A Programming Logic for Part of the Agent Language 3APL

Koen V. Hindriks¹, Frank S. de Boer¹, Wiebe van der Hoek¹,²,³, and John-Jules Ch. Meyer¹

¹ Institute of Information and Computing Sciences, Utrecht University, The Netherlands
{koenh,frankb,wiebe,jj}@cs.uu.nl
² Department of Philosophy, Utrecht University, The Netherlands
Wiebe.vanderHoek@phil.uu.nl
³ Department of Computer Science, The University of Liverpool, United Kingdom

Abstract. 3APL is an agent programming language based on the concept of an intelligent agent. An intelligent agent is a computational entity with a mental state consisting of its beliefs and goals. The operational semantics of the language 3APL is specified by a formal semantics in terms of a so-called transition system. An operational semantics allows operational reasoning about agents, but does not allow for a compositional style of reasoning based on the structure of the agent itself. For this purpose, in this paper we construct a denotational semantics which corresponds to the operational semantics and provides the basis for a semantics of a programming logic for (part of) 3APL. The programming logic is a variant of a modal logic with operators for reasoning about the actions and the beliefs of an agent. Our results clarify the relation between more practical approaches to agents, represented by agent programming languages, and more theoretical work on agents, represented by so-called agent logics.

1 Introduction

One of the main challenges in the field of intelligent agent programming is to provide a satisfactory account of the relation between formal, logical specifications of intelligent agents and the agents in agent programming languages. Our work on the agent programming language 3APL (pronounced "triple-a-p-l"; cf. [4,5]) has been an attempt to bridge the gap between the two by introducing a well defined programming language to build and implement agents.

In [4] we discuss the agent programming language 3APL (pronounced "triple-a-p-l") in detail. The programming language 3APL is a combination of imperative programming and logic programming. From imperative programming the
language inherits the full range of regular programming constructs. These constructs are used for determining the course of action of an agent. From logic programming, the language inherits the proof as computation model as a basic means of computation for querying the beliefs of an agent. 3APL is a language closely related to a number of other agent programming languages like AGENT-0 ([12]), AgentSpeak(L) ([11]), and GOLOG ([9]) as we have shown elsewhere [3,2,7]. There are at least two perspectives possible on what the programming language 3APL is. First, it is an agent programming language which supports the construction of programs by viewing them as intelligent agents. Secondly, it can be viewed upon as a language for knowledge-based or database programming.

For the specification and verification of agent systems we need a programming logic that enables us to reason about agents written in an agent language like 3APL. So far, no satisfying account of the relationship between agent programming languages and agent logics has been given. In this paper, we introduce a programming logic for (part of) the agent programming language 3APL to bridge the gap between agent programming frameworks and agent logics. We present a general framework for studying a family of agent programming languages, or knowledge update languages, and their related proof logics. The framework is general in the sense that we study abstract actions and not just any particular set of concrete actions. Concrete actions, like an insert action ins(ϕ) or delete action del(ϕ) for a proposition ϕ, can then be plugged in into the general framework, and a programming logic for that particular language is then obtained in a straightforward way.

In previous papers ([5,4]) presenting the language 3APL, an operational semantics was provided by means of a Plotkin-style transition system ([10]). The main contributions of this paper are that we provide a denotational semantics and a programming logic for the language. The programming logic is a variant of a modal logic for reasoning about ‘regular’ 3APL agents (which means that the agents are build by means of the regular programming constructs from imperative programming like sequential composition, etc.). The logic provides operators for reasoning about the actions as well as the beliefs of an agent. Moreover, we prove an equivalence result which shows that the operational semantics and denotational semantics are equivalent for knowledge bases, given certain relations between the operational and denotational semantics of basic actions. The semantics of the programming logic is based on this denotational semantics for the programming language. The equivalence result both shows that the language 3APL is mathematically well founded, and relates the programming logic to 3APL agents. The formal equivalence result thereby makes sure that properties proven in the logic are properties of 3APL agents. The particular properties we take into account in this paper are partial correctness properties.

2 The Agent Programming Language 3APL

Most agent researchers agree that agents have a complex mental state which is made up of an informational component like beliefs and a motivational compo-
nent like intentions or goals. Agents are supposed to be pro-active as well as reactive and may have reflective capabilities to modify their plans and goals. Accordingly, the agent programming language 3APL has operators for manipulating the beliefs of an agent, and so-called practical reasoning rules which enable an agent to plan for new or revise old intentions. Here, intentions are taken to be plan or program like structures. In this section, we review part of the agent programming language and introduce its formal operational semantics. The focus is on the beliefs and goals of the agent and we do not discuss the practical reasoning rules of 3APL (cf. [4]). In the next section we then show how to design a logical semantics which can serve as the basis for a semantics for a programming logic for 3APL.

To represent their beliefs, agents need some knowledge representation language. In principle, this can be any language, but here we will use a first order language as the knowledge representation language used by 3APL agents. Because it is important for presenting the programming logic later on, we explicitly define this language. As is usual, terms are defined from a non-empty set of constants Cons, function symbols Func, and countably infinite variables Var. The set of all terms is denoted by Term.

**Definition 1.** (knowledge representation language)
The knowledge representation language $L_0$ is then inductively defined by:

1. $p(t_1, \ldots, t_n) \in L_0$ for predicates $p \in \text{Pred}$ and terms $t_1, \ldots, t_n \in \text{Term}$,
2. if $\phi_1, \phi_2 \in L_0$, then $\neg \phi_1, \phi_1 \land \phi_2 \in L_0$, and
3. if $x \in \text{Var}$, $\phi \in L_0$, then $\forall x(\phi) \in L_0$.

By convention, we use $\phi$ (possibly subscripted) for formulas from $L_0$. $L_0$ is used to represent the agent’s beliefs.

**Definition 2.** A belief base $\sigma$ of an agent is any set of sentences from $L_0$.

As mentioned in the introduction, the proof as computation model of logic programming is used to enable an agent to query its beliefs. For this reason, substitutions play an important role in the semantics of the language. However, we only use simple grounding substitutions and therefore our definition of substitution differs somewhat from more standard definitions.

**Definition 3.** (substitution)

1. A substitution $\theta$ is a set of bindings $x = c$, where $x \in \text{Var}$ and $c \in \text{Cons}$,
2. The domain of a substitution is denoted by $\text{dom}(\theta)$, and defined as: $\text{dom}(\theta) = \{x \mid \exists c(x = c \in \theta)\}$,
3. Given an expression $e$ and a substitution $\theta$, the expression $e\theta$ is the expression obtained by simultaneously substituting $c_i$ for $x_i$ for all $x_i = c_i \in \theta$,
4. The composition of two substitutions $\theta, \gamma$ is defined by:
   $$\theta\gamma = \theta \cup \{x = t \mid x = t \in \gamma, x \not\in \text{dom}(\theta)\}.$$
We introduce some notation. \( \emptyset \) denotes the empty substitution. The set of all substitutions is denoted by \( \text{Subs} \).

The second basic notion associated with agents is that of a \textit{goal}. A goal in 3APL is similar to an imperative program. Assignment statements that are the basic actions from imperative programming, however, are replaced in 3APL with actions for manipulating high-level information. Goals are built from these basic actions \( a(t) \), and tests \( \phi \), where \( \phi \) is a formula from \( \mathcal{L}_0 \), possibly containing free variables. Basic actions define the agent capabilities and the execution of such a capability results in changes to the mental state of the agent. Basic actions thus are mental state transformers. More concretely, a basic action could range from sensory actions performed by a robot to database updates. Tests do not change the beliefs of the agent, but can be used by the agent to introspect its beliefs and to compute values or bindings for free variables in the test as in logic programming. More complex goals can be constructed by composing two goals with one of two programming constructs, sequential composition \( ; \) and nondeterministic choice \( + \). We refer the reader to [4,8] for a discussion of the other constructs like practical reasoning rules, procedure abstractions and concurrency and communication in 3APL.

\begin{definition}(goals)\end{definition}
Let \( \text{Bact} \) be the set of all basic actions. Then the set of goals \( \text{Goal} \) is inductively defined as:

1. \( \text{Bact} \subseteq \text{Goal} \),
2. \( \phi \in \text{Goal} \), for \( \phi \in \mathcal{L}_0 \),
3. if \( \pi_1, \pi_2 \), then \( \pi_1; \pi_2, \pi_1 + \pi_2 \in \text{Goal} \),

\begin{flushleft} \textit{Operational Semantics} \end{flushleft} The operational semantics of the programming language is provided by a so-called transition semantics which defines the possible computation steps that an agent can perform given its current mental state. The symbol \( \rightarrow \) is used to denote a computation step. After performing a computation step both the beliefs and the goals of the agent may have changed. The computation step relation \( \rightarrow \) is a relation on mental states and is inductively defined by means of a transition system. A transition system consists of a set of derivation rules of the form \( \text{premise(s)} \rightarrow \text{conclusion} \). A transition rule allows to derive new possible computation steps from given computation steps listed in the premises. A special symbol \( E \) is used below to denote successful termination, and \( E; \pi \) is identified with \( \pi \).

The semantics of the basic capabilities of the agent are not fixed except for their type. Basic actions provide an agent with update capabilities upon its beliefs. Because the specifics of these basic actions are not fixed by 3APL but are considered plug in features, we assume a so-called transition function \( \mathcal{T} \) of type \( \text{Bact} \times \mathcal{L}_0 \rightarrow \mathcal{L}_0 \) that specifies what type of update is associated with a basic action \( a(t) \in \text{Bact} \). The execution of a basic action then amounts to changing the mental state in accordance with the transition function \( \mathcal{T} \). The empty substitution \( \emptyset \) is associated with a computation step due to a basic action.
(subscripted to $\rightarrow$), since a basic action does not compute any bindings for variables. The use of substitutions associated with $\rightarrow$ is explained below.

**Definition 5. (transition rules for basic actions)**

$$T(a(t), \sigma) = \sigma'$$

$$\langle a(t), \sigma \rangle \rightarrow \langle E, \sigma' \rangle$$

The semantics of tests is derived from the usual consequence relation for first order logic $\models$. Tests provide the agent with the ability to introspect its beliefs and to retrieve information from its current beliefs. The information retrieved are computed values or bindings for the free variables in the condition of a test entailed by the current belief base of the agent. The mechanism is that of logic programming. Somewhat more formal, a test $\phi$ is a check whether or not there is an instantiation of $\phi$ that is entailed by the belief base of the agent. The instantiation of is computed by assigning values to the free variables in the test and results in a substitution. The bindings retrieved from the belief base thus are recorded in a substitution $\theta$ and this substitution is associated with the computation step relation $\rightarrow$. This is because the computed bindings must be passed on to the remaining goal after the test has been executed (compare the rule for sequential composition). For example, the test $\text{meet}(10\text{am}, \text{Place})$? where $\text{Place}$ is a variable can compute the binding $\text{Place} = \text{utrecht}$ if the belief base implies $\text{meet}(10\text{am}, \text{utrecht})$.

**Definition 6. (transition rules for tests)**

$$\sigma \models \phi \theta, \text{dom}(\theta) = \text{Free}(\phi)$$

$$\langle \phi?, \sigma \rangle \rightarrow_\theta \langle E, \sigma \rangle$$

A sequential composition $\pi_1; \pi_2$ is executed by first executing $\pi_1$ and passing any computed bindings to $\pi_2$, which explains why the substitution $\theta$ is applied to $\pi_2$ in the rule for sequential composition below. Nondeterministic choice goals $\pi_1 + \pi_2$ are executed by executing one of the subgoals $\pi_1$ or $\pi_2$ and dropping the other. Below, we only give the rule for selecting the left subgoal.

**Definition 7. (sequential composition and nondeterministic choice)**

$$\langle \pi_1, \sigma \rangle \rightarrow_\theta \langle \pi_1', \sigma' \rangle$$

$$\langle \pi_1; \pi_2, \sigma \rangle \rightarrow_\theta \langle \pi_1'; \pi_2\theta, \sigma' \rangle$$

$$\langle \pi_1; \pi_2, \sigma \rangle \rightarrow_\theta \langle \pi_1', \sigma' \rangle$$

$$\langle \pi_1; \pi_2, \sigma \rangle \rightarrow_\theta \langle \pi_1', \sigma' \rangle$$

3 A Programming Logic for 3APL

The programming logic $\mathcal{L}$ for 3APL is an extension of the knowledge representation language used by the agents to represent their beliefs. The language $\mathcal{L}$ extends $\mathcal{L}_0$ with two modal operators. The first operator corresponds to the actions an agent can perform whereas the second is a modal operator for reasoning about the beliefs of an agent.
Definition 8. \textit{(language \( \mathcal{L} \))}

The language \( \mathcal{L} \) is inductively defined by:

1. \( \mathcal{L}_0 \subseteq \mathcal{L} \),
2. if \( \varphi_1, \varphi_2 \in \mathcal{L} \), then \( \neg \varphi_1, \varphi_1 \land \varphi_2 \in \mathcal{L} \),
3. if \( x \in \text{Var} \), \( \varphi \in \mathcal{L} \), then \( \forall x(\varphi) \in \mathcal{L} \),
4. if \( \pi \in \text{Goal} \), \( \varphi \in \mathcal{L} \), then \( [\pi]_\varphi \in \mathcal{L} \),
5. if \( \varphi \in \mathcal{L} \), then \( \text{B}\varphi \in \mathcal{L} \).

The notion of a sentence of \( \mathcal{L} \) is somewhat more involved than the usual definition for first order language because variables in the action modality \([\pi]_\varphi \) bind variables in \( \varphi \). This binding corresponds to the implicit binding mechanism of the programming language based on the parameter mechanism for 3APL. Free variables in a goal \( \pi \) need to be instantiated during a computation and the computed bindings need to be passed on. These same values also need to be used to evaluate the conditions \( \varphi \) evaluated in \([\pi]_\varphi \). A formula \( \forall(\varphi) \) denotes the \textit{universal closure} of formula \( \varphi \), in which all free variables of \( \varphi \) are universally quantified.

Definition 9. \textit{(sentence)}

A sentence \textit{from} \( \mathcal{L} \) is defined by:

- if \( \text{Free}(\varphi) = \{x\} \), then \( \forall x(\varphi) \) is a sentence,
- if \( \text{Free}(\varphi) \subseteq \text{Free}(\pi) \), then \([\pi]_\varphi \) is a sentence,
- if \( \varphi_1, \varphi_2 \) are sentences, then \( \neg \varphi_1, \varphi_1 \land \varphi_2, \forall x(\varphi_1), [\pi]_\varphi_1 \) and \( \text{B}\varphi_1 \) are sentences.

The semantics for the programming logic \( \mathcal{L} \) is provided by a so-called \textit{denotational semantics}. Below, it is proven that the \textit{operational semantics} introduced for the programming language 3APL in the previous section is equivalent with the denotational semantics. The theorem shows how to relate an agent logic to an agent programming language, which remains unclear in, for example, [12] and [13]. Because the logical semantics is formally linked to the semantics of the programming language, properties proven in the agent logic really are properties of 3APL agent programs. The denotational semantics moreover has the important feature that it is \textit{compositional}.

The two basic observations that are used to construct a denotational semantics are that, first, it is possible to represent the belief base of an agent by a \textit{set of first order structures} and, second, that the usual \textit{set of objects} in such structures can be identified with the set of constants of the knowledge representation language \( \mathcal{L}_0 \) since agents can only have knowledge of things which can be referred to by these constants. In effect, a \textit{domain closure assumption} is made that every possible thing an agent can know of has a name.

Instead of the usual first order structures, we introduce the notion of a \textit{world state} which plays a similar role in our semantics. A world state is used to specify the semantics of the belief operator \( \text{B} \) and could be used to extend our current framework to model sensory actions in the agent’s environment. A world state
$w$ is a function which maps closed terms of the form $f(c_1, \ldots, c_n)$ to constants (the fixed domain in the semantics for $\mathcal{L}$) and maps closed atoms of the form $p(c_1, \ldots, c_n)$ to $1$ (true) or $0$ (false) where $c_i$ are constants. This is all the information we need to interpret arbitrary closed terms and sentences from the knowledge representation language $\mathcal{L}_0$. Closed terms are interpreted by a world state as follows: $w(f(t_1, \ldots, t_n)) = w(f(c_1, \ldots, c_n))$ where $c_i = w(t_i)$ for complex terms, and $w(c) = c$ for simple terms, that is, constants. The constants in the language thus are mapped in the semantics onto themselves, and their interpretation is the same in all world states.

A set of world states is called an epistemic state. This terminology is justified by our first observation that a belief base of an agent can be identified with a set of world states, namely those world states which make every sentence in the belief base true. An epistemic state thus consists of the world states that are compatible with the beliefs of an agent, which is similar to the semantics for modal logics of knowledge. Epistemic states are denoted by $e$ and the set of all epistemic states by $\mathcal{E}$.

Now we are able to define the meaning of goals in terms of these notions. Recall that basic actions are interpreted as update functions on the belief base of an agent. The semantics of basic actions therefore is represented as a function mapping one epistemic state to another since, semantically, a belief base is represented by an epistemic state. In the denotational semantics, the function $B$ of type $\textsc{Bact} \times \mathcal{E} \to \mathcal{E}$ defines the semantics of basic actions and is the counterpart in the logical semantics of the function $\mathcal{T}$ of the operational semantics.

To make sure, however, that both types of semantics are equivalent, we need to impose a constraint on the transition function $\mathcal{T}$ of the operational semantics. The function $\mathcal{T}$ is directly defined in terms of the syntactical representation of the agent’s belief base. Syntax, however, distinguishes too much and allows the possibility that two logically equivalent belief bases are updated in quite different ways by the same action. This possibility is absent in a more semantical setup as is the case with the function $B$. Because a belief base in the denotational semantics is identified with a set of world states it is not possible to distinguish between logically equivalent belief bases as $p \wedge q$ and $q \wedge p$, for example. For this reason, we impose the following coherence constraint on the transition function $\mathcal{T}$.

**Constraint** For two logically equivalent belief bases $\sigma$ and $\sigma'$, we require that $\mathcal{T}(a(t), \sigma) = \mathcal{T}(a(t), \sigma')$. That is, the effect of a basic action is not allowed to rely on the syntactical representation of the belief base of an agent.

Goals essentially perform two functions. They are update operators on the belief base of an agent and they can be used to introspect the belief base. In the former case, a new updated belief base results whereas in the latter case a set of bindings for variables may be returned. Since, semantically, we can represent a belief base by an epistemic state it is natural to interpret goals as a mapping from such epistemic states and a substitution representing the computed bindings for variables so far to (sets of) pairs consisting of two things: (i) the updated
epistemic states and (ii) the newly computed bindings combined with the old ones, that is, a pair consisting of an epistemic state and a substitution. Note that goals do not change the ‘current’ world state, and that the denotational semantics for basic actions is well defined because of the constraint on \( T \) introduced above.

**Definition 10. (denotational semantics for goals)**
The interpretation function \([.]\) for goals of type \( \text{Goal} \times \mathcal{E} \times \text{Subs} \rightarrow \wp(\mathcal{E} \times \text{Subs}) \) is inductively defined by:

1. \([a(t)](e, \theta) = \{(B(a(t)\theta, e), \theta)\}\),
2. \([\phi\psi](e, \theta) = \{ (e, \theta') \mid \forall w \in e(w) \models \phi(\theta) \}, \text{dom}(\theta') = \text{Free}(\phi)\},
3. \([\pi_1 \cdot \pi_2](e, \theta) = \bigcup_{(e', \theta') \in [\pi_1](e, \theta)} [\pi_2](e', \theta')\), and
4. \([\pi_1 + \pi_2](e, \theta) = [\pi_1](e, \theta) \cup [\pi_2](e, \theta)\).

A formal semantics for the logic \( \mathcal{L} \) can be formulated using this denotational semantics for goals. Although the definition below may seem circular, since \( \models \) is also used in the definition of \([.]\), this circularity can be circumvented since tests are conditions formulated in the language \( \mathcal{L}_0 \) and not in the full language \( \mathcal{L} \) with operators \([\pi] \) for reasoning about goals. The semantics is only defined for sentences from \( \mathcal{L} \) which allows us to do without the notion of valuation functions from traditional first order logic. Sentences from \( \mathcal{L} \) are evaluated relative to a world state and an epistemic state.

A formula from the knowledge representation language \( \mathcal{L}_0 \) is interpreted in the current world state, as usual. The semantics of quantifiers is defined in terms of the fixed domain defined as the set of constants from \( \mathcal{L}_0 \). The semantics of the goal modality \([\pi] \) is provided by the denotational semantics for goals \([.]\). The belief modality is interpreted with respect to the current epistemic state; \( B\varphi \) is true given an epistemic state \( e \) iff all of the world states \( w \in e \) entail \( \varphi \) (that is, all world states in \( e \) are compatible with \( \varphi \)).

\[
\begin{align*}
w, e \models p(t_1, \ldots, t_n) & \iff w(p(c_1, \ldots, c_n)) = 1, \text{ where } c_i = w(t_i), \\
w, e \models \varphi \land \psi & \iff w, e \models \varphi \text{ and } w, e \models \psi, \\
w, e \models \neg \varphi & \iff \exists w, e \not\models \varphi, \\
w, e \models \forall x(\varphi) & \iff \forall c \in \text{Cons}(w, e) \models \varphi\{x = c\}, \\
w, e \models [\pi]\varphi & \iff \forall (e', \theta) \in [\pi](e, \varphi)(w, e' \models \varphi(\theta)), \\
w, e \models B\varphi & \iff \forall w' \in (e(w'), e \models \varphi).
\end{align*}
\]

### 3.1 Some Validities

In this section, we list a few validities. The first validity in the lemma below expresses that tests are used to introspect the beliefs of the agent. The other two validities are the usual axioms for sequential composition and nondeterministic choice from dynamic logic \([1]\). The axiom for sequential composition expresses that if after executing \( \pi_1 \); \( \pi_2 \) necessarily \( \varphi \) holds, then necessarily after executing \( \pi_1 \), we know that in case \( \pi_2 \) is executed consecutively \( \varphi \) will hold, and vice versa. The axiom for nondeterminism states that only if after execution of both \( \pi_1 \) and \( \pi_2 \) \( \varphi \) holds, we know that after executing \( \pi_1 + \pi_2 \varphi \) holds, and vice versa. Finally,
the fourth validity expresses that the execution of a goal does not modify the current world state.

The following lemma is useful in proving these validities.

**Lemma 1.**

- $[\pi \theta](e, \theta') = [\pi](e, \theta')$
- $\langle e', \theta' \rangle \in [\pi](e, \theta) \Rightarrow \theta \subseteq \theta'$

**Lemma 2.**

- $\models [\varphi] \psi \iff \forall (B\phi \rightarrow \psi)$
- $\models [\pi_1; \pi_2] \varphi \iff [\pi_1][\pi_2]\varphi$
- $\models [\pi_1; \pi_2] \varphi \iff ([\pi_1]\varphi \land [\pi_2]\varphi)$
- $\models [\pi] \phi \iff \phi, \text{ for } \phi \in \mathcal{L}_0$
- $\models [\pi] \forall \varphi \iff \forall [\pi]\varphi$

**Proof:** We prove the two first validities; the other three are left to the reader.

$- w, e \models [\varphi] \psi \iff$
  $\forall \varphi'(e', \theta) \in [\varphi] (e, \varnothing)(w, e' \models \psi \theta) \iff$
  $\forall \theta((\forall w \in e(w = \varphi \theta) \land \text{dom}(\theta) = \text{Free}(\phi)) \Rightarrow w, e \models \psi \theta) \iff$
  $\forall \theta((w, e \models B\phi \theta \land \text{dom}(\theta) = \text{Free}(\phi)) \Rightarrow w, e \models \psi \theta) \iff$
  $w, e \models \forall (B\phi \rightarrow \psi)$

$- w, e \models [\pi_1; \pi_2] \varphi \iff$
  $\forall \varphi'(e', \theta) \in [\pi_1; \pi_2] (e, \varnothing)(w, e' \models \varphi \theta) \iff$
  $\forall \varphi'(e', \theta') \in \bigcup_{\varphi''}(e''(\varnothing))([\pi_2][\varphi''][\varphi''')(w, e' \models \varphi \theta' \theta') \iff$
  $\forall \varphi''(e'', \theta'') \in [\pi_1][\varnothing](w, e'' \models ([\pi_2]\varphi)\theta'') \iff$
  $w, e \models [\pi_1; \pi_2] \varphi$

$\Box$

The first two validities below express that the B operator is normal and that an agent has positive introspection. The third validity expresses that an agent is aware of any changes in its beliefs due to executing goals. Finally, the fourth validity states that the order of the universal quantifier and the belief modality can be swapped, which is a consequence of the fact that the domain is the same in all world states.

**Lemma 3.**

- $\models B(\varphi \rightarrow \psi) \rightarrow (B\varphi \rightarrow B\psi),$
- $\models BB\varphi \Leftrightarrow B\varphi,$
- $\models B[\pi]B\varphi \Leftrightarrow [\pi]B\varphi,$
- $\models (\forall B\varphi) \Leftrightarrow B \forall \varphi.$
Proof: We only present the proof for the last validity.

\[ w, e \models \forall B \varphi \text{ iff } \]
\[ \forall \theta \left( \text{dom}(\theta) = \text{Free}(B \varphi) \Rightarrow w, e \models B \varphi(\theta) \right) \text{ iff } \]
\[ \forall \theta \left( \text{dom}(\theta) = \text{Free}(\varphi) \Rightarrow w, e \models B \varphi(\theta) \right) \text{ iff } \]
\[ \forall w' \in e \left( \forall \theta \left( \text{dom}(\theta) = \text{Free}(\varphi) \Rightarrow w', e \models \varphi(\theta) \right) \right) \text{ iff } \]
\[ \forall w' \in e \left( w', e \models \forall \varphi \right) \text{ iff } \]
\[ w, e \models B \forall \varphi \]

3.2 Equivalence of Operational and Denotational Semantics

In this subsection, we prove that the operational and denotational semantics are equivalent. The theorem below formally states this fact, and the corollary following the theorem states that the logic is suitable for proving partial correctness properties of agents. A partial correctness property expresses a condition on the final state in case the program finishes execution, that is, in case the agent successfully terminates.

Theorem 1. Let \( e_\sigma \) (\( e_\sigma' \)) be the set of world states that satisfy \( \sigma \) (resp. \( \sigma' \)).

\[ \forall \sigma' \left( \langle \pi, \sigma \rangle \xrightarrow{\alpha}^{*} \langle E, \sigma' \rangle \text{ iff } \langle e_{\sigma'}, \theta \rangle \in [\alpha](e_\sigma, \emptyset) \right) \]

Proof: By induction on the structure of \( \pi \).

\( \pi = a: \) By definition.

\( \pi = \phi?: \)
\[ \langle \phi?, \sigma \rangle \xrightarrow{\theta} \langle E, \sigma \rangle \text{ iff } \]
\[ \sigma \models \phi \theta \text{ and } \text{dom}(\theta) = \text{Free}(\phi) \text{ iff } \]
\[ \langle e_{\sigma}, \theta \rangle \in [\phi?](e_\sigma). \]

\( \pi = \pi_1; \pi_2: \)
\[ \langle \pi_1; \pi_2, \sigma \rangle \xrightarrow{\alpha}^{*} \langle E, \sigma' \rangle \text{ iff } \]
\[ \text{there are } \sigma'' \text{ and } \theta_1, \theta_2 \text{ such that } \langle \pi_1, \sigma' \rangle \xrightarrow{\theta_1}^{*} \langle E, \sigma'' \rangle \text{ and } \langle \pi_2, \sigma'' \rangle \xrightarrow{\theta_2}^{*}. \]

The corollary shows that in case an agent believes \( \varphi \) after successfully terminating execution of goal \( \pi \), then this fact can be expressed in the logic \( \mathcal{L} \). It also shows that in case a property \( [\pi]B \phi \) holds in the logic, the property really expresses a property of the program; that is, upon termination of goal \( \pi \), it must be the case that the agent believes \( \phi \).

Corollary 1. Let \( e_\sigma \) be the set of world states that satisfy \( \sigma \) and \( \phi \in \mathcal{L}_0 \) a sentence from the knowledge language \( \mathcal{L}_0 \). Then:

\[ \forall \sigma' \left( \langle \pi, \sigma \rangle \xrightarrow{\alpha}^{*} \langle E, \sigma' \rangle \Rightarrow \sigma' \models \phi \right) \text{ iff } w, e_\sigma \models [\alpha]B \phi. \]
Proof: Immediate from theorem 1. □

4 Conclusion

The programming logic for reasoning about a (subset of) 3APL agents presented here is a variant of a modal logic with operators for reasoning about the beliefs and the actions of an agent. As we showed, the programming logic is suitable for proving so-called partial correctness properties of agents. The logic is a variant of dynamic logic and can be used to prove properties of terminating 3APL agents. In the future, we would also like to deal with non-terminating behaviours of agents. A variant of a temporal logic might be more suitable to this end.

Because 3APL goals are plan-like structures composed of the basic capabilities of an agent, the action modality in the logic suffices to reason about the goals and plans of the agent. In other words, there is no need for another modality to represent the goals of an agent in contrast with agent logics based on, for example, the BDI approach. From this lack of a goal modality in our programming logic we can conclude that declarative goals are virtually absent in the programming language 3APL. Because 3APL is a member of a family of agent languages, this same conclusion holds for other languages like AGENT0, AgentSpeak and ConGolog as well. Although we think these agent programming languages offer a viable and clear approach to building agents, we also believe that it is interesting to take declarative goals more seriously and incorporate these also into agent programming. This consideration has given rise to a new approach to agent programming called GOAL. In a complementary paper about the programming language GOAL [6], we show how to incorporate declarative goals into agent programming and provide a temporal logic for reasoning about GOAL agents.

A framework for proving properties of agents has been presented by providing a programming logic for 3APL agents. The logic as well as the programming language are abstract in the sense that agent capabilities are a plugin feature and need to be specified by the user. In the future, we would like to investigate in more detail the specification of these basic capabilities in the programming logic. Also, it would be interesting to investigate the effect of using other knowledge representation formalisms upon the programming logic.

Acknowledgements We would like to acknowledge Hector Levesque for many helpful discussions about the semantics of the knowledge representation language.

References


