Modal logics for communicating rule-based agents

Natasha Alechina and **Mark Jago** and **Brian Logan**[1]

Abstract. In this paper, we show how to establish correctness and time bounds (e.g., quality of service guarantees) for multi-agent systems composed of communicating rule-based agents. The formal models of multi-agent systems we study are transition systems where each transition corresponds to either a rule firing or an act of communication by an agent. We present a complete and sound modal logic which formalises how the beliefs of communicating rule-based agents change over time. Using a simple example, we show how this logic can be used to specify temporal properties of belief change in multi-agent systems in a precise and realistic way, and how existing modal logic techniques such as model-checking can be used to state and verify properties of agents.

1 INTRODUCTION

There has recently been considerable interest in rule-based approaches to various aspects of agent technology. For example, rules and rule engines (rule interpreters) have been advocated as a means of managing the business logic of applications in a wide range of domains such as insurance, financial services, government, telecom customer care and billing, and ecommerce [19]. Another important application of rule-based approaches in agent-based systems is the semantic web, for example, ontological reasoning in DAML/OWL and, more generally, in rule extensions to ontologies such as OWL+Rules [17] and SWRL[18]), which significantly increase the expressive power of ontology languages [17].

At the same time there has been a growing use of rule engines as key components of agent based systems. Much of this work is based on mature rule-based systems technology such as the Jess rule engine [14], e.g., the FIPA-OS JessAgent, the use of Jess in reasoning about ontologies (DAML JessKB) and in web service composition (DAML-S Virtual Machine [20]) and supporting tools such as Jena which convert XML/RDF to Jess facts, and, more generally, the Java Rules API.[2] However, while the adoption of rule-based approaches brings great benefits in terms of flexibility and ease of development, these approaches also raise new challenges for the agent developer, namely how to ensure correctness of rule-based designs (will a rule-based agent produce the correct output for all legal inputs), termination (will a rule-based agent produce an output at all) and response time (how much computation will a rule-based agent have to do before it generates an output).

As a motivating example, consider an agent which acts as a provider agent for a company which sells goods or services on the web. Such an agent may require a combination of ontological rules to match a potential customer's request against the goods and services it offers (e.g., to realise that dog food is a kind of pet food [21]) and business rules to determine, e.g., the appropriate level of discount to be offered. The designer or developer of such an agent may wish to ensure that the agent's behaviour is correct, e.g., that the agent does not offer a customer an inappropriate discount, or accept an order from a customer with an unpaid bill, and/or may wish to offer certain quality of service guarantees, e.g., to bound the time a potential customer has to wait between the acknowledgement of a request for a price and receiving a quote.

The upper limit on deliberation (or response) time in rule-based systems is a well-established problem. However previous work has studied expert and diagnostic systems as single isolated systems [15], and has focused mainly on termination (or worst case response time), rather than more general issues of correctness and quality of service. In this paper, we show how to establish correctness and time bounds for multi-agent systems composed of rule-based agents. The formal models of multi-agent systems we study are transition systems where each transition corresponds to a rule firing by an agent (or an act of communication, which we also model as an effect of a rule firing). To solve a particular problem or to achieve a particular state, agents typically need to fire multiple rules; the number of transitions until the goal state is reached gives an estimate of the time required for deliberation. We present a modal epistemic logic where the agent's rules and its knowledge in each state determine conditions on the accessibility relation in the model. The novelty of our work is in representing a multi-agent system of communicating rule-based agents as a transition system, and proposing a modal logic to describe these systems. In addition to being able to state time-related properties of the system, this also enables us to use existing modal logic techniques such as model-checking to state and verify properties of agents.

The remainder of the paper is organised as follows: is section 2 we introduce our model of communicating rule-based agents. In sections 3, 4 and 5 we introduce the language and its semantics, present a complete and sound axiomatisation of our logic, and state decidability. In section 7 we briefly illustrate the logic by giving an example and show how properties expressed in the logic can be verified using an existing model checker, before discussing related work in section 8.

2 COMMUNICATING RULE-BASED AGENTS

We assume that each agent has a *program*, consisting of Horn clause rules, and a working memory, which contains facts. The set of rules comprising the agent's program and the facts contained in the agent's working memory together constitute the agent's beliefs.

To define the agent's internal language more precisely, we fix a set of predicate symbols $\mathcal{P}$, a set of variables $\mathcal{X}$ and a set of constants $\mathcal{D}$. A literal λ is a predicate symbol of n arguments followed by n variables or constants and possibly preceded by a negation symbol '$\neg$'. For example, $LuxuryProduct(x)$ and

[1] School of Computer Science, University of Nottingham, Nottingham, NG8 1BB, UK, email: {nza,mtw,bsl}@cs.nott.ac.uk
[2] Jess is the reference implementation for the Java Rules API.

$LuxuryProduct(Porsche)$ are both literals. When every argument of the predicate symbol in a literal is an element of $\mathcal{D}$, e.g., $LuxuryProduct(Porsche)$, that literal is called a *ground liter* . Each rule-based agent has a finite set $\mathcal{R}$ of rules, which are of the form

$$\lambda_1, \dots, \lambda_n \rightarrow \lambda$$

where λ_i, λ are literals. λ is called the *consequent* of the rule (denoted by $\text{cons}(\lambda_1, \dots, \lambda_n \rightarrow \lambda)$), and each λ_i is an *antecedent* of the rule. An example of a rule is:[3]

$$PremiumCustomer(x), LuxuryProduct(y) \rightarrow \\ Discount(x, y, 7.5\%)$$

We assume that the rules do not contain functional symbols. Given a rule $\lambda_1, \dots, \lambda_n \rightarrow \lambda$, we denote an *instance* of this rule as $\delta(\lambda_1, \dots, \lambda_n \rightarrow \lambda)$, where δ is some substitution function from the set of variables of the rule into $\mathcal{D}$. For example, if δ assigns $Miller$ to x and $Porsche$ to y, then

$$\delta(PremiumCustomer(x), LuxuryProduct(y) \rightarrow \\ Discount(x, y, 7.5\%)) \\ = \\ PremiumCustomer(Miller), LuxuryProduct(Porsche) \rightarrow \\ Discount(Miller, Porsche, 7.5\%)$$

A rule $\lambda_1, \dots, \lambda_n \rightarrow \lambda$ *matches* if there is a substitution δ such that the agent's working memory contains $\delta(\lambda_1), \dots, \delta(\lambda_n)$. *Firing* the matching rule instance $\delta(\lambda_1, \dots, \lambda_n \rightarrow \lambda)$ adds the ground literal $\delta(\lambda)$ to the agent's working memory. In what follows, we assume that the agents fire at most one rule per cycle (we discuss other rule application strategies in section 6), and that rule matching is refractory, i.e., that each rule instance is fired at most once.

Consider a set $\mathcal{A} = \{1, \dots, n\}$ of n agents. To model communication between agents, we assume that agents have special communication constructs in their language: '$\text{tell}(i, j)\lambda$', which we will refer to as a *tell* and '$\text{ask}(i, j)\lambda$', which we will refer to as an *ask*. In both cases, i and j are agents and λ is a literal not containing an ask or a tell. $\text{tell}(i, j)\lambda$ stands for 'i tells j that λ' and $\text{ask}(i, j)\lambda$ stands for 'i asks j whether λ is the case'. The position in which these formulas may appear in a rule depends on which agent's program the rule belongs to. Agent i may have an ask or a tell with arguments (i, j), in the *consequent* of a rule, e.g.:

$$\lambda_1, \dots, \lambda_n \rightarrow \text{ask}(i, j)\lambda$$

Whereas agent j may have the same expressions in the *antecedent* of the rule. For example:

$$\text{tell}(i, j)\lambda \rightarrow \lambda$$

is a well-formed rule for agent j which makes it trust i when i informs it that λ is the case. No other occurrences of $\text{tell}(i, j)$ and $\text{ask}(i, j)$ are allowed. In particular, i must be distinct from j in all the cases just listed, for our agents neither ask nor tell themselves anything.

When a rule has either an ask or a tell as its consequent, we call it a *communication rule*, or *c-rule* for short. All other rules are known as *deduction rules*, or *d-rules*. These include rules with asks and tells in the antecedent (as well as rules containing neither an ask nor a tell).

[3] The business rules in the running example are taken from the RuleML tutorial [8].

Firing a *c-rule* instance with the consequent $\text{tell}(i, j)\lambda$ adds the literal $\delta(\text{tell}(i, j)\lambda)$ both to the working memory of i and of j. Intuitively, i has a record that it told j that $\delta(\lambda)$, and j has a record of being told by i that $\delta(\lambda)$. Similarly, if the consequent of a *c-rule* instance is of the form $\delta(\text{ask}(i, j)\lambda)$, then the corresponding ask is added to the working memories of both i and j.

In this paper, for simplicity we do not consider the interaction between agents and their environment. However we could model the environment as a distinguished agent, e.g., with a different rule application strategy, and interpret tells from the agents to the environment as actions, and tells from the environment to the agents as sensing.

3 LANGUAGE

First we define agent i's internal language $\mathcal{L}_i$. For simplicity, we assume that all agents have the same finite set of predicate symbols $\mathcal{P}$ and a finite set of constants $\mathcal{D}$ (although this assumption is not essential). We denote the set of all possible substitutions $\delta : \mathcal{X} \longrightarrow \mathcal{D}$ by Σ. Note that given finite sets $\mathcal{X}$ and $\mathcal{D}$, the set of all possible substitutions Σ and the set of all possible rule instances are finite as well. Literals are denoted by $\lambda, \lambda_1, \lambda_2, \dots, \text{tell}(i, j)\lambda, \text{ask}(i, j)\lambda, \dots$ $(i, j \in \mathcal{A})$, ground literals are denoted by $\delta(\lambda)$, $\delta(\lambda_1)$, ..., $\text{tell}(i, j)\delta(\lambda)$, $\text{ask}(i, j)\delta(\lambda)$, ..., where $\delta \in \Sigma$, and rules of the form $\lambda_1, \dots \lambda_n \rightarrow \lambda$ are denoted by $\rho, \rho_1, \rho_2, \dots$. Note that only rules and ground literals are formulas of $\mathcal{L}_i$, and that rules are well-formed only if the conditions on the occurrences of asks and tells are satisfied.

In the modal language $\mathcal{ML}$ over $\mathcal{L}_i$, we have a belief operator B_i for each agent $i \in \mathcal{A}$. The primitive wffs of $\mathcal{ML}(\mathcal{P}, \mathcal{D})$ are:

$$\text{B}_i\ \delta(\lambda) \mid \text{B}_k \text{tell}(i, j)\delta(\lambda) \mid \text{B}_k \text{ask}(i, j)\delta(\lambda) \mid B_i\rho$$

where $\delta(\lambda)$ is a ground literal, i and j are two different agents (we assume that $|\mathcal{A}| \geq 2$), $k \in \{i, j\}$, and ρ is a well-formed rule of agent i. If ϕ_1 and ϕ_2 are both $\mathcal{ML}(\mathcal{P}, \mathcal{D})$ wffs, the complex wffs of $\mathcal{ML}(\mathcal{P}, \mathcal{D})$ are then given by

$$\neg\phi_1 \mid \phi_1 \wedge \phi_2 \mid \Diamond\phi_1$$

where $\Diamond$ is the next state modality; $\vee$ and $\rightarrow$ are defined as usual, and $\Box\phi =_{df} \neg\Diamond\neg\phi$.

4 SEMANTICS

In this section we define the transition systems we use to interpret the language introduced above. The states of the transition system are tuples of local states of the agents and are used to interpret the basic belief formulas of the language; intuitively, $B_i\alpha$ is true if the formula α is either a rule of agent i's program, or is contained in the agent's local state (working memory) in that state. Transitions between states correspond to the agents firing a single rule instance each, in parallel. In the case in which an agent fires an instance of a d-rule, a single new formula is added to the agent's state. In the case in which an agent fires an instance of a *c-rule* involving another agent j, the conclusion of the rule is added both to the agent's state and to the agent j's state.

The formal definition of models is as follows.

Definition 1 (Models) *Given a group of agents $\mathcal{A} = \{1, \dots, n\}$, a multi-agent model M is an $n + 3$-tuple*

$$\langle S, \mathcal{A}, T, V_1, \dots, V_n \rangle$$

where S is a set of states, T is the transition relation and each $V_i : S \longrightarrow \wp(\mathcal{L}_i)$ is the labelling function for agent $i \in \mathcal{A}$ assigning a set of $\mathcal{L}$-formulas to each state.

*We say that a rule $\lambda_1, \ldots, \lambda_n \rightarrow \lambda$ is s-δ-i-*applicable *if s is a state, δ a substitution, i an agent, $\delta(\lambda_1), \ldots, \delta(\lambda_n) \in V_i(s)$ and $\delta(\lambda) \notin V_i(s)$.*

The following conditions on the assignments V_i (for all $i \in A$) and the accessibility relation T hold in all models:

1. *for every $i \in \mathcal{A}$, any two states $s, s' \in S$, and every rule formula ρ, $\rho \in V_i(s)$ if, and only if, $\rho \in V_i(s')$*
2. *For any two states s and s', $T(s, s')$ holds if, and only if, for every agent i, $V_i(s') = V_i(s) \cup \{\delta_i(\operatorname{cons}(\rho_i)\} \cup C_i$, where ρ_i is some s-δ_i-i-applicable rule if such a rule exists (otherwise $\{\delta_i(\operatorname{cons}(\rho_i)\} = \emptyset$), and $C_i = \{\delta_j \operatorname{cons}(\rho_j) : \delta_j \operatorname{cons}(\rho_j)$ is of the form $\operatorname{tell}(j,i)\alpha$ or $\operatorname{ask}(j,i)\alpha\}$).*

Condition 1 says that the agent's program does not change, and condition 2 says that each agent fires a single applicable rule instance if it has one, otherwise its state does not change apart from possibly as a result of communication from other agents.

A state transition system can be visualised as follows. Each state contains n finite sets of formulas, corresponding to the beliefs of each of the n agents. Each applicable rule instance of agent i corresponds to a possible 'action' by i, namely the addition of the consequent of the rule to the next state. A transition by the system is an n-tuple of such actions (if agent j has no applicable rule instance, the jth component of the tuple is a null action). If in a state s, each agent i has k_i different applicable rule instances, then there are $k_1 \times \cdots \times k_n$ different transitions from s, one for every possible combination of agent 'actions'. This also means that on different execution paths originating in s, applicable rule instances of agent i may be fired in a different order. Suppose there are two such rule instances in s, $\delta_1(\rho_1)$ and $\delta_2(\rho_2)$. Then there is a path to some state s' reached by agent i firing $\delta_1(\rho_1)$ (and other agents firing their rule instances) and from s' to s'' where i gets to fire $\delta_2(\rho_2)$. There is also a path where the order in which $\delta_1(\rho_1)$ and $\delta_2(\rho_2)$ are fired is reversed, and it is also possible that in the next state agent i will acquire other applicable rule instances which on some execution paths will be fired earlier than the one instance remaining from s. If none of the agents has an applicable rule instance, then only one transition is possible (a null action by all agents), to the same state.

The relation of a formula ϕ being satisfied in a model M ($M \Vdash \phi$) is defined as follows:

$M, s \Vdash B_i\alpha$ iff $\alpha \in V_i(s)$, for $i \in \mathcal{A}, \alpha \in \mathcal{L}(\mathcal{P}, \mathcal{D})$
$M, s \Vdash \neg\phi$ iff $M, s \nVdash \phi$
$M, s \Vdash \phi \wedge \psi$ iff $M, s \Vdash \phi$ and $M, s \Vdash \psi$
$M, s \Vdash \Diamond\phi$ iff there is a $u \in S$ such that $T(s, u)$ and $M, u \Vdash \phi$

Although our agents share a common language, each has its own program. Given a program (set of rules) $\mathcal{R}_i$ for each agent $i \in \mathcal{A}$, we define the *program set* $\mathbb{R} = \{\mathcal{R}_1, \ldots, \mathcal{R}_n\}$. We want to pick out those models in which agents believe all the rules in their program and no other rules. Such models comprise the class $\mathbf{M}_{\mathbb{R}}$.

Definition 2 (The class $\mathbf{M}_{\mathbb{R}}$) *A model $M \in \mathbf{M}_{\mathbb{R}}$, where $\mathbb{R} = \{\mathcal{R}_1, \ldots, \mathcal{R}_n\}$, iff for every state s in M, $M, s \Vdash B_i\rho$ iff $\rho \in \mathcal{R}_i$. Given a set of $\mathcal{ML}$-formulas Γ and an $\mathcal{ML}$-sentence ϕ, we write $\Gamma \Vdash_{\mathbb{R}} \phi$ iff every model of Γ which is in the class $\mathbf{M}_{\mathbb{R}}$ is also a model of ϕ.*

We now ask, given a set of programs $\mathbb{R} = \{R_1, \ldots R_n\}$ for n agents in the language $\mathcal{L}(\mathcal{P}, \mathcal{D})$, what logic corresponds to the class $\mathbf{M}_{\mathbb{R}}$?

5 AXIOMATISATION

The logic $\Lambda_{\mathbb{R}}$ over $\mathcal{ML}$ is defined as follows. First, fix a set of agents $\mathcal{A} = \{1, \ldots, n\}$, then fix a program set $\mathbb{R} = \{\mathcal{R}_1, \ldots, \mathcal{R}_n\}$. Then we define

$$\delta \operatorname{app}_i(\lambda_1, \ldots, \lambda_n \rightarrow \lambda) \overset{df}{=} \mathrm{B}_i\delta(\lambda_1) \wedge \ldots \wedge \mathrm{B}_i\delta(\lambda_n) \wedge \neg\mathrm{B}_i\delta(\lambda)$$

and $\operatorname{app}(\rho) \overset{df}{=} \bigvee_{\delta \in \Sigma} \delta \operatorname{app}_i(\rho)$, where $\rho \in R_i$. The logic $\Lambda_{\mathbb{R}}$ has the following axiom schemata and rules:

Cl all classical propositional tautologies
K $\Box(\phi \rightarrow \psi) \rightarrow (\Box\phi \rightarrow \Box\psi)$
A1 $\mathrm{B}_i\rho$, where $\rho \in \mathcal{R}_i$ (agent i believes its rules)
A2 $\neg\mathrm{B}_i\rho$, where $\rho \notin \mathcal{R}_i$ (agent i only believes its rules)
A3 $\mathrm{B}_i\alpha \rightarrow \Box\mathrm{B}_i\alpha$ (agents are monotonic reasoners)
A4 $\mathrm{B}_i(\lambda_1, \ldots, \lambda_n \rightarrow \lambda) \wedge \mathrm{B}_i\delta(\lambda_1) \wedge \ldots \wedge \mathrm{B}_i\delta(\lambda_n) \rightarrow \Diamond\mathrm{B}_i\delta(\lambda)$
for each $\delta \in \Sigma$ (if a rule matches, its consequent belongs to some successor state)
A5 $\Diamond(\mathrm{B}_i\alpha \wedge \mathrm{B}_i\beta) \rightarrow (\mathrm{B}_i\alpha \vee \mathrm{B}_i\beta)$
where α and β are not of the form $\operatorname{tell}(j,i)\lambda$ or $\operatorname{ask}(j,i)\lambda$ (at most one new belief of agent i is added in each transition as a result of agent i firing a rule)
A6 $\Diamond\mathrm{B}_i\alpha \rightarrow \big(\mathrm{B}_i\alpha \vee \bigvee_{\lambda_1,\ldots,\lambda_n \rightarrow \lambda \in \mathcal{R}_i, \delta(\lambda)=\alpha} \mathrm{B}_i\delta(\lambda_1) \wedge \ldots \wedge \mathrm{B}_i\delta(\lambda_n)\big)$
where α is not of the form $\operatorname{tell}(j,i)\beta$ or $\operatorname{ask}(j,i)\beta$ (new beliefs of agent i only arise as a result of firing a rule of i, unless it is a communication from another agent j)
A7 $\delta^1_1 \operatorname{app}_1(\rho^1_1) \wedge \ldots \wedge \delta^1_{k1} \operatorname{app}_1(\rho^1_{k1}) \wedge \bigwedge_{(\delta,\rho)\notin\{(\delta^1_1,\rho^1_1),\ldots,(\delta^1_{k1},\rho^1_{k1})\}} \neg\delta \operatorname{app}_1(\rho) \quad \wedge \ldots$
$\delta^n_1 \operatorname{app}_n(\rho^n_1) \wedge \ldots \wedge \delta^n_{kn} \operatorname{app}_n(\rho^n_{kn}) \wedge \bigwedge_{(\delta,\rho)\notin\{(\delta^n_1,\rho^n_1)\ldots(\delta^n_{kn},\rho^n_{kn})\}} \neg\delta \operatorname{app}_n(\rho) \rightarrow \Box \bigvee_{f(i)\in\{1\ldots ki\}} (\mathrm{B}_1\delta_{f(1)}(\operatorname{cons}(\rho_{f(1)})) \wedge \ldots \wedge \mathrm{B}_n\delta_{f(n)}(\operatorname{cons}(\rho_{f(n)}))))$
for each possible set of applicable rule instances $\{\delta^i_1(\rho^i_1), \ldots, \delta^i_{ki}(\rho^i_{ki})\}$ for each agent i, provided this set is non-empty for at least one of the agents. This axiom constrains all possible successors of each state to states resulting from firing applicable rule instances in parallel by all agents.
A8 $\bigwedge_{\rho\in\mathcal{R}} \neg \operatorname{app}\rho \rightarrow \Diamond \bigwedge_{\rho\in\mathbb{R}} \neg \operatorname{app}\rho$ (the terminating state has a transition to itself)
A9 $\mathrm{B}_i\operatorname{tell}(i,j)\lambda \leftrightarrow \mathrm{B}_j\operatorname{tell}(i,j)\lambda$ (communication involving tells is perfect)
A10 $\mathrm{B}_i\operatorname{ask}(i,j)\lambda \leftrightarrow \mathrm{B}_j\operatorname{ask}(i,j)\lambda$ (communication involving asks is perfect)
MP From ϕ and $\phi \rightarrow \psi$ derive ψ
N From ϕ derive $\Box\phi$

The notion of derivation is standard.

Theorem 1 *$\Lambda_{\mathbb{R}}$ is sound and complete for $\mathbf{M}_{\mathbb{R}}$: $\Gamma \vdash_{\mathbb{R}} \phi$ iff $\Gamma \Vdash_{\mathbb{R}} \phi$.*

Due to the lack of space, we can only sketch a proof here. Soundness is proved as usual by induction of the length of a derivation. For completeness, we build a canonical model for a $\Lambda_{\mathbb{R}}$-consistent set of formulas and prove that this model is in $\mathbf{M}_{\mathbb{R}}$.

Theorem 2 *The satisfiability problem for $\mathbf{M}_{\mathbb{R}}$ is decidable.*

The proof is omitted due to lack of space; it uses filtration to prove that every satisfiable formula has a model of size bounded by the size of the formula and the size of $\mathbb{R}$.

6 ADDITIONAL AXIOMS

In this section, we discuss additional axiom schemata which we can use to express interesting properties of agents.

First, we consider properties relating to agent communication. The property of agent 1 trusting agent 2 with respect to some literal λ can be expressed as a *c-rule* belonging to agent 1. This is reflected in the following axiom:

Trust(1,2,λ) $\mathrm{B}_1(\mathrm{tell}(2,1)\lambda \to \lambda)$

The following axiom says that agent 1 is cooperative in that it believes that it should answer agent 2's queries concerning λ, if it already believes that λ:

Answer(1,2,λ) $\mathrm{B}_1(\mathrm{ask}(2,1)\lambda, \lambda \to \mathrm{tell}(1,2)\lambda)$

We can state more general properties of trust and cooperativeness by replacing the axioms above by axiom schemas: e.g., trust any agent i rather than a particular agent, and do so with respect to all formulas, rather than a particular predicate.

Another kind of property we can express relates to the rule application strategy of the agents. The logic described above is based on the assumption that agents may fire matching rule instances in any order. This is not a very realistic assumption (although it still allows us to establish an upper bound on when a certain belief will be derived, because all possible rule orderings will include the actual one). For example, an agent's rules (and rule instances) may be ordered, or more recent information may be acted upon first, or queries answered in preference to applying internal deduction rules. Logics corresponding to an ordered rule application strategy were studied in detail in [5]; unfortunately, rule ordering in general cannot be captured by a single, simple axiom schema.

However, there are some rule application strategies which are relatively simple to capture in modal logic. For example, the 'all rules at each cycle' strategy can be easily modelled.[4] This strategy requires the agent to apply all of its rules to all of its ground beliefs in order to transit to the next state. Note that rules are not applied immediately to the *consequences* of matching rule instances; the rules are applied only once in each transition. To model the 'all rules at each cycle' strategy, we remove axiom **A5** (which says that at most one new formula is derived) and add the axiom schema which says that there is at most one successor to each state: $\Diamond\phi \to \Box\phi$. Note that **A7** then becomes redundant.

7 EXAMPLE

In this section we show how the logic defined above can be used to specify temporal properties of belief change in multi-agent systems in a precise and realistic way. We then go on to show how these properties can be verified using an existing model checker.

Consider a simple multi-agent system which implements the business rules example introduced in section 2. Suppose there are two agents involved in giving a quote. Agent 1 is in charge of quoting for products, and has data relating to current prices and the following discount rules:

[4] This is also the strategy that is used in step logic [13].

R0 $Quote(x,y) \to \mathrm{ask}(1,2)PremiumCustomer(x)$
R1 $\mathrm{tell}(2,1)PremiumCustomer(x) \to PremiumCustomer(x)$
R2 $Quote(x,y), PremiumCustomer(x), LuxuryProduct(y) \to Discount(x,y,7.5\%)$

Rule **R0** states that if a quote is required for customer x, ask agent 2 whether x is a premium customer. **R1** states that agent 1 should believe agent 2 if agent 2 tells agent 1 that x is a premium customer, and **R2** states that premium customers get at 7.5% discount. Agent 2 is in charge of customer data, and has rules to determine who counts as a premium customer:

R3 $Spending(x, min5000, 2005) \to PremiumCustomer(x)$
R4 $\mathrm{ask}(1,2)PremiumCustomer(x), PremiumCustomer(x) \to \mathrm{tell}(2,1)PremiumCustomer(x)$

Suppose we are interested in quality of service guarantees of the form: every request for a quote is answered within at most 4 timesteps. To simplify, we have reformulated this as a question of whether agent 1 believes that a particular customer qualifies for a discount in at most 4 steps, provided that the information on customer spending and luxury products is immediately available. This can be re-expressed as a problem of verifying the formula

Q $\Box^4\mathrm{B}_1 Discount(Miller, Porsche, 7.5\%)$

(where $\Box^n$ stands for n nestings of $\Box$) in a state where agent 1 believes $Quote(Miller, Porsche)$ and $LuxuryProduct(Porsche)$, and agent 2 believes $Spending(Miller, min5000, 2005)$. This reformulation presupposes that no other queries are being processed by agents 1 and 2.

It is straightforward to verify this property using existing model checking techniques. As proof of concept, we converted the example multi-agent system above into a system specification for the Mocha model checker [6] and verified that the formula **Q** is true. The translation into *reactive modules*, the description language used by Mocha, is straightforward: ground literals in the agent's working memory are represented as boolean *state variables*, rules are represented by *atoms* (which describe the initial condition and transition relation for a group of related variables), and agents by *modules* (collections of atoms specifying which state variables are visible from outside the module). The multi-agent system is then simply a parallel composition of agent modules (this translation assumes an 'all rules at each cycle' rule application strategy, but 'one instance at a time' is also possible). The translation of $\mathcal{ML}$ formula **Q** expressing the property to be verified into the ATL specification language used by Mocha is similarly straightforward.

This example is extremely simplified, and gives the response time guarantee in terms of 'inference steps' rather than of units of time. However, if we know the number of beliefs an agent has and the rule-matching strategy it uses, we can produce a mapping from 'steps' to time durations, and verify quality of service guarantees such as 'every request for a quote will be answered within n milliseconds'.

8 RELATED WORK

Our work relates both to the field of epistemic logics (and their use in verifying properties of agents) and to work on verifying properties of rule-based programs, such as bounded response time.

A considerable amount of work has been done in the area of model-checking multi-agent systems (see, e.g., [9, 7]). However, the emphasis of this work is on correctness rather than the timing properties of agents, which is our primary interest. An exception is our recent joint work on automatic verification of space and time requirements for resource-bounded agents [3, 2]. However this work focuses on verifying properties of a single agent and does not consider the multi-agent case.

Another strand of related work is verifying temporal properties of agent systems, in particular real-time properties, where actions are assumed to take non-trivial time. In [22], Singh proposed a framework for modelling agent systems where actions have duration and can be executed in parallel. As far as we know, it was not applied to actions as inferences. Recent work in dynamic and temporal epistemic logic [23] has a similar motivation to our work (modelling knowledge change) but assumes that the agent's knowledge is deductively closed before and after the update. There are, however, approaches in epistemic logic which explicitly reflect time required for a certain inference. For example, step logics [13] study development of agents' belief sets over discrete linear time. In [16] Grant, Kraus and Perlis present a semi-decidable formalism for expressing agent knowledge and inference steps with explicit time increments, which is implemented in Prolog. Other related work [12, 1, 4] also describes epistemic logics where each inference step takes the agent into the next (or some future) moment in time.

There has also been 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 Chen and Cheng on predicting the response time of OPS5-style production systems. In [10], 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 [11], 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.

9 CONCLUSION

We have presented a sound, complete and decidable logic which describes how the beliefs of communicating agents which reason using rules evolve over time. This logic can be used to express and verify temporal properties of multi-agent systems such as 'if agent i asks agent j λ, agent j is guaranteed to reply within n inference cycles'. This is useful, for example, for verifying quality of service guarantees, which may have the form of 'each query is going to be answered within n milliseconds'.

We are aware of a number of limitations of the work presented here. We currently assume that the agent's reasoning is monotonic (they never discard beliefs, only acquire new ones). It is, however, possible to expand our framework to incorporate rules which delete information, either to restore consistency or due to memory limitations and we plan to investigate this in future work. (An example of a similar logic which models a reasoner who deletes formulas to save memory can be found in [3].) Finally, the assumptions on communications are only made for the sake of simplicity; it is very easy to modify the semantics to allow for communication delays and lossy communication channels.

ACKNOWLEDGEMENTS

This work was partially supported by the Royal Society grant 'Model-checking resource-bounded agents'.

REFERENCES

[1] T. Ågotnes and M. Walicki, 'Strongly complete axiomatizations of "knowing at most" in standard syntactic assignments', in *Pre-proceedings of the Sixth International Workshop on Computational Logic in Multi-agent Systems (CLIMA VI)*, London, UK, (June 2005).

[2] A. Albore, N. Alechina, P. Bertoli, C. Ghidini, B. Logan, and L. Serafini, 'Model-checking memory requirements of resource-bounded reasoners', in *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI'06)*. AAAI Press, (2006). (to appear).

[3] N. Alechina, P. Bertoli, C. Ghidini, M. Jago, B. Logan, and L. Serafini, 'Verifying space and time requirements for resource-bounded agents', Technical Report T05-10-03, ITC-irst, Trento, Italy, (2005).

[4] N. Alechina, B. Logan, and M. Whitsey, 'A complete and decidable logic for resource-bounded agents', in *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2004)*, pp. 606–613, New York, (July 2004). ACM Press.

[5] N. Alechina, B. Logan, and M. Whitsey, 'Modelling communicating agents in timed reasoning logics', in *Proceedings of the Ninth European Conference on Logics in Artificial Intelligence (JELIA 2004)*, LNAI Vol. 3229, pp. 95–107, Lisbon, (September 2004). Springer.

[6] R. Alur, T. A. Henzinger, F. Y. C. Mang, S. Qadeer, S. K. Rajamani, and S.Tasiran, 'MOCHA: Modularity in model checking', in *Computer Aided Verification*, pp. 521–525, (1998).

[7] M. Benerecetti, F. Giunchiglia, and L. Serafini, 'Model checking multiagent systems.', *J. Log. Comput.*, **8**(3), 401–423, (1998).

[8] H. Boley, B. Grosof, and S. Tabet. RuleML tutorial, 2005. http://www.ruleml.org/papers/tutorial-ruleml.html.

[9] R. Bordini, M. Fisher, W. Visser, and M. 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)*, pp. 896–903, New York, (2004). ACM Press.

[10] J.-R. 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*, p. 203. IEEE Computer Society, (1995).

[11] A. M. K. Cheng and H. 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).

[12] H. N. Duc, 'Reasoning about rational, but not logically omniscient, agents', *Journal of Logic and Computation*, **7**(5), 633–648, (1997).

[13] J. Elgot-Drapkin, M. Miller, and D. Perlis, 'Memory, reason and time: the Step-Logic approach', in *Philosophy and AI: Essays at the Interface*, 79–103, MIT Press, Cambridge, Mass., (1991).

[14] E. Friedman-Hill, *Jess in Action: Java Rule-Based Systems*, Manning Publications Co., 2003.

[15] M. P. Georgeff and A. L. Lansky, 'Reactive reasoning and planning', in *Proceedings of the Sixth National Conference on Artificial Intelligence, AAAI-87*, pp. 677–682, (1987).

[16] 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).

[17] I. Horrocks and P. F. Patel-Schneider, 'A proposal for an OWL rules language.', in *Proceedings of the 13th international conference on World Wide Web, WWW 2004*, pp. 723–731. ACM, (2004).

[18] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. SWRL: A semantic web rule language, 2003. http://www.daml.org/rules/proposal.

[19] Q. H. Mahmoud. Getting started with the Java rule engine API (JSP 94): Toward rule-based applications, 2005. http.

[20] M. Paolucci, A. Ankolekar, N. Srinivasan, and K. P. Sycara, 'The DAML-S virtual machine.', in *International Semantic Web Conference*, LNCS Vol. 2870, pp. 290–305. Springer, (2003).

[21] M. Paolucci and K. Sycara, 'Autonomous semantic web services', *IEEE Internet Computing*, **7**(5), 34 – 41, (2003).

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

[23] H.P. van Ditmarsch, W. van der Hoek, and B.P. Kooi, 'Dynamic epistemic logic with assignment', in *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2005)*, pp. 141–148. ACM New York, (2005).