An Operational Semantics for the Single Agent Core of AGENT0

Koen V. Hindriks, Frank S. de Boer, Wiebe van der Hoek and John-Jules Meyer

University Utrecht, Department of Computer Science
P.O. Box 80.089, 3508 TB Utrecht, The Netherlands
{koenh,frankb,wiebe,jj}@cs.uu.nl

Abstract

AGENT0 is an agent-oriented programming (AOP) language, designed by Shoham [10]. An AGENT0 agent is an entity with a complex mental state, consisting of beliefs and commitments. On the one hand, in [10] the properties of the mental state of agents are formally introduced by a modal *logic*. On the other hand, the *programming language* AGENT0 for programming agents is only informally introduced. The link between the modal logic and the programming language AGENT0, however, is unclear. One of the more important mismatches is that the logic does not specify how the mental states of agents change over time. In the programming language so-called *commitment rules* are used for this purpose.

In this paper, we provide a clear and formal operational semantics for AGENT0, focusing on individual agents, which specifies both the static and the dynamic aspects of the mental states of AGENT0 agents. The proposal for a semantics for AGENT0 is split up in two separate layers. By introducing two layers a separation of concerns is established. The first and most basic layer specifies the meaning of the AGENT0 programming constructs. The second layer specifies the control loop of the AGENT0 interpreter.

The formal semantics discussed in this paper, we believe, yields a better insight into the use and a precise definition of the working of *commitment rules* in AGENT0 and the style of decision-making of AGENT0 agents. It also allows for a detailed comparison with other agent programming languages and provides a first step towards the design of a specification logic that is directly based on a formal semantics for AGENT0.

1 Introduction

AGENT0 is an experimental programming language to program *intelligent agents* [14], designed by Shoham [10]. An intelligent agent in AGENT0 is an entity with a mental state, consisting of the *beliefs* and *commitments* of the agent, that is capable of interacting with its environment and deciding what to do. So-called *commitment rules* provide the basic means for decision-making and the introduction of new commitments.

Since its introduction there have been several proposals for other agent programming languages in the literature. It is not so clear, however, how these various programming languages are related to each other. Moreover, it is not so easy to compare them because of the fact that not all of them have a clear and formal semantics. The main result of this paper is a proposal for an operational semantics for AGENT0 which is formal and facilitates a precise comparison with other agent languages, notably AgentSpeak(L) ([8]) and our own programming language 3APL ([3, 4]). The proposed semantics provides a detailed analysis of AGENT0 and - we believe - enhances our insight into agents and agent programming in general. Although our main focus is on providing a formal semantics which provides a basis for comparison with other languages and with clarifying the notion of agent programming in general, we also discuss some of the more methodological issues raised in Shoham's papers [9, 10, 11]. In particular, we discuss the reasons for specifying a formal semantics for the agent programming language and the relation between the logic proposed in [10, 9] and AGENT0.

In the first section, we motivate the need for a formal semantics for the *programming language* AGENT0 and argue that such a semantics cannot be derived from the modal logic for reasoning about beliefs, choices, etc. as introduced in [10, 9]. Then we give an informal overview of the programming language AGENT0. In section four, we then identify a subset of AGENT0 useful for programming single agents which we call the *single agent core of AGENT0.* It is this subset we provide a formal semantics for. In the next two sections, a formal, operational semantics for AGENT0 is proposed. The semantics is split up into two parts. The first part specifies the meaning of the basic constructs in the language, whilst the second part specifies the semantics for the AGENT0 interpreter used to execute agent programs. Our account reveals an interesting difference in the style of decision-making between AGENT0 agents and agents programmed in AgentSpeak(L) or 3APL.

2 Defining The Agent-Oriented Paradigm

The programming language AGENT0 supports the construction of agent programs. In [10], agents are defined as entities "whose state is viewed as consisting of mental components such as beliefs, capabilities, choices, and commitments." AGENT0 supports the construction of such agents by offering programming constructs which are viewed as formal counterparts of these mental concepts. The programming constructs stand in rough correspondence to their common sense counterparts. That is, the goal is to obtain a close enough resemblance to be suggestive and useful when programming agents, or, as Shoham puts it, to "make engineering sense out of these abstract concepts" [9].

The emphasis on mental states in agent-oriented programming makes it imperative to state precisely and explicitly what such a state is. For this reason, part of the agent programming paradigm in [9, 10, 11] is to provide a formal semantics of the mental state of agents. The means by which the mental terminology is made precise in these papers is provided by a formal *modal logic.* This approach is also used to define the mental components in the programming language PLACA [12], a successor of AGENT0.

The modal logic used to define the modal components of AGENT0 agents is in most detail outlined in [9]. The basis for the modal logic is an "explicit-time point-based logic". This temporal logic is extended with a KD45 operator ([1]) for modelling *belief* and a KD4 operator for modelling *commitment.* In terms of these operators, a *choice* of an agent is defined as commitment to itself, and the capability to (see to it that) X (where X is a proposition) is defined as a conditional: if the agent chooses to X, then X should be the case.

On the other hand, in [10] an informal account is provided of the agent *programming language.* The features of agents written in the programming language include constructs for referring to time points, beliefs, capabilities, actions, commitment rules for selecting actions, and communication. These constructs only roughly correspond to the operators of the logic.

There are more differences than similarities. To name only a few, the programming language includes an explicit representation of actions whereas the logic does not, commitments are defined in terms of these actions in the programming language (chosen actions) whereas commitment is a primitive notion in the logic, and the dynamics of the execution of agents is not modelled in the logic but is implemented by commitment rules in the programming language.

For these and other, similar reasons, there is no clear link between the modal logic and the programming language AGENT0. The formal semantics of the logic, therefore, contrasts with the lack of a formal semantics for the agent programming language itself. Still, it is important to have a formal semantics which gives a precise definition of the meaning of the basic constructs in the language AGENT0 and of the control loop of the interpreter which is used to run agents.

There are a number of reasons which motivate the need for a formal semantics. First of all, the construction of a formal semantics requires a detailed analysis and may reveal inconsistencies or gaps in informal accounts of software systems. As our analysis will show, there are several such gaps in the informal account of AGENT0. Such an incomplete specification may give rise to ad hoc implementations that differ in important aspects. Secondly, a formal semantics explains in a precise and explicit manner the meaning of the programming language constructs and thereby enhances our understanding of agent programming and agents in general. It also provides a formal definition of the rule-based decision-making and other features of agents. A formal semantics thus facilitates the programming of agent systems. Third, a formal semantics allows for a detailed comparison with other agent programming languages. In particular, the proposal of a formal semantics for AGENT0 highlights an interesting difference in the decision-making of AGENT0 agents and that of agents written in AgentSpeak(L) ([8]) and 3APL ([3, 4]). Finally, a formal semantics is a first step towards a programming logic for reasoning about agent programs written in AGENT0.

3 Overview of the programming language AGENT0

Agents are controlled by *agent programs.* In AGENT0 these programs are executed by an interpreter which continuously executes a *control loop* consisting of two phases: A phase in which the mental state is updated and a second phase in which actions (current commitments) are executed. In the first phase received messages from other agents are processed and commitment rules are fired to add new commitments to the current ones.

We will now give a somewhat more detailed overview of the constructs used in AGENT0 to program agents. First of all, the *beliefs* of an agent are simple, timed, atomic formulas, also called *facts*, from a first order language with *explicit time*, written like `(1march/10:00 (employee(John)))`. These facts may also contain variables. The programming language has a number of different types of *actions.* The first, most basic type is a so-called *private action* and is written like `(DO 18april/9:00 issue_boarding_pass)`. Note that these actions are also timed. The time indicates the time associated with the execution of the action (if it is executed). Three *communication* actions are provided. An action to inform an agent of a fact, written like `(INFORM 1march/2:00 smith (18april/10:00 (flight sf ny #293)))` which informs Smith that on 18 april 10:00 a flight with number 293 from San Francisco to New York is scheduled. Two more communication primitives, the `REQUEST` and the `UNREQUEST` primitives, are supported. These primitives allow an agent to send requests to perform an action to another agent and to request an agent to drop a commitment to a particular action. Finally, conditional actions of the form `(IF mntlcond action)` where `mntlcond` is a condition on the mental state and refrain actions of the form `(REFRAIN action)` can recursively be constructed from more simple actions.

Commitments of an agent consist of actions that an agent has chosen to perform at a particular time. The decision to perform an action is regulated by *commitment rules* of the form `(COMMIT msgcond mntlcond (agent action))` where `msgcond` is a condition on the received messages and `mntlcond` is a condition on the current mental state of an agent. The rule can be fired if both conditions are satisfied. Conditions on the mental state are boolean combinations of simple conditions on the beliefs of the form `(B fact)` and simple conditions on the commitments of the form `((CMT agent) action)`. Conditions on the messages are boolean combinations of simple message conditions of the form `(agent INFORM fact)` or `(agent REQUEST action)` [1] Finally, these conditions may include a number of different types of variables. The agent associated with `action` is the agent to which the commitment is made. As an example, the commitment rule

```
(COMMIT (?a REQUEST ?action) (B (now (myfriend ?a))) (?a ?action))
```

can be fired if a request to perform `?action` from agent `?a` has been received and the agent believes that `?a` is a friend. The constant `now` is a special constant referring to the current time. Upon firing, the agent commits to action `?action` and records that the commitment is made to agent `?a`. Thus, commitment rules provide a mechanism for making decisions in AGENT0. In the interpreter for AGENT0, this decision-making occurs in the first phase.

To summarise, the agent programming language AGENT0 includes features for representing the domain of interest (beliefs), to perform actions (simple, conditional and refrain actions), for interacting with agents (communication primitives), and for making commitments (by means of commitment rules). Each agent has it own associated set of capabilities, and actions and beliefs are explicitly associated with a particular time.

4 The Single Agent Core of AGENT0

The number of features present in the language AGENT0 is somewhat overwhelming. It is hard to understand the interaction between so many features and difficult to program with no clear understanding of the meaning of so many constructs. For simplicity, we therefore define a semantics for a subset of AGENT0 constructs. This semantics is a first approximation to a semantics which includes all features. The subset we consider in this paper includes all features except for multi-agent communication and reference to time. We call this subset the *single agent core of AGENT0.*

The single agent core as we have defined it includes beliefs, capabilities, three types of action, commitments, and commitment rules. The specification of the formal semantics for this core is based on the informal explanation of AGENT0 in [10]. Although we have tried to stay as close as possible to the intended meaning of AGENT0 constructs, our semantics is a reconstruction from the informal text in [10]. Our strategy for defining the semantics is

[1] It is not so clear why message conditions of the form `(agent UNREQUEST action)` are not allowed. Message conditions of this form would allow an agent, in case it is requested to drop a commitment but it still wants to achieve the goal associated with the commitment, to commit to an alternative plan of action.

to separate the semantic systems specifying the meaning of the basic constructs of the programming language like beliefs and rules and the interpreter for the language. This strategy is based on our research into our own agent language 3APL (cf. [3, 4]).

In the following sections, the single agent core is introduced and the syntax is formally defined. The syntax of the language is recast in a somewhat different notation. This notation serves our purposes better and is more suitable as a means for comparison of AGENT0 with other languages than the idiosyncratic syntax of AGENT0 as presented in [13, 10].

4.1 Beliefs

An AGENT0 agent essentially is a mental entity that operates on and manipulates a database of beliefs. The language for beliefs used in AGENT0 is a simple fragment of first-order logic, namely the set of literals (atomic statements and their negations). Beliefs are built from terms, predicates and negation. Here, following [10], we define a term as either a constant or a variable. These restrictions on the beliefs of an agent seem unnecessarily restrictive. Since no functions are allowed, nor Prolog-like programs, the computational expressiveness at this level is restricted to a bare minimum (and basically consists of pattern-matching). However, for our purposes this is of no real interest.

Definition 4.1 *(terms, atoms, literals)*
Let $\langle \mathsf{Pred}, \mathsf{Cons} \rangle$ be a signature, where Pred is a set of *predicate symbols*, and Cons is a set of *constant symbols*, and let Var be a set of countably infinite *variables*. Then the set of *terms* Term, the set of *atoms* At, and the set of *literals* Lit are defined by:

- $\mathsf{Var} \subseteq \mathsf{Term}$,
- $\mathsf{Cons} \subseteq \mathsf{Term}$,
- if $P \in \mathsf{Pred}$ of arity n, and $t_1, \ldots, t_n \in \mathsf{Term}$, then $P(t_1, \ldots, t_n) \in \mathsf{At}$,
- if $P \in \mathsf{Pred}$ of arity n, and $t_1, \ldots, t_n \in \mathsf{Term}$, then $P(t_1, \ldots, t_n), \neg P(t_1, \ldots, t_n) \in \mathsf{Lit}$.

4.2 Actions

When communication and time is left out of AGENT0, three types of actions remain: (i) *simple actions* of the form `(DO <privateaction>)`, constructed from a given set of so-called *private actions*, (ii) *conditional actions* of the form `(IF <mntlcond> <action>)`, where `<mntlcond>` expresses a condition on the mental state of the agent, and (iii) *refrain actions* of the form `(REFRAIN <action>)`. The refrain action `(REFRAIN <action>)` is a type of action that precludes commitment to actions of the form `<action>`. [2]

First, we formally define the syntax of the most basic actions, called *private actions*, and for reasons that will become clear below we currently postpone the formal introduction of the other types of actions.

Definition 4.2 *(private actions)*
Let Asym be a set of action symbols. Then the set of *private actions* Bact is defined by:

$$\mathsf{Bact} = \{\mathsf{a}(t_1, \ldots, t_n) \mid \mathsf{a} \in \mathsf{Asym} \text{ of arity } n, \text{ and } t_1, \ldots, t_n \in \mathsf{Term}\}$$

[2] According to Shoham, the refrain action "is really a non-action ([10], p. 72)

Actions The definition of actions provides an instructive example of the mismatch between the modal logic for defining the mental state and the programming language. Although in [10] it is stated that "in the programming language" actions are "introduce[d] [...] as syntactic sugar", actually, in the programming language an explicit construct `DO` is introduced and the so-called private actions may range from 'retrieval primitives', 'mathematical procedures' to robotic, physical actions. As an example, in the manual [13] of AGENT0, a number of basic or primitive actions programmed in Lisp are provided. These primitive actions are *not* propositions, but are explicit actions useful for programming agents in AGENT0. Moreover, the programming language allows complex conditional and refrain actions which have no counterpart in the modal logic. And finally, a number of communicative actions are supported by the programming language which are not present in the logic.

In the modal logic as defined in [9, 10], the most primitive actions, called *private actions*, are represented by propositions. In the logic no explicit and distinct representation for actions is present nor are there any modalities for actions incorporated into the logic as in, for example, dynamic logic ([2]). In [5], in particular this feature is criticised. The specific axioms of the logic which are proposed in [9] turn out to have the highly counter-intuitive consequence that if an agent cannot do something, then it believes that it can do it. To secure consistency within the logic, as a consequence, an agent cannot believe that it is incapable of anything. In [5], the main conclusion is that the core of this problem is due (in part at least) to the absence of an explicit representation of action.

For these reasons, there is a mismatch between the programming language and the modal logic in [10] in the representation of actions. In particular, whereas in the logic commitments are represented as obligations to a fact holding, in the programming language an agent commits to actions to change its mental state. As a consequence, the requirement that commitments to actions should be "internally consistent" makes only sense in the logic but not in the programming language. Moreover, the principle of "good faith" which requires that a commitment to see to it that a proposition holds implies that the agent believes that the proposition will hold cannot straightforwardly be translated to a statement about the programming language. Also, the persistence conditions with respect to beliefs and commitments are only discussed informally in [10]; no formal semantic account for either the logic or the programming language is provided.

4.3 Variables

In [10] a number of different types of variables are introduced. The types of variables in AGENT0 include variables ranging over agents, beliefs, and action statements as well as first order variables. For our purposes, the first type - agent variables - are not very interesting since we focus on the single agent core of AGENT0. Therefore, we do not include these variables. The second type of variables in the list, variables ranging over beliefs, provides a kind of higher-order feature concerning information, whereas the third type provides a higher-order feature concerning actions. They are, however, not included in the BNF syntax definition in [10]. In the absence of any complex beliefs or complex actions constructed by means of regular programming operators, we have doubts concerning the use of both types of variables. In this context, it seems to allow only for very general pattern matching. For example, a rule could be programmed expressing that if the agent is committed to any action, then inform another agent that the agent is currently busy. In [10], some examples are provided of variables ranging over beliefs, but no interesting examples are offered for variables ranging over actions.

The intended semantics of these variables is also not entirely clear. As far as variables are concerned, we therefore discuss only first order variables.

In [10], two types of first order variables are introduced, 'existentially quantified' and 'universally quantified' variables. Whereas the 'existentially quantified' variable is used similarly as variables are used in logic programming, the semantics of the 'universally quantified' variable is less clear. The use of a universally quantified variable, denoted by the prefix "?!", is illustrated in [10] by the following example:

`(IF (B (t (emp ?!x acme))) (INFORM a (t (emp ?!x acme)))).`

As explained in [10], this conditional action results in informing all employees which are believed to have acme of that fact. The scope of the 'universally quantified' variable seems to be the entire conditional.

The combination of both types of variables, however, leads to a problem concerning the order of the quantifiers. For example, what does a statement `IF (B (friend ?!x ?y)) (INFORM a (friend ?!x ?y)` mean? Does it mean that agent `a` should be informed in case everybody has a friend or in case there is somebody who is everybody's friend?

Because of this problem, in this paper we only allow one type, the 'existentially quantified' variable ranging over the domain of discourse of the agent, and do not consider a 'universally quantified' variable. Also, in PLACA [12], 'universally quantified' variables seem to have been left out of the language, and in AGENT0 they were not included in the actual implementation.

The types of variables allowed in the programming language provide another example of the mismatch between the logic and the programming language. The different types of variables in the programming language do not have counterparts in the logic. From the informal text, moreover, it is not easy to reconstruct the intended semantics of the different types of variables in the programming language.

4.4 Actions and Mental State Conditions

An AGENT0 agent is allowed to inspect both its beliefs and its commitments for decision-making. Also, during the execution of (conditional) actions an agent is allowed to do so. The beliefs of an agent are simple facts (literals) from a predicate logic. The commitments of an agent are the actions it has selected for execution. Together, the beliefs and commitments of an agent make up its mental state.

Since conditions on mental states may refer to actions and (conditional) actions may refer to mental state conditions, actions and mental state conditions are defined by simultaneous induction. Perhaps in our definition of mental state conditions we deviate the most from the syntax of AGENT0 as introduced in [10]. Our reason for doing so is that we want to be as precise as possible and at the same time aim at an operational semantics which can be readily implemented. Although the logic-like notation in [10] is very suggestive, the operational meaning of the notation is not so clear (although in [10] the opposite is claimed). In particular, to understand the parameter mechanism of the programming language it is important to have a formal semantics. Our notation is more suited for this purpose, though it is less suggestive as the notation introduced in [10]. We discuss these issues more extensively below when the operational semantics is defined. Actions are defined starting with private actions and mental state conditions are defined as four-tuples consisting of: (i) a set of literals the agent should believe, (ii) a set of literals the agent should not believe, (iii) a set of actions an agent should be committed to, and, finally, (iv) a set of actions the agent should not be committed to. A mental state condition is fulfilled if all of these conditions hold.

Definition 4.3 *(actions and mental state conditions)*
The set of actions Act, and the set of mental state conditions $\mathcal{L}^m$ is defined by simultaneous induction:

- The set of actions Act:
  - $\mathsf{Bact} \subseteq \mathsf{Act}$,
  - if $a \in \mathsf{Act}$ and $c_1, \ldots, c_n \in \mathcal{L}^m$, then $(c_1, \ldots, c_n : a) \in \mathsf{Act}$, called *conditional actions*,
  - if $a \in \mathsf{Act}$, then $\delta a \in \mathsf{Act}$, called *refrain actions*,
- The set of mental state conditions $\mathcal{L}^m$:
  - if $L^+, L^- \subseteq \mathsf{Lit}$ and $C^+, C^- \subseteq \mathsf{Act}$, then $\langle L^+, L^-, C^+, C^- \rangle \in \mathcal{L}^m$.

Informally, when executed a conditional action $(c_1, \ldots, c_n : a)$ performs the action a if one of the conditions c_i is satisfied. A refrain action removes commitments to actions.

4.5 Commitment Rules

The decision-making of AGENT0 agents is implemented by so-called *commitment rules.* Commitment rules introduce new commitments. They do not remove or modify the current commitments of the agent, but simply add new ones. A commitment rule consists of two parts: (i) a condition on the mental state of an agent and (ii) an action (recall that we do not discuss communication in this paper which explains the absence of message conditions in commitment rules). The action part represents the new commitment that is to be made if the rule is fired. A commitment rule thus (almost) has the same structure as a conditional action, but for clarity and to be able to keep them apart we introduce some new notation for commitment rules:

Definition 4.4 *(commitment rules)*
The set of *commitment rules* Rule is defined by:

- if $\langle L^+, L^-, C^+, C^- \rangle \in \mathcal{L}^m$, and $a \in \mathsf{Act}$, then $C^+, C^- \leftharpoondown L^+, L^- \mid a \in \mathsf{Rule}$. [3]

4.6 Agent Programs

Now we have introduced the basic constructs in the language AGENT0, we are able to define what an agent (program) is. Syntactically, an agent program is a set of capabilities, a set of initial beliefs, and a set of commitment rules. The set of capabilities in the program defines the expertise of the agent. Capabilities consist of a mental state condition and a private action. [4] The condition specifies under what (mental) condition the agent is capable of executing the action. The initial beliefs specify what the agent believes, at the start of execution. Finally, the commitment rules determine what types of decisions - new commitments - the agent will make. Note that initially the set of commitments is supposed to be empty.

[3] In the BNF syntax of AGENT0, multiple actions are allowed in a rule instead of a single action a. Since this feature can be simulated by rules with a single action, however, we have restricted rules to the more simple format with a single action in the body.

[4] In the BNF grammar of AGENT0 in [10] every type of action is allowed. In the main text (p. 77) only private actions are allowed.

Definition 4.5 *(agent program)*
An *agent program* is a tuple $\langle \mathsf{Cap}, \sigma_0, \Gamma \rangle$, where

- Cap is a set of *capabilities*, i.e. a set of actions of the form $(c_1, \ldots, c_n : a)$, where $c_i \in \mathcal{L}^m$ for all i and $a \in \mathsf{Bact}$,
- $\sigma_0 \subseteq \mathsf{Lit}$ is the set of *initial beliefs*, and
- $\Gamma \subseteq \mathsf{Rule}$ is a set of *commitment rules.*

5 Operational Semantics for the Core of AGENT0

In this section, we propose a formal semantics that specifies the meaning of the basic constructs of AGENT0. The semantics of mental state conditions, actions and commitment rules is defined. The semantics in this as well as the next section is based on the informal account of the programming language in [10] and only discusses the logical approach when it offers a different perspective or there is a difference between the two. Because the informal account is not always precise or detailed enough, there are a number of gaps in the account in [10] which we had to fill in to specify a semantics for AGENT0. In section 6, we define the semantics of the main control loop of an interpreter for AGENT0.

5.1 Transition Systems

The semantics we provide for AGENT0 is an *operational* semantics. For this purpose, we use Plotkin-style *transition systems* ([7]). Such a transition system defines a transition relation on agents, where each transition corresponds with a single computation step. This type of semantics can be viewed as specifying an abstract machine on which agent programs can be executed. Formally, a transition system is a deductive system that allows the *derivation* of transitions of a program. A transition system consists of a set of *transition rules* that specify the meaning of each programming construct in the language.

In agent-oriented programming, the notion of a mental state is the basic concept. Agent programs can be viewed as transition functions on mental states. The transition relation defined in this section therefore is a relation on mental states. It specifies how computation steps of an agent program transform mental states.

Definition 5.1 *(mental state)*
A *mental state* is a pair $\langle \Pi, \sigma \rangle$, where $\Pi \subseteq \mathsf{Act}$ is a set of actions, also called *commitments*, and σ is a set of *beliefs.*

We assume that there are no occurrences of free variables in an agent's belief base.

5.2 Semantics of Mental State Conditions

In [10], no (informal) explanation is given of the semantics for mental state conditions. The use of a logic-like notation suggests that the formal semantics of the modal logic used in [10, 9] should fill in this gap. However, the logic does not provide an appropriate account. First of all, the parameter mechanism and scope of free variables in the programming language is not explained by the logic. Secondly, the logic does not specify when an agent does not believe something given a particular mental state. The reason is that the logic only offers the usual modality for belief. However, from $\mathsf{B}p$, for example, it is not possible to derive that the agent does not believe q, that is, $\neg\mathsf{B}q$. For this purpose, an operator for 'only knowing' might be more appropriate.

Our semantics, however, is directly derived from the mental state as used in the operational semantics below. Although we cannot be sure that this semantics fully corresponds to that of the intended semantics, it probably provides a good approximation and completes the specification of the semantics for the language. Moreover, it is a precise semantics which can be evaluated on its merits and deficiencies. The semantics of beliefs is derived from the semantics of first order logic; we use $\models$ to denote the usual consequence relation of first order logic. To specify the parameter mechanism used in AGENT0 which is used to obtain bindings for free variables, we use the notion of a substitution. A substitution is a finite set of pairs (also called bindings) of variable-term pairs (for a more explicit and formal definition, see [6]).

Definition 5.2 *(semantics of mental state conditions)*
Let θ be a substitution. A mental state condition $c = \langle L^+, L^-, C^+, C^- \rangle$ is *satisfied* in a mental state $M = \langle \Pi, \sigma \rangle$ relative to θ, notation: $M \models c\theta$, if:

- for each $\varphi \in L^+$, we have that $\sigma \models \varphi\theta$,
- for each $a \in C^+$, we have that $a\theta \in \Pi$,
- for all $\varphi \in L^-$ and substitutions γ we have that $\sigma \not\models \varphi\gamma$, and
- for all $a \in C^-$ and substitutions γ we have that $a\gamma \notin \Pi$.

Thus, a mental state condition is satisfied if (i) it is possible to instantiate the free variables in L^+ and C^+ uniformly such that the agent's beliefs imply the literals in L^+ and the agent's commitments contain the actions in C^+, and (ii) it is not possible to instantiate a literal in L^- or action in C^- such that the agent respectively believes the instantiated literal or has committed itself to the instantiated action. Somewhat simplified, a mental state condition is satisfied if the agent believes L^+ and does not believe any of the literals in L^-, and the agent is committed to C^+ and is not committed to any of the actions in C^-.

5.3 Executing Commitments

Since the commitments of an agent consist of the actions the agent has selected for execution, the semantics of commitments is provided by a semantics for action execution. A semantics for actions is provided relative to the meaning of the most basic or private actions. These actions define the basic capabilities of the agent and are assumed to be given. In [10] the meaning of private actions is not discussed in great detail. However, a number of remarks suggest that private actions should be taken as updates on the set of beliefs of the agent. [5] This is the view we will take here. For this purpose, we introduce a (partial) function $\mathcal{T} : \mathsf{Bact} \times \wp(\mathsf{Lit}) \to \wp(\mathsf{Lit})$ which specifies what type of update is performed by each private action. [6] The computation step resulting from executing a private action then formally is

[5] On p. 61 of [10], it is remarked that there is no distinction made "between actions and facts, and the occurrence of an action will be represented by the corresponding fact holding." On p. 76 we are explained that the belief database may be updated "as a result of taking a private action" .

[6] We assume that the transition function does not introduce any free variables into the belief base of an agent in compliance with our previous constraint on belief bases.

defined by the following transition rule.

Definition 5.3 *(private actions)*
Let a be a private action.

$$\frac{\mathcal{T}(a,\sigma)=\sigma'}{\langle\{\ldots,a,\ldots\},\sigma\rangle \longrightarrow \langle\{\ldots\},\sigma'\rangle}$$

A conditional action $(c_1,\ldots,c_n : a)$ is executed by testing whether there are bindings for one of the conditions c_i such that it is satisfied in the current mental state of the agent and, if the test succeeds, by committing to the action a instantiated with the computed bindings. Variables in the mental state condition thus retrieve data from the belief base and current commitments by means of pattern-matching. The values retrieved are recorded in a substitution θ and used to instantiate the action a. If the test fails, either the conditional action could be removed from the commitments or not. We have chosen to retain the conditional action as a commitment. The transition rule for conditional actions is specified at the commitment level. Note that for the execution of a conditional action the context of execution, i.e. the entire mental state, is required.

Definition 5.4 *(conditional actions)*
Let θ be a substitution and $\Pi = \{\ldots,(c_1,\ldots,c_n : a),\ldots\}$.

$$\frac{\Pi,\sigma \models c_i\theta \text{ for some } i}{\langle(c_1,\ldots,c_n : a),\ldots\},\sigma\rangle \longrightarrow \langle\{\ldots,a\theta,\ldots\},\sigma\rangle}$$

A refrain action δa removes actions of the form a from the set of commitments. All possible instantiations of a are removed from the current commitments. In this way, it prevents the execution of these actions. It is not clear from [10] if and when a refrain action itself is removed from the set of commitments. We have chosen to retain the refrain action itself after executing it, since this type of action is probably most often used for *safety reasons.* I.e., for example, to prevent destructive behaviour or to prevent certain undesirable effects of the action which is to be refrained from. [7]

Definition 5.5 *(refrain actions)*
Let θ be a substitution.

$$\frac{\delta a, a\theta \in \Pi}{\langle\Pi,\sigma\rangle \longrightarrow \langle\Pi\setminus\{a\theta\},\sigma\rangle}$$

5.4 Applying Commitment Rules

A commitment rule $C^+, C^- \leftharpoonup L^+, L^- \mid a$ in AGENT0 is used to make new commitments. A commitment rule does not remove old commitments, it only introduces new ones. It is possible to make a new commitment if the mental state condition $\langle L^+, L^-, C^+, C^-\rangle$ is satisfied in the current mental state.

[7]We think that a refrain action makes more sense in a multi-agent setting, where requests from other agents to refrain from a particular action could be received. In the single agent setting, the conditions specifying the circumstances in which an agent should refrain from an action could probably just as well be explicitly listed in the condition part concerning the action in the list of capabilities in the program.

A unique feature of AGENT0 is that *all* applicable instances of a commitment rule are fired. Thus, a new commitment is made for every set of bindings for the free variables in the mental state condition of the rule which can be derived from the current belief base and commitments of the agent.

Formally, a rule fires for each substitution that satisfies the mental state condition of the rule in the current mental state. This also is an important difference between the meaning of conditional actions and commitment rules, besides the fact that rules are never removed from the agent program. It should be noted that a variant of the rule is used to prevent any undesired interference with variables in the rule and current commitments (cf. also [3]). We use $\mathsf{Free}(e)$ for the set of all (free) variables in e and $dom(\theta)$ for the variables in the domain of substitution θ.

Definition 5.6 *(rule application)*
Let Θ be the set of *all* substitutions θ such that $\Pi, \sigma \models c\theta$ and $dom(\theta) \subseteq \mathsf{Free}(c)$, where $c = \langle L^+, L^-, C^+, C^- \rangle$.

$$\frac{}{\langle \Pi, \sigma \rangle \longrightarrow \langle \Pi \cup \{a\theta \mid \theta \in \Theta\}, \sigma \rangle}$$

if $C^+, C^- \leftharpoondown L^+, L^- \mid a$ is a variant of one of the commitment rules of the agent such that no free variables in the rule occur in the set of commitments Π.

Alternative Semantics for Rule Application Whereas commitment rules do not change the current commitments of an agent, it will prove useful for defining an interpreter for AGENT0 to allow also rules which do modify the current commitments of an agent. In particular, it will be useful to have rules for dropping commitments. We use rules of the form $C^+, C^- \leftarrow L^+, L^- \mid a$ for this purpose and call these rules *decommitment rules.* The semantics is similar to that of commitment rules but instead of adding new commitments, these rules delete commitments, and instead of checking whether a condition holds, a check is performed to see if a mental state condition fails to hold. Thus, if the mental state condition of a decommitment rule fails to hold, then the agent is allowed to drop the associated commitment of the rule.

Definition 5.7 *(rule application)*
Let Θ be the set of *all* substitutions θ such that $\Pi, \sigma \not\models c\theta$ and $dom(\theta) \subseteq \mathsf{Free}(c)$, where $c = \langle L^+, L^-, C^+, C^- \rangle$.

$$\frac{}{\langle \Pi, \sigma \rangle \longrightarrow \langle \Pi \setminus \{a\theta \mid \theta \in \Theta\}, \sigma \rangle}$$

if $C^+, C^- \leftarrow L^+, L^- \mid a$ is a variant of one of the decommitment rules of the agent such that no free variables in the rule occur in the set of commitments Π.

5.5 Decision-Making in AGENT0, AgentSpeak(L) and 3APL

In AGENT0 there are no operators for constructing complex actions. For example, sequential composition or recursive structures like procedures or plan rules are absent in AGENT0. This lack of constructs for programming control flow and abstraction is one of the things which suggests that AGENT0 supports a *bottom-up approach.* By a *bottom-up approach* we mean here that in contrast with a *top-down approach* the agent does not decide what to do next by fixing a high-level goal and computing plans, but decides what to do next by looking only at the circumstances the agent finds itself in. This lack of goal-driven behaviour of agents has been one of the reasons to design the successor language PLACA with such features in [12]. PLACA is similar to AGENT0, but it allows planning by means of a plan library.

Another feature which also suggests AGENT0 supports a bottom-up approach is the type of rules allowed to program an agent. The rules to program agents are condition-action rules. We mean by this that the rules do not *modify* any existing (high-level) goals of the agent by substituting plans for achieving them, but just *add* new commitments to the set of commitments if the conditions of a rule are satisfied. In this sense we could say that a language like AGENT0 is *rule-driven*, while languages like AgentSpeak(L) ([8]) and 3APL ([3, 4]) are *goal-driven*.

With the bottom-up and top-down approach two different styles of decision-making can be associated. These different styles of decision-making give rise to two different styles of control loops for interpreters for agent programs. The different styles of decision-making can be explained by introducing two distinct *practical syllogisms* for decision-making; one corresponding to goal-driven interpreters and one corresponding to the rule-driven interpreters:

Practical Syllogism corresponding to the Goal-Driven Approach (PSG):
If (1) the agent intends to achieve a goal g, and (2) believes that g will not be achieved unless the plan p will be executed, then (Concl) the agent intends to execute plan p [8].

Practical Syllogism corresponding to the Rule-Driven Approach (PSR):
If (1) the agent believes it is in situation S and (2) the agent already has made commitments Π concerning a set of actions, then (Concl) the agent should commit to perform action a.

An explanation of the PSG syllogism is to view it as a reasoning scheme which may be used by the agent to achieve a goal by means of *some* plan. The PSR syllogism is best explained as a reasoning scheme to guarantee the commitment to *all* actions of a particular form. Thus, the first might profitably be used to infer a *possible* means to achieve a goal, while the second is more suited to be used as a means to infer the *necessity* to perform an action, i.e. to guarantee that some actions are performed in certain circumstances. The two approaches therefore are dual approaches and correspond to the duality of the possibility and necessity modalities. For this reason, it is interesting to note that in [10] the concept of *obligation* is taken as basic instead of that of *motivation*. In [10] Shoham actually somewhat overstates, we think, the contrast between the two different modes of decision making. According to him, the decision making in AGENT0 "reflects absolutely no motivation of the agent, and merely describes the actions to which the agent is obligated." (p. 67) The tools for programming an agent in AGENT0 on the one hand, and AgentSpeak(L) and 3APL on the other hand, thus are derived from two different perspectives on decision making.

6 The AGENT0 Interpreter

Apart from the basic language constructs, in [10] also a control loop or interpreter for executing agent programs is introduced. To complete the formal specification of AGENT0, therefore,

[8]It is probably advantageous for the agent to add some condition stating that the plan should not interfere to much with other goals of the agent. For simplicity, we have left such a condition out.

in this section we present a language in which such an interpreter can be specified. This language is based on previous work of the authors as reported in [4]. There are some notable differences, however. The original language from [4] is somewhat simplified, the semantics of some of the basic operators has been changed to suit the AGENT0 semantics, and a new update operator has been introduced. The language for programming interpreters introduced below is a set-based, imperative language with operators for referring to the basic notions of the agent language itself and for programming selection strategies. First, this so-called *meta language* is introduced and then it is shown how to define an interpreter for AGENT0 in this language.

6.1 Auxiliary Notions

For the purpose of defining an interpreter for AGENT0, it is convenient to introduce some notation. The notation is introduced to highlight a distinction between on the one hand *executing commitments* and on the other hand *making new or dropping old commitments.* Basically, the notation consists of a kind of labelling of the transition relation as defined in the previous section with the information that is relevant in defining a control loop for an interpreter. The relevant information consists in (i) the action that is executed or the rule that is applied (written above the transition relation $\longrightarrow$), and (ii) the commitments that have been deleted or removed (this information is added as subscripts to the transition relation $\longrightarrow$ where the set Δ is the set of commitments that have been deleted and Λ is the set of commitments that have been added).

Notation 6.1 Let M, M' be mental states and Δ and Λ be sets of actions.

- $M \xrightarrow{a}_{\{a\},\emptyset} M'$ for a computation step due to a private action,
- $M \xrightarrow{(c_1,\ldots,c_n:a)}_{\{(c_1,\ldots,c_n:a)\},\{a'\}} M'$ for a computation step due to a conditional action,
- $M \xrightarrow{\delta a}_{\{a'\},\emptyset} M'$ for a computation step due to a refrain action,
- $M \xrightarrow{\rho}_{\emptyset,\Lambda} M'$ for a computation step due to the application of the commitment rule ρ, and
- $M \xrightarrow{\rho}_{\Delta,\emptyset} M'$ for a computation step due to the application of the decommitment rule ρ.

When we abstract from the specific action that has been executed, the first three (labelled) transitions can also be written as $M \xrightarrow{\pi}_{\Delta,\Lambda} M'$ where π denotes the action that has been executed. Moreover, we write $M \xrightarrow{\pi}$ if there is an M' (and sets Δ, Λ) such that $M \xrightarrow{\pi}_{\Delta,\Lambda} M'$. Similarly, the last two transitions due to the application of rules can also be written as $M \xrightarrow{\rho}_{\Delta,\Lambda} M'$. We write $M \xrightarrow{\rho}_{\Delta,\Lambda}$ if there is an M' such that $M \xrightarrow{\rho}_{\Delta,\Lambda} M'$.

In case, for a mental state M, we have that $M \xrightarrow{\pi}$ for some commitment π, we say that action π is *executable in* M; if for some rule ρ $M \xrightarrow{\rho}$, we say that the (de)commitment rule ρ is *applicable in* M.

6.2 Syntax

The syntax of the meta language is very similar to imperative programming and consists of assignment, and the regular operators for sequential composition, nondeterministic choice, and iteration. The basic terms and variables, however, range over a special domain: the commitments and rules of agent programs. Two types of terms are thus distinguished, respectively the commitment and rule terms. The commitment and rule terms of the meta language are used to access the commitments and rules of the agent program of the object language in the meta language. The terms refer to sets of commitments or sets of rules. The operators of the meta language for building complex terms are the usual set operators.

Definition 6.2 Let $\mathsf{Var}_\Pi, \mathsf{Var}_\Gamma$ be given disjoint, countably infinite sets of variables.
Then the set of *commitment terms* $\mathfrak{T}_\Pi$ is defined by: (i) $\mathsf{Var}_\Pi \subseteq \mathfrak{T}_\Pi$, (ii) $\emptyset, \Pi \in \mathfrak{T}_\Pi$, (iii) if $g_0, g_1 \in \mathfrak{T}_\Pi$, then $g_0 \cap g_1, g_0 \cup g_1, g_0 - g_1 \in \mathfrak{T}_\Pi$.
The set of *rule terms* $\mathfrak{T}_\Gamma$ is defined by: (i) $\mathsf{Var}_\Gamma \subseteq \mathfrak{T}_\Gamma$, (ii) $\emptyset, \Gamma \in \mathfrak{T}_\Gamma$, (iii) if $r_0, r_1 \in \mathfrak{T}_\Gamma$, then $r_0 \cap r_1, r_0 \cup r_1, r_0 - r_1 \in \mathfrak{T}_\Gamma$.

The commitment and rule terms are the usual set terms, constructed from the set operators $\cap$, etc. $\emptyset$ is a constant denoting the empty set. Furthermore, Γ is a rule term constant denoting the set of rules of an agent program. The commitment term Π, however, is a variable which denotes the commitments of the current object mental state during execution. As a notational convention, strings denoting variables start with upper case while all other strings for terms denote arbitrary terms including variables.

The meta language includes assignment of sets of commitments or rules to respectively commitment or rule variables, tests for equality on the commitment and rule terms, and the regular programming constructs for sequential composition, nondeterministic choice, and iteration. It also includes three special actions to control the type of computation steps and updates that are performed at the object level. The action *apply*$(R, \mathit{Del}, \mathit{Add})$ selects an applicable rule from the input set R, computes the commitments that would be removed from and added to the current commitments if the rule would be applied, and returns these sets respectively in *Del* and *Add*; then the selected rule is removed from R and this is repeated until no rules from R are applicable anymore. In the end, *Del* consists of all the commitments that would have been removed by applying all these rules and *Add* consists of all the commitments that would have been added. If there is no applicable rule in the set R, then the output variables *Del* and *Add* return the empty set. The action *ex*(G) is used for the execution of committed actions from the input set G. The set G is updated every time an action is executed. It repeatedly executes until no actions from G are executable anymore. Finally, the action *upd*$(\mathit{del}, \mathit{add})$ first removes the commitments in the input term *del* from the current commitments and then adds the actions in *add* again. The first three meta actions are calls to the object agent system to perform object transition steps corresponding to execution of actions or application of rules.

Definition 6.3 The set of *meta statements* $\mathfrak{S}$ is defined by:

- if $G \in \mathsf{Var}_\Pi, g \in \mathfrak{T}_\Pi$, then $G := g \in \mathfrak{S}$,
- if $R \in \mathsf{Var}_\Gamma, r \in \mathfrak{T}_\Gamma$, then $R := r \in \mathfrak{S}$,
- if $g, g' \in \mathfrak{T}_\Pi$, then $g = g', g \neq g' \in \mathfrak{S}$,

- if $r, r' \in \mathfrak{T}_\Gamma$, then $r = r', r \neq r' \in \mathfrak{S}$,
- if $G \in \mathsf{Var}_\Pi$, then $ex(G) \in \mathfrak{S}$,
- if $R \in \mathsf{Var}_\Gamma$ and $Del, Add \in \mathsf{Var}_\Pi$, then $apply(R, Del, Add) \in \mathfrak{S}$,
- if $del, add \in \mathfrak{T}_\Pi$, then $upd(del, add) \in \mathfrak{S}$, and
- if $\beta, \beta' \in \mathfrak{S}$, then $\beta; \beta', \beta + \beta', \beta^* \in \mathfrak{S}$.

Remark 6.4 If in some context a *planning system* is defined, then we could also introduce an action $plan(g, r, G', R')$ for adding new rules to the set of rules of the program. (g, r are input terms; in G' the set of goals for which a plan has been found could be stored, and in R' the set of plans found.) The suggestion of incorporating a planning system in the language *PLACA* (cf. [12]) possibly can be viewed as such an action. Note that in this case, the rule term Γ no longer is a constant.
End Remark

6.3 Semantics

In this section the operational semantics of the meta language is defined. The transition relation of the meta transition system is denoted by $\Longrightarrow$. The transition relation $\Longrightarrow$ is a relation on meta configurations, which are pairs consisting of a program statement and a meta state. Meta level states should include the information about object level features an agent interpreter should be able to access. Among these features are the object mental state and the commitment rules of an agent program. Furthermore, a meta state should keep track of the values of variables used in the meta program.

Definition 6.5 A *meta state*, or *m-state* τ is a tuple $\langle\langle\Pi, \sigma\rangle, <_\Pi, \Gamma, V\rangle$, where $\langle\Pi, \sigma\rangle$ is an *object mental state*, $<_\Pi$ is an ordering on the set of actions Act, Γ is a set of *object rules*, and V is a *variable valuation* of type : $(\mathsf{Var}_\Pi \rightarrow \wp(\mathsf{Act})) \cup (\mathsf{Var}_\Gamma \rightarrow \wp(\mathsf{Rule}))$.

The ordering on the set of actions is used to define priorities on action types, as will be explained below. An *m-configuration* is a pair $\langle\beta, \tau\rangle$ where β is a program statement and τ is an m-state. We also write an m-configuration as a triple $\langle\beta, \langle\Pi, \sigma\rangle, V\rangle$, where the constant set of rules and ordering on actions is dropped from the configuration.

Definition 6.6 *(semantics of terms)*
Let $\tau = \langle\langle\Pi, \sigma\rangle, <_\Pi, \Gamma, V\rangle$ be an m-state, and T range over goal and rule terms. Then the interpretation function $[\![\cdot]\!]_\tau : (\mathfrak{T}_\Pi \rightarrow \wp(\mathsf{Act})) \cup (\mathfrak{T}_\Gamma \rightarrow \wp(\mathsf{Rule}))$ is defined by:

- $[\![T]\!]_\tau = V(T) \cap \Pi$, for $T \in \mathsf{Var}_\Pi \cup \mathsf{Var}_\Gamma$,
- $[\![\Pi]\!]_\tau = \Pi$, $[\![\Gamma]\!]_\tau = \Gamma$,
- $[\![\emptyset]\!]_\tau = \emptyset$,
- $[\![T_0 \oplus T_1]\!]_\tau = [\![T_0]\!]_\tau \oplus [\![T_1]\!]_\tau$, for $\oplus \in \{\cap, \cup, -\}$,

We will drop the subscript referring to the state in the rest of this paper, as it will be clear from the context which state is referred to.

The semantics of each of the four special primitives of the meta language is formally introduced next. In this paper, we do not define the formal semantics for the well-known constructs from imperative programming, but refer the reader to [7, 4].

The action $ex(G)$ is an action that executes as many actions from G as possible, in a non-deterministic order. The formal definition of the semantics of the action $ex(G)$, therefore, is defined by iteration. $ex(G)$ is executed by choosing an executable action with highest priority from $[\![G]\!]$, deleting this action from $[\![G]\!]$, and executing the selected action at the object level, until no more actions from $[\![G]\!]$ can be executed. When $ex(G)$ terminates, the content of the variable G contains the remaining actions which are not executable in the current mental state. The second transition rule below specifies the termination condition of the iteration.

Definition 6.7 *(transition rule for ex)*

$$\frac{M \xrightarrow{\pi}_{\Delta,\Lambda} M', \pi \in [\![G]\!]}{\langle ex(G), M, V\rangle \Longrightarrow \langle ex(G), M', V\{([\![G]\!] \setminus \Delta) \cup \Lambda / G\}\rangle}$$

and there is no $\pi' \in [\![G]\!]$ such that $\pi < \pi'$ and $M \xrightarrow{\pi'}$.

$$\frac{M \not\xrightarrow{\pi} \text{ for all } \pi \in [\![G]\!]}{\langle ex(G), M, V\rangle \Longrightarrow \langle E, M, V\rangle}$$

The action $apply(R, Del, Add)$ nondeterministically selects an applicable rule from R, and stores in the variables Del and Add respectively the set of commitments that would have been removed and the set of commitments that would have been added if the rule would have been actually applied. This process is repeated until no more rules in R are applicable.

Definition 6.8 *(transition rule for apply)*

$$\frac{M \xrightarrow{\rho}_{\Delta,\Lambda}, \rho \in [\![R]\!]}{\begin{array}{c}\langle apply(R, Del, Add), M, V\rangle \Longrightarrow \\ \langle apply(R, Del, Add), M, V\{[\![R]\!] \setminus \{\rho\}/R, [\![Del]\!] \cup \Delta/Del, [\![Add]\!] \cup \Lambda/Add\}\rangle\end{array}}$$

$$\frac{M \not\xrightarrow{\rho} \text{ for all } \rho \in [\![R]\!]}{\langle apply(R, Del, Add), M, V\rangle \Longrightarrow E, M, V\rangle}$$

Finally, an update action $upd(del, add)$ is defined on the set of commitments. Its arguments are two sets of actions: the first set is a set of actions that are to be deleted from the commitments - which is done first - and the second set is a set of actions which are to be added to the set of commitments. The update action always succeeds.

Definition 6.9 *(transition rule for upd)*

$$\frac{}{\langle upd(del, add), \langle \Pi, \sigma\rangle, V\rangle \Longrightarrow \langle E, \langle (\Pi \setminus [\![del]\!]) \cup [\![add]\!], \sigma\rangle, V\rangle}$$

6.4 Interpreter

One of the basic differences between a number of agent languages resides in their *control structure*. The main purpose of a control structure for an agent language is to specify which commitments to deal with first and which rules to apply during the execution of an agent program. The strategy for executing commitments and applying rules in AGENT0, is to execute all executable commitments and apply all applicable rules in every cycle of the control loop of the interpreter. In this section we use the meta language to specify an interpreter for AGENT0, based on the informal account in [10].

The strategy used in AGENT0 for decision-making corresponds to the PSR syllogism that we discussed in section 5. The main control loop in the AGENT0 interpreter executes two consecutive phases:

1. in the first phase, *update* the commitments (the commitment rules of the agent program are used in this phase),

2. in the second phase, *execute* commitments made previously (this phase is independent from the agent program).

The update phase in the control loop again consists of two distinct steps. On the one hand, new commitments are added by firing the applicable commitment rules of the agent program, and, on the other hand, the feasibility of the agent's commitments is checked. The test to determine whether or not a commitment is feasible consists of checking a condition on the mental state to see whether or not the agent believes that it is capable of executing the commitment. In case a commitment is no longer considered feasible, the commitment is removed. It is here that the decommitment rules which we introduced are useful. By means of these rules, we can implement the feasibility check in the interpreter below. The order imposed on these two steps in the AGENT0 interpreter is not discussed in [10]. We have chosen to first apply the applicable commitment rules and then to check for feasibility. We believe that this choice makes the most sense.

The execution phase, i.e. the second phase in the loop above, boils down to executing as many commitments as possible in AGENT0. Since the actions executed are simple actions we might presume that each of the actions executed is executed completely. This remark applies in particular to conditional actions, which are executed by first performing a test and in case the test succeeds executing the action part.

An implicit order on the type of actions is assumed. In particular, the refrain actions have a higher priority than private or conditional actions. The reason is that refrain actions should always be executed first to prevent actions from which the agent should refrain to be executed. The priority on actions is used in defining the semantics of the meta action *ex*.

	AGENT0
	`REPEAT`
Step 1	$R := \Gamma$; $Del := \emptyset$; $Add := \emptyset$;
	`apply`(R, Del, Add);
	`upd`(Del, Add);
Step 2	$R := \Delta$; $Del := \emptyset$; $Add := \emptyset$;
	`apply`(R, Del, Add);
	`upd`(Del, Add);
Step 3	$G := \Pi$;
	`ex`(G);
	`UNTIL FALSE`;

Table 6.4

The meta program defining the interpreter is given in table 6.4. It implements the three separate steps of the interpreter. Step 1 corresponds to firing the set of commitment rules Γ of the agent program. After initialising the variables R, Del and Add, this step is implemented by the meta action *apply* which fires as many rules as possible and the action *upd* for actually updating the mental state of the agent.

Step 2 corresponds to the check for feasibility of committed actions, and is again implemented by the meta action *apply*. A set of decommitment rules Δ is presupposed. Each of the rules in this set is of the form $C^+, C^- \leftarrow L^+, L^- \mid a$ where the mental state condition $c = \langle L^+, L^-, C^+, C^- \rangle$ corresponds to the feasibility or capability conditions for the action a. Recall that by the definition of the semantics for decommitment rules, only in case the capability conditions associated with an action fail, the action is to be removed from the set of commitments.

Step 3 corresponds to the execution of as many commitments as possible. The meta action *ex* implements this step of the interpreter. The order on actions is assumed to give higher priority to refrain actions than to any of the other action types. The actions are executed completely by iteratively repeating execution for the remaining part of (conditional) actions.

7 Conclusion

By abstracting from a number of features of AGENT0, we have been able to construct an operational semantics for AGENT0. We used a two-layered approach by separating the semantics of the basic programming constructs and the semantics of the control structure in the interpreter for AGENT0. This approach yields a clear and intuitive definition of AGENT0, as well as for other agent languages. The benefits of a formal semantics in general and for AGENT0 in particular is that it provides for a precise specification for an implementation of the language. In contrast, in the original, informal specification in [10] a number of gaps were identified. Moreover, the formal semantics allows for a formal comparison, and thereby clarifies a number of differences between rule-based agent languages (cf. [4]).

The specification of a formal semantics for AGENT0, we believe, also resulted in a better understanding of the use of so-called commitment rules in AGENT0. We distinguished a bottom-up approach used in AGENT0 and a top-down approach used in AgentSpeak(L) and 3APL. Corresponding to this distinction, AGENT0 can be characterised as *rule-driven*, while AgentSpeak(L) and 3APL can be characterised as *goal-driven*.

References

[1] Brian F. Chellas. *Modal logic: an introduction.* Cambridge University Press, 1980.

[2] David Harel. *First-order dynamic logic.* LNCS 68. Berlin: Springer, 1979.

[3] Koen V. Hindriks, Frank S. de Boer, Wiebe van der Hoek, and John-Jules Ch. Meyer. Formal Semantics for an Abstract Agent Programming Language. In Munindar P. Singh, Anand Rao, and Michael J. Wooldridge, editors, *Intelligent Agents IV (LNAI 1365)*, pages 215–229. Springer-Verlag, 1998.

[4] Koen V. Hindriks, Frank S. de Boer, Wiebe van der Hoek, and John-Jules Ch. Meyer. Control Structures of Rule-Based Agent Languages. In J. P. Müller, M. P. Singh, and A. S. Rao, editors, *Intelligent Agents V (LNAI 1555)*, pages 381–396. Springer-Verlag, 1999.

[5] Andrew J.I. Jones. Practical Reasoning, California-style: Some Remarks on Shoham's Agent-oriented Programming (AOP), 1993.

[6] J.W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, 1987.

[7] G. Plotkin. A structural approach to operational semantics. Technical report, Aarhus University, Computer Science Department, 1981.

[8] Anand S. Rao. AgentSpeak(L): BDI Agents Speak Out in a Logical Computable Language. In W. van der Velde and J.W. Perram, editors, *Agents Breaking Away (LNAI 1038)*, pages 42–55. Springer-Verlag, 1996.

[9] Yoav Shoham. Implementing the Intentional Stance. In Robert Cummins and John Pollock, editors, *Philosophy and AI: Essays at the Interface*, chapter 11, pages 261–277. MIT Press, 1991.

[10] Yoav Shoham. Agent-oriented programming. *Artificial Intelligence*, 60:51–92, 1993.

[11] Yoav Shoham. Agent Oriented Programming: An overview of the framework and summary of recent research. In Jan-Michael Friedrich Masuch and László Pólos, editors, *Knowledge representation and reasoning under uncertainty, logic at work (LNAI 808)*, pages 123–129. Berlin [etc.], Springer-Verlag, 1994.

[12] Sarah Rebecca Thomas. *PLACA, An Agent Oriented Programming Language.* PhD thesis, Department of Computer Science, Stanford University, 1993.

[13] Mark C. Torrance. The AGENT0 Manual, 1991.

[14] Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.