Time, Knowledge, and Choice

(PRELIMINARY REPORT)

Michael Wooldridge

Department of Computing
Manchester Metropolitan University
Chester Street, Manchester M1 5GD
United Kingdom

M.Wooldridge@doc.mmu.ac.uk

Abstract. This article considers the link between theory and practice in agent-oriented programming. We begin by rigorously defining a new formal specification language for autonomous agents. This language is the expressive branching time logic CTL*, enriched by the addition of two further modal connectives, for representing *knowledge* and *seeing to it that* (stit). These connectives are *grounded*: given a concrete semantics in terms of the states and actions of an agent. This grounding makes it possible to establish a precise relationship between the specification language and deterministic automata, and in particular, the automatic synthesis of agents from logical specifications becomes a possibility. This possibility, and the potential problems associated with it, are discussed at length. The paper closes with a summary of future research issues and directions.

1 Introduction

It has become common practice in Artificial Intelligence (AI) to use non-classical logics to characterize desirable properties of autonomous intelligent agents and multi-agent systems; see [29] for a recent survey of such work. These theories are generally intended to serve as *specifications* for future autonomous agents. (This point is made explicitly in [6, p223].) And yet, if one adopts this viewpoint, then one immediately runs into difficulties. It is by no means clear how one might take even a simple specification expressed in a logic such as that defined in [6], and from it systematically derive an implementation that satisfies the specification. Perhaps the key difficulty is that such logics are typically given an abstract, *ungrounded* semantics in terms of so-called *possible worlds* [5]. This style of semantics is in many respects attractive (in particular, the associated mathematics of *correspondence theory* is very elegant). However, in general, it is not clear what possible worlds should correspond to within an agent, and there is thus no clearly defined relationship between the logic and agents as they might actually be built[1]. If the *agent-oriented programming* paradigm ([22]) is ever to become a workaday reality, this issue must be addressed.

In this paper, we hope to bring the theory and practice of agent-oriented programming closer together. We define an expressive branching time logic for developing agent

[1] The obvious exception is the situated automata paradigm [20].

specifications, which contains modal connectives for representing *knowledge* and *seeing to it that* (stit) [3]. This logic provides the agent designer with a rich collection of primitives for specifying agents. For example, one can use the logic to write formulae that express the sense of ‘if the agent ever knows φ, then it should see to it that ψ’, without worrying about *how* the agent brings about ψ. However, the semantics of this logic are *grounded*: given a precise interpretation in terms of the states and actions of automata-like agents. This grounding allows us to precisely define the circumstances under which an agent can be said to *satisfy* a specification, and suggests the possibility of automatically synthesizing agents from specifications given in the logic. This paper can thus be seen to build on the situated automata paradigm, where agents were in part synthesized from specifications given in terms of a modal logic of knowledge [20].

The remainder of the paper is structured as follows. We begin, in the following section, by rigorously defining the new logic. In section 3, we examine the relationship between this logic and automata-like agents, and define the conditions under which an agent satisfies a specification. We also speculate on the automatic synthesis of agents. The paper ends, in section 4, with a discussion on the relationship of the new logic to other work, and future research issues.

2 A Logic of Time, Knowledge, and Choice

We now introduce the agent specification language, which we shall call $\mathcal{L}$: this language is a logic of time, knowledge, and choice. The remainder of this sub-section provides an overview of the key semantic and syntactic features of $\mathcal{L}$, and the sub-sections that follow rigorously define these aspects. A discussion on the properties of $\mathcal{L}$ appears in section 2.3, and a discussion on the relationship of $\mathcal{L}$ to other formalisms appears in section 4.

It is intended that $\mathcal{L}$ will be used for specifying (and ultimately, verifying and synthesizing) systems that contain an *environment* part and an *agent* part. Intuitively, the agent part represents an active autonomous component, that is able to perceive the environment, and change the environment state through the performance of *actions*. It is the agent that is the primary focus of study in this article. The environment part of the system represents ‘everything else’ in the world — although in reality this ‘everything else’ will contain other objects of interest (in particular, other agents), for our purposes it is acceptable to simply lump these other components into one. The *state* of a system can thus be characterized as an agent state/environment state pair.

$\mathcal{L}$ is a *branching time* logic: formulae are interpreted over tree-like branching time structures, which represent all conceivable ways that the system could evolve. Nodes in a time structure correspond to *system states*, and arcs correspond to the performance of actions by the agent. A key notion is that of a *path* through a time structure. A path represents a *history*, or *course of events*: intuitively, one way that the system could evolve. Note that a path is associated with the notion of *choice* on the part of the agent: at each state along the path, the agent must select one action to perform, from a repertoire of alternatives. The performance of this action constrains the possible evolution of the system, and thus commits the agent to some subset of the possible futures that were available to it from that state. Thus, we have a sense of a path encapsulating choices made by an agent; this notion will turn out to be important later in the paper.

Syntactically, $\mathcal{L}$ is a propositional version of the expressive branching time logic CTL* [8], extended by the addition of two further modal connectives: K and S. The K connective is used to represent the agent's *knowledge*. The semantics of K are given in the standard way, by using *possible worlds* semantics [12]. However, the semantics are *grounded*, in the way proposed by Rosenschein for his situated automata paradigm [20], and equivalently, by members of the distributed systems community [9]. Thus a formula $\mathsf{K}\varphi$ represents the fact that the information φ is *implicit* within the agent's state; φ need not be *represented* within the agent at all, and there is no sense of the information being immediately *available* to the agent:

> 'In this model, knowledge is an *external* notion. We don't imagine a processor scratching its head wondering whether or not it knows a fact φ. Rather, a programmer ... would say, from the outside, that the processor knew φ because in all global states [indistinguishable] from its current state (intuitively, all the states the processor could be in, for all it knows), φ is true'. [10, p6]

The semantics of K ensure that its logic corresponds to the normal modal system S5 [5], which is widely accepted as a logic of *idealized knowledge* [12].

The S connective is used to represent an agent *seeing to it that* (stit) [3, 16, 1, 2]. Intuitively, $\mathsf{S}\varphi$ represents the fact that φ is satisfied as a necessary consequence of a choice that the agent has made (the formula schema $\mathsf{S}\varphi$ is more usually written as $[i \text{ stit } \varphi]$ in the stit literature, where i is a term denoting an agent; since we are considering only a single agent, we find it more convenient to use the short form). Alternatively, one might say that $\mathsf{S}\varphi$ if φ is a necessary consequence of the agent's decision making process. Our intention is that the semantics of S should correspond closely to the semantics of stit modalities as given in, for example, [2, p153]. However, rather than using the somewhat abstract semantic scheme described in [2], we take care to *ground* the semantics of S, giving it a concrete interpretation in terms of the states of the agent and its actions[2]. Thus, the semantics of both K and S are ultimately given in terms of automata. This decision has some advantages when we use $\mathcal{L}$ for specification, and in the synthesis of automata from $\mathcal{L}$ specifications, an issue we discuss in section 3.1.

Turning to the CTL* connectives of $\mathcal{L}$, a distinction is made between *state formulae* and *path formulae* [8, pp156–157]. A path formula, as its name suggests, is one that is interpreted with respect to a *path* through a branching time structure. Paths correspond to histories or runs of the system. In contrast, a state formula is interpreted with respect to a system state. If φ is a path formula, then the CTL* path quantifier A can be applied to φ to yield a formula $\mathsf{A}\varphi$ that is satisfied in some state if φ is satisfied on all paths originating from that state. Intuitively, $\mathsf{A}\varphi$ represents the fact that φ is *inevitable*. If φ is a path formula, then the CTL* connective $\bigcirc$ can be applied to φ to yield a formula $\bigcirc\varphi$ that is satisfied on a path p iff φ is satisfied on the path obtained from p by removing its first state. Intuitively, $\bigcirc$ means 'in the next state'. If both φ and ψ are path formulae,

[2] Segerberg proposed a similar grounding in his logic of *bringing it about* [21]. However, Segerberg's starting point was the notion of a program (in the dynamic logic sense), rather than automata.

$$
\begin{array}{rcl}
\langle \mathit{prop} \rangle & ::= & \text{any member of } \Phi \\
\langle \mathit{path\text{-}fmla} \rangle & ::= & \langle \mathit{state\text{-}fmla} \rangle \\
& \mid & \neg \langle \mathit{path\text{-}fmla} \rangle \\
& \mid & \langle \mathit{path\text{-}fmla} \rangle \vee \langle \mathit{path\text{-}fmla} \rangle \\
& \mid & \bigcirc \langle \mathit{path\text{-}fmla} \rangle \\
& \mid & \langle \mathit{path\text{-}fmla} \rangle \mathsf{U} \langle \mathit{path\text{-}fmla} \rangle \\
\langle \mathit{state\text{-}fmla} \rangle & ::= & \mathsf{true} \\
& \mid & \langle \mathit{prop} \rangle \\
& \mid & \neg \langle \mathit{state\text{-}fmla} \rangle \\
& \mid & \langle \mathit{state\text{-}fmla} \rangle \vee \langle \mathit{state\text{-}fmla} \rangle \\
& \mid & \mathsf{A} \langle \mathit{path\text{-}fmla} \rangle \\
& \mid & \mathsf{K} \langle \mathit{state\text{-}fmla} \rangle \\
& \mid & \mathsf{S} \langle \mathit{state\text{-}fmla} \rangle \\
\langle \mathit{wff} \rangle & ::= & \langle \mathit{state\text{-}fmla} \rangle
\end{array}
$$

Fig. 1. Syntax of $\mathcal{L}$

then they can be combined using the binary CTL* connective U to make a formula $\varphi \mathsf{U} \psi$ that is satisfied on a path p iff φ is satisfied at all points along p *until* ψ is satisfied. Note that state formulae are also path formula, which can be interpreted on a path p by interpreting them in the first state in p. Finally, we will later introduce a number of other connectives, that will be *derived* from the basic stock described above. These connectives can be considered as *abbreviations* for those described here.

The remainder of this section is structured as follows: section 2.1 formally introduces the syntax of $\mathcal{L}$, and section 2.2 formally introduces the semantics. In section 2.3, we discuss the properties of $\mathcal{L}$.

2.1 Syntax of $\mathcal{L}$

The logic $\mathcal{L}$ is an extension of propositional CTL* [8], which is in turn an extension of classical propositional logic. Formulae of $\mathcal{L}$ (more accurately, well-formed formulae) are thus built from a set Φ of *primitive propositions*, and may be combined using the classical propositional connectives $\vee$ ('or') and $\neg$ ('not'). The remaining classical connectives ($\wedge$ — 'and', $\Rightarrow$ — 'if ... then ...', $\Leftrightarrow$ — 'iff') are assumed to be introduced as abbreviations, as we indicated above. We also assume the language contains a logical constant true, for truth. In addition to these symbols, the alphabet of $\mathcal{L}$ contains the CTL* path quantifier A ('on all paths'), the temporal connectives $\bigcirc$ ('next') and U ('until'), and finally the modal connectives K ('knows') and S ('sees to it that'). The syntax of $\mathcal{L}$ is then defined by the grammar that appears in Figure 1.

2.2 Semantics of $\mathcal{L}$

We present the semantics of $\mathcal{L}$ in two parts: first, in the following sub-section, we define the basic semantic concepts and structures that underpin $\mathcal{L}$; we then formally define the semantics of the language with reference to these structures.

Basic concepts: Our intention is that $\mathcal{L}$ will allow us to represent the properties of a system containing an *environment* part and an active, autonomous *agent* part, which is able to change the state of the environment through the performance of *actions*. We assume that the system is populated by a *single* agent in this paper — in section 4, we comment on the multi-agent case.

At any instant, it is assumed that the environment part of the system may be in any of a set $E = \{e, e', \ldots\}$ of *environment states*. Similarly, the agent may be in any of a set $L = \{l, l', \ldots\}$ of *local*, or *agent states*. (The internal structure of an environment or agent state is not an issue here; see [9, pp335–339] for a discussion.) The state of a system at any instant may therefore be represented as an ordered pair (e, l), where $e \in E$ and $l \in L$. Let $G = E \times L$ be the set of all *global*, or *system states*. We use $g, g', \ldots$ to stand for elements of G. We assume a selection function $es : G \to E$, which takes a global state and returns its environment state part, and a similar function $ls : G \to L$, which takes a global state and returns its local state part. (Formal definitions of these functions are trivial.)

The state of a system may be modified by the performance of an action by the agent. Let $A = \{a, a', \ldots\}$ be the set of all actions available to the agent. It is assumed that the environment is *non-deterministic*, in that the performance of an action in some state may result in a set of possible outcomes.

We now introduce the idea of *knowledge*. Suppose that $g = (e, l)$ and $g' = (e', l')$ are global states. Under what circumstances does the agent have *the same information* in g and g'? Clearly, it will implicitly carry the same information in g and g' iff $l = l'$. In this case, g and g' are said to be *indistinguishable* to the agent, and are thus *knowledge equivalent* [11, pp46–47].

Definition 1. *Let* $\sim \subseteq G \times G$ *be a* knowledge equivalence relation*, defined by* $g \sim g'$ *iff* $ls(g) = ls(g')$.

It is convenient to define a similar relation $\approx$ for environment states.

Definition 2. *Let* $\approx \subseteq G \times G$ *be an* environment equivalence relation*, defined by* $g \approx g'$ *iff* $es(g) = es(g')$.

A standard result is that $\sim$ is indeed an equivalence relation [9, pp342–343]; it is not difficult to see that $\approx$ also has this property. We will later use $\sim$ to give a Kripke-style semantics to the K modality.

From any global state, the agent has available a set A of actions, from which it must select just one to perform. Choosing one of these actions for execution commits the agent to some subset of the futures that were possible from the state. This leads naturally to the idea of the possible histories of the system branching infinitely into the future from each state. We assume that the system has some defined starting point (i.e., the past is finite), and let the total tree relation $R \subseteq G \times G$ represent all possible courses of system history. A pair (G, R), where G is a set of global states, and $R \subseteq G \times G$ is a total tree relation over G, is known as a *branching time structure*. Thus $(g, g') \in R$ iff the state

g could be transformed into the state g' by the execution of an action. We use a labeling function $Act : R \to A$ to associate arcs in R with the action the arc corresponds to. We refer to a triple (G, R, Act), where (G,R) is a branching time structure and $Act : R \to A$ is a labeling function, as a *labeled branching time structure*.

We now introduce some technical apparatus for manipulating branching time structures.

Definition 3. *Let (G,R) be a branching time structure. A* finite path *through R is a sequence $(g_0, g_1, \ldots, g_k)$ such that $\forall u \in \{0, \ldots, k-1\}$, we have $(g_u, g_{u+1}) \in R$. Let $fpaths(G,R)$ denote the set of finite paths though R. An* infinite path *(or just 'path') through R is a sequence $(g_u \mid u \in I\!N)$ such that $\forall u \in I\!N$, we have $(g_u, g_{u+1}) \in R$. Let $paths(G,R)$ be the set of infinite paths over (G,R). If p is a (finite or infinite) path and $u \in I\!N$, then the uth element of p is $p(u)$. (Thus $p(0)$ is the first element of p.) If $p \in fpaths(G,R)$ then the last element of p is given by $last(p)$. If p is a (finite or infinite) path and $u \in I\!N$, then the path obtained from p by removing its first u elements is denoted by $p^{(u)}$. Let $root(R)$ denote the root (intuitively, the starting state) of R.*

A path p through a labeled branching time structure (G, R, Act) can be visualized as follows:

$$p : (e_0, l_0) \xrightarrow{a_0} (e_1, l_1) \xrightarrow{a_0} \cdots \xrightarrow{a_{k-1}} (e_k, l_k) \xrightarrow{a_k} \cdots$$

where $\forall u \in I\!N$, we have $p(u) = (e_u, l_u)$, and $Act(p(u), p(u+1)) = a_u$. It should be clear that a branching time structure (G,R) intuitively represents all possible *executions* or *runs* of the agent in the environment, whereas a path through (G,R) represents just *one* possible execution/run of the agent.

Choice and determinism: We now discuss the notion of *choice* with respect to the structures we have just introduced. We begin by defining a *choice equivalence relation* over global states. Intuitively, we write $g \equiv g'$, for global states g and g', iff g and g' are possible outcomes of a choice just made by the agent.

Definition 4. *Global states g and g' are said to be* choice equivalent *with respect to labeled branching time structure (G, R, Act) iff $\exists g'' \in G$ such that $\{(g'', g), (g'', g')\} \subseteq R$ and $Act(g'', g) = Act(g'', g')$.*

It is not difficult to see that $\equiv$ is an equivalence relation. An illustration of $\equiv$ appears in Figure 2. In this figure, the agent is faced by two choices when the system is in state g_0: execute either action a or action a'. If the agent executes a, then as a result, the system will either end up in state g_2 or state g_3. If the agent chooses to execute a', then the system will either end up in state g_1 or state g_4. Clearly, the two choices that the agent has available to it when the system is in state g_0 induce two equivalence classes of possible outcomes, and so $g_1 \equiv g_4$, and $g_2 \equiv g_3$, but $g_1 \not\equiv g_2$, $g_1 \not\equiv g_3$, $g_4 \not\equiv g_2$, and $g_4 \not\equiv g_3$. The $\equiv$ relation will be used to define a semantics for the stit modality, 'S'.

Later in this paper, we find it useful to talk about the notion of choice relative to paths (rather than states, as in the $\equiv$ relation). We now introduce some notation and terminology that captures this sense of choice. Suppose on some path p, at 'time' $u \in I\!N$,

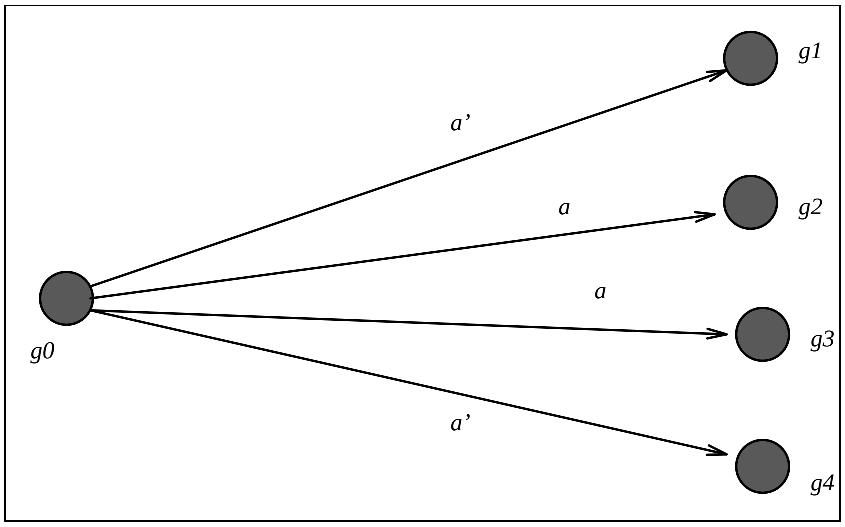

Fig. 2. Choice Equivalence

the agent is in state l, and chooses some action a_1. Sometime later, the agent is once again in state l, but this time chooses action a_2, where $a_1 \neq a_2$. Intuitively, on the basis of the same information (implicit within its state), the agent is selecting different actions. This behaviour is *non-deterministic*. We can make a similar observation about the way that an agent changes its internal state. Suppose that we have two states g and g' such that $g = g'$. Then we would expect that, since these states carry the same information in the agent ($g \sim g'$) and the environment is indistinguishable in these states ($g \approx g'$), a deterministic agent would subsequently choose to be in the same internal state, and hence carry the same information.

We are interested in paths, (and in fact models), where the agent's behaviour *is* deterministic. The reason for this interest in deterministic behaviour is that we shall later examine the relationship between deterministic automata and the semantic structures that underpin $\mathcal{L}$. We can formalize this intuitive notion of determinism as follows.

Definition 5. *Let* (G, R, Act) *be a labeled branching time structure. Then* (G, R, Act) *is said to be* deterministic *iff:*

1. $\forall g, g' \in G$ *such that* $g \sim g'$*, if* $(g, g'') \in R$ *and* $(g', g''') \in R$*, then* $Act(g, g'') = Act(g', g''')$*; and*
2. $\forall g, g' \in G$ *such that* $g = g'$*, if* $(g, g'') \in R$ *and* $(g', g''') \in R$*, then* $g'' \sim g'''$.

If p *is a (finite or infinite) path through a deterministic labeled branching time structure, then we say* p *is also deterministic.*

Next, we introduce the idea of a *choice function*. Such a function is an abstract representation of an agent's decision-making process.

Definition 6. *A* choice function*, c, has the signature $c : L \to A$, i.e., on the basis of an agent's local state, it selects an action. Let C be the set of all choice functions.*

If p is a deterministic path, then it is possible to extract from p a choice function, representing that which has guided the agent's decision making on that path. (If p is non-deterministic, then no such choice function can be recovered, as the action selected from a particular local state may not be uniquely defined.) We define a function $\hat{c}$, that reads off a choice function from a deterministic path.

Definition 7. *If p is a deterministic path, then $\hat{c}(p) \in C$ is the choice function associated with p:*

$$\hat{c}(p) \overset{\text{def}}{=} \{l \mapsto a \mid \exists u \in I\!N \text{ such that } ls(p(u)) = l \text{ and } Act(p(u), p(u+1)) = a\}.$$

Similarly, we define a *next state* function to be one that characterizes the way an agent changes state.

Definition 8. *A* next state *function, ns, has the signature $ns : E \times L \to L$, i.e., on the basis of the environment and internal state, it selects an internal state. Let NS be the set of all next state functions.*

As for choice functions, we define a function $\widehat{ns}$ that reads off a next state function from a deterministic path.

Definition 9. *If p is a deterministic path, then $\widehat{ns}(p) \in NS$ is the next state function associated with p:*

$$\widehat{ns}(p) \overset{\text{def}}{=} \{g \mapsto l \mid \exists u \in I\!N \text{ such that } p(u) = g \text{ and } ls(p(u+1)) = l\}.$$

Formal semantics of $\mathcal{L}$: We now introduce *logical models* for $\mathcal{L}$.

Definition 10. *A model, M, for $\mathcal{L}$, is a structure:*

$$M = (G, R, A, Act, \pi)$$

where (G, R, Act) is a labeled branching time structure (over the set of actions A); and $\pi : \Phi \times G \to \{T, F\}$ is a valuation function*, which says for every primitive proposition $p \in \Phi$, and every global state $g \in G$, whether p is true (T) or false (F) in g.*

Semantic rules for $\mathcal{L}$: We must define the semantics of state and path formulae separately. The semantics of path formulae are given via the path formula satisfaction relation '$\models$', which holds between pairs of the form (M, p) (where M is an $\mathcal{L}$-model and p is a path in M), and path formulae of $\mathcal{L}$. The rules defining this relation are given in Figure 3. The semantics of state formulae are given via the state formula satisfaction relation, which for convenience we also write as '$\models$'. (Context will always make it clear which relation we mean.) This relation holds between pairs of the form (M, g), (where

$(M,p) \models \varphi$	iff $(M,p(0)) \models \varphi$ (where φ is a state formula)
$(M,p) \models \neg\varphi$	iff $(M,p) \not\models \varphi$
$(M,p) \models \varphi \vee \psi$	iff $(M,p) \models \varphi$ or $(M,p) \models \psi$
$(M,p) \models \bigcirc\varphi$	iff $(M,p^{(1)}) \models \varphi$
$(M,p) \models \varphi \mathsf{U} \psi$	iff $\exists u \in I\!N$ such that $(M,p^{(u)}) \models \psi$ and
	$\forall v \in I\!N,$ if $(0 \leq v < u),$ then $(M,p^{(v)}) \models \varphi$

Fig. 3. Path Formulae Semantics

$(M,g) \models \mathsf{true}$	
$(M,g) \models p$	iff $\pi(p,g) = T$ (where $p \in \Phi$)
$(M,g) \models \neg\varphi$	iff $(M,g) \not\models \varphi$
$(M,g) \models \varphi \vee \psi$	iff $(M,g) \models \varphi$ or $(M,g) \models \psi$
$(M,g) \models \mathsf{K}\varphi$	iff $\forall g' \in G,$ if $g \sim g',$ then $(M,g') \models \varphi$
$(M,g) \models \mathsf{S}\varphi$	iff (i) $\forall g' \in G,$ if $g \equiv g'$ then $(M,g') \models \varphi,$ and
	(ii) $\exists g' \in G$ such that $(M,g') \not\models \varphi$
$(M,g) \models \mathsf{A}\varphi$	iff $\forall p \in paths(G,R),$ if $p(0) = g,$ then $(M,p) \models \varphi$

Fig. 4. State Formulae Semantics

M is an $\mathcal{L}$-model and *g* is a global state in *M*), and state formulae. The rules defining this relation appear in Figure 4.

With the exception of the rule for S, all of the rules in Figures 3 and 4 represent either standard definitions, or adaptations of such. The rule for S, however, requires further explanation[3].

A formula Sφ is intended to represent the fact that the agent sees to it that φ is satisfied [3]; more precisely, that φ is a necessary consequence of a choice made by the agent. The semantic rule for S may be paraphrased as follows. Sφ is satisfied in some state *g* iff:

1. before being in state *g*, the system was in some state g', and the choice made by the agent in state *g* guaranteed that φ would be satisfied in subsequent states;
2. φ actually *is* forced to be satisfied as a result of the agent's choice — φ could possibly have been false.

The first part of the semantic rule for S is generally referred to as the *positive* condition, and the second as the *negative* condition. The semantic account presented here is closely based on [30, p334]; the reader may like to compare this account with the various other accounts of stit semantics given in the literature: [3, p191], [16, p381], [1,

[3] In an earlier version of this paper, (presented at the ATAL-95 workshop), we used a more elaborate semantics for the stit modality, with a very much more complex formalization of choice. The semantics given here correspond more closely to 'standard' stit semantics, and should be regarded as the more up-to-date version.

p153], [4, p466], [31, p460] (more precisely, our S modality is a *deliberative stit*, or *dstit* connective; the non-deliberative stit modality does not have the negative condition).

Throughout the remainder of the paper, we assume the usual definitions of satisfiability, validity, validity in a model, and so on.

Derived connectives: In addition to the basic connectives defined above, it is useful to *derive* some further connectives. These derived connectives do not add to the expressive power of the language, but are intended to make formulae more concise and readable. First, we introduce the *existential path quantifier*, E, which is defined as the dual of the universal path quantifier A:

$$\mathsf{E}\varphi \overset{\text{def}}{=} \neg\mathsf{A}\neg\varphi.$$

Thus a formula $\mathsf{E}\varphi$ is interpreted as 'on some path, φ', or 'optionally φ'.

It is also convenient to introduce further temporal connectives. The unary connective $\Diamond$ means 'sometimes'. Thus the path formula $\Diamond\varphi$ will be satisfied on some path if φ is satisfied at some point along the path. The unary $\Box$ connective means 'now, and always'. Thus $\Box\varphi$ will be satisfied on some path if φ is satisfied at all points along the path. We also have a weak version of the U connective: a formula $\varphi\mathsf{W}\psi$ is read 'φ *unless* ψ'.

$$\Diamond\varphi \overset{\text{def}}{=} \mathsf{true}\mathsf{U}\varphi \qquad \Box\varphi \overset{\text{def}}{=} \neg\Diamond\neg\varphi \qquad \varphi\mathsf{W}\psi \overset{\text{def}}{=} (\varphi\mathsf{U}\psi) \vee \Box\varphi.$$

Thus $\varphi\mathsf{W}\psi$ means that either: (i) φ is satisfied until ψ is satisfied, or else (ii) φ is always satisfied. It is *weak* because it does not require that ψ be eventually satisfied.

2.3 Properties of $\mathcal{L}$

After introducing a new logic by means of its syntax and semantics, it is usual to illustrate its properties, typically by means of a Hilbert-style axiom system. However, no complete axiomatization is currently known for CTL*, the logic that underpins $\mathcal{L}$[4]. For this reason, instead of attempting a complete axiomatization, we simply present some valid formulae of $\mathcal{L}$.

First, observe that the semantics of $\mathcal{L}$ generalize those of classical propositional logic, and so standard propositional modes of reasoning can be used in $\mathcal{L}$.

Lemma 11.

1. *If φ is a substitution instance of a classical tautology, then $\models \varphi$.*
2. *If $\models \varphi \Rightarrow \psi$ and $\models \varphi$, then $\models \psi$.*

Next, we turn to the knowledge connective, K. This is essentially a normal modal necessity connective, with semantics given via the equivalence relation $\sim$. Thus the logic of K corresponds to the well-known normal modal system S5 [12].

[4] The system presented in [24] reportedly contains an error in the proof of completeness.

Lemma 12.

1. *Axiom* K-*K:* $\models \mathsf{K}(\varphi \Rightarrow \psi) \Rightarrow ((\mathsf{K}\varphi) \Rightarrow (\mathsf{K}\psi))$.
2. *Axiom* K-*T:* $\models \mathsf{K}\varphi \Rightarrow \varphi$.
3. *Axiom* K-*5:* $\models \neg\mathsf{K}\varphi \Rightarrow \mathsf{K}\neg\mathsf{K}\varphi$.
4. K-*necessitation: If* $\models \varphi$ *then* $\models \mathsf{K}\varphi$.

Turning to the CTL* connectives in $\mathcal{L}$, the universal path quantifier A also behaves rather like a normal modal necessity connective with logic S5 [24].

Lemma 13.

1. *Axiom* A-*K:* $\models \mathsf{A}(\varphi \Rightarrow \psi) \Rightarrow ((\mathsf{A}\varphi) \Rightarrow (\mathsf{A}\psi))$.
2. *Axiom* A-*T:* $\models \mathsf{A}\varphi \Rightarrow \varphi$.
3. *Axiom* A-*5:* $\models \neg\mathsf{A}\varphi \Rightarrow \mathsf{A}\neg\mathsf{A}\varphi$.
4. A-*necessitation: If* $\models \varphi$ *then* $\models \mathsf{A}\varphi$.

In addition, state formulae have a number of other properties with respect to path quantifiers.

Lemma 14. *If* φ *is a state formula, then* $\models \varphi \Leftrightarrow \mathsf{E}\varphi \Leftrightarrow \mathsf{A}\varphi$.

Next, we turn to the S connective. The proof theoretic properties of stit modalities have been investigated in detail by Xu [30, 31], and the following discussion borrows freely from his work. Suppose that the semantic rule for S did not include the second, negative condition. Then S would be a normal modal connective with a semantics given via the choice equivalence relation, $\equiv$. As we noted above, the modal logic of equivalence relations is S5 [5]. However, the negative condition in the semantic rule for S makes the logic of S weaker than S5. We claim that the logic of S has the following properties:

Lemma 15.

1. *If* $\models \varphi$ *then* $\models \neg\mathsf{S}\varphi$.
2. *If* $\models \neg\varphi$, *then* $\models \neg\mathsf{S}\varphi$.
3. *Axiom* S-*K:* $\models \mathsf{S}(\varphi \Rightarrow \psi) \Rightarrow ((\mathsf{S}\varphi) \Rightarrow (\mathsf{S}\psi))$.
4. *Axiom* S-*T:* $\models \mathsf{S}\varphi \Rightarrow \varphi$.
5. *Axiom* S-*4:* $\models \mathsf{S}\varphi \Rightarrow \mathsf{S}\mathsf{S}\varphi$.
6. $\not\models \neg\mathsf{S}\varphi \Rightarrow \mathsf{S}\neg\mathsf{S}\varphi$.

Lemma 15(1) is a variant of the necessitation rule from normal modal logic. To see that this rule is sound, assume the antecedent. Then φ is valid. Hence it cannot be the case that $(M,g) \models \mathsf{S}\varphi$ for any (M,g), as this would contradict the negative condition of the semantic rule for S. Thus the antecedent implies that $(M,g) \models \neg\mathsf{S}\varphi$ for arbitrary (M,g), which is our result. The proof of Lemma 15(2) is obvious: if φ is inconsistent, it cannot ever satisfy the positive condition in the semantic rule for S, hence $\mathsf{S}\varphi$ is inconsistent, and $\neg\mathsf{S}\varphi$ is valid.

Lemma 15(3) corresponds to axiom K from normal modal logic: if one sees to it that $\varphi \Rightarrow \psi$, and one also sees to it that φ, then one sees to it that ψ. To see that this formula schema is valid, assume that $(M,g) \models \mathsf{S}(\varphi \Rightarrow \psi)$ and $(M,g) \models \mathsf{S}\varphi$, for arbitrary (M,g). We need to show that this assumption implies $(M,g) \models \mathsf{S}\psi$. This, in turn, requires us to prove the positive and negative conditions of the semantic rule for S hold. The positive part is obvious, following exactly the proof for K in normal modal logic. Now suppose that the negative part of the semantic rule for $\mathsf{S}\psi$ was *not* satisfied. Then it must be that ψ is satisfied in all states in M. But in this case, $\varphi \Rightarrow \psi$ would also be satisfied in all states in M. But this implies that it could not be the case that $(M,g) \models \mathsf{S}(\varphi \Rightarrow \psi)$, which is a contradiction. Hence the negative part of the semantic rule for $\mathsf{S}\psi$ must also be satisfied, and we have our result. Lemma 15(4) and (5) correspond to axioms T and 4 from normal modal logic, respectively, and the proofs of these properties are obvious.

Since the stit modality is based on an equivalence relation, one might expect to see other S5-like properties for S. However, Lemma 15(6) shows that the axiom corresponding to 5 from normal modal logic does not hold for S. To see why, assume it did. Then let φ be an inconsistent formula, such as false. For arbitrary (M,g), we would thus have $(M,g) \models \neg\mathsf{S}\varphi$. But it could not then be the case that $(M,g) \models \mathsf{S}\neg\mathsf{S}\varphi$, because the validity of $\neg\mathsf{S}\varphi$ would contradict the negative condition of the S semantic rule. Xu shows how one can obtain a weakened version of 5 that holds for S modalities, by using a 'historical necessity' connective (axiom T4, [30, p336]).

3 On the Specification and Synthesis of Agents

An important motivation for the work presented in this paper is that there is often only an intuitive relationship between a formalism ostensibly intended for reasoning about agents, and agents as they might actually be built. A key objective of this work is to develop a formalism that provides useful and powerful abstractions for the agent designer, while having a well-defined interpretation in terms of real systems. To this end, we focus in this section on the relationship between $\mathcal{L}$ and agents. We begin by formally defining our agents (which in fact correspond to Moore type sequential machines), and go on to discuss the notion of $\mathcal{L}$-formulae as specifications for agents, and the automatic synthesis of agents from $\mathcal{L}$-specifications.

Definition 16. *An agent, α, is a structure:*

$$\alpha = (L, E, A, ns, c, l_0)$$

where:

- $L = \{l, l', \ldots\}$ *is a set of* local*, or* agent states*, (as above);*
- $E = \{e, e', \ldots\}$ *is a set of* external*, or* environment states*, (as above), representing inputs to the agent;*
- $A = \{a, a', \ldots\}$ *is a set of* actions*, (as above), representing those that the agent might perform;*
- $ns : E \times L \to L$ *is a* next state *function (as above);*
- $c : L \to A$ *is a* choice *function (as above); and*
- $l_0 \in L$ *is an* initial state.

In section 2.2, we indicated that a labeled branching time structure is intended to represent all possible executions (runs) of an agent in some environment. We now formally define this relationship.

Definition 17. *Let (G, R, Act) be a labeled branching time structure, and*

$$\alpha = (L, E, A, ns, c, l_0)$$

be an agent. Then (G, R, Act) represents *α, (notation $represents((G, R, Act), \alpha)$) iff:*

1. *$ls(root(R)) = l_0$;*
2. *$\forall g, g' \in G$, if $(g, g') \in R$ then $Act(g, g') = c(ls(g))$; and*
3. *$\forall g, g' \in G$, if $(g, g') \in R$, then $ns(g) = ls(g')$.*

This relation can be extended to models: we abuse notation and write $represents(M, \alpha)$ if the labeled branching time structure in M represents agent α.

We can now begin to consider the notion of $\mathcal{L}$ as an *agent specification language*. An agent specification will be given as an $\mathcal{L}$-formula φ, which will typically take the form $\varphi = \bigwedge_{i=1}^{n} \psi_i$, where each ψ_i, for $1 \leq i \leq n$, represents a desirable property of the agent. These properties will typically be either *liveness* properties (intuitively stating that 'something good will eventually happen'), or *safety* properties (intuitively stating that 'nothing bad ever happens') [7]. A specification identifies a class of models: those in which the specification formula is valid. We will say an agent α satisfies a specification φ iff φ is valid in any model that represents α.

Definition 18. *An agent α* satisfies *specification φ (notation $\{\alpha\} \models \varphi$) iff for all deterministic models M, we have $represents(M, \alpha)$ implies $M \models \varphi$.*

3.1 On the Automatic Synthesis of Agents

An obvious problem is how one goes from an agent specification to an implementation that satisfies the specification. In mainstream software engineering, this transformation is achieved through an iterative process of *refinement*, gradually moving from a high-level, abstract representation of a system's desirable properties, down to concrete data structures and programs [14]. However, it is not clear how this scheme might work for languages such as $\mathcal{L}$, which have possible worlds semantics. So, instead of attempting such manual refinement, we propose to *automatically synthesize* agents from specifications, in the way pioneered by Rosenschein and Kaelbling in their situated automata paradigm [20]. This process becomes feasible because we have been careful in $\mathcal{L}$ to *ground* the semantics of the K and S modalities in the states and actions of automata.

To see how the automatic synthesis might be achieved, first note that, given a deterministic model M, it is possible to read off an agent α such that $represents(M, \alpha)$.

Definition 19. *Let $M = (G, R, A, Act, \pi)$ be a deterministic model. Then the agent $\alpha = (L, E, A, ns, c, l_0)$ corresponding to M is denoted by $agent(M)$, and is defined by:*

$$\begin{aligned}
L &\stackrel{\text{def}}{=} \{ls(g) \mid g \in G\} \\
E &\stackrel{\text{def}}{=} \{es(g) \mid g \in G\} \\
A &\stackrel{\text{def}}{=} \text{ran } Act \\
ns &\stackrel{\text{def}}{=} \bigcup\{\widehat{ns}(p) \mid p \in fpaths(G,R)\} \\
c &\stackrel{\text{def}}{=} \bigcup\{\hat{c}(p) \mid p \in fpaths(G,R)\} \\
l_0 &\stackrel{\text{def}}{=} ls(root(R))
\end{aligned}$$

Clearly, for all deterministic models M, the relation $represents(M, agent(M))$ holds. Two points to note:

- in general, models may contain infinitely many states, and so it would not be possible in practice to read off an agent from *any* model;
- if a model is non-deterministic, then the next-state and choice functions will not be well-defined, meaning that it is impossible to read off an agent from such models (this is why we focused on deterministic models).

We comment below on the significance of these points. The next problem to be addressed is that of taking a specification φ and from it generating a model M such that $M \models \varphi$. The idea here is to use an algorithm based on *constructive theorem proving*. In constructive theorem proving, one demonstrates the satisfiability of a formula by systematically searching for a model of the formula. If such a model is found, the formula is satisfiable; otherwise it is unsatisfiable. One popular approach to constructive theorem proving is the method of semantic tableaux [23]. This method has been applied to modal and temporal logics, (including the S5 logic of knowledge [12] and a restricted form of CTL* known as CTL [7]), as well as combinations of such logics [28, 18]. Tableaux methods for temporal and modal logics do not, in fact, return models, but structures (often called Hintikka structures) from which models may be extracted in a systematic way. These structures, which generally resemble labeled graphs or multi-graphs, have the property of being *finite representations* of models. Thus the first point noted above is not likely to be an issue.

We are currently developing an algorithm, based on semantic tableaux, that will automatically generate models (and hence agents) from $\mathcal{L}$-specifications.

4 Discussion

In this final section, we discuss the relationship between the work presented in this paper, and other work in both AI and mainstream computer science. We also provide some pointers to future work issues.

The work presented in this paper may be seen as contribution to Shoham's agent-oriented programming (AOP) paradigm [22]. The key idea of AOP is that of directly programming agents in terms of *mentalistic* constructs (such as knowledge, belief, desire, intention, and so forth). The rationale for this approach is that these constructs provide powerful *abstraction tools* for reasoning about complex systems. A stated aim of this article was to develop an agent specification language that provides such abstractions. Perhaps the most obvious ways in which our work differs from that of Shoham (and other AOP researchers [19]) are: (i) the insistence on *grounded* semantics, and (ii) the use of a stit modality (as opposed to, for example, desires and intentions). The use of grounded semantics was justified at length in the preceding section. It is therefore worth focusing on the use of the stit modality.

There are at least three reasons for choosing to use stit modalities. First, they provide a powerful level of abstraction for specifying agents. When we specify that the agent stit φ, we are not at all concerned with *how* the agent brings about φ, nor need we be concerned with this aspect until we come to implement the agent. This separation of *what* we want our agent to do from *how* the agent is to do it is, of course, very desirable from the point of view of software engineering.

The second advantage to the stit modality is that, as we have demonstrated in this paper, it can be given a grounded semantics. It is not so obvious how one might go about grounding, say, desires and intentions in a BDI logic [18]. (The *intuitive* grounding of beliefs and desires is obvious, but establishing a *formal* grounding, in the way that we have done for K and S in this paper, seems more problematic.) A final advantage is that slits are not *mentalistic* constructs, in the way that, for example, desires and intentions are. We therefore need not worry about the relationship between the S modality and the components of an agents cognitive makeup. The stit modality is simply an abstraction tool: a convenient way of specifying desirable properties of agents.

It may not be obvious that the stit modality actually adds anything to the expressive power of the logic. For example, one might suppose that, rather than specifying $\Diamond\mathsf{S}\varphi$ (eventually see to it that φ), one could simply specify $\Diamond\varphi$, (eventually φ), doing away with the stit modality completely. However, we argue that this latter specification contains no notion of *agency*: $\Diamond\varphi$ does not require that the *agent* bring about φ, but simply that φ be satisfied. The former specification thus expresses a stronger, more explicit requirement.

We are aware of very little other work in AI that uses stit modalities; the only other work that comes immediately to mind is [15], who uses a deontic-stit logic to examine some legal issues associated with agency. Krogh's logic is not given a model-theoretic semantics, though this was not really an issue in his work. Also, his logic is only intended as a specification language in a fairly abstract sense. Other researchers have come quite close to the notion of stit: Werner, for example, has developed a language in which a formula $\triangleright\ \varphi$ expresses the fact that φ is forced to come about as a result of the 'strategies' the agent is following [25]. In earlier work, we also used a similar idea in an attempt to give a grounded semantics to goals [27].

Another way of looking at the work presented in this paper is as an extension of the situated automata paradigm: the basic idea (of automatically synthesizing an automata-like machine from a declarative, modal logic specification), was pioneered by Rosenschein and Kaelbling [20]. This article extends their work in several ways, perhaps most importantly by the use of a branching time logic and the introduction of a stit modality.

Finally, note that there are some similarities between the work programme sketched out in this paper, and the automatic synthesis of reactive systems from temporal logic specifications in mainstream computer science. For example, in [17], Pnueli and Rosner show that the synthesis of an '$\forall\exists$' reactive module can be viewed as a proof problem in branching temporal logic.

Issues for Future Work

There are many obvious ways in which the work presented in this article needs to be extended:

Automatic synthesis: We are currently developing an algorithm to achieve the automatic synthesis process outlined in section 3. In addition to the development of the basic algorithm, issues such as computational complexity and expressiveness of the specification language must also be addressed. Temporal logics of knowledge are known to have a hard decision problem, and our stit modality will almost certainly add to the complexity [13].

The multi-agent case: Another obvious area of work is the extension of the logic presented here to the more general multi-agent case. While the principles remain the same, a key difficulty in extending the logic in this way will be dealing with *concurrency* [26].

Example specifications: We have used the language $\mathcal{L}$ to develop several small agent specifications (omitted due to space restrictions). However, we need to gain experience not only with larger specifications, but with multi-agent specifications.

Acknowledgments: Jörg Müller gave helpful comments on the first draft of this paper. The ATAL-95 attendees also gave a lot of suggestions.

References

1. N. Belnap. Backwards and forwards in the modal logic of agency. *Philosophy and Phenomenological Research*, LI(4):777–807, December 1991.
2. N. Belnap. Before refraining: Concepts for agency. *Erkenntnis*, 34:137–169, 1991.
3. N. Belnap and M. Perloff. Seeing to it that: a canonical form for agentives. *Theoria*, 54:175–199, 1988.
4. N. Belnap and M. Perloff. The way of the agent. *Studia Logica*, 51:463–484, 1992.
5. B. Chellas. *Modal Logic: An Introduction.* Cambridge University Press: Cambridge, England, 1980.
6. P. R. Cohen and H. J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42:213–261, 1990.
7. E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 996–1072. Elsevier Science Publishers B.V.: Amsterdam, The Netherlands, 1990.
8. E. A. Emerson and J. Y. Halpern. 'Sometimes' and 'not never' revisited: on branching time versus linear time temporal logic. *Journal of the ACM*, 33(1):151–178, 1986.
9. R. Fagin, J. Y. Halpern, and M. Y. Vardi. What can machines know? on the properties of knowledge in distributed systems. *Journal of the ACM*, 39(2):328–376, 1992.

10. J. Y. Halpern. Reasoning about knowledge: An overview. In J. Y. Halpern, editor, *Proceedings of the 1986 Conference on Theoretical Aspects of Reasoning About Knowledge*, pages 1–18. Morgan Kaufmann Publishers: San Mateo, CA, 1986.
11. J. Y. Halpern. Using reasoning about knowledge to analyze distributed systems. *Annual Review of Computer Science*, 2:37–68, 1987.
12. 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.
13. J. Y. Halpern and M. Y. Vardi. The complexity of reasoning about knowledge and time. I. Lower bounds. *Journal of Computer and System Sciences*, 38:195–237, 1989.
14. C. B. Jones. *Systematic Software Development using VDM (second edition)*. Prentice Hall, 1990.
15. C. Krogh. The rights of agents. In M. Wooldridge, J. P. Müller, and M. Tambe, editors, *Intelligent Agents Volume II — Proceedings of the IJCAI-95 Workshop on Agent Theories, Architectures, and Languages*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 1996. (In this volume).
16. M. Perloff. *STIT* and the language of agency. *Synthese*, 86:379–408, 1991.
17. A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proceedings of the Sixteenth ACM Symposium on the Principles of Programming Languages*, January 1989.
18. A. S. Rao. Decision procedures for propositional linear-time Belief-Desire-Intention logics. In M. Wooldridge, J. P. Müller, and M. Tambe, editors, *Intelligent Agents Volume II — Proceedings of the IJCAI-95 Workshop on Agent Theories, Architectures, and Languages*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 1996. (In this volume).
19. A. S. Rao and M. P. Georgeff. An abstract architecture for rational agents. In C. Rich, W. Swartout, and B. Nebel, editors, *Proceedings of Knowledge Representation and Reasoning (KR&R-92)*, pages 439–449, 1992.
20. S. Rosenschein and L. P. Kaelbling. The synthesis of digital machines with provable epistemic properties. In J. Y. Halpern, editor, *Proceedings of the 1986 Conference on Theoretical Aspects of Reasoning About Knowledge*, pages 83–98. Morgan Kaufmann Publishers: San Mateo, CA, 1986.
21. K. Segerberg. Bringing it about. *Journal of Philosophical Logic*, 18:327–347, 1989.
22. Y. Shoham. Agent-oriented programming. *Artificial Intelligence*, 60(1):51–92, 1993.
23. R. M. Smullyan. *First-Order Logic*. Springer-Verlag: Heidelberg, Germany, 1968.
24. C. Stirling. Completeness results for full branching time logic. In *REX School-Workshop on Linear Time, Branching Time, and Partial Order in Logics and Models for Concurrency*, Noordwijkerhout, Netherlands, 1988.
25. E. Werner. What can agents do together: A semantics of co-operative ability. In *Proceedings of the Ninth European Conference on Artificial Intelligence (ECAI-90)*, pages 694–701, Stockholm, Sweden, 1990.
26. M. Wooldridge. This is MYWORLD: The logic of an agent-oriented testbed for DAI. In M. Wooldridge and N. R. Jennings, editors, *Intelligent Agents: Theories, Architectures, and Languages (LNAI Volume 890)*, pages 160–178. Springer-Verlag: Heidelberg, Germany, January 1995.
27. 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)*, pages 234–238, Vienna, Austria, 1992.
28. 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.

29. M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.
30. M. Xu. Decidability of deliberative *STIT* theories with multiple agents. In D. M. Gabbay and H. J. Ohlbach, editors, *Temporal Logic — Proceedings of the First International Conference (LNAI Volume 827)*, pages 332–348. Springer-Verlag: Heidelberg, Germany, July 1994.
31. M. Xu. On the basic logic of *STIT* with a single agent. *Journal of Symbolic Logic*, 60(2):459–483, June 1995.