lower bounds on the time taken to fire a single rule and then show how this can be generalised to compute upper and lower bounds for sets of rules. The more we know about the agent's architecture and program, the more accurate the upper and lower bounds will be.

The three phases of the rule interpreter, namely, matching rules against working memory, selecting which rule(s) to fire from the conflict set, and firing the rule(s), each has an associated time cost.

Consider a naive rule interpreter which simply matches each condition in each rule against each fact in working memory to generate a conflict set containing all possible rule instances. We assume that rule instances are simply appended to the conflict set in the order in which they are generated and that all rule instances are fired. In this case, the upper bound on rule execution is given by $\alpha\, w^p + \beta\, n + \gamma\, n$, where w is the number of working memory elements, p is the number of patterns (condition elements), n is the size of the resulting conflict set ($n \leq w^p$) and α, β and γ are constants which scale the basic operations in each phase to some common measure of time, e.g., the time required to perform a single match. The lower bound on rule execution is given by $\alpha\, w$, since with a naive interpreter, we must match every working memory element against at least one condition element to discover that no rules match. If the conflict set is empty, the second and third terms are 0.

In practice, the time required to fire a single rule can be reduced in several ways. For example, we can reduce the cost of matching by indexing the working memory elements to determine which condition elements to consider and/or cache the results of previous matches (e.g., using the Rete [12] or TREAT [16] algorithms). Caching the results of previous matches can significantly increase the upper bound for rule execution. For example, the Rete algorithm [12] has an upper bound on rule execution of $\alpha\, w^{2p-1} + \beta\, n + \gamma\, n$. However caching also significantly reduces the lower bound, giving a minimum rule execution time for Rete of α. For conflict resolution strategies that only fire a single rule at each cycle, the cost of computing the conflict set can be reduced by only computing the first element of the conflict set [17]. In this case, the time required to fire the single rule instance is constant and the upper bound on rule execution becomes $\alpha\, w^p + \beta + \gamma$, and the corresponding lower bound α.

If we know the number of working memory elements which match each condition element appearing in the agent's rules, and know the order in which condition elements are matched, e.g., the structure of the Rete network, we can provide tighter bounds on rule execution. For example, we could assume that the Rete network is always optimal, i.e., performs the minimal number of comparisons.[1]

In many cases, it is convenient to consider upper and lower bounds for sets of rules rather than a single rule. The upper and lower bounds for a set of rules is the minimum and maximum times necessary to fire all the rules in the set. There are at least three cases in which this can be useful:

- if there is more than one rule that brings about a condition of interest, then to establish whether a particular property holds, we may want to to establish the upper and lower bounds for the execution of this set of rules.

- if the rule we are interested in has a precondition which must be made true by other rules, the to compute the time required to fire the rule of interest, we need to know how long the other rules take to satisfy its precondition. For example, given a rule $P \rightarrow Q$, if there is a set of rules that make P true, then the time to make Q true is the time required to execute the ruleset + the time required to fire $P \rightarrow Q$ given the resulting working memory. If we also have $R \rightarrow Q$, then the time to make Q true, given that the initial working memory is sufficient to produce both P and R, is the least lower bound for P and R, and, as upper bound, whatever the next smallest bound is.

- if the agent program consists of several sets of rules which are executed in order, e.g., rules for perception form a set and are all executed before the rules for deliberation, which in turn are executed before the rules for action selection. In this case the upper bound for the complete execution cycle can be no greater than the sum of the upper bounds for each ruleset, and the lower bound for the cycle must be at least the sum of the lower bounds for each ruleset.

In such cases we model the ruleset by a single *abstract rule* whose upper and lower bounds are determined by the underlying ruleset. For example, consider the simple ruleset shown in figure 1 which computes the nearest tile to an agent in Tileworld [18]

When executed by a naive rule interpreter which repeatedly executes the (lexicographically) first rule to match to quiesence, the lower bound is

$$\alpha\, w$$

when there are no tiles and the upper bound is

$$w \times (\alpha\, w^p + \beta\, n + \gamma)$$

[1] For example, Tan et al. [22] show how to model the time cost of matching for a range of caching strategies, and how to establish whether a Rete network known to be optimal at compile time remains optimal in the face of working memory updates.

```
;;; No tile can be nearer than one we
;;; are holding.
RULE holding_tile
     (have_tile ?tile)
==>
     (ADD nearest_tile ?tile)

;;; If there is no nearest tile from
;;; the last cycle, pick a tile at
;;; random as the nearest.
RULE random_nearest_tile
     (NOT have_tile)
     (NOT nearest_tile)
     (seen_tile ?tile)
==>
     (ADD nearest_tile ?tile)

;;; Check to see if there is tile nearer
;;; than the current nearest.
RULE see_nearer_tile
     (NOT have_tile)
     (nearest_tile ?tile)
     (seen_tile ?nearer_tile)
     (WHERE dist(sim_myself, nearer_tile) <
            dist(sim_myself, tile))
==>
     (ADD nearest_tile ?nearer_tile)
```

Figure 1: An example ruleset

if the entire conflict set must be enumerated to fire the first matching rule at each cycle. If we can assume that the agent has seen at least one tile and the time required by the rest of the agent's program does not depend on the identity of the the nearest tile, the execution of the ruleset can be abstracted into a single rule, for example

```
RULE abstract_nearest_tile
     (seen_tile ?tile)
==>
     (ADD nearest_tile ?tile)
```

with the same upper and lower bounds as the ruleset.[2].

4 Modelling a rule-based multi-agent system as a state transition system

It is intuitively clear that any rule-based multi-agent system can be modelled as a state transition system.

In this section, we give a precise translation from a specific type of a rule-based multi-agent system to a state transition system, and characterise the size of the resulting transition system in terms of the size of agent programs. In the next section, we use existing results on model-checking complexity of RTCTL (Real time CTL, [11]) to characterise time required to verify temporal properties of a multi-agent system as a function of the size of the agent programs.

We assume that agents' programs are propositional. There is an obvious translation from a predicate program where all variables come from a finite fixed domain, to propositional rules (all possible rule instances). The time bounds on rule firing cycles which we use below, however, come from the original program, where the rules contain variables, to account for the overhead of pattern matching.

[2]For a more plausible rule interpreter which indexes the contents of working memory and fires the first matching rule without enumerating the entire conflict set, the lower bound is O(1) when the agent has seen at least one tile, and the upper bound is cubic in the number of tiles seen by the agent

All variables used in the agents' rules (and by the environment) are from a finite set $P = \{p_1, \ldots, p_n\}$. Each agent i in the set of agents A ($A = \{1, \ldots, m\}$) has a set of input, output and internal variables, $var(i) = in(i) \cup out(i) \cup int(i)$, with $var(i) \subseteq P$. The environment also has a set of variables $var(e) = in(e) \cup out(e) \cup int(e)$ with $var(e) \subseteq P$. For each agent i, $in(i) \subseteq out(e)$, and $in(e) \subseteq \bigcup_i out(i)$.

The agents execute sense-think-act cycle asynchronously. At the 'sense' part of the cycle, each agent reads its input variables from the environment. For simplicity, we assume that the 'sense' part of the cycle is infallible; this assumption is not essential and may be relaxed. At the 'think' part of the cycle, which corresponds to the rule firing cycle of the agent, the agent sets the values of its output and internal variables; at the 'act' phase, it writes the values of its output variables to the environment. The environment in turn may update its internal and output variables at that stage.

We assume that the 'sense' and 'act' parts of the cycle do not consume any time, or rather a very small constant which can be ignored. The 'think' part of the cycle has different durations for each agent, depending on the agent's program and state. We can associate lower and upper bounds $(t_l(s_i), t_u(s_i))$ on the duration of the 'think' cycle of agent i which commences in state s_i. The meaning of this pair is that the 'think' cycle starting in state s_i is going to take at least $t_l(s_i)$ time units and at most $t_u(s_i)$ time units, where the time unit is the time required to compute one rule match. The upper and lower bounds are integer values.

Given the agent system, the corresponding state transition system $\mathbf{S}$ consists of a set of states S, a transition relation R and an assignment of propositional variables to states V.

Each $s \in S$ is a tuple $\langle s_1, \ldots, s_m, s_e \rangle$, where s_i is a local state of the agent i and s_e is the state of the environment. We will denote the set of all local agent states in S by $loc(S)$ (the set of all s_i such that they are a member of some tuple $s \in S$) and environment states by $env(S)$ (the set of all s_e such that they are a member of some tuple $s \in S$). The local state of the agent encodes information about the truth values of the agent's variables, but also about the control state of the agent's program (which rules have been fired etc.), whether the agent is ready to sense and act (assume $ready_i$ is an output variable for each agent i), and a clock variable the use of which will be explained below (clock is not a propositional variable from P, but an integer-valued variable).

The assignment V is a function from pairs (s_i, p) to {**true**, **false**}, where either $s_i \in loc(S)$ and $p \in var(i)$, or $s_i \in env(S)$ and $p \in var(e)$. The assignment function is determined by the agent system in a straightforward way ($V(s_i, p) = \mathbf{true}$ if p is true in the corresponding agent state). Assignments do not have to be consistent between agents (e.g., in state s, $V(s_1, p)$ may be **true**, $V(s_2, p)$ may be **false**, and $V(e, p)$ may be **true**).

The transition relation $R \subseteq S \times S$ is defined in terms of three auxiliary relations: R_s ('sense'), R_t ('think'), and R_a ('act'), as $R = R_s \circ R_t \circ R_a$. We assume that from every state s it is possible to make an R transition to some state (even if it is a 'halt' state with a transition to itself).

$R_s(s, s')$ if, and only if, $s_e = s'_e$ (environment state does not change), and for each agent i either $ready_i = \mathbf{false}$, in which case its state does not change, or $ready_i = \mathbf{true}$. In the latter case, for each of i's input variables p, $V(s'_i, p) = V(s_e, p)$ (the value of agent i's input variable in s'_i is equal to the value it had for the environment at s). From the definition it follows that R_s is a deterministic transition; note that this corresponds to the case where the sensing is infallible, and all input variables are read correctly.

$R_t(s, s')$ if, and only if, $s_e = s'_e$ (environment state does not change), and the agents' states are updated in one of the two ways, defined below as a 'busy' transition or 'update' transition.

Recall that for each agent i and state s in which a 'think' transition is possible, we have a pair of lower and upper bounds $(t_l(s_i), t_u(s_i))$. Each agent also has a clock variable x_i, which counts how many 'think' cycles have been spent for the given rule-firing cycle, and a variable $ready_i$, which is set to false if the agent's rule firing cycle is not complete. The value of the clock and the bounds determine whether the agent performs a 'busy' transition or an 'update' transition, or if both of them are enabled.

If $x_i < t_l(s_i)$ (the time the agent has spent thinking so far is less than the lower bound on the length of the rule firing cycle), the agent does nothing but increments x_i (s'_i is like s_i but the clock value $x'_i = x_i + 1$). This is a 'busy' transition. The values $(t_l(s_i), t_u(s_i))$ persist into the next state and $ready_i$ is false (until the clock value is reset to 1).

If $x_i = t_u(s_i)$ (the time spent thinking equals to the upper bound), the agent has to update its state. The values of its internal and output variables are set in accordance with its rules, the clock variable is set to 1 and $ready_i$ to **true**. This is an 'update' transition.

If $t_l(s_i) \leq x_i \leq t_u(s_i)$, the agent can perform both a 'busy' transition and an 'update' transition (in this case, R_t is a non-deterministic transition relation).

$R_a(s, s')$ if, and only if, $s_i = s'_i$ for each agent i (agent's state does not change), and for each of the

environment's input variables p, $V(s'_e, p)$ is changed to reflect the real agent system behaviour. Namely, in case p is an output variable of a single agent i with $ready_i = \mathbf{true}$, then $V(s'_e, p) = V(s_i, p)$. If several agents with the $ready$ variable set to true output p, their values for p do not have to be the same; in this case the environment has a rule for choosing a value for p. Internal and output variables of the environment are also updated at s'_e. This transition may be non-deterministic.

Proposition 1 *Given a rule-based multi-agent system with m agents, n propositional variables, r propositional rules, and t the largest upper bound on the 'think' transition in any agent, the size of the corresponding state transition system is bounded by $2^{m(n+r+c)} \times t^m$, where c is a constant.*

Proof. The number of different local states for every agent is $2^{n+r+c} \times t$: since each local state is encoding an assignment to n propositional variables, control state of the agent program which may depend on r (e.g., which rule is scheduled to be fired), and the value of the clock which can have one of t values; c represents the rest of control information, such as the value of $ready_i$.

The number of global states is bounded by the number of all possible combinations of local states, which is $(2^{n+r+c} \times t)^m = 2^{m(n+r+c)} \times t^m$.

The size of the transition relation R is significantly smaller than S^2.

$\dashv$

4.1 Example

Below, we introduce a simple example of an agent-environment system, and describe a corresponding state transition system.

Assume that the system consists of two agents and an environment. Each agent has two variables p_i, q_i and one rule $p_i \rightarrow q_i$. For the first agent, the time bounds on the rule firing cycle when p_1 is true are $[2, 3]$; for the second agent, if p_2 is true, the rule can be fired in one cycle (bounds are $[1, 1]$). The input and output variables are as follows:

$in(1) = \{p_1\}, out(1) = \{q_1\}, int(1) = \emptyset.$

$in(2) = \{p_2\}, out(2) = \{q_2\}, int(2) = \emptyset.$

$in(e) = \{q_1, q_2\}, out(e) = \{p_1, p_2\}, int(e) = \emptyset.$

Suppose the system starts in state s where the local states are as follows:

in s_i ($i \in \{1, 2\}$), p_i, q_i are false, $ready_i$ is true, $x_i = 1$

in s_e, p_1, p_2 are true and q_1, q_2 are false.

The resulting system looks as follows:

5 Model-checking temporal properties of agents

We express temporal properties of agents in RTCTL (Real Time CTL) [11]. Since our time bounds are integers, we can express all the upper and lower bound properties in CTL using nestings of operators, but RTCTL provides a more convenient notation.

We define formulas of temporal logic RTCTL-B extended with a belief operator B_i, one for each agent i. A formula of RTCTL-B is defined by

$$\phi = p \mid B_i p \mid \phi \vee \phi \mid \neg\phi \mid EX\phi \mid EG\phi \mid EG_{[a,b]}\phi \mid E(\phi U \phi) \mid E(\phi U_{[a,b]} \phi)$$

where p is a propositional variable, and $B_i p$ means: agent i believes p, and a, b are non-negative integers. $B_i p$ can be treated as a special kind of propositional variable. The meaning of temporal logic formulas is defined relative to a notion of a path in a state transition system. A *path* in $\mathbf{S} = (S, R, V)$ is an infinite sequence of states from S: $s^0, s^1, \ldots$, such that for each pair s^i, s^{i+1}, $R(s^i, s^{i+1})$. $EX\phi$ is true in a state of a transition system if there is a path starting from that state where in the next state ϕ holds. $EG\phi$ means: there is a path where ϕ holds in every state, and $E(\phi U \psi)$ means: there is a path where ϕ holds until ψ becomes true. The meaning of bounded EG and EU operators is as follows: $EG_{[a,b]}\phi$ holds if there is a

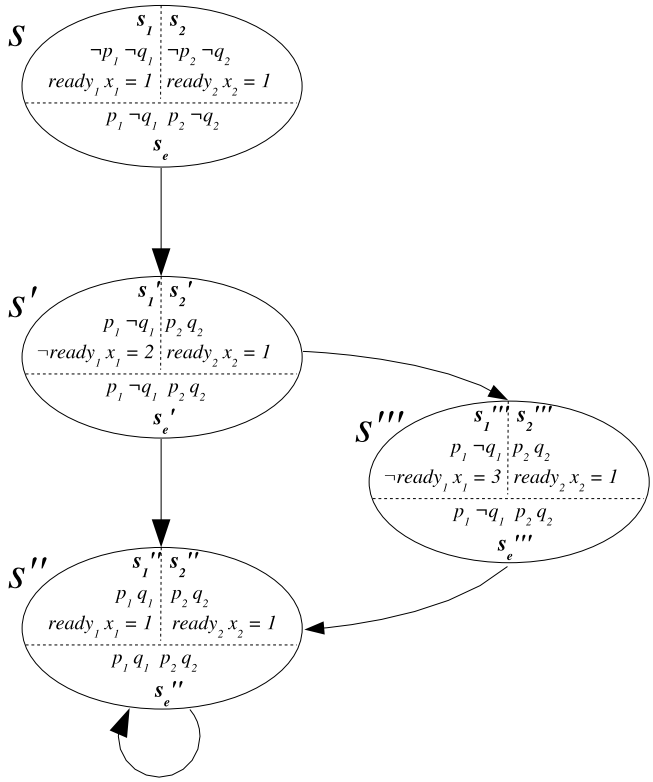

Figure 2: Example state transition system

path, where ϕ holds in all the states reachable by more than a and fewer than b steps; $E(\phi U_{[a,b]}\psi)$ holds if there is a path, where a state satisfying ψ is reachable by more than a and fewer than b steps, and in all preceding states, ϕ holds.

The truth conditions of RTCTL-B in a state transition $\mathbf{S} = (S, R, V)$ are defined as follows (where $s \in S$ and $\mathbf{S}, s \models \phi$ is read as 's satisfies ϕ in $\mathbf{S}$'):

$\mathbf{S}, s \models p$ iff $V(s_e, p) = \mathbf{true}$

$\mathbf{S}, s \models B_i p$ iff $V(s_i, p) = \mathbf{true}$

$\mathbf{S}, s \models \phi \vee \psi$ iff $\mathbf{S}, s \models \phi$ or $\mathbf{S}, s \models \psi$

$\mathbf{S}, s \models \neg\phi$ iff not $\mathbf{S}, s \models \phi$

$\mathbf{S}, s \models EX\phi$ iff there is a path $s^0, ^1, s^2, \ldots$, where $s^0 = s$ and $\mathbf{S}, s^1 \models \phi$

$\mathbf{S}, s \models EG\phi$ iff there is a path $s^0, s^1, s^2, \ldots$, where $s^0 = s$ and for each s^i on the path, $\mathbf{S}, s^i \models \phi$

$\mathbf{S}, s \models EG_{[a,b]}\phi$ iff there is a path $s^0, s^1, s^2, \ldots$, where $s^0 = s$ and for each s^i with $a \leq i \leq b$ on the path, $\mathbf{S}, s^i \models \phi$

$\mathbf{S}, s \models E(\phi U \psi)$ iff there exists a path $s^0, s^1, s^2, \ldots$, where $s^0 = s$, there exists $i \geq 0$ such that $\mathbf{S}, s^i \models \psi$ and for all $j < i$, $\mathbf{S}, s^j \models \phi$

$\mathbf{S}, s \models E(\phi U_{[a,b]} \psi)$ iff there exists a path $s^0, s^1, s^2, \ldots$, where $s^0 = s$, there exists i with $a \leq i \leq b$ such that $\mathbf{S}, s^i \models \psi$ and for all $j < i$, $\mathbf{S}, s^j \models \phi$

A formula ϕ is true in a model if every state satisfies it.

Other boolean connectives can be defined in a standard way, and also the following temporal operators:

$AX\phi \stackrel{df}{=} \neg EX\neg\phi$ (on all paths, in the next state ϕ holds)

$EF\phi \stackrel{df}{=} E(\top U \phi)$, where $\top$ is a logical tautology (there exists a path where after finitely many steps ϕ holds)

$AG\phi \stackrel{df}{=} \neg EF\neg\phi$ (ϕ is true in all states on all paths)

$AF\phi \stackrel{df}{=} \neg EG\neg\phi$ (on each path, ϕ is true at some finitely reachable point)

$A(\phi U \psi) \stackrel{df}{=} \neg E(\neg\psi U(\neg\phi \wedge \neg\psi)) \wedge \neg EG\neg\psi$

As an example of a property expressible in RTCTL-B, consider 'if p_1 is true, then after at most 3 steps agent 1 will believe q_1':

$$AG(p_1 \rightarrow A(\top U_{[0,3]} B_1 q_1))$$

Given a state transition system $\mathbf{S}$ as defined above, and an RTCTL-B formula ϕ, the problem whether all states of the system satisfy ϕ is decidable in linear time (if the upper and lower bounds a, b are written in unary). This follows from the model-checking complexity of RTCTL[11] and the fact that formulas of the form $B_i p$ can be treated as propositional variables.

Theorem 1 *Given a rule-based multi-agent system with m agents, n propositional variables, r propositional rules, and t the largest upper bound on the 'think' transition in any agent, the problem whether a property expressed as an RTCTL-B formula ϕ is true in the system is decidable in time $O(|\phi| \times 2^{m(n+r)} \times t^m)$.*

Proof. From linear time complexity of RTCTL-B model-checking and the size of corresponding state transition system established in Proposition 1. $\dashv$

6 Related work

A considerable amount of work has been done in the area of model-checking multi-agent systems (see, e.g., [8, 7]. However, the emphasis of this work is on correctness rather than the timing properties of agents, which is our primary interest.

There has been also considerable work on the execution properties of rule based systems, both in AI and in the active database community. Perhaps the most relevant is that of of Chen and Cheng on predicting the response time of OPS5-style production systems. In [9], they show how to compute the response time of a rule based program in terms of the maximum number of rule firings and the maximum number of basic comparisons made by the Rete network. In [10], Cheng and Tsai describe a tool for detecting the worst-case response time of an OPS5 program by generating inputs which are guaranteed to force the system into worst-case behaviour, and timing the program with those inputs. Also relevant are the complexity analyses of various matching algorithms, e.g., [12, 16, 17] and work on the design of optimal match networks, e.g., [22]. As sketched in section 3, this work provides a rich framework for the analysis of upper and lower bounds on the firing of individual rules. However, to our knowledge, there has been little work to date on rule abstraction. Results for upper and lower bounds on sets of rules typically assume that all rules are compiled into a single global match network, making it difficult to compute the upper and lower bound for an arbitrary set of rules.

The literature also contains a number of analyses of average case complexity for match networks, e.g., [2]. The assumptions required by this type of analysis (e.g., numbers of working memory elements per class, average numbers of additions and deletions from working memory per cycle etc.) can also be used in tightening the lower and upper bounds, and this work could be used to verify average case properties, such as whether on average, the length of time the agent spends planning is shorter than the rate at which environment changes. Some authors have also advocated the use of empirical studies to characterise the average case performance of rule based systems, e.g., [15]; again if such data are available, they could be used in verifying average case properties.

There has also been some work on characterising termination conditions for rule based programs, e.g., [1, 23, 10]. However this is less relevant to the framework presented in this paper, since we do not require that the agent's production system terminates. So long as each (abstract or concrete) rule instance executes in bounded time and the set of states is finite, we can express and verify properties of the agent.

Another strand of relevant work is verifying temporal properties of agent systems, in particular real-time properties, where actions are assumed to take non-trivial time. In [20], Singh proposed a framework for

modelling agent systems where actions have duration and can be executed in parallel. However, to the best of our knowledge, complexity issues for this framework have not been explored. In [13], a semi-decidable formalism for expressing what knowledge an agent has at what time was presented, and implemented in Prolog. This work has very similar motivation to the work presented here. However, a particular kind of conflict resolution strategy was assumed (*all rules at one cycle'* in our terminology).[3] In [3, 4], a more flexible step-counting logic was presented, which could model various conflict-resolution strategies used by an agent. The model checking problem for this logic is decidable. However, there is no automatic verification procedure associated with this framework, while the approach presented in this paper can be easily adapted to use existing model-checkers, such as Mocha [5].

7 Conclusions

In this paper, we have shown how to model a multi-agent system involving rule-based agents and to assign durations to the steps comprising an agent's *think* cycle. The agent's internal actions can be modelled either at a low level, where each match and rule firing is modelled as a separate transition, or at a more abstract level, where firing a whole rule set is assigned upper and lower time bounds.

We have shown how an existing modelling framework, state transition systems, can be used to specify the system, and how an existing verification language, RTCTL, can be used to specify temporal properties of agents. For a discrete flow of time, the problem of whether all runs of the automaton satisfy the property is decidable. There are tools, such as Mocha [5], which enable automatic verification of the agent's response properties.

The framework for verifying bounds on deliberation time in rule-based agents proposed in this paper brings together two well-established strands of work; however we believe that the combined approach has not been tried before, and promises interesting results on verifying response times of existing rule-based systems. In future work, we plan to study in more detail potential abstraction techniques (generalising a rule set to a single rule) which would reduce the state space of the transition system.

Acknowledgements

This work reported in this paper was partially supported by the Royal Society project 'Model-checking resource-bounded agents'.

References

[1] Alexander Aiken, Jennifer Widom, and Joseph M. Hellerstein. Behavior of database production rules: termination, confluence, and observable determinism. In *Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data*, pages 59–68. ACM Press, 1992.

[2] Luc Albert. Average case complexity analysis of RETE pattern-match algorithm and average size of join in database. In *Proceedings of the Ninth Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 223–241. Springer-Verlag, 1989.

[3] Natasha Alechina, Brian Logan, and Mark Whitsey. A complete and decidable logic for resource-bounded agents. In Nicholas R. Jennings, Charles Sierra, Liz Sonenberg, and Milind Tambe, editors, *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2004)*, volume 2, pages 606–613, New York, July 2004. ACM Press.

[4] Natasha Alechina, Brian Logan, and Mark Whitsey. Modelling communicating agents in timed reasoning logics. In José Júlio Alferes and Jõao Leite, editors, *Proceedings of the Ninth European Conference on Logics in Artificial Intelligence (JELIA 2004)*, number 3229 in LNAI, pages 95–107, Lisbon, September 2004. Springer.

[3] It would be interesting to check whether for a restricted set of temporal properties (upper and lower bounds) their formalism is decidable.

[5] Rajeev Alur, Thomas A. Henzinger, Freddy Y. C. Mang, Shaz Qadeer, Sriram K. Rajamani, and Serdar Tasiran. MOCHA: Modularity in model checking. In *Computer Aided Verification*, pages 521–525, 1998.

[6] Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa. Developing multi-agent systems with a FIPA-compliant agent framework. *Software Practice and Experience*, 31(2):103–128, 2001.

[7] Massimo Benerecetti, Fausto Giunchiglia, and Luciano Serafini. Model checking multiagent systems. *J. Log. Comput.*, 8(3):401–423, 1998.

[8] Rafael Bordini, Michael Fisher, Willem Visser, and Michael Wooldridge. State-space reduction techniques in agent verification. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-2004)*, pages 896–903, New York, 2004. ACM Press.

[9] Jeng-Rung Chen and A. M. K. Cheng. Predicting the response time of OPS5-style production systems. In *Proceedings of the 11th Conference on Artificial Intelligence for Applications*, page 203. IEEE Computer Society, 1995.

[10] Albert Mo Kim Cheng and Hsiu yen Tsai. A graph-based approach for timing analysis and refinement of OPS5 knowledge-based systems. *IEEE Transactions on Knowledge and Data Engineering*, 16(2):271–288, 2004.

[11] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts, 1999.

[12] Charles Forgy. Rete: A fast algorithm for the many patterns/many objects match problem. *Artificial Intelligence*, 19(1):17–37, 1982.

[13] J. Grant, S. Kraus, and D. Perlis. A logic for characterizing multiple bounded agents. *Autonomous Agents and Multi-Agent Systems*, 3(4):351–387, 2000.

[14] J. E. Laird, A. Newell, and P. S. Rosenbloom. SOAR: An architecture for general intelligence. *Artificial Intelligence*, 33:1–64, 1987.

[15] Jay S. Lark, Lee D. Erman, Stephanie Forrest, and Kim P. Gostelow. Concepts, methods, and languages for building timely intelligent systems. *Real-Time Systems*, 2(1-2):127–148, 1990.

[16] Daniel P. Miranker. TREAT: A better match algorithm for ai production systems. In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI'87)*, pages 42–47. AAAI Press, 1987.

[17] Daniel P. Miranker, David A. Brant, Bernie Lofaso, and David Gadbois. On the performance of lazy matching in production systems. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 685–692. AAAI Press, 1990.

[18] Martha E. Pollack and Marc Ringuette. Introducing the Tileworld: Experimentally evaluating agent architectures. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 183–189, Boston, MA, 1990. AAAI.

[19] S. Poslad, P. Buckle, and R. G. Hadingham. The FIPA-OS agent platform: Open source for open standards. In *Proceedings of the Fifth International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agents (PAAM2000)*, pages 355–368, Manchester, April 2000.

[20] Munindar P. Singh. Toward a model theory of actions: How agents do it in branching time. *Computational Intelligence*, 14(3):287–305, 1998.

[21] Aaron Sloman and Brian Logan. Building cognitively rich agents using the SIM_AGENT toolkit. *Communications of the ACM*, 42(3):71–77, March 1999.

[22] Jack S. E. Tan, Jaideep Srivastava, and Sashi Shekhar. On the construction of efficient match networks. In *Proceedings of the 1992 ACM/SIGAPP Symposium on Applied computing*, pages 353–362. ACM Press, 1992.

[23] Hsiu yen Tsai and Albert Mo Kim Cheng. Termination analysis of OPS5 expert systems. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI '94)*, pages 193–198. American Association for Artificial Intelligence, 1994.