An Operational Semantics for a BDI Agent-Oriented Programming Language*

Álvaro F. Moreira[1] Rafael H. Bordini[2]

[1]Departamento de Informática (DEIN)
Centro de Ciências Exatas e Tecnologia (CCET)
Universidade de Caxias do Sul (UCS)
95070-560, Caxias do Sul, RS, Brazil
afmoreir@ucs.tche.br

[2]Departamento de Informática Teórica (INT)
Instituto de Informática (II)
Universidade Federal do Rio Grande do Sul (UFRGS)
CP 15064, 91501-970, Porto Alegre, RS, Brazil
bordini@inf.ufrgs.br

Abstract

This paper presents a structural operational semantics for AgentSpeak(L), a logic-based, agent-oriented programming language that is grounded on the BDI model for cognitive agents. This language was first presented by Rao in 1996 in a paper where he defined loosely an interpreter for it, and also sketched its operational semantics. Further formalisation was given by d'Inverno and Luck using the Z formal specification language. A semantic description for AgentSpeak(L) following traditional approaches—which would allow further formal work on it, such as comparison with other languages as well as facilitating the proof of some of its properties—was still missing. This paper makes up for that missing formal semantics using a standard notation in the field of programming languages, and points out to future directions of theoretical research contributing to the incipient paradigm of agent oriented programming languages.

Keywords: Logic-Based Agent Programming Languages, BDI Agents, Structural Operational Semantics.

*This work has been partially supported by CNPq and FAPERGS.

1 Introduction

In [8], a logic-based, agent-oriented programming language called AgentSpeak(L), was introduced. This language is based on the BDI (Belief-Desire-Intention) architecture [10] for cognitive agents. In that paper, Rao presented the operation of an abstract interpreter for that language. He also defined a proof theory for it with which, he claimed, it would be possible to prove known properties that are satisfied by BDI systems, as proven with BDI Logics [9, 12] previously. Further, he claimed that there is a one-to-one correspondence between his interpreter and the proof system. In this way, he proposed what can be considered as the first viable approach to bridging the ever so quoted gap between BDI theory and practice: an old worry of the BDI community.

Rao's sketched proof system was in fact presented in the usual style of the inference rules defining a transition relation between configurations of a transition system that gives the structural operational semantics of a programming language [7]. However, Rao only provided a few rules to give a flavour of how that proof system should look like, so a complete formal operational semantics was missing. Further formalisation of the abstract interpreter and a few missing details were given in [3]. Their formalisation was done using the Z formal specification language [14]. Those Z specifications, so they claim, can also be used as a framework for the formal specification of other BDI systems. They can be useful in the implementation of a prototype AgentSpeak(L) interpreter (through animation), but still Z appears not the best notation for, e.g., proving properties of a programming language. That is the reason why we have chosen a traditional framework for this semantic description. In this paper, we give formal semantics to AgentSpeak(L) using the standard method of structural operational semantics.

AgentSpeak(L) has many similarities with traditional logic programming, which is a trait that would favour its becoming a popular language: it should prove quite intuitive for those familiar with logic programming. Besides, it has a neat notation and provides quite elegant specifications of BDI agents. This could contribute to its being considered also for the production of formal, high-level specification of BDI systems. However, the fact that Rao's abstract interpreter had not, until very recently [6], been actually implemented accounts for the language's lack of popularity. A rigorous analysis of the language would further increase interest in it, and would allow future extensions to be formally grounded. It would also make it possible to precisely

state and prove properties of the language, and show that it is consistent with BDI logics. This is the rationale behind this paper.

Also, agent oriented programming [11] and agent oriented software engineering [15] are becoming widely recognised as the potential basis for the implementation of next generation computing systems. Even though the foundations of agent oriented programming languages were proposed by Shoham in 1993, few agent oriented languages have been put forward and they still lack good formalisations and/or implementations. Agent oriented programming is an incipient but very promising area of application for formal methods supporting the implementation of large-scale working systems, and our work contributes in that direction.

In previous papers, we concentrated on more practical aspects of AgentSpeak(L). In order to be able to run our AgentSpeak(L) applications before a purpose-built interpreter is available, we have found the means to running AgentSpeak(L) programs within Sloman's SIM_AGENT framework [13]. We have implemented a tool which automatically converts AgentSpeak(L) programs into running code within SIM_AGENT: we call this tool SIM_Speak [6], which was is fact the first (to the best of our knowledge) working interpreter for AgentSpeak(L). More recently, we developed an AgentSpeak(L) interpreter from scratch (in C++) and we in fact extended the language so that it is possible to specify certain relationships among plans. This allows for the automatic generation of efficient intention selection functions by using decision-theoretic scheduling; the extended language is called AgentSpeak(XL) [1]. One of the future work that depends on this first complete operational semantics, as we shall mention in the conclusion, deals precisely with proving the correctness of such an implementation, and extending the semantics to account for the language extensions.

This paper is structured as follows. In the next section we discuss the syntax and the basic intuitions behind AgentSpeak(L) constructs (such as beliefs, plans, and goals), while in Section 3 we give the rules of the operational semantics for the language and we explain the intuitions behind each of the rules. We conclude with comments on the various intended uses of this formal semantics for AgentSpeak(L).

2 Syntax of AgentSpeak(L)

2.1 Beliefs

The atomic formulas of the language are predicates given by the following grammar:

$$at \quad ::= \quad \mathtt{P}(t_1, \ldots, t_n) \quad (n \geq 0)$$

where $\mathtt{P}$ is a predicate symbol and $t_1, \ldots, t_n$ are standard terms of first order logic. We use *at* as a metavariable for atomic formulas. The subset of formulas of first order predicate logic that is used in AgentSpeak(L) programs is given by the following grammar:

$$\varphi \quad ::= \quad at \mid \neg at \mid \varphi \wedge \varphi'.$$

We call a *belief* an atomic formula *at* with no variables and we use b as a metavariable for beliefs. An AgentSpeak(L) program consists of a set of *beliefs* and of a set of *plans*. Syntactically, the set of beliefs of an AgentSpeak(L) program is a sequence of beliefs:

$$\mathit{beliefs} \quad ::= \quad b_1 \ldots b_n \quad (n \geq 0).$$

When a programmer specifies a set of beliefs in an AgentSpeak(L) program, he is defining what the agent initially believes about the environment where it is situated. As the environment changes, this set of beliefs changes accordingly (through perception). This generates *events* which can trigger the execution of plans.

2.2 Plans

Besides beliefs, a programmer also specifies a set of plans. Plans determine a course of action to be pursued in case certain events occur. When writing a plan the programmer should state which event the plan is supposed to deal with. More than one plan can be associated with an event; if that event occurs, it is necessary to check whether the conditions under which each plan is considered applicable to handling the event; then the body of one of these applicable plans can be executed.

In summary, when writing a plan, the programmer has to define: the event *te* that can trigger the plan, called the plan's *triggering event*; the conditions φ that must hold for this plan to be considered applicable, called

the *context* of the plan; and finally a sequence h of formulæ to be executed. These formulæ can be goals to pursue, the updating of beliefs (as explained later), or actions to be performed, in order for the agent to act properly on the triggering event. We now discuss each of these parts in turn.

Triggering Events The first component in the head of a plan is a *triggering event*. Triggering events are given by the following grammar:

$$te \quad ::= \quad +at \quad | \quad -at \quad | \quad +g \quad | \quad -g.$$

A triggering event can then be the addition or the deletion of a belief from an agent's belief base ($+at$ and $-at$, respectively), or the addition or the deletion of a goal ($+g$ and $-g$, respectively).

Context As we said before, for a plan to be applicable it must be in agreement with the agents's beliefs. Each plan then, have in its head a formula φ that specifies the condition that must be a logical consequence of the agent's beliefs if the plan is to be considered.

Actions, Goals, and Belief Updating AgentSpeak(L) is concerned specifically with agent's reasoning. Because of that, we abstract away from the details of which actions are available and how they are actually performed. We assume the agent has at its disposal a set of *actions* and we use a as a metavariable ranging over them.

A *goal* in AgentSpeak(L) is given by the following grammar:

$$g \quad ::= \quad !at \quad | \quad ?\varphi.$$

A goal $!at$ is called an *achievement goal*. A goal $?\varphi$ is called a *test goal* (or a `query goal`). Achievement goals have to do with the execution of subplans whilst test goals are used to instantiate variables from unification with current beliefs of the agent while the plan is being executed.

Unlike Rao [8], we allow the body of plans to include the updating of the agent's own beliefs. (Although Rao pointed out to these constructs, he did not include them in the presentation of his abstract interpreter). This *updating of beliefs* is given by the grammar:

$$u \quad ::= \quad +b \quad | \quad -b.$$

A sequence h of actions and goals of a plan (forming a plan's *body*) is then given by the grammar:

$$h \quad ::= \quad a \quad | \quad g \quad | \quad u \quad | \quad h \,;\, h'.$$

A plan in AgentSpeak(L) is then given by the following grammar:

$$p \quad ::= \quad te : \varphi \leftarrow h,$$

where $te : \varphi$ is the plan's head; we use *head* as a metavariable for plan heads. The set of plans of an agent is written as a list of plans:

$$\mathit{plans} \quad ::= \quad p_1 \ldots p_n \quad (n \geq 1).$$

In AgentSpeak(L), an agent is simply a specified of a set of beliefs (the agent's initial belief base) and a set of plans (the agent's plan library). An agent specification is therefore given by the following grammar:

$$\mathit{agent} \quad ::= \quad \mathit{beliefs}\ \mathit{plans}.$$

3 Operational Semantics for AgentSpeak(L)

An AgentSpeak(L) *configuration* C is a tuple $C = \langle I, E, A, R, Ap, \iota, \rho, \varepsilon \rangle$ where:

- I is a set of *intentions* $\{i, i', \ldots\}$. Each intention i is a stack of partially instantiated plans $[p_1 \ddagger p_2 \ldots p_n]$. We use the symbol $\ddagger$ to separate the plans in an intention.
- E is a set of *events* $\{(te, i), (te', i'), \ldots\}$. Each event is a pair (te, i), where te is a triggering event and the intention i has the plans associated with te.
- A is a set of *actions*.
- R is a set of *relevant plans*. In Definition 1 bellow we state precisely how the set of relevant plans is obtained.
- Ap is a set of *applicable plans*. The way this set is obtained is given in Definition 2 bellow.
- Each configuration C also has three components called ι, ε, and ρ. They keep a record of a particular intention, event and applicable plan (respectively) being considered along the execution of an agent.

A plan is considered relevant in relation to a triggering event if it has been written to deal with that event. In practice, that is verified by trying to unify the triggering event part of the plan with the triggering event that has been selected from E for treatment. In the definition bellow we write mgu for the unifying procedure that computes the most general substitution of two triggering events.

Definition 1 *Given the plans of an agent and a triggering event te, the set* $\mathsf{RelPlans}(plans, te)$ *of relevant plans is given as follows:*

$$\mathsf{RelPlans}(plans, te) = \{p\sigma \mid p \in plans \ \wedge \ \sigma = \mathsf{mgu}(te, \mathsf{TrEv}(p))\}$$

A plan of an agent is applicable if it is both relevant and its context is a logical consequence of the agent's beliefs.

Definition 2 *Given a set of relevant plans R and the beliefs of an agent, the set of applicable plans* $\mathsf{AppPlans}(R, beliefs)$ *is defined as follows:*

$$\mathsf{AppPlans}(R, beliefs) = \{p\theta \mid p \in R \ \wedge \ \theta \text{ is s. t. } beliefs \models \mathsf{Ctxt}(p)\theta\}$$

An agent can also perform a test goal. The evaluation of a test goal $?\varphi$ consists in testing if the formula φ is a logical consequence of the agent's beliefs. One of the effects of this test is the production of a set of substitutions:

Definition 3 *Given the beliefs of an agent and a formula φ the set of substitutions* $\mathsf{Test}(beliefs, \varphi)$ *produced by testing φ against beliefs is defined as follows:*

$$\mathsf{Test}(beliefs, \varphi) = \{\theta \mid beliefs \models \varphi\theta\}.$$

We now present the semantic rules for AgentSpeak(L). In order to keep the rules neater we adopt the following notations:

- If C is an AgentSpeak(L) configuration, we write C_E to make reference to the component E of C. Similarly for all the other components of C.
- We write $C_\iota = \_$ (the underline symbol) to indicate that there is no intention being considered in the agent's execution. Similarly for C_ρ and C_ϵ.
- We write $i[p]$ to denote the intention that has plan p on its top.

We also define some auxiliary syntactic functions to be used in the semantics. If p is a plan of the form $t : \varphi \leftarrow h$, we define:

- $\mathsf{TrEv}(p) = t$ (returns the triggering event of p)
- $\mathsf{Ctxt}(p) = \varphi$ (returns the context of p)

We have organised the presentation of the semantics in groups of related rules. We next discuss each of these groups.

Event selection: the rule below assumes the existence of a selection function S_E that selects events from a set of events E. The selected event is removed from E and it is assigned to the ϵ component of the configuration.

$$\textbf{SelEv} \quad \frac{S_E(C_E) = (te, i)}{C, \mathit{beliefs} \longrightarrow C', \mathit{beliefs}} \qquad C_\varepsilon = \_, \quad C_{Ap} = C_R = \{\}$$

$$\mathit{where:} \quad \begin{array}{rcl} C'_E & = & C_E - (te, i) \\ C'_\varepsilon & = & (te, i) \end{array}$$

Relevant plans: the rule $\mathbf{Rel_1}$ initialises the R component with the set of relevant plans. If no plan is relevant, the event is discarded from ϵ by $\mathbf{Rel_2}$.

$$\mathbf{Rel_1} \quad \frac{\mathsf{RelPlans}(\mathit{plans}, te) \neq \{\}}{C, \mathit{beliefs} \longrightarrow C', \mathit{beliefs}} \qquad C_\varepsilon = (te, i), \quad C_{Ap} = C_R = \{\}$$

$$\mathit{where:} \quad C'_R = \mathsf{RelPlans}(\mathit{plans}, te)$$

$$\mathbf{Rel_2} \quad \frac{\mathsf{RelPlans}(\mathit{plans}, te) = \{\}}{C, \mathit{beliefs} \longrightarrow C', \mathit{beliefs}} \qquad C_\varepsilon = (te, i), \quad C_{Ap} = C_R = \{\}$$

$$\mathit{where:} \quad C'_\varepsilon = \_$$

Applicable plans: the rule $\mathbf{Appl_1}$ initialises the Ap component with the set of applicable plans. If no plan is applicable, the event is discarded from ϵ by $\mathbf{Appl_2}$. In either case the relevant plans are also discarded.

$$\mathbf{Appl_1} \quad \frac{\mathsf{AppPlans}(C_R, \mathit{beliefs}) \neq \{\}}{C, \mathit{beliefs} \longrightarrow C', \mathit{beliefs}} \qquad C_\varepsilon \neq \_, \quad C_{Ap} = \{\}, C_R \neq \{\}$$

$$\mathit{where:} \quad \begin{array}{rcl} C'_R & = & \{\} \\ C'_{Ap} & = & \mathsf{AppPlans}(C_R, \mathit{beliefs}) \end{array}$$

$$\textbf{Appl}_2 \quad \frac{\mathsf{AppPlans}(C_R, beliefs) = \{\}}{C, beliefs \longrightarrow C', beliefs} \qquad C_\varepsilon \neq \_, \quad C_{Ap} = \{\}, C_R \neq \{\}$$

$$\begin{array}{rrcl} \textit{where:} & C'_R & = & \{\} \\ & C'_\varepsilon & = & \_ \end{array}$$

Selection of applicable plan: this rule assumes the existence of a selection function S_{Ap} that selects a plan from a set of applicable plans Ap. The plan selected is then assigned to the ρ component of the configuration and the set of applicable plans is discarded.

$$\textbf{SelApplic} \quad \frac{S_{Ap}(C_{Ap}) = p}{C, beliefs \longrightarrow C', beliefs} \qquad C_\varepsilon \neq \_, \quad C_{Ap} \neq \{\}$$

$$\begin{array}{rrcl} \textit{where:} & C'_\rho & = & p \\ & C'_{Ap} & = & \{\} \end{array}$$

Preparing the set of intentions: events can be classified as external or internal (depending one whether they were generated from the agent's perception, or whether they were generated by the previous execution of other plans, respectively). Rule **ExtEv** says that if the event ϵ is external (that is indicated by T in the intention associated to ϵ) a new intention is created and its single plan is the plan p annotated in the ρ component. If the event is internal, rule **IntEv** says that the plan in ρ should be put on top of the intention associated with the event. Either way, both the event and the plan can be discarded from the ϵ and ι components, respectively.

$$\textbf{ExtEv} \quad \frac{}{C, beliefs \longrightarrow C', beliefs} \qquad C_\varepsilon = (te, \mathsf{T}), \quad C_\rho = p$$

$$\begin{array}{rrcl} \textit{where:} & C'_I & = & C_I \cup \{\ [p]\ \} \\ & C'_\varepsilon & = & \_ \\ & C'_\rho & = & \_ \end{array}$$

$$\textbf{IntEv} \quad \frac{}{C, beliefs \longrightarrow C', beliefs} \qquad C_\varepsilon = (te, i), \quad C_\rho = p$$

$$\begin{array}{rrcl} \textit{where:} & C'_I & = & C_I \cup \{\ i[p]\ \} \\ & C'_\varepsilon & = & \_ \\ & C'_\rho & = & \_ \end{array}$$

Note that, in rule **IntEv**, the whole intention i that generated the internal event needs to be inserted back in C_I, with p on its top. This is related to suspended intentions and will be explained further when we present rule **Achieve**.

Selection of intention: this rule uses a function that selects an intention (that is, a stack of plans) for processing.

$$\textbf{SelInt} \quad \frac{S_I(C_I) = i}{C, \mathit{beliefs} \longrightarrow C', \mathit{beliefs}} \qquad C_\iota = \_$$

$$\mathit{where:} \quad C'_\iota \;=\; i$$

Executing the body of plans: this group of rules expresses the effects of executing the body of plans. The plan being executed is always the one on the top of the intention that has been previously selected. Observe that all the rules in this group discard the intention ι. After that, another intention can be eventually selected. We discuss each of the rules in this group in turn, according to formula that is selected (the one in the beginning of the plan on the top of the intention):

Basic Actions – the action a on the body of the plan is added to the set of actions A. The action is removed from the body of the plan and the intention is updated to reflect this removal.

$$\textbf{Action} \quad \frac{}{C, \mathit{beliefs} \longrightarrow C', \mathit{beliefs}} \qquad C_\iota = i[\mathit{head} \texttt{ <- } a\,; h']$$

$$\begin{array}{lrcl} \mathit{where:} & C'_\iota & = & \_ \\ & C'_A & = & C_A \cup \{a\} \\ & C'_I & = & (C_I - \{C_\iota\}) \cup \{i[\mathit{head} \texttt{ <- } h']\} \end{array}$$

Achievement Goals – this rule register a new internal event in the set of events E. This event can then be eventually selected (see rule **SelEv**).

$$\textbf{Achieve} \quad \frac{}{C, \mathit{beliefs} \longrightarrow C', \mathit{beliefs}} \qquad C_\iota = i[\mathit{head} \texttt{ <- } !at; h]$$

$$\begin{array}{lrcl} \mathit{where:} & C'_\iota & = & \_ \\ & C'_E & = & C_E \cup \{(+!at, C_\iota)\} \\ & C'_I & = & (C_I - \{C_\iota\}) \end{array}$$

Note how the intention that generated the internal event is removed from the set of intentions C_I. This denotes the idea of *suspended intentions*. If the formula being executed in a plan is an achievement goal, a plan (i.e., an intended means) for that goal needs to be pushed on top of that intention. Only when that plan is finished, execution can be resumed to the point after the achievement goal in the previous plan. This is the reason why rule **IntEv** inserts back in C_I the whole intention with the new plan on its top.

Test Goals – these rules are used when a test goal $?\varphi$ should be executed. Both rules try to produce a set of substitutions that can make φ a logical consequence of the agent's beliefs. The rule $\mathbf{Test_1}$ says basically that nothing is done if no substitution is found, and the rule $\mathbf{Test_2}$ says that one of the substitutions is applied to the plan.

$$\mathbf{Test_1} \quad \frac{\mathsf{Test}(\mathit{beliefs}, \varphi) = \{\}}{C, \mathit{beliefs} \longrightarrow C', \mathit{beliefs}} \qquad C_\iota = i[\mathit{head} \texttt{ <- } ?\varphi; h]$$

$$\mathit{where:} \quad \begin{array}{lcl} C'_\iota & = & \_ \\ C'_I & = & (C_I - \{C_\iota\}) \cup \{i[\mathit{head} \texttt{ <- } h]\} \end{array}$$

$$\mathbf{Test_2} \quad \frac{\mathsf{Test}(\mathit{beliefs}, \varphi) \neq \{\}}{C, \mathit{beliefs} \longrightarrow C', \mathit{beliefs}} \qquad C_\iota = i[\mathit{head} \texttt{ <- } ?\varphi; h]$$

$$\mathit{where:} \quad \begin{array}{lcl} C'_\iota & = & \_ \\ C'_I & = & (C_I - \{C_\iota\}) \cup \{i[(\mathit{head} \texttt{ <- } h)\theta]\} \quad \theta \in \mathsf{Test}(\mathit{beliefs}, \varphi) \end{array}$$

Updating Beliefs – the rule **AddBel** simply adds a new event to the set of events E. The goal $+at$ is removed from the body of the plan and the set of intentions is updated properly. The rule **DelBel** works similarly. In both rules, the set of beliefs of the agent should be modified in a way that either the predicate at follows from the new set of beliefs (rule **AddBel**) or it does not (rule **DelBel**).

$$\mathbf{AddBel} \quad \frac{}{C, \mathit{beliefs} \longrightarrow C', \mathit{beliefs}'} \qquad C_\iota = i[\mathit{head} \texttt{ <- } +at; h]$$

$$\mathit{where:} \quad \begin{array}{lcl} C'_\iota & = & \_ \\ \mathit{beliefs}' & \models & at \\ C'_E & = & C_E \cup \{(+at, C_\iota)\} \\ C'_I & = & (C_I - \{C_\iota\}) \cup \{i[\mathit{head} \texttt{ <- } h']\} \end{array}$$

$$\textbf{DelBel}\quad \frac{}{C, beliefs \longrightarrow C', beliefs'} \qquad C_\iota = i[head \texttt{<-}\ -at; h']$$

$$\begin{array}{lllll} where: & C'_\iota & = & \_ \\ & beliefs' & \not\models & at \\ & C'_E & = & C_E \cup \{(-at, C_\iota)\} \\ & C'_I & = & (C_I - \{C_\iota\}) \cup \{i[head \texttt{<-}\ h']\} \end{array}$$

Any formula in the body of a plan fits in one of the cases above. There is only two final rules to conclude the semantics.

Removing intentions: the two rules below can be seen as *clearing house* rules. The rule $\textbf{ClearInt}_\textbf{1}$ simply removes an intention from the set of intentions of an agent when there is nothing left (goal or action) in that intention. The rule $\textbf{ClearInt}_\textbf{2}$ removes from the intention what is left from the plan that had been put on the top of the intention on behalf of the achievement goal $!at$ (which is also removed as it has been accomplished).

$$\textbf{ClearInt}_\textbf{1}\quad \frac{}{C, beliefs \longrightarrow C', beliefs} \qquad C_\iota = [head \texttt{<-}\ ]$$

$$\begin{array}{lllll} where: & C'_\iota & = & \_ \\ & C'_I & = & C_I - \{C_\iota\} \end{array}$$

$$\textbf{ClearInt}_\textbf{2}\quad \frac{}{C, beliefs \longrightarrow C', beliefs} \qquad C_\iota = i'[head' \texttt{<-}\ !at; h'\ \ddagger\ head \texttt{<-}\ ]$$

$$\begin{array}{lllll} where: & C'_\iota & = & \_ \\ & C'_I & = & (C_I - \{C_\iota\}) \cup \{i'[head' \texttt{<-}\ h']\} \end{array}$$

4 Conclusion

Agent oriented programming and agent oriented software engineering are becoming widely recognised as important contributions of the Multi-Agent Systems research community to the conception and the implementation of the next generation computing systems. Agent oriented programming languages are, however, in their infancy. So far, they have been mostly considered within the multi-agent systems community, and at a conceptual level. This means that they lack both good formalisations and/or implementations. In our research, we have tackled both aspects: this paper is an initial contribution to the former, and we dealt with the latter in [1, 6].

Although this paper provides only a new start to the formal aspects of AgentSpeak(L), it sets the ground for important theoretical work in the area of agent oriented programming languages. In effect, we presented here the first complete formalisation of the basic AgentSpeak(L) language given with traditional tools in formal semantics of programming languages; this has the advantage of allowing the use of well know techniques for proving properties of the language. This semantics will also be instrumental in varied future work, e.g., in comparing this language and its extensions (see below) with other programming languages for which structural operational semantics exist. In [4], it has been shown that 3APL [5] can simulate AgentSpeak(L); an operational semantics to extensions of AgentSpeak(L) will facilitate further comparison with 3APL as well as comparison with other agent oriented programming languages [11, 2].

Another use that we have planned for this operational semantics is to prove that the AgentSpeak(L) interpreters we have implemented are correct and complete in relation to it. Next, it would be interesting to use this formal semantics of AgentSpeak(L) to prove known properties of BDI system. This way, we would be pursuing the solution that Rao proposed for grounding implementations of BDI agents on the well studied BDI logics. The impossibility of doing so at present is frequently mentioned as one serious drawback in multi-agent systems practice [8].

In [1] we introduced an extension to AgentSpeak(L) called AgentSpeak(XL); that paper concentrated on one particular extension of the language that allows the automatic generation of efficient intention selection functions. Several other extensions of AgentSpeak(L) were mentioned in that paper, which we have already informally defined, and others are being designed. However, the formal specification of the semantics of these extensions is yet to be done. This planned work of giving formal semantics to such extensions can be firmly grounded on the operational semantics given in this paper.

References

[1] Rafael H. Bordini, Ana L. C. Bazzan, Rafael O. Jannone, Daniel M. Basso, Rosa M. Vicari, and Victor R. Lesser. AgentSpeak(XL): Efficient intention selection in BDI agents via decision-theoretic task scheduling. In *Proceedings of the First International Joint Conference on Au-*

tonomous Agents and Multi-Agent Systems (AAMAS-2002, featuring 6th AGENTS, 5th ICMAS and 9th ATAL), 15-19 July, 2002, Bologna, Italy, New York, NY, 2002. ACM Press. To appear.

[2] Giuseppe de Giacomo, Yves Lespérance, and Hector J. Levesque. ConGolog: A concurrent programming language based on the situation calculus. *Artificial Intelligence*, 121:109–169, 2000.

[3] Mark d'Inverno and Michael Luck. Engineering AgentSpeak(L): A formal computational model. *Journal of Logic and Computation*, 8(3):1–27, 1998.

[4] Koen V. Hindriks, Frank S. de Boer, Wiebe van der Hoek, and John-Jules Ch. Meyer. A formal embedding of agentspeak(l) in 3apl. In Grigoris Antoniou and John K. Slaney, editors, *Advanced Topics in Artificial Intelligence, 11th Australian Joint Conference on Artificial Intelligence (AI'98), Brisbane, Australia, 13–17 July, 1998 (Selected Papers)*, number 1502 in Lecture Notes in Computer Science, pages 155–166, Heidelberg, 1998. Springer-Verlag.

[5] Koen V. Hindriks, Frank S. de Boer, Wiebe van der Hoek, and John-Jules Ch. Meyer. Control structures of rule-based agent languages. In Jörg P. Müller, Munindar P. Singh, and Anand S. Rao, editors, *Intelligent Agents V—Proceedings of the Fifth International Workshop on Agent Theories, Architectures, and Languages (ATAL-98), held as part of the Agents' World, Paris, 4–7 July, 1998*, number 1555 in Lecture Notes in Artificial Intelligence, pages 381–396, Heidelberg, 1999. Springer-Verlag.

[6] Rodrigo Machado and Rafael H. Bordini. Running AgentSpeak(L) agents on SIM_AGENT. In *Pre-Proceedings of the Eighth International Workshop on Agent Theories, Architectures, and Languages (ATAL-2001), August 1–3, 2001, Seattle, WA*, 2001. Proceedings to appear as a volume of LNCS – Intelligent Agents Series.

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

[8] Anand S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In Walter Van de Velde and John Perram, editors, *Proceedings of the Seventh Workshop on Modelling Autonomous Agents*

in a Multi-Agent World (MAAMAW'96), 22–25 January, Eindhoven, The Netherlands, number 1038 in Lecture Notes in Artificial Intelligence, pages 42–55, London, 1996. Springer-Verlag.

[9] Anand S. Rao. Decision procedures for propositional linear-time belief-desire-intention logics. In Michael Wooldridge, Jörg P. Müller, and Milind Tambe, editors, *Intelligent Agents II—Proceedings of the Second International Workshop on Agent Theories, Architectures, and Languages (ATAL'95), held as part of IJCAI'95, Montréal, Canada, August 1995*, number 1037 in Lecture Notes in Artificial Intelligence, pages 33–48, Berlin, 1996. Springer-Verlag.

[10] Anand S. Rao and Michael P. Georgeff. BDI agents: From theory to practice. In Victor Lesser and Les Gasser, editors, *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS'95), 12–14 June, San Francisco, CA*, pages 312–319, Menlo Park, CA, 1995. AAAI Press / MIT Press.

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

[12] Munindar P. Singh, Anand S. Rao, and Michael P. Georgeff. Formal methods in DAI: Logic-based representation and reasoning. In Gerhard Weiß, editor, *Multiagent Systems—A Modern Approach to Distributed Artificial Intelligence*, chapter 8, pages 331–376. MIT Press, Cambridge, MA, 1999.

[13] Aaron Sloman and Riccardo Poli. SIM_AGENT: A toolkit for exploring agent designs. In Michael Wooldridge, Jörg P. Müller, and Milind Tambe, editors, *Intelligent Agents II—Proceedings of the Second International Workshop on Agent Theories, Architectures, and Languages (ATAL'95), held as part of IJCAI'95, Montréal, Canada, August 1995*, number 1037 in Lecture Notes in Artificial Intelligence, pages 392–407, Berlin, 1996. Springer-Verlag.

[14] J. Michael Spivey. *The Z Notation: A Reference Manual*. Prentice Hall, Hemel Hempstead, second edition, 1992.

[15] Michael J. Wooldridge, Nicholas R. Jennings, and David Kinny. The Gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.