Practical Reasoning with Procedural Knowledge

(A LOGIC OF BDI AGENTS WITH KNOW-HOW)

Michael Wooldridge

Department of Computing
Manchester Metropolitan University
Chester Street, Manchester M1 5GD
United Kingdom
M.Wooldridge@doc.mmu.ac.uk

Abstract. In this paper, we present a new logic for specifying the behaviour of multi-agent systems. In this logic, agents are viewed as *BDI* systems, in that their state is characterised in terms of *beliefs*, *desires*, and *intentions*: the semantics of the BDI component of the logic are based on the well-known system of Rao and Georgeff. In addition, agents have available to them a library of plans, representing their 'know-how': procedural knowledge about how to achieve their intentions. These plans are, in effect, programs, that specify how a group of agents can work in parallel to achieve certain ends. The logic provides a rich set of constructs for describing the structure and execution of plans. Some properties of the logic are investigated, (in particular, those relating to plans), and some comments on future work are presented.

1 Introduction

There is currently much international interest in computer systems that go under the banner of *intelligent agents* [16]. Crudely, an intelligent agent is a system that is situated in a dynamic environment, of which it has an incomplete view, and over which it can exert partial control through the performance of actions. Agents will typically be allocated several (possibly conflicting) tasks, and will be required to make decisions about how to achieve these tasks in time for these decisions to have useful consequences [8].

An obvious research problem is to devise software architectures that are capable of satisfying these requirements. Various solutions have been proposed, many of which are reviewed in [16]. One solution in particular, that is currently the subject of much ongoing research, is the *belief-desire-intention* (BDI) architecture [10]. A representative BDI architecture, (the PRS [4]), is illustrated in Figure 1. As this figure shows, a BDI architecture typically contains four key data structures. An agent's *beliefs* correspond to information the agent has about the world, which may be incomplete or incorrect. Beliefs may be as simple as variables, (in the sense of, e.g., PASCAL programs), but implemented BDI agents typically represent beliefs symbolically (e.g., as PROLOG-like facts [4]). An agent's *desires* intuitively correspond to the tasks allocated to it. (Implemented BDI agents require that desires be consistent, although *human* desires often fail in this respect.)

An agent's *intentions* represent desires that it has committed to achieving. The intuition is that an agent will not, in general, be able to achieve *all* its desires, even if these

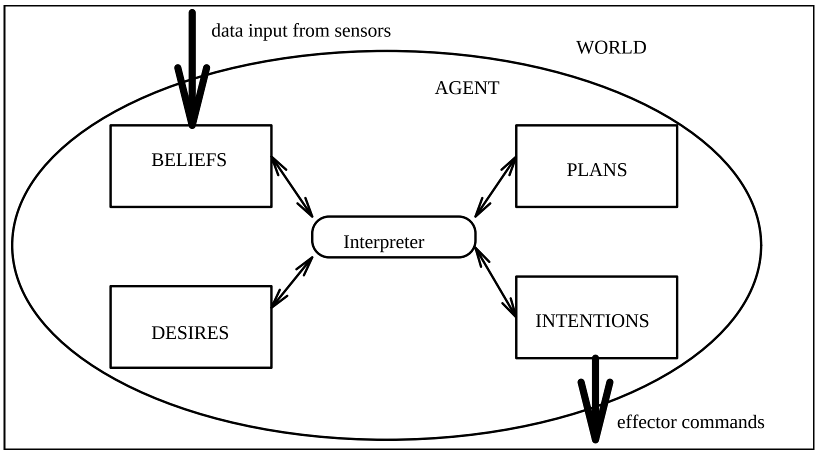

Fig. 1. A BDI Agent Architecture

desires *are* consistent. Agents must therefore fix upon some subset of available desires and commit resources to achieving them. These chosen desires are *intentions*. An agent will typically continue to try to achieve an intention until either it believes the intention is satisfied, or else it believes the intention is no longer achievable [2].

The final data structure in a BDI agent is a *plan library*. A plan library is a set of plans (a.k.a. *recipes*) that specify courses of action that may be followed by an agent in order to achieve its intentions. An agent's plan library represents its *procedural knowledge*, or *know-how*. A plan contains two parts: a *body*, or *program*, which defines a course of action; and a *descriptor*, which states both the circumstances under which the plan can be used (i.e., its pre-condition), and what intentions the plan may be used in order to achieve (i.e., its post-condition).

The *interpreter* in Figure 1 is responsible for updating beliefs from observations made of the world, generating new desires (tasks) on the basis of new beliefs, and selecting from the set of currently active desires some subset to act as intentions. Finally, the interpreter must select an action to perform on the basis of the agent's current intentions and procedural knowledge.

In order to give a formal semantics to BDI architectures, a range of *BDI logics* have been developed by Rao and Georgeff [9, 11]. These logics are extensions to the branching time logic CTL* [3], which also contain normal modal connectives for representing beliefs, desires, and intentions. Most work on BDI logics has focussed on possible relationships between the three 'mental states' [9], and more recently, on developing proof methods for restricted forms of the logics [11].

In short, the aim of this paper is to extend the basic BDI framework [9, 11] with an apparatus that allows us to represent the plans (options) that agents have available to them, and how these plans can be executed. As such, this paper builds on earlier attempts to represent BDI agents with plan libraries [12, 7], as well as more general attempts to represent agents that can act in complex ways [2]. We begin, in the following subsection, with a brief rationale for our work, and then in section 2, we introduce the basic semantic objects that underpin our new logic, and then formally define plans and the semantics of plan execution. In section 3, we present the new logic itself, which we shall call $\mathcal{L}$. Some properties of the logic are investigated, and some conclusions are presented in section 4.

Motivation: Some previous attempts have been made to graft a logic of plans onto the basic BDI framework [12, 7]. However, these logics treat plans as *syntactic* objects. This makes it unclear what plans *denote*, and also makes the semantics of quantifying over plans somewhat complex. For these reasons, we here introduce a new BDI logic of planning agents: $\mathcal{L}$. The new logic is similar in many respects to those defined in [12, 7], (in particular, the BDI semantics are based on [9]). However, we give a *semantic* account of plans, that has the flavour of dynamic logic [5].

2 Plans and Plan Execution

As noted above, we intend our logic $\mathcal{L}$ to let us represent the properties of reasoning agents, each of which has associated with it a *plan library*, containing plans that it can use in order to achieve its intentions. In this section, we formally define plans and the semantics of plan execution. We begin by introducing the basic semantic objects of our logic.

2.1 Worlds, Situations, and Paths

The logic $\mathcal{L}$ that we develop in section 3 allows us to represent the properties of a system that may evolve in different ways, depending upon the choices made by *agents* within it. We let D_{Ag} be the set of all agents, and use a binary *branching time relation*, R, to model all possible courses of system history. The relation R holds over a set T of *time-points*, i.e., $R \subseteq T \times T$. Any time-point may be transformed into another through the execution of a *primitive action* by some agent: arcs in R thus correspond to the performance of such actions. We let D_{Ac} be the set of all primitive actions, and assume an arc labeling function *Act* that gives the action associated with every arc in R. Similarly, we assume a function *Agt*, which gives the agent associated with every primitive action.

Definition 1. A *world* is a pair (T', R'), where $T' \subseteq T$ is a non-empty set of time points, and $R' \subseteq T' \times T'$ is a total, backwards linear branching time relation on T'. Let $W = \{w, w', \ldots\}$ be the set of all worlds (over T). If $w \in W$, then we write T_w for the set of time points in w, and R_w for the branching time relation in w.

Definition 2. A pair (w, t), where $w \in W$ and $t \in T_w$, is known as a *situation*. If $w \in W$, then the set of all situations in w is denoted by S_w, i.e., $S_w = \{(w, t) \mid t \in T_w\}$. Let $S = \bigcup_{w \in W} S_w$ be the set of all situations. We use s (with decorations: $s', s_1, \ldots$) to stand for members of S.

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

$\langle$*sit-set*$\rangle$	$::=$	any element of $\wp(S)$	(conditions)
$\langle$*plan-body*$\rangle$	$::=$	any element of D_{Ac}	(primitive actions)
	$\mid$	$\langle$*plan-body*$\rangle$; $\langle$*plan-body*$\rangle$	(sequential composition)
	$\mid$	$\langle$*plan-body*$\rangle$ ‘$\mid$’ $\langle$*plan-body*$\rangle$	(non-deterministic choice)
	$\mid$	$\langle$*plan-body*$\rangle$ $\parallel$ $\langle$*plan-body*$\rangle$	(parallel composition)
	$\mid$	$\langle$*plan-body*$\rangle*$	(iteration)
	$\mid$	$\langle$*sit-set*$\rangle$?	(test actions)

Fig. 2. Plan Body Structure

Definition 3. Let $w \in W$ be a world. Then a *finite path* through w is a sequence

$$(t_0, t_1, \ldots, t_k)$$

of time points, such that $\forall u \in \{0, \ldots, k-1\}$, we have $(t_u, t_{u+1}) \in R_w$. Let *fpaths*(w) denote the set of finite paths through w. An *infinite path* (or just ‘path’) through w is a sequence $(t_u | u \in \mathbb{N})$, such that $\forall u \in \mathbb{N}$, we have $(t_u, t_{u+1}) \in R_w$. Let *paths*(w) denote the set of paths through w. If p is a (finite or infinite) path and $u \in \mathbb{N}$, then $p(u)$ denotes the $u+1$'th element of p (where this is defined). Thus $p(0)$ is the first time-point in p, $p(1)$ is the second, and so on. If p is a (finite or infinite) path and $u \in \mathbb{N}$, then the path obtained from p by removing its first u time-points is denoted by $p^{(u)}$ (where this is defined).

2.2 Plan Structure

An agent's plan library is a set of ‘recipes’, which the agent can use use in order to bring about its intentions. Plans are actually *multi-agent* plans, which closely resemble parallel programs. A plan contains a *plan body*, which represents the ‘program’ part of the plan, and a *plan descriptor*, which characterizes the pre- and post-conditions of the plan. The atomic components of plan bodies are *actions*, i.e., elements of the set D_{Ac}. Actions are composed into plan bodies by the use of *plan constructors*: these are precisely the kind of constructs that one would expect to find in a parallel programming language[1], allowing for sequential and parallel composition, iteration, and choice. Formally, the set D_B, of all plan bodies, is defined by the grammar in Figure 2. We use β (with decorations: $\beta', \beta_1, \ldots$) to stand for members of D_B.

There are two points to note about this definition. First, although we have used a grammar to define plan bodies, and they thus appear to be syntactic objects, they are in fact *semantic* objects, built up from other semantic objects (actions and sets of situations). There are good reasons for emphasizing this point. In our language, $\mathcal{L}$, we will need to be able to quantify over plans (and plan bodies). Hence there must be terms in the language which stand for plans (and plan bodies), and both plans and plan bodies

[1] It is worth noting that we do not have any of the CSP-like primitives for communication and synchronization that one finds in parallel languages like occam [6].

must appear in the domain of the language. Thus plans are not syntactic constructs: they are semantic objects. The second point to note is that a test action takes as its argument a *set of situations*: the test action $c?$ will succeed if the current situation is a member of the set c. A more natural representation for conditions might appear to be formulae of the language, so a test action $\varphi?$ would succeed if the formula φ was satisfied in the current situation. But this would confuse syntax and semantics: test actions are semantic objects, since they are part of plan bodies, which in turn are contained within the domain of the language. In contrast, formulae are syntactic constructs. Putting formulae of the object-language into the domain of the object-language would, in effect, make $\mathcal{L}$ a kind of *self-referential meta-language*, and such languages have a number of difficulties associated with them [16].

A plan body is not, in itself, of much use to an agent, as it specifies neither the circumstances under which the plan may be used, nor what it is good for. A plan body is thus very much like an undocumented fragment of program code. For this reason, we introduce *plan descriptors*, which characterize the *pre-* and *post-conditions* associated with plans. The plan descriptor associated with a plan body represents both when the plan body can be executed, and what execution of the plan body will achieve.

Definition 4. A *plan descriptor*, δ, is a binary relation $\delta \subseteq S \times S$, with the constraint that if $((w, t), (w', t')) \in \delta$, then $w = w'$. Let Δ be the set of all plan descriptors.

Plan descriptors are interpreted as follows. If $\delta \in \Delta$ is intended to characterize the behaviour of a plan body $\beta \in D_B$, then: (i) dom δ represents the set of situations from which execution of β may legally commence — intuitively, dom δ represents the *pre-condition* of β; (ii) ran δ represents the set of situations that may arise as a result of executing β from one of the situations in dom δ — intuitively, ran δ represents the *post-condition* of β; and (iii) if $(s, s') \in \delta$, then s' is a situation that could possibly arise as a result of executing β starting in situation s.

The constraint on plan descriptors, (that if $((w, t), (w', t')) \in \delta$ then $w = w'$), ensures that plan execution always happens *within* worlds, rather than *between* worlds.

This method for characterizing the pre- and post-conditions of a plan might at first sight appear to be somewhat roundabout — a more obvious approach would be to characterize these conditions as formulae of $\mathcal{L}$. However, this approach would run into exactly the same difficulties that we outlined above with respect to test actions, in that putting formulae into the domain of the language is problematic. It is worth noting that the approach we have adopted is essentially identical to the way that programs are represented in dynamic logic, where the behaviour of a program is represented as a binary relation over program states [5].

Definition 5. A *plan* is a pair (β, δ), where $\beta \in D_B$ is a plan body, and $\delta \in \Delta$ is a plan descriptor, intended to represent the behaviour of β. Let $D_\Pi = D_B \times \Delta$ be the set of all plans; we use π (with decorations: $\pi', \pi_1, \ldots$) to stand for members of D_Π. If $\pi \in D_\Pi$, then let $\hat{\beta}(\pi) \in D_B$ denote the body of π, and $\hat{\delta}(\pi) \in \Delta$ denote the descriptor in π. Thus dom $\hat{\delta}(\pi)$ represents the pre-condition of π, and ran $\hat{\delta}(\pi)$ represents the post-condition. If $s \in$ dom $\hat{\delta}(\pi)$, then let $\hat{\delta}(\pi)(s)$ denote the image of s through $\hat{\delta}(\pi)$, i.e., $\hat{\delta}(\pi)(s) = \{s' \mid (s, s') \in \hat{\delta}(\pi)\}$.

If $\beta \in D_B$ is a plan body, then we denote by *agents*(β) the set of all agents that could possibly be required to perform the actions in β:

$$\begin{aligned}
agents(\alpha) &\stackrel{\text{def}}{=} \{Agt(\alpha)\} && \text{(where } \alpha \in D_{Ac}\text{)}\\
agents(\beta \oplus \beta') &\stackrel{\text{def}}{=} agents(\beta) \cup agents(\beta') && \text{(where } \oplus \in \{;, |, ||\}\text{)}\\
agents(\beta *) &\stackrel{\text{def}}{=} agents(\beta)\\
agents(c?) &\stackrel{\text{def}}{=} \emptyset.
\end{aligned}$$

2.3 Plan Execution

We now turn to the semantics of plan execution. We define a 4-place meta-level predicate *exec*, such that *exec*(β, p, u, v) holds just in case the plan body $\beta \in D_B$ is executed on path p between times $u, v \in I\!N$. Formally, the *exec* predicate is defined inductively by six equations: one each for the plan body constructors, and one for the execution of primitive actions. The first equation represents the base case, where a primitive action is executed.

$$exec(\alpha, p, u, v) \text{ iff } v = u + 1 \text{ and } Act(p(u), p(u+1)) = \alpha \quad \text{(where } \alpha \in D_{Ac}\text{)} \tag{1}$$

The second equation captures the semantics of sequential composition: $\beta; \beta'$ will be executed between times u and v iff there is some time point n between u and v such that β is executed between u and n, and β' is executed between n and v.

$$exec(\beta; \beta', p, u, v) \text{ iff } \exists n \in \{u, \ldots, v\} \text{ s.t. } exec(\beta, p, u, n) \text{ and } exec(\beta', p, n, v) \tag{2}$$

The semantics of non-deterministic choice are even simpler: $\beta \mid \beta'$ will be executed between times u and v iff either β or β' is executed between those times.

$$exec(\beta \mid \beta', p, u, v) \text{ iff } exec(\beta, p, u, v) \text{ or } exec(\beta', p, u, v) \tag{3}$$

For the execution of parallel plan bodies $\beta \mid\mid \beta'$, we require that both β and β' are executed over the path, with the same start and end times. The semantics of concurrency clearly represent a simplification, which we make in order to prevent the formalism becoming complicated by tangential side issues.

$$exec(\beta \mid\mid \beta', p, u, v) \text{ iff } exec(\beta, p, u, v) \text{ and } exec(\beta', p, u, v) \tag{4}$$

The semantics of iteration rely upon the fact that executing $\beta *$ is the same as either (i) doing nothing, or (ii) executing β once and then executing $\beta *$. This leads to the following *fixed point* equation, where the right hand side is defined in terms of the left hand side.

$$exec(\beta *, p, u, v) \text{ iff } u = v \text{ or } exec(\beta; (\beta *), p, u, v) \tag{5}$$

(The reader may like to compare this equation with the fixed-point semantics given to loops in imperative programming languages [13].) Finally, we have an equation that defines the semantics of test actions (the free variable w, that appears on the right hand side of this equation, is the world through which p is a path, and in practice this variable will always be bound).

$$exec(c?, p, u, v) \text{ iff } (w, p(u)) \in c \tag{6}$$

Finally, we mention some assumptions relating to plans. First, the notion of *soundness*. Intuitively, a plan is sound if its plan body is completely correct with respect to its plan descriptor. That is, a plan $\pi \in D_{\Pi}$ is sound iff whenever its body $\hat{\beta}(\pi)$ is executed from a situation (w, t) such that $(w, t) \in \text{dom}\,\hat{\delta}(\pi)$, it will terminate in some situation (w, t') such that $((w, t), (w, t')) \in \hat{\delta}(\pi)$. For simplicity, we shall assume that (i) all plans are sound; and (ii) plan bodies are only executed when their pre-condition holds. Intuitively, condition (ii) requires that agents only execute a plan when they *know* the pre-condition of the plan is satisfied, i.e., they are *competent* with respect to plan pre-conditions. Formally, soundness is expressed as follows: $\forall \pi \in D_{\Pi}$, $\forall w \in W$, $\forall p \in paths(w)$, $\forall u, v \in I\!N$, if $exec(\hat{\beta}(\pi), p, u, v)$ then $(w, p(u)) \in \text{dom}\,\hat{\delta}(\pi)$ and $(w, p(v)) \in \hat{\delta}(\pi)((w, p(u)))$.

3 A Logic of BDI Agents with Procedural Knowledge

In this section, we formally define our logic $\mathcal{L}$, which is an extension to the expressive branching time logic CTL* [3]. The logic builds on the work of Rao and Georgeff [9], and our own previous work in agent theory [15].

$\mathcal{L}$ contains the usual connectives and quantifiers of sorted first-order logic: we take as primitive the connectives $\neg$ (not) and $\vee$ (or), and the universal quantifier $\forall$ (for all), and define the remaining classical connectives and existential quantifier in terms of these. As $\mathcal{L}$ is based on CTL*, a distinction is made between *state formulae* and *path formulae*. The idea is that $\mathcal{L}$ is interpreted over a tree-like branching time structure. Formulae that express a property of nodes in this structure are known as *state formulae*, whereas formulae that express a property of paths through the structure are known as *path formulae*. State formulae can be ordinary first-order formulae, but various other additional modal connectives are also provided for making state formulae. Thus $(\mathsf{Bel}\ i\ \varphi)$ is intended to express the fact that the agent denoted by i believes φ (where φ is some state formula). The semantics of belief are given in terms of an accessibility relation over possible worlds, in much the standard modal logic tradition [1], with the properties required of belief accessibility relations ensuring that the logic of belief corresponds to the normal modal system KD45 (weak-S5). The state formulae $(\mathsf{Goal}\ i\ \varphi)$ and $(\mathsf{Int}\ i\ \varphi)$ mean that agent i has a desire or intention of φ, respectively: the logics of desire and intention correspond to the normal modal system KD. (Note that worlds in $\mathcal{L}$ are not instantaneous states (as in [15]), but are themselves branching time structures: the intuition is that belief accessible worlds represent an agent's uncertainty not only about how the world actually is, but also about its past and future; similarly for desires and intentions [9].)

In addition, $\mathcal{L}$ contains various connectives for representing the plans possessed by agents. The state formula $(\mathsf{Has}\ i\ \pi)$ is used to represent the fact that in the current state, agent i is in possession of the plan denoted by π. The state formulae $(\mathsf{Pre}\ \pi)$ and $(\mathsf{Post}\ \pi)$ represent the fact that the pre- and post-conditions of the plan π respectively are satisfied in the current world-state. The formula $(\mathsf{Body}\ \pi\ \beta)$ is used to represent the fact that β is the body of the plan denoted by π. We also have a connective $(\mathsf{Holds}\ c)$, which means that the *condition* denoted by c is satisfied in the current world state.

Turning to path formulae, $(\mathsf{Exec}\ \beta)$ means that the plan body denoted by β is executed on the current path. State formulae may be related to path formulae by using the CTL* *path quantifier* A. This connective means 'on all paths'. It has a dual, existential connective E, meaning 'on some path'. Thus $\mathsf{A}\varphi$ means that the path formula φ is satisfied on all histories originating from the current world state, and $\mathsf{E}\varphi$ means that φ is satisfied on at least one history that originates from the current world state. Path formulae may be built up from state formulae (or other path formulae) by using two *temporal connectives*: the U connectives means 'until', and so a formula $\varphi \mathsf{U} \psi$ means 'φ is satisfied until ψ is satisfied'. The $\bigcirc$ connective means 'next', and so $\bigcirc\varphi$ means that φ will be satisfied in the next state.

3.1 Syntax

$\mathcal{L}$ is a *many sorted* logic, which permits quantification over various types of individuals: agents, actions, plans, plan bodies, sets of agents (groups), sets of situations (conditions), and other individuals in the world. All of these sorts must have a corresponding set of terms in the alphabet of the language.

Definition 6. The alphabet of $\mathcal{L}$ contains the following symbols:

1. A denumerable set *Pred* of *predicate symbols*;
2. A denumerable set *Fun* of *function symbols*, the union of the following mutually disjoint sets:
   - Fun_{Ag} — functions that return agents;
   - Fun_{Ac} — functions that return actions;
   - Fun_{Π} — functions that return plans;
   - Fun_{B} — functions that return plan bodies;
   - Fun_{Gr} — functions that return sets of agents (groups);
   - Fun_{C} — functions that return sets of situations (conditions);
   - Fun_{U} — functions that return other individuals.
3. A denumerable set *Var* of *variable symbols*, the union of the mutually disjoint sets Var_{Ag}, Var_{Ac}, Var_{Π}, Var_{B}, Var_{Gr}, Var_{C}, and Var_{U}.
4. The *operator symbols* true, Bel, Goal, Int, Agts, $=$, $\in$, A, Pre, Post, Body, Has, Holds, Exec, U, and $\bigcirc$.
5. The *classical connectives* $\vee$ (or) and $\neg$ (not), and the *universal quantifier*, $\forall$.
6. The punctuation symbols), (, and $\cdot$.

Associated with each predicate and function symbol is a natural number called its *arity*, given by the function $arity : Pred \cup Fun \to \mathbb{N}$. Predicates of arity 0 are known as *proposition symbols*, and functions of arity 0 are known as *constants*.

$$\begin{array}{rl@{\qquad}rl}
\langle \textit{ag-term} \rangle ::= & \text{any element of } \mathit{Term}_{Ag} & \langle \Pi\textit{-term} \rangle ::= & \text{any element of } \mathit{Term}_{\Pi} \\
\langle \beta\textit{-term} \rangle ::= & \text{any element of } \mathit{Term}_{B} & \langle \textit{gr-term} \rangle ::= & \text{any element of } \mathit{Term}_{Gr} \\
\langle \textit{c-term} \rangle ::= & \text{any element of } \mathit{Term}_{C} & \langle \textit{term} \rangle ::= & \text{any element of } \mathit{Term} \\
\langle \textit{pred-sym} \rangle ::= & \text{any element of } \mathit{Pred} & \langle \textit{var} \rangle ::= & \text{any element of } \mathit{Var}
\end{array}$$

$$\begin{array}{l}
\langle \textit{state-fmla} \rangle \; ::= \\
\quad \begin{array}{ll}
\mathsf{true} & \mid \langle \textit{pred-sym} \rangle(\langle \textit{term} \rangle, \ldots, \langle \textit{term} \rangle) \mid \\
(\mathsf{Bel}\ \langle \textit{ag-term} \rangle\ \langle \textit{state-fmla} \rangle) & \mid (\mathsf{Goal}\ \langle \textit{ag-term} \rangle\ \langle \textit{state-fmla} \rangle) \mid \\
(\mathsf{Int}\ \langle \textit{ag-term} \rangle\ \langle \textit{state-fmla} \rangle) & \mid (\mathsf{Agts}\ \langle \beta\textit{-term} \rangle\ \langle \textit{gr-term} \rangle) \mid \\
(\langle \textit{term} \rangle = \langle \textit{term} \rangle) & \mid (\langle \textit{ag-term} \rangle \in \langle \textit{gr-term} \rangle) \mid \\
(\mathsf{Pre}\ \langle \Pi\textit{-term} \rangle) & \mid (\mathsf{Post}\ \langle \Pi\textit{-term} \rangle) \mid \\
(\mathsf{Body}\ \langle \Pi\textit{-term} \rangle\ \langle \beta\textit{-term} \rangle) & \mid (\mathsf{Has}\ \langle \textit{ag-term} \rangle\ \langle \Pi\textit{-term} \rangle) \mid \\
(\mathsf{Holds}\ \langle \textit{c-term} \rangle) & \mid \mathsf{A}\langle \textit{path-fmla} \rangle \mid \\
\neg \langle \textit{state-fmla} \rangle & \mid \langle \textit{state-fmla} \rangle \vee \langle \textit{state-fmla} \rangle \mid \\
\forall \langle \textit{var} \rangle \cdot \langle \textit{state-fmla} \rangle &
\end{array} \\
\langle \textit{path-fmla} \rangle \; ::= \\
\quad \begin{array}{ll}
(\mathsf{Exec}\ \langle \beta\textit{-term} \rangle) & \mid \langle \textit{state-fmla} \rangle \mid \\
\langle \textit{path-fmla} \rangle \mathsf{U} \langle \textit{path-fmla} \rangle & \mid \bigcirc \langle \textit{path-fmla} \rangle \mid \\
\neg \langle \textit{path-fmla} \rangle & \mid \langle \textit{path-fmla} \rangle \vee \langle \textit{path-fmla} \rangle \mid \\
\forall \langle \textit{var} \rangle \cdot \langle \textit{path-fmla} \rangle &
\end{array} \\
\langle \textit{fmla} \rangle \; ::= \; \langle \textit{state-fmla} \rangle
\end{array}$$

Fig. 3. Syntax

Definition 7. A *sort* is either Ag, Ac, Π, B, Gr, C, or U. If σ is a sort, then the set Term_σ, of *terms of sort* σ, is defined as follows:

1. if $x \in \mathit{Var}_\sigma$, then $x \in \mathit{Term}_\sigma$;
2. if $f \in \mathit{Fun}_\sigma$, $\mathit{arity}(f) = n$, and $\{\tau_1, \ldots, \tau_n\} \subseteq \mathit{Term}$, then $f(\tau_1, \ldots, \tau_n) \in \mathit{Term}_\sigma$

where the set *Term*, of all terms, is defined by

$$\mathit{Term} = \bigcup \{\mathit{Term}_\sigma \mid \sigma \in \{Ag, Ac, \Pi, B, Gr, C, U\}\}.$$

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

The syntax of the language is then defined by the grammar in Figure 3 (it is assumed that predicate and function symbols are applied to the appropriate number of arguments).

3.2 Semantics

In addition to the various semantic sets discussed above, the world may contain other objects (such as, for example, blocks and tables), given by the set D_U. The objects over which we can quantify in $\mathcal{L}$ together constitute a *domain*.

Definition 8. A *domain* is a structure: $D = (D_{Ag}, D_{Ac}, D_{\Pi}, D_B, D_{Gr}, D_C, D_U)$ where:

- $D_{Ag} = \{1, \ldots, n\}$ is a non-empty set of agents;
- $D_{Ac} = \{\alpha, \alpha', \ldots\}$ is a non-empty set of actions;
- $D_{\Pi} = \{\pi, \pi', \ldots\}$ is a non-empty set of plans;
- D_B is a set of plan bodies;
- $D_{Gr} = \wp(D_{Ag}) - \{\emptyset\}$ is the set of non-empty subsets of D_{Ag}, i.e., the set of agent groups over D_{Ag};
- D_C is a non-empty set of situations; and
- D_U is a non-empty set of other individuals

such that (i) all actions in elements of D_B are members of D_{Ac}; (ii) all plan bodies in elements of D_{Π} must be in D_B; and (iii) any plan bodies contained in elements of D_B are also in D_B. If D is a domain, then we denote by $\bar{D}$ the set $\bigcup\{D_\sigma \mid \sigma \in \{Ag, Ac, \Pi, B, Gr, C, U\}\}$. If D is a domain and $u \in I\!N$, then by $\bar{D}^u$ we mean the set of u-tuples over $\bar{D}$.

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

Definition 9. A *predicate interpretation*, Φ, is a function

$$\Phi : Pred \times W \times T \to \wp(\bigcup_{u \in I\!N} \bar{D}^u)$$

such that $\forall Q \in Pred, \forall n \in I\!N, \forall w \in W, \forall t \in T_w,$ if $arity(Q) = n$ then $\Phi(Q, w, t) \subseteq \bar{D}^n$ (i.e., predicate interpretations preserve arity).

Definition 10. An *interpretation for functions*, F, is a second-order function

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

such that (i) $\forall f \in Fun, \forall n \in I\!N,$ if $arity(f) = n$ then $\operatorname{dom} F(f) \subseteq \bar{D}^n$ (i.e., function interpretations preserve arity), and (ii) F preserves sorts.

Similarly, a variable assignment associates variables with elements of the domain.

Definition 11. A *variable assignment*, V, is a function $V : Var \to \bar{D}$, such that if $x \in Var_\sigma$, then $V(x) \in D_\sigma$, (i.e., variable assignments preserve sorts).

We now introduce a derived function $[\![\ldots]\!]_{V,F}$, which gives the *denotation* of an arbitrary term.

Definition 12. If V is a variable assignment and F is a function interpretation, then by $[\![\ldots]\!]_{V,F}$, we mean the function $[\![\ldots]\!]_{V,F} : Term \to \bar{D}$, which interprets arbitrary terms relative to V and F:

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

Since V and F will always be clear from context, reference to them will be suppressed. We can now define models for $\mathcal{L}$.

Definition 13. A *model*, M, for $\mathcal{L}$, is a structure

$$M = (T, R, W, D, Act, Agt, P, BR, DR, IR, F, \Phi)$$

where:

- T is the set of all time points;
- $R \subseteq T \times T$ is a total, backwards-linear branching time relation over T;
- W is a set of worlds, such that $\forall w \in W$, we have:
  1. $T_w \subseteq T$;
  2. R_w is the relation obtained from R by removing from it any arcs that contain components not in T_w;
- $D = (D_{Ag}, D_{Ac}, D_{\Pi}, D_B, D_{Gr}, D_C, D_U)$ is a domain;
- $Act : R \to D_{Ac}$ associates an action with every arc in R;
- $Agt : D_{Ac} \to D_{Ag}$ associates an agent with every action;
- $P : D_{Ag} \times W \times T \to \wp(D_{\Pi})$ gives the *plan library* of every agent in every situation;
- $BR : D_{Ag} \to \wp(W \times T \times W)$ associates with every agent a serial, transitive, euclidean *belief accessibility relation*;
- $DR : D_{Ag} \to \wp(W \times T \times W)$ associates with every agent a serial *desire accessibility relation*;
- $IR : D_{Ag} \to \wp(W \times T \times W)$ associates with every agent a serial *intention accessibility relation*;
- $F : Fun \to (\bigcup_{u \in I\!N} \bar{D}^u \to \bar{D})$ interprets functions;
- $\Phi : Pred \times W \times T \to \wp(\bigcup_{u \in I\!N} \bar{D}^u)$ interprets predicates.

The formal semantics of the language are defined in two parts, for path formulae and state formulae respectively. The semantics of path formulae are given via the path formula satisfaction relation, '$\models$', which holds between structures of the form (M, V, w, p), (where M is a model, V is a variable assignment, w is a world in M, and p is a path through w), and path formulae. The rules defining this relation are given in Figure 4. 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 is intended. The state formula satisfaction relation holds between structures of the form (M, V, w, t), (where M is a model, V is a variable assignment, w is a world in M, and $t \in T_w$ is a time-point in w) and state formulae. The rules defining this relation are also given in Figure 4. We assume the standard interpretation for validity. Thus a path formula φ is valid, (notation: $\models_{\mathcal{P}} \varphi$), iff for all (M, V, w, p), we have $(M, V, w, p) \models \varphi$. Similarly, a state formula φ is valid iff for all (M, V, w, t) we have $(M, V, w, t) \models \varphi$. We write $\models_{\mathcal{S}} \varphi$ to indicate that the state formula φ is valid. Satisfiability for path and state formulae are defined in the obvious way.

Path Formulae Semantics	
$(M, V, w, p) \models \varphi$	iff $(M, V, w, p(0)) \models \varphi$ (where φ is a state formula)
$(M, V, w, p) \models \neg\varphi$	iff $(M, V, w, p) \not\models \varphi$
$(M, V, w, p) \models \varphi \vee \psi$	iff $(M, V, w, p) \models \varphi$ or $(M, V, w, p) \models \psi$
$(M, V, w, p) \models \forall x \cdot \varphi$	iff $(M, V \dagger \{x \mapsto d\}, w, p) \models \varphi$
	for all $d \in \bar{D}$ s.t. x and d are of the same sort
$(M, V, w, p) \models \varphi \mathsf{U} \psi$	iff $\exists u \in I\!N$ such that $(M, V, w, p^{(u)}) \models \psi$ and
	$\forall v \in I\!N,$ if $(0 \leq v < u),$ then $(M, V, w, p^{(v)}) \models \varphi$
$(M, V, w, p) \models \bigcirc\varphi$	iff $(M, V, w, p^{(1)}) \models \varphi$
$(M, V, w, p) \models (\mathsf{Exec}\ \beta)$	iff $\exists u \in I\!N$ such that $exec(\llbracket\beta\rrbracket, p, 0, u)$
State Formulae Semantics	
$(M, V, w, t) \models \mathsf{true}$	
$(M, V, w, t) \models Q(\tau_1, \ldots, \tau_n)$	iff $\langle\llbracket\tau_1\rrbracket, \ldots, \llbracket\tau_n\rrbracket\rangle \in \Phi(Q, w, t)$
$(M, V, w, t) \models \neg\varphi$	iff $(M, V, w, t) \not\models \varphi$
$(M, V, w, t) \models \varphi \vee \psi$	iff $(M, V, w, t) \models \varphi$ or $(M, V, w, t) \models \psi$
$(M, V, w, t) \models \forall x \cdot \varphi$	iff $(M, V \dagger \{x \mapsto d\}, w, t) \models \varphi$
	for all $d \in \bar{D}$ s.t. x and d are of the same sort
$(M, V, w, t) \models (\mathsf{Bel}\ i\ \varphi)$	iff $\forall w' \in W,$ if $(w, t, w') \in BR(\llbracket i\rrbracket),$ then $(M, V, w', t) \models \varphi$
$(M, V, w, t) \models (\mathsf{Goal}\ i\ \varphi)$	iff $\forall w' \in W,$ if $(w, t, w') \in DR(\llbracket i\rrbracket),$ then $(M, V, w', t) \models \varphi$
$(M, V, w, t) \models (\mathsf{Int}\ i\ \varphi)$	iff $\forall w' \in W,$ if $(w, t, w') \in IR(\llbracket i\rrbracket),$ then $(M, V, w', t) \models \varphi$
$(M, V, w, t) \models (\mathsf{Agts}\ \beta\ g)$	iff $agents(\llbracket\beta\rrbracket) = \llbracket g\rrbracket$
$(M, V, w, t) \models (\tau = \tau')$	iff $\llbracket\tau\rrbracket = \llbracket\tau'\rrbracket$
$(M, V, w, t) \models \mathsf{A}\varphi$	iff $\forall p \in paths(w),$ if $p(0) = t,$ then $(M, V, w, p) \models \varphi$
$(M, V, w, t) \models (\mathsf{Pre}\ \pi)$	iff $(w, t) \in \mathrm{dom}\ \hat{\delta}(\llbracket\pi\rrbracket)$
$(M, V, w, t) \models (\mathsf{Post}\ \pi)$	iff $(w, t) \in \mathrm{ran}\ \hat{\delta}(\llbracket\pi\rrbracket)$
$(M, V, w, t) \models (\mathsf{Body}\ \pi\ \beta)$	iff $\hat{\beta}(\llbracket\pi\rrbracket) = \llbracket\beta\rrbracket$
$(M, V, w, t) \models (\mathsf{Has}\ i\ \pi)$	iff $\llbracket\pi\rrbracket \in P(\llbracket i\rrbracket, w, t)$
$(M, V, w, t) \models (\mathsf{Holds}\ c)$	iff $(w, t) \in \llbracket c\rrbracket$

Fig. 4. Semantics of $\mathcal{L}$

3.3 Derived Connectives

In addition to the basic connectives defined above, it is useful to introduce some *derived* constructs. 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 assume that the remaining connectives of classical logic, (i.e., $\wedge$ — 'and', $\Rightarrow$ — 'if… then…', and $\Leftrightarrow$ — 'if, and only if') have been defined as normal, in terms of $\neg$ and $\vee$. Similarly, we assume that the existential quantifier, $\exists$, has been defined as the dual of $\forall$. Next, we introduce the *existential path quantifier*, E, which is defined as the dual of the universal path quantifier A. Thus a formula $\mathsf{E}\varphi$ is interpreted as 'on some path, φ', or 'optionally, φ':

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

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: $\varphi\mathsf{W}\psi$ 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.

Talking about groups: The language $\mathcal{L}$ provides us with the ability to use simple (typed) set theory to relate the properties of agents and groups of agents. The operators $\subseteq$ and $\subset$ relate groups together, and have the obvious set-theoretic interpretation; $(\mathsf{Singleton}\ g\ i)$ means g is a singleton group with i as the only member; $(\mathsf{Singleton}\ g)$ simply means g is a singleton.

$$\begin{array}{ll} (g \subseteq g') \overset{\text{def}}{=} \forall i \cdot (i \in g) \Rightarrow (i \in g') & (\mathsf{Singleton}\ g\ i) \overset{\text{def}}{=} \forall j \cdot (j \in g) \Rightarrow (j = i) \\ (g \subset g') \overset{\text{def}}{=} (g \subseteq g') \wedge \neg(g = g') & (\mathsf{Singleton}\ g) \overset{\text{def}}{=} \exists i \cdot (\mathsf{Singleton}\ g\ i) \end{array}$$

$(\mathsf{Agt}\ \beta\ i)$ means that i is the only agent required to perform plan body β.

$$(\mathsf{Agt}\ \beta\ i) \overset{\text{def}}{=} \forall g \cdot (\mathsf{Agts}\ \beta\ g) \Rightarrow (\mathsf{Singleton}\ g\ i)$$

Talking about plans: Next, we introduce some operators that will allow us to conveniently represent the structure and properties of plans. First, we introduce two constructs, $(\mathsf{Pre}\ \pi\ \varphi)$ and $(\mathsf{Post}\ \pi\ \varphi)$, that allow us to represent the pre- and post-conditions of plans as formulae of $\mathcal{L}$. Thus $(\mathsf{Pre}\ \pi\ \varphi)$ means that φ corresponds to the pre-condition of π — that φ is satisfied in just those situations where the pre-condition of π is satisfied:

$$(\mathsf{Pre}\ \pi\ \varphi) \overset{\text{def}}{=} \mathsf{A}\Box((\mathsf{Pre}\ \pi) \Leftrightarrow \varphi).$$

Similarly, $(\mathsf{Post}\ \pi\ \varphi)$ means that φ is satisfied in just those situations in which the post-condition of π is satisfied:

$$(\mathsf{Post}\ \pi\ \varphi) \overset{\text{def}}{=} \mathsf{A}\Box((\mathsf{Post}\ \pi) \Leftrightarrow \varphi).$$

These definitions say that if $(\mathsf{Pre}\ \pi\ \varphi)$, then $(M, V, w, t) \models \varphi$ iff $(w, t) \in \operatorname{dom} \hat{\delta}(\llbracket\pi\rrbracket)$, and if $(\mathsf{Post}\ \pi\ \varphi)$, then $(M, V, w, t) \models \varphi$ iff $(w, t) \in \operatorname{ran} \hat{\delta}(\llbracket\pi\rrbracket)$. We write $(\mathsf{Plan}\ \pi\ \varphi\ \psi\ \beta)$ to express the fact that plan π has pre-condition φ, post-condition ψ, and body β:

$$(\mathsf{Plan}\ \pi\ \varphi\ \psi\ \beta) \overset{\text{def}}{=} (\mathsf{Pre}\ \pi\ \varphi) \wedge (\mathsf{Post}\ \pi\ \psi) \wedge (\mathsf{Body}\ \pi\ \beta).$$

It is often useful to be able to talk about the *structure* of plans: how their bodies are put together, in terms of the constructors ;, $|$, $||$, and so on. In order to do this, we introduce some logical functions (i.e., functions denoted by elements of the set *Fun*). We introduce one function for each of the plan constructors:

$$\textit{seq} \text{ for } ; \quad \textit{par} \text{ for } \| \quad \textit{test} \text{ for } ? \quad \textit{or} \text{ for } | \quad \textit{iter} \text{ for } *.$$

We require that these functions satisfy certain properties. For example, for all $\beta, \beta' \in \mathit{Term}_B$, we require that $\mathit{seq}(\beta, \beta')$ returns the plan body $[\![\beta]\!];[\![\beta']\!]$, i.e., that $\mathit{seq}(\beta, \beta')$ returns the plan body obtained by conjoining the plan bodies denoted by β and β' with the sequential composition constructor. Similarly, we require than $\mathit{par}(\beta, \beta')$ returns $[\![\beta]\!] \;\|\; [\![\beta']\!]$, that $\mathit{or}(\beta, \beta')$ returns $[\![\beta]\!] \mid [\![\beta']\!]$, that $\mathit{iter}(\beta)$ returns $[\![\beta]\!]*$, and finally, that $\mathit{test}(c)$ returns $[\![c]\!]?$, for all $c \in \mathit{Term}_C$. These functions allow us to construct plan bodies within our language. However, complex plan bodies written out in full using these functions become hard to read. To make such expressions more readable, we introduce a *quoting convention*. The idea is best illustrated by example. We write

$$\begin{array}{ll}
\ulcorner \beta; \beta' \urcorner & \text{to abbreviate } \mathit{seq}(\beta, \beta') \\
\ulcorner \beta; (\beta' \;\|\; \beta'') \urcorner & \text{to abbreviate } \mathit{seq}(\beta, \mathit{par}(\beta', \beta'')) \\
\ulcorner \beta; (\beta' \;\|\; \beta'')* \urcorner & \text{to abbreviate } \mathit{seq}(\beta, \mathit{iter}(\mathit{par}(\beta', \beta'')))
\end{array}$$

and so on. In the interests of consistency, we shall generally use quotes even where they are not strictly required.

Next, we introduce a construct that makes test actions more readable. Let $c \in \mathit{Term}_C$ be a term denoting a situation set, (i.e., a condition), and let φ be a state formula. Then $(c \equiv \varphi)$ represents the fact that φ is satisfied in just those situations denoted by c:

$$(c \equiv \varphi) \stackrel{\text{def}}{=} \mathsf{A}\Box((\mathsf{Holds}\ c) \Leftrightarrow \varphi).$$

Hereafter, instead of writing $c?$ we write $\varphi?$, where it is understood that $(c \equiv \varphi)$. Thus, when we write $(\mathsf{Exec}\ \ulcorner(\mathsf{Bel}\ i\ p)?\urcorner)$, it should be understood that this abbreviates $\forall c \cdot (c \equiv (\mathsf{Bel}\ i\ p)) \Rightarrow (\mathsf{Exec}\ \ulcorner c? \urcorner)$. Any formula abbreviated in this way may be systematically rewritten into the fully expanded form. Note that one property of this style of abbreviation is $\models_{\mathcal{P}} (\mathsf{Exec}\ \ulcorner \varphi? \urcorner) \Leftrightarrow \varphi$.

The readability of plan body expressions may be further improved by the introduction of derived constructs corresponding to the high-level statement-types one would expect to find in a standard imperative language such as PASCAL. First, the *if... then...* construct:

$$\ulcorner \texttt{if}\ \varphi\ \texttt{then}\ \beta\ \texttt{else}\ \beta' \urcorner \stackrel{\text{def}}{=} \ulcorner (\varphi?; \beta) \mid (\neg\varphi; \beta') \urcorner.$$

While and *repeat* loops are similarly easy to define:

$$\ulcorner \texttt{while}\ \varphi\ \texttt{do}\ \beta \urcorner \stackrel{\text{def}}{=} \ulcorner (\varphi?; \beta)*; \neg\varphi? \urcorner$$
$$\ulcorner \texttt{repeat}\ \beta\ \texttt{until}\ \varphi \urcorner \stackrel{\text{def}}{=} \ulcorner \beta; \texttt{while}\ \neg\varphi\ \texttt{do}\ \beta \urcorner.$$

Finally, we define an `await` construct:

$$\ulcorner \texttt{await}\ \varphi \urcorner \stackrel{\text{def}}{=} \ulcorner \texttt{repeat}\ \mathsf{true}?\ \texttt{until}\ \varphi \urcorner.$$

Thus `await` φ will be executed on a path p if there is some point on p at which φ is true. There is thus a close relationship between `await` and the temporal ‘sometimes’ connective: $\models_{\mathcal{P}} (\mathsf{Exec}\ \ulcorner \texttt{await}\ \varphi \urcorner) \Leftrightarrow \Diamond\varphi$.

3.4 Some 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 $\mathcal{L}$[2]. For this reason, instead of attempting a complete axiomatization, we simply identify some valid formulae of $\mathcal{L}$, focusing in particular on plan execution. First, notice that the semantics of $\mathcal{L}$ generalize those of sorted first-order logic, and hence, in turn, propositional logic. Thus $\mathcal{L}$ admits propositional and sorted first-order reasoning, as one might expect. In addition, the semantics of the BDI component of $\mathcal{L}$ ensure that axioms corresponding to the normal modal system KD45 (weak-S5) are valid for the Bel modalities, and axioms corresponding to the normal modal system KD are valid for Goal and Int modalities. Rao and Georgeff prove that these axioms together constitute a sound and complete axiomatization of this 'basic BDI system' [11]. With respect to the CTL* component of the logic, it is not difficult to see that the axioms one would expect of CTL* are valid in $\mathcal{L}$ [14].

Turning to plans, it is not difficult to see that the logic of plan execution is similar to that of many program logics. For example, one can show that if the plan body $\ulcorner \beta \mid \beta' \urcorner$ is executed, then the pre-condition of β or the pre-condition of β' must hold prior to execution:

$$\models_{\mathcal{P}} (\mathsf{Plan}\, \pi\, \varphi\, \psi\, \beta) \wedge (\mathsf{Plan}\, \pi'\, \varphi'\, \psi'\, \beta') \wedge (\mathsf{Exec}\, \ulcorner \beta \mid \beta' \urcorner) \Rightarrow (\mathsf{Exec}\, \ulcorner (\varphi \vee \varphi')?; (\beta \mid \beta') \urcorner)$$

As a corollary, one can show that agents believe that plans behave in this way:

$$\models_{\mathcal{S}} (\mathsf{Bel}\ i\ (\mathsf{Plan}\, \pi\, \varphi\, \psi\, \beta)) \wedge (\mathsf{Bel}\ i\ (\mathsf{Plan}\, \pi'\, \varphi'\, \psi'\, \beta')) \wedge (\mathsf{Bel}\ i\ \mathsf{A}(\mathsf{Exec}\, \ulcorner \beta \mid \beta' \urcorner)) \Rightarrow (\mathsf{Bel}\ i\ \varphi \vee \varphi').$$

In a similar way, one can prove various properties of derived constructs such as `while` loops and `if` statements.

4 Concluding Remarks

It is now widely accepted that the technology of multi-agent systems will play a key role in the development of future distributed systems. As the use of multi-agent technology becomes more commonplace, so the need for a firm theoretical foundations for it will grow. In this paper, we hope to have contributed to such a foundation, by presenting a new logic that can be used to give an abstract semantics to a significant class of intelligent agent architectures. In these so-called BDI architectures, the internal state of an agent is characterised by symbolic data structures loosely corresponding to beliefs, desires, and intentions. In addition, such agents have available to them a library of plans, representing their 'know-how': procedural knowledge about how to achieve goals.

Acknowledgments: This work was carried out while I was a visiting researcher at Daimler-Benz research institute in Berlin. I would like to thank Kurt Sundermeyer for arranging the visit, and in particular, Stefan Bussmann, Michael Georgeff, Afsaneh Haddadi, and Anand Rao, who all read and carefully commented on earlier drafts of this paper.

[2] The system presented in [14] reportedly contains an error in the proof of completeness.

References

1. B. Chellas. *Modal Logic: An Introduction.* Cambridge University Press: Cambridge, England, 1980.
2. P. R. Cohen and H. J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42:213–261, 1990.
3. 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.
4. M. P. Georgeff and A. L. Lansky. Reactive reasoning and planning. In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*, pages 677–682, Seattle, WA, 1987.
5. D. Harel. Dynamic logic. In D. Gabbay and F. Guenther, editors, *Handbook of Philosophical Logic Volume II — Extensions of Classical Logic*, pages 497–604. D. Reidel Publishing Company: Dordrecht, The Netherlands, 1984. (Synthese library Volume 164).
6. C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21:666–677, 1978.
7. D. Kinny, M. Ljungberg, A. S. Rao, E. Sonenberg, G. Tidhar, and E. Werner. Planned team activity. In C. Castelfranchi and E. Werner, editors, *Artificial Social Systems — Selected Papers from the Fourth European Workshop on Modelling Autonomous Agents in a Multi-Agent World, MAAMAW-92 (LNAI Volume 830)*, pages 226–256. Springer-Verlag: Heidelberg, Germany, 1992.
8. A. S. Rao and M. Georgeff. BDI Agents: from theory to practice. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 312–319, San Francisco, CA, June 1995.
9. A. S. Rao and M. P. Georgeff. Modeling rational agents within a BDI-architecture. In R. Fikes and E. Sandewall, editors, *Proceedings of Knowledge Representation and Reasoning (KR&R-91)*, pages 473–484. Morgan Kaufmann Publishers: San Mateo, CA, April 1991.
10. 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.
11. A. S. Rao and M. P. Georgeff. Formal models and decision procedures for multi-agent systems. Technical Note 61, Australian AI Institute, Level 6, 171 La Trobe Street, Melbourne, Australia, June 1995.
12. A. S. Rao, M. P. Georgeff, and E. A. Sonenberg. Social plans: A preliminary report. In E. Werner and Y. Demazeau, editors, *Decentralized AI 3 — Proceedings of the Third European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW-91)*, pages 57–76. Elsevier Science Publishers B.V.: Amsterdam, The Netherlands, 1992.
13. D. A. Schmidt. *Denotational Semantics.* Allyn and Bacon: Newton, MA, 1986.
14. 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.
15. M. Wooldridge. Coherent social action. In *Proceedings of the Eleventh European Conference on Artificial Intelligence (ECAI-94)*, pages 279–283, Amsterdam, The Netherlands, 1994.
16. M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.

This article was processed using the LaTeX macro package with LLNCS style