

Extending the Operational Semantics of a BDI Agent-Oriented Programming Language for Introducing Speech-Act Based Communication

Álvaro F. Moreira¹, Renata Vieira², and Rafael H. Bordini³

¹ Departamento de Informática Teórica
Instituto de Informática
Universidade Federal do Rio Grande do Sul
Porto Alegre RS, 91501-970, Brazil
`afmoreira@inf.ufrgs.br`

² Centro de Ciências Exatas
Universidade do Vale do Rio dos Sinos
CP 275, CEP 93022-000, São Leopoldo, RS, Brazil
`renata@exatas.unisinos.br`

³ Department of Computer Science,
University of Liverpool,
Liverpool L69 7ZF, U.K.
`R.Bordini@csc.liv.ac.uk`

Abstract. Work on agent communication languages has since long striven to achieve adequate speech act semantics; partly, the problem is that references to an agent’s architecture (in particular a BDI-like architecture) would be required in giving such semantics more rigorously. On the other hand, BDI agent-oriented programming languages have had their semantics formalised for an abstract versions only, neglecting “practical” aspects such as communication primitives; this means that, at least in what concerns communication, implementations of BDI programming languages have been *ad hoc*. This paper tackles, however preliminarily, both these problems by giving semantics to speech-act based messages received by an AgentSpeak(L) agent. AgentSpeak(L) is a BDI, agent-oriented, logic programming language that has received a great deal of attention in recent years. The work in this paper builds upon a structural operational semantics to AgentSpeak(L) that we have given in previous work. The contribution of this paper is two-fold: we here extend our earlier work on providing a solid theoretical background on which to base existing implementations of AgentSpeak(L) interpreters, as well as we shed light on a more computationally grounded approach to giving semantics for (the core) illocutionary forces used in speech-act based agent communication languages.

1 Introduction

The AgentSpeak(L) programming language was introduced by Rao in [15]. The language was quite influential in the definition of other agent-oriented programming languages. After a period of apparent disinterest, work related to AgentSpeak(L) has been done in many fronts recently, from developing interpreters [3] and applications to formal semantics and model-checking [7,4]. AgentSpeak(L) is particularly interesting, in comparison to other agent-oriented languages, in that it retains the most important aspects of the BDI-based reactive planning systems on which it was based, it has a working interpreter, and at the same time its formal semantics and relation to BDI logics [16] is being thoroughly studied [14,5,6].

In our previous work on giving operational semantics to AgentSpeak(L) [14], we considered only the main constructs of the language, as originally defined by Rao. However, Rao's definition considered an abstract programming language; for AgentSpeak(L) to be useful in practice, various extensions to it have been proposed [3]. Still, the interpreter described in that paper does not yet support communication, which is essential when it comes to engineering *multi-agent systems*. Speech act theory (see Section 3) is particularly adequate as a foundation for communication among intentional agents. Through communication, an agent can share its internal state (beliefs, desires, intentions) with other agents, as well as it can influence other agents' states. Speech-act based communication for AgentSpeak(L) agents has already been used, in a very simple way (so as to allow model checking), in [4].

This paper deals exactly with this aspect of AgentSpeak(L), by extending its operational semantics to account for the main illocutionary forces related to communicating AgentSpeak(L) agents. Our semantics tells exactly how to implement the processing of messages received by an AgentSpeak(L) agent (how its computational representation of Beliefs-Desires-Intentions are changed when a message is received). Note that in implementations of the BDI architecture, the concept of *plan* is used to simplify aspects of deliberation and knowing what course of action to take in order to achieve desired states of the world. Therefore, an AgentSpeak(L) agent *sends* a message whenever a communicative action appears in the body of an intended plan that is being executed. The important issue is then how to interpret a message that has been *received*. This is precisely the aspect of agent communication that we consider in this paper.

In extending the operational semantics of AgentSpeak(L) to account for inter-agent communication, we also touch upon another long-standing problem in the area of multi-agent systems, namely the semantics of speech acts. As Singh pointed out [19], semantics of agent communication languages such as KQML and FIPA have incurred in the mistake (for those concerned with general interoperability and legacy systems) of emphasising *mental agency*. The problem is that if the semantics makes reference to an agent believing (or intending a state satisfying) a certain proposition, there is no way to ensure that any software using that communication language complies with the expected underlying semantics of belief (or intention, or mental attitudes in general). This problem is

avoided here as the AgentSpeak(L) agents for which such speech-act based communication language is being used, are indeed agents for which a precise notion of Belief-Desire-Intention has been given [5,6]. Another way of putting this, is that we give a more “computationally grounded” [21] semantics of speech-act based agent communication.

Previous attempts to give semantics to agent communication languages, e.g. [12], were based on the “pre-conditions – action – post-conditions” approach, referring to agent mental states in modal languages based on Cohen and Levesque’s work on intention [10]. Our semantics for communication, besides being more instrumental in implementations (as it serves as the specification for an interpreter), can also be used in the proof of communication properties [23]. More recently, the work reported in [11,20] also provides an operational semantics for an agent communication language. However, again they do not consider the effects of communication in terms of BDI agents. To the best of our knowledge, our work is the first to give operational semantics incorporating the core illocutionary forces in a BDI programming language.

The paper is organised as follows. Section 2 gives a very brief (informal) overview of AgentSpeak(L), and Section 3 provides the general background on speech-act based agent communication. We then present formally the syntax and semantics of AgentSpeak(L) in Section 4; this is the syntax and semantics of AgentSpeak(L) first presented in [14], and reproduced here so that the paper is self-contained. Section 5 provides the main contribution in this paper, i.e., the semantics of speech-act based communication for AgentSpeak(L) agents. Brief conclusions and plans for future work are given in the last section.

2 An Overview of AgentSpeak(L)

The AgentSpeak(L) programming language was introduced in [15]. It is a natural extension of logic programming for the BDI agent architecture, and provides an elegant abstract framework for programming BDI agents. The BDI architecture is, in turn, the predominant approach to the implementation of “intelligent” or “rational” agents [22]. An AgentSpeak(L) agent is created by the specification of a set of base beliefs and a set of plans. A *belief atom* is simply a first-order predicate in the usual notation, and belief atoms or their negations are termed *belief literals*. An *initial set of beliefs* is just a collection of ground belief atoms.

AgentSpeak(L) distinguishes two types of goals: *achievement goals* and *test goals*. Achievement and test goals are predicates (as for beliefs) prefixed with operators ‘!’ and ‘?’ respectively. Achievement goals state that the agent wants to achieve a state of the world where the associated predicate is true. (In practice, these initiate the execution of *subplans*.) A *test goal* returns a unification for the associated predicate with one of the agent’s beliefs; they fail otherwise. A *triggering event* defines which events may initiate the execution of a plan. An *event* can be internal, when a subgoal needs to be achieved, or external, when generated from belief updates as a result of perceiving the environment. There

are two types of triggering events: those related to the *addition* ('+') and *deletion* ('-') of mental attitudes (beliefs or goals).

Plans refer to the *basic actions* that an agent is able to perform on its environment. Such actions are also defined as first-order predicates, but with special predicate symbols (called action symbols) used to distinguish them from other predicates. A plan is formed by a *triggering event* (denoting the purpose for that plan), followed by a conjunction of belief literals representing a *context*. The context must be a logical consequence of that agent's current beliefs for the plan to be *applicable*. The remainder of the plan is a sequence of basic actions or (sub)goals that the agent has to achieve (or test) when the plan, if applicable, is chosen for execution.

```

+concert(A,V) : likes(A)
  ← !book_tickets(A,V).

+!book_tickets(A,V) : ¬busy(phone)
  ← call(V);
  ...;
  !choose_seats(A,V).

```

Fig. 1. Examples of AgentSpeak(L) Plans

Figure 1 shows some examples of AgentSpeak(L) plans. They tell us that, when a concert is announced for artist *A* at venue *V* (so that, from perception of the environment, a belief `concert(A,V)` is *added*), then if this agent in fact likes artist *A*, then it will have the new goal of booking tickets for that concert. The second plan tells us that whenever this agent adopts the goal of booking tickets for *A*'s performance at *V*, if it is the case that the telephone is not busy, then it can execute a plan consisting of performing the basic action `call(V)` (assuming that making a phone call is an atomic action that the agent can perform) followed by a certain protocol for booking tickets (indicated by '...'), which in this case ends with the execution of a plan for choosing the seats for such performance at that particular venue.

3 Background on Speech-Act Based Agent Communication Languages

As BDI theory is based on the philosophical literature on practical reasoning [8], agent communication in multi-agent systems is inspired by philosophical studies on the speech act theory, in particular the work of Austin [1] and Searle [17].

Speech act theory is based on the conception of language as action, and different types of speech actions are explained according to their illocutionary force, which represents the speaker's intention for a certain semantic content.

In natural language, one has an illocutionary force associated to a utterance (or locutionary act) such as “the door is open” and another to a utterance “open the door”. The former intends belief revision, whereas the latter intends a change in the plans of the hearer. As seen in the utterances above, in natural language the illocutionary force is implicit. When the theory is adapted to agent communication the illocutionary force is made explicit, to facilitate the computational processing of the communication act.

Other pragmatical factors related to communication such as social roles and conventions have been discussed in the literature [13,2,18]. Illocutionary forces may require the existence of certain relationships between speaker and hearer for them to be felicitous. A *command*, for instance, requires a subordination relation between the individuals involved in the communication, whereas such subordination is not required in a *request*. To some extent, this has been made explicit in the semantics of communicating AgentSpeak(L) agents that we give in Section 5. We require that an agent specification provides “trust” and “power” relations to other agents, as the semantics of illocutionary forces in messages being processed by that agent depends on them. This is a simple mechanism, which should incorporate more elaborate conceptions of trust and power from the literature on the subject (see, e.g., [9]).

The “Agent Communication Language” developed in the context of the “Knowledge Sharing Effort” project, attempt to define a practical agent communication language that included high level (speech-act based) communication as proposed in the distributed artificial intelligence literature. The Knowledge Query and Manipulation Language (KQML) is a language that adds intentional context to communication messages [12]. KQML “performatives” refer to illocutionary forces and they make explicit the agent intentions with a message being sent.

In this paper, as a first attempt to give semantics to communication aspects of BDI programming languages, we focus on four illocutionary forces (which are particularly relevant for AgentSpeak(L) agents), named exactly as the associated KQML performatives; we also consider two new illocutionary forces, which we introduce in Section 5. The four KQML performatives which we consider in the semantics given in this paper are listed below, where S denotes the agent that sends the message, and R denotes the agent that receives the message.

tell: S informs R that the sentence in the message (i.e., the message content) is true of S — that is, the sentence is in the knowledge base of S (or S believes that the content of the message is true, for BDI-like agents);
untell: the message content is *not* in the knowledge base of S ;
achieve: S requests that R try to achieve a state of the world where the message content is true;
unachieve: S wants to revert the effect of an **achieve** previously sent.

A complete semantic representation for speech acts needs to consider action and planning theories, and to conceive communication as a special kind of action which has direct effect over the mental states of agents. Most formalisms used for

the representation of actions use the “pre-conditions – action – post-conditions” approach. For the description of mental states, most of the work in the area is based on the Cohen and Levesque’s work on intention [10].

In [12], an initial attempt to introduce semantic principles to KQML was made. The communication process is defined according to the influence that messages have on the agents’ beliefs and intentions. The semantics is given through the description of agent states before and after sending or receiving a message, in terms of pre- and post-conditions. Pre-conditions describe the required state for an agent to send a message and for the receiver to accept and process it. Post-conditions describe the interlocutors’ states after an utterance or after receiving and processing a message.

Agent states are described through mental attitudes whose arguments may be propositions or state of the world descriptions. Mental attitudes are, e.g., belief (*bel*), knowledge (*know*), desire (*want*), and intention (*intend*). In the style introduced in [12], the semantics for $tell(S, R, X)$ (S tells R that S believes that X is true), for instance, is given as:

- Pre-conditions on the states of S and R :
 - $Pre(S): bel(S, X) \wedge know(S, want(R, know(R, bel(S, X))))$
 - $Pre(R): intend(R, know(R, bel(S, X)))$
- Post-conditions on S and R :
 - $Pos(S): know(S, know(R, bel(S, X)))$
 - $Pos(R): know(R, bel(S, X))$
- Action completion:
 - $know(R, bel(S, X))$

Post-condition and *completion* hold unless a *sorry* or *error* are returned (i.e., when R was unable to process the *tell* message).

As we mentioned in the introduction, the problem with the usual approach to giving semantics to an agent communication language is that it makes reference to an agent’s mental attitudes, and there is no way to ensure that any software in general, using that communication language, complies with the expected underlying semantics of the mental attitudes. This is true of the semantic approaches to both KQML and FIPA (see [19] for a discussion). As an example, consider a legacy software wrapped in an agent that uses KQML or FIPA to interoperate with other agents. One could not prove communication properties of the system, as there is not such a thing as a precise definition of when that legacy system believes or intends a formula X , as appears in the semantics of the communication language. In our approach, it is possible to prove such properties given that in [5,6] we gave a precise definition of what it means for an AgentSpeak(L) agent to believe, desire, or intend a certain formula.

4 Syntax and Semantics of AgentSpeak(L)

This section presents the syntax and semantics of AgentSpeak(L), as originally in [14] and then used in [5,6]. The semantics is given in the style of Plotkin’s structural operational semantics, a standard notation for semantics of programming languages.

4.1 Abstract Syntax

An AgentSpeak(L) agent specification ag is given by the following grammar:

$$\begin{array}{ll}
ag & ::= bs \ ps \\
bs & ::= b_1 \dots b_n \quad (n \geq 0) \\
at & ::= P(t_1, \dots, t_n) \quad (n \geq 0) \\
ps & ::= p_1 \dots p_n \quad (n \geq 1) \\
p & ::= te : ct \leftarrow h \\
te & ::= +at \mid -at \mid +g \mid -g \\
ct & ::= at \mid \neg at \mid ct \wedge ct \mid \top \\
h & ::= a \mid g \mid u \mid h; h \\
a & ::= A(t_1, \dots, t_n) \quad (n \geq 0) \\
g & ::= !at \mid ?at \\
u & ::= +at \mid -at
\end{array}$$

In AgentSpeak(L), an agent is simply specified by a set bs of beliefs (the agent's initial belief base) and a set ps of plans (the agent's plan library). The atomic formulæ at of the language are predicates where P is a predicate symbol and t_1, \dots, t_n are standard terms of first order logic. We call a *belief* an atomic formula at with no variables and we use b as a metavariable for beliefs. The set of initial beliefs of an AgentSpeak(L) program is a sequence of beliefs bs .

A plan in AgentSpeak(L) is given by p above, where te is the *triggering event*, ct is the plan's context, and h is sequence of actions, goals, or belief updates; $te : ct$ is referred as the *head* of the plan, and h is its *body*. Then the set of plans of an agent is given by ps as a list of plans. Each plan has in its head a formula ct that specifies the conditions under which the plan can be executed. The formula ct must be a logical consequence of the agent's beliefs if the plan is to be considered applicable.

A triggering event te 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). A sequence h of actions, goals, and belief updates defines the body of a plan. We assume the agent has at its disposal a set of *actions* and we use a as a metavariable ranging over them. They are given as normal predicates except that an action symbol A is used instead of a predicate symbol. Goals g can be either *achievement goals* ($!at$) or *test goals* ($?at$). Finally, $+at$ and $-at$ (in the body of a plan) represent operations for updating (u) the belief base by, respectively, adding and removing at .

4.2 Semantics

An agent and its circumstance form a configuration of the transition system giving operational semantics to AgentSpeak(L). The transition relation:

$$\langle ag, C \rangle \longrightarrow \langle ag', C' \rangle$$

is defined by the semantic rules given in the next section.

An agent's circumstance C is a tuple $\langle I, E, A, R, Ap, \iota, \rho, \varepsilon \rangle$ where:

- I is a set of *intentions* $\{i, i', \dots\}$. Each intention i is a stack of partially instantiated plans.
- E is a set of *events* $\{(te, i), (te', i'), \dots\}$. Each event is a pair (te, i) , where te is a triggering event and the plan on top of intention i is the one that generated te .
When the belief revision function, which is not part of the AgentSpeak(L) interpreter but rather of the general architecture of the agent, updates the belief base, the associated events are included in this set.
- A is a set of *actions* to be performed in the environment.
An action expression included in this set tells other architecture components to actually perform the respective action on the environment, thus changing it.
- R is a set of *relevant plans*. In Definition 1 below 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 below.
- Each circumstance C also has three components called ι , ε , and ρ . They keep record of a particular intention, event and applicable plan (respectively) being considered along the execution of an agent.

Auxiliary Functions

We define some auxiliary syntactic functions to be used in the semantics. If p is a plan of the form $te : ct \leftarrow h$, we define $\text{TrEv}(p) = te$ and $\text{Ctx}(p) = ct$, which retrieve the triggering event and the context of the plan, respectively. We use these to define the auxiliary functions below, which will be needed in the semantic rules.

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 below, we write mgu for the procedure that computes the most general unifying substitution of two triggering events.

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

$$\text{RelPlans}(ps, te) = \{p\theta \mid p \in ps \wedge \theta = \text{mgu}(te, \text{TrEv}(p))\}.$$

A plan 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 bs of an agent, the set of applicable plans $\text{AppPlans}(bs, R)$ is defined as follows:*

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

An agent can also perform a test goal. The evaluation of a test goal $?at$ consists in testing if the formula at 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 bs of an agent and a formula at , the set of substitutions $\text{Test}(bs, at)$ produced by testing at against bs is defined as follows:

$$\text{Test}(bs, at) = \{\theta \mid bs \models at\theta\}.$$

Notation

In order to keep the semantic rules neat, we adopt the following notations:

- If C is an AgentSpeak(L) agent circumstance, we write C_E to make reference to the component E of C . Similarly for all the other components of C .
- We write $C_i = _$ (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 use i, i', \dots to denote intentions, and we write $i[p]$ to denote an intention that has plan p on its top, i being the remaining plans in that intention.

We use the following notation for AgentSpeak(L) selection functions: S_E for the event selection function, S_{Ap} for the applicable plan selection function, and S_I for the intention selection function.

The semantic rules are given in Appendix A. They appeared in [14,5], but we include them here so that the paper is self-contained.

5 Semantics of Communicating AgentSpeak(L) Agents

In order to endow AgentSpeak(L) agents with the capability of processing communication messages, we first change the syntax of atomic propositions so that we can annotate, for each belief, what is its source. This annotation mechanism provides a very neat notation for making explicit the sources of an agent’s belief. It has advantages in terms of expressive power and readability, besides allowing the use of such explicit information in an agent’s reasoning (i.e., in selecting plans for achieving goals).

By using this information source annotation mechanism, we also clear up some practical problems in the implementation of AgentSpeak(L) interpreters concerning internal beliefs (the ones added during the execution of a plan). In the interpreter reported in [3], we temporarily dealt with the problem by creating a separate belief base where the internal beliefs are included or removed; this extra belief base, together with the current list of percepts, is then used in the belief revision process.

The following new grammar rule is used instead of the one given in Section 4, so that we can annotate each *atomic proposition* with its source: either a term identifying which was the agent in the society (*id*) that previously sent the information in a message, **self** to denote internal beliefs, or **percept** to indicate that the belief was acquired through perception of the environment.

$$at ::= P(t_1, \dots, t_n)[an_1, \dots, an_m]$$

where $n \geq 0$, $m > 0$, and $an_i \in \{\text{percept}, \text{self}, \text{id}\}$, $0 \leq i \leq m$.

Note that with this new language construct, it is possible to make sure, in a plan context, what was the source of a belief before using that plan as an intended means. All of these details that are consequence of the new syntax that we introduce here are in fact hidden in the `mgu` function used in the auxiliary functions of the semantics in Section 4, as well as in the logical consequence relation that is also referred to in the semantics. We intend to make such details more clear in future work.

Next, we need to change the definition of an agent’s circumstance, as we now need a set M which represents an agent’s mail box. As usual in practice (and used, e.g., in [4]), we assume that the implementation of the AgentSpeak(L) interpreter provides, as part of the overall agent architecture, a mechanism for receiving and sending messages asynchronously; messages are stored in an mail box and one of them is processed by the agent at the beginning of a reasoning cycle. The format of those messages is $\langle Ilf, id, content \rangle$, where $Ilf \in \{Tell, Untell, Achieve, Unachieve, TellHow, UntellHow\}$ is the illocutionary force associated with the message, id identifies the agent that sent the message (as mentioned above), and $content$ is the message content, which can be either an atomic proposition (at) or a plan (p).

An agent’s circumstance C is now defined as a tuple $\langle I, E, M, A, R, Ap, \iota, \rho, \varepsilon \rangle$ where M is the set of messages that the agent has received and has not processed yet, and everything else is as in Section 4. For processing messages, a new selection function is necessary, which operates in the same way as the three selection functions described in the previous section as well. The new selection function is called S_M , and selects one particular message from M .

Further, in processing messages we now need two more “given” functions, in the same way that the selection functions are assumed as given in an agent’s specification. $\text{Trust}(id)$ returns true if id identifies a trusted information source. It is used to decide whether *Tell* messages will be processed at all. Even though we annotate the information source when a belief is acquired from communication, the agent may ignore messages arriving from untrusted agents. $\text{Power}(id)$, is true if the agent has a subordination relation towards agent id . In that case, messages of type *Achieve* should be respected. The idea of having user-defined “trust” and “power” functions has already been used in practice in [4].

We now extend the semantics to cope with the processing of speech-act based messages received by an AgentSpeak(L) agent. The new semantic rules are as follows.

Receiving a Tell Message

$$\text{TellRec} \frac{S_M(C_M) = \langle Tell, id, at \rangle \quad \text{Trust}(id)}{\langle ag, C \rangle \longrightarrow \langle ag', C' \rangle}$$

$$\begin{aligned} \text{where: } C'_M &= C_M - \{ \langle Tell, id, at \rangle \} \\ bs' &\models \begin{cases} at[sources \cup \{id\}], & \text{if } bs \models at[sources] \\ at[id], & \text{otherwise} \end{cases} \\ C'_E &= C_E \cup \{ \langle +at[id], \top \rangle \} \end{aligned}$$

The content of the message is added to the belief base in case it was not there previously. If the information is already there, the sender of the message is included in the set of sources giving accreditation to that belief.

Receiving an Untell Message

$$\text{UnTellRec} \frac{S_M(C_M) = \langle \text{Untell}, id, at \rangle \quad \text{Trust}(id)}{\langle ag, C \rangle \longrightarrow \langle ag', C' \rangle}$$

$$\text{where: } C'_M = C_M - \{\langle \text{Untell}, id, at \rangle\}$$

$$bs' \not\models at[id], \quad \text{if } bs \models at[id]$$

$$bs' \models at[sources - \{id\}], \quad \text{if } bs \models at[sources]$$

$$C'_E = C_E \cup \{\langle -at[id], \top \rangle\}$$

The sender of the message is removed from the set of sources giving accreditation to the belief. If the sender was the only source for that information, the belief is removed from the belief base.

Receiving an Achieve Message

$$\text{AchieveRec} \frac{S_M(C_M) = \langle \text{Achieve}, id, at \rangle \quad \text{Power}(id)}{\langle ag, C \rangle \longrightarrow \langle ag, C' \rangle}$$

$$\text{where: } C'_M = C_M - \{\langle \text{Achieve}, id, at \rangle\}$$

$$C'_E = C_E \cup \{\langle +!at, \top \rangle\}$$

If the sender has power over the AgentSpeak(L) agent, the agent will try to execute a plan whose triggering event is $+!at$; that is, it will try to achieve the goal associated with the propositional content of the message. All that needs to be done is to include an external event in the set of events (recall that external events have the triggering event associated with the true intention \top).

Note that, interestingly, now it is possible to have a new focus of attention (each of the stacks of plans in the set of intentions I) being started by an addition (or deletion, see below) of an achievement goal. Originally, only a change of belief from perception of the environment started a new focus of attention; the plan chosen for that event could, in turn, have achievement goals in its body, thus pushing new plans onto the stack.

Receiving an Unachieve Message

$$\text{UnAchieveRec} \frac{S_M(C_M) = \langle \text{Unachieve}, id, at \rangle \quad \text{Power}(id)}{\langle ag, C \rangle \longrightarrow \langle ag, C' \rangle}$$

$$\text{where: } C'_M = C_M - \{\langle \text{Unachieve}, id, at \rangle\}$$

$$C'_E = C_E \cup \{\langle -!at, \top \rangle\}$$

Similarly to the previous rule, except that now a deletion (rather than addition) of achievement goal is included in the set of events. If the agent has a

plan with such triggering event, that plan should handle all aspects of dropping an intention. However, doing so in practice may require the alteration of the set of intentions, thus requiring special mechanisms which are not available in AgentSpeak(L) as yet (neither formally, nor in implemented AgentSpeak(L) interpreters, to the best of our knowledge).

Receiving a Tell-How Message

$$\text{TellHowRec} \frac{S_M(C_M) = \langle \text{TellHow}, id, p \rangle \quad \text{Trust}(id)}{\langle ag, C \rangle \longrightarrow \langle ag', C' \rangle}$$

$$\text{where: } C'_M = C_M - \{\langle \text{TellHow}, id, p \rangle\}$$

$$ps' = ps \cup \{p\}$$

The concept of plans in reactive planning systems such as those defined by AgentSpeak(L) agents is associated with Singh's notion of know-how [18]. Accordingly, we use the *TellHow* performative when an external source wants to inform an AgentSpeak(L) agent of a plan it uses for handling certain types of events (given by the plan's triggering event). If the source is trusted, the plan (which is in the message content) is simply added to the agent's plan library.

Receiving an Untell-How Message

$$\text{UntellHowRec} \frac{S_M(C_M) = \langle \text{UntellHow}, id, p \rangle \quad \text{Trust}(id)}{\langle ag, C \rangle \longrightarrow \langle ag', C' \rangle}$$

$$\text{where: } C'_M = C_M - \{\langle \text{UntellHow}, id, p \rangle\}$$

$$ps' = ps - \{p\}$$

Similarly to the rule above. An external source may find that a plan is no longer valid, or efficient, for handling the events it was supposed to handle. It may then want to inform an AgentSpeak(L) agent of that fact. Thus, when receiving and *UntellHow*, the agent drops the associated plan (in the message content) from its plan library.

6 Conclusion

We have given formal semantics to the processing of speech-acted based messages received by an AgentSpeak(L) agent. The operational semantics we have used in previous work proved quite handy: in this extension, all we had to do (apart from minor changes in the syntax of the language and in the configuration of the transition system) was to provide new semantic rules, one for each illocutionary force used in the communication language. In giving semantics to communicating AgentSpeak(L) agents, we have provided the means for precise implementation of AgentSpeak(L) interpreters with such functionality, as well as given a more computationally grounded semantics of speech-act based agent communication.

If on one hand Singh's [19] proposal for a social-agency based semantics may be the best way towards giving semantics to general purpose agent communication languages such as FIPA or KQML, on the other hand, within the context of a BDI agent programming language, a mental agency approach to semantics of communication can be used without any of the drawbacks pointed out by Singh.

Future work should consider other performatives (in particular *AskIf*, *AskAll*, *Reply*, and for plans *AskHow* and *ReplyHow*), as well as giving a better formal treatment of information sources (that are annotated to atomic propositions) — unification and logical consequence of annotated atomic propositions was not formalised in this paper. Further communication aspects such as ontological agreement among AgentSpeak(L) agents, and reasoning about information sources (e.g., in executing test goals or choosing plans based on those annotations) should also be considered in future work. We also expect that this work will be used when speech-act based communication is implemented in existing AgentSpeak(L) interpreters.

References

1. J. L. Austin. *How to Do Things with Words*. Oxford University Press, London, 1962.
2. T. T. Ballmer and W. Brennenstuhl. *Speech Act Classification: A Study in the Lexical Analysis of English Speech Activity Verbs*. Springer-Verlag, Berlin, 1981.
3. R. H. Bordini, A. L. C. Bazzan, R. O. Jannone, D. M. Basso, R. M. Vicari, and V. R. Lesser. AgentSpeak(XL): Efficient intention selection in BDI agents via decision-theoretic task scheduling. In C. Castelfranchi and W. L. Johnson, editors, *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-2002), 15-19 July, Bologna, Italy*, pages 1294–1302, New York, NY, 2002. ACM Press.
4. R. H. Bordini, M. Fisher, C. Pardavila, and M. Wooldridge. Model checking AgentSpeak. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-2003), Melbourne, Australia, 14-18 July, 2003*. To appear.
5. R. H. Bordini and Á. F. Moreira. Proving the asymmetry thesis principles for a BDI agent-oriented programming language. In J. Dix, J. A. Leite, and K. Satoh, editors, *Proceedings of the Third International Workshop on Computational Logic in Multi-Agent Systems (CLIMA-02), 1st August, Copenhagen, Denmark*, Electronic Notes in Theoretical Computer Science 70(5). Elsevier, 2002. URL: <<http://www.elsevier.nl/locate/entcs/volume70.html>>. CLIMA-02 was held as part of FLoC-02. This paper was originally published in Datalogiske Skrifter number 93, Roskilde University, Denmark, pages 94–108.
6. R. H. Bordini and Á. F. Moreira. Proving BDI properties of agent-oriented programming languages: The asymmetry thesis principles in AgentSpeak(L). *Annals of Mathematics and Artificial Intelligence*, 2003. Accepted for publication in a Special Issue on Computational Logic and Multi-Agency.
7. R. H. Bordini, W. Visser, M. Fisher, C. Pardavila, and M. Wooldridge. Model checking multi-agent programs with CASP. In *Proceedings of the Fifteenth Conference on Computer-Aided Verification (CAV-2003), Boulder, CO, 8-12 July, 2003*. Tool description. To appear.

8. M. E. Bratman. *Intentions, Plans and Practical Reason*. Harvard University Press, Cambridge, MA, 1987.
9. C. Castelfranchi and R. Falcone. Principles of trust for MAS: Cognitive anatomy, social importance, and quantification. In Y. Demazeau, editor, *Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS'98), Agents' World, 4-7 July, Paris*, pages 72-79, Washington, 1998. IEEE Computer Society Press.
10. P. R. Cohen and H. J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42(3):213-261, 1990.
11. F. S. de Boer, R. M. van Eijk, W. Van Der Hoek, and J.-J. C. Meyer. Failure semantics for the exchange of information in multi-agent systems. In C. Palamidessi, editor, *Eleventh International Conference on Concurrency Theory (CONCUR 2000), University Park, PA, 22-25 August*, number 1877 in LNCS, pages 214-228. Springer-Verlag, 2000.
12. Y. Labrou and T. Finin. A semantics approach for KQML—a general purpose communication language for software agents. In *Proceedings of the Third International Conference on Information and Knowledge Management (CIKM'94)*. ACM Press, Nov. 1994. URL: <http://www.cs.umbc.edu/kqml/papers/>.
13. S. C. Levinson. The essential inadequacies of speech act models of dialogue. In H. Parret, M. Sbisà, and J. Verschuren, editors, *Possibilities and limitations of pragmatics: Proceedings of the Conference on Pragmatics at Urbino, July, 1979*, pages 473-492. Benjamins, Amsterdam, 1981.
14. A. F. Moreira and R. H. Bordini. An operational semantics for a BDI agent-oriented programming language. In *Proceedings of the Workshop on Logics for Agent-Based Systems (LABS-02), held in conjunction with the Eighth International Conference on Principles of Knowledge Representation and Reasoning (KR2002), April 22-25, Toulouse, France*, pages 45-59, 2002.
15. A. S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In W. Van de Velde and J. 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 LNAI, pages 42-55, London, 1996. Springer-Verlag.
16. A. S. Rao and M. P. Georgeff. Decision procedures for BDI logics. *Journal of Logic and Computation*, 8(3):293-343, 1998.
17. J. R. Searle. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, Cambridge, 1969.
18. M. P. Singh. *Multiagent Systems—A Theoretic Framework for Intentions, Know-How, and Communications*. Number 799 in LNAI. Springer-Verlag, Berlin, 1994.
19. M. P. Singh. Agent communication languages: Rethinking the principles. *IEEE Computer*, 31(12):40-47, December 1998.
20. R. M. van Eijk, F. S. de Boer, W. Van Der Hoek, and J.-J. C. Meyer. A verification framework for agent communication. *Autonomous Agents and Multi-Agent Systems*, 6(2):185-219, 2003.
21. M. Wooldridge. Computationally grounded theories of agency. In E. Durfee, editor, *Proceedings of the Fourth International Conference on Multi-Agent Systems (ICMAS-2000), 10-12 July, Boston*, pages 13-20, Los Alamitos, CA, 2000. IEEE Computer Society. Paper for an Invited Talk.
22. M. Wooldridge. *Reasoning about Rational Agents*. The MIT Press, Cambridge, MA, 2000.
23. M. Wooldridge. Semantic issues in the verification of agent communication languages. *Autonomous Agents and Multi-Agent Systems*, 3(1):9-31, 2000.

A Semantic Rules

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 circumstance.

$$\text{SelEv} \frac{S_E(C_E) = \langle te, i \rangle}{\langle ag, C \rangle \longrightarrow \langle ag, C' \rangle} \quad \# C_\epsilon = -, C_{Ap} = C_R = \{\}$$

$$\text{where: } C'_E = C_E - \langle te, i \rangle$$

$$C'_\epsilon = \langle te, i \rangle$$

Relevant Plans: The rule **Rel₁** initialises the R component with the set of relevant plans. If no plan is relevant, the event is discarded from ϵ by **Rel₂**.

$$\text{Rel}_1 \frac{\text{RelPlans}(ps, te) \neq \{\}}{\langle ag, C \rangle \longrightarrow \langle ag, C' \rangle} \quad \# C_\epsilon = \langle te, i \rangle \quad C_{Ap}, C_R = \{\}$$

$$\text{where: } C'_R = \text{RelPlans}(ps, te)$$

$$\text{Rel}_2 \frac{\text{RelPlans}(ps, te) = \{\}}{\langle ag, C \rangle \longrightarrow \langle ag, C' \rangle} \quad \# C_\epsilon = \langle te, i \rangle \quad C_{Ap}, C_R = \{\}$$

$$\text{where: } C'_\epsilon = -$$

Applicable Plans: The rule **App₁** initialises the Ap component with the set of applicable plans. If no plan is applicable, the event is discarded from ϵ by **App₂**. In either case the relevant plans are also discarded.

$$\text{App}_1 \frac{\text{AppPlans}(bs, C_R) \neq \{\}}{\langle ag, C \rangle \longrightarrow \langle ag, C' \rangle} \quad \# C_\epsilon \neq -, C_{Ap} = \{\}, C_R \neq \{\}$$

$$\text{where: } C'_R = \{\}$$

$$C'_{Ap} = \text{AppPlans}(bs, C_R)$$

$$\text{App}_2 \frac{\text{AppPlans}(bs, C_R) = \{\}}{\langle ag, C \rangle \longrightarrow \langle ag, C' \rangle} \quad \# C_\epsilon \neq -, C_{Ap} = \{\}, C_R \neq \{\}$$

$$\text{where: } C'_R = \{\}$$

$$C'_\epsilon = -$$

$$C'_E = C_E \cup \langle te, i \rangle$$

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 circumstance and the set of applicable plans is discarded.

$$\text{SelAppI} \frac{S_{Ap}(C_{Ap}) = p}{\langle ag, C \rangle \longrightarrow \langle ag, C' \rangle} \quad \# C_\epsilon \neq -, C_{Ap} \neq \{\}$$

$$\text{where: } C'_\rho = p$$

$$C'_{Ap} = \{\}$$

Preparing the Set of Intentions: Events can be classified as external or internal (depending on 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 (which 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.

$$\begin{array}{c}
\mathbf{ExtEv} \frac{}{\langle ag, C \rangle \longrightarrow \langle ag, C' \rangle} \\
\# C_\epsilon = \langle te, \top \rangle, \quad C_\rho = p \\
\text{where: } C'_I = C_I \cup \{ [p] \} \\
C'_\epsilon = C'_\rho = -
\end{array}
\qquad
\begin{array}{c}
\mathbf{IntEv} \frac{}{\langle ag, C \rangle \longrightarrow \langle ag, C' \rangle} \\
\# C_\epsilon = \langle te, i \rangle, \quad C_\rho = p \\
\text{where: } C'_I = C_I \cup \{ i[p] \} \\
C'_\epsilon = C'_\rho = -
\end{array}$$

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, in rule **Achieve**.

Intention Selection: This rule uses a function that selects an intention (i.e., a stack of plans) for processing.

$$\mathbf{IntSel} \frac{S_I(C_I) = i}{\langle ag, C \rangle \longrightarrow \langle ag, C' \rangle} \quad \# C_\iota = - \\
\text{where: } C'_\iota = i$$

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.

– *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.

$$\mathbf{Action} \frac{}{\langle ag, C \rangle \longrightarrow \langle ag, C' \rangle} \quad \# C_\iota = i[\text{head} \leftarrow a; h] \\
\text{where: } C'_\iota = - \\
C'_A = C_A \cup \{ a \} \\
C'_I = (C_I - \{ C_\iota \}) \cup \{ i[\text{head} \leftarrow h] \}$$

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

$$\mathbf{Achieve} \frac{}{\langle ag, C \rangle \longrightarrow \langle ag, C' \rangle} \quad \# C_\iota = i[\text{head} \leftarrow !at; h] \\
\text{where: } C'_\iota = - \\
C'_E = C_E \cup \{ \langle +!at, C_\iota \rangle \} \\
C'_I = C_I - \{ C_\iota \}$$

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* (see [5] for details).

– *Test Goals*: these rules are used when a test goal $?at$ should be executed. Both rules try to produce a set of substitutions that can make at a logical consequence of the agent’s beliefs. The rule **Test₁** says basically that nothing is done if no substitution is found, and the rule **Test₂** says that one of the substitutions is applied to the plan.

$$\begin{array}{l}
\mathbf{Test}_1 \frac{\mathbf{Test}(bs, at) = \{\}}{\langle ag, C \rangle \longrightarrow \langle ag, C' \rangle} \\
\# C_i = i[\mathit{head} \leftarrow ?at; h] \\
\text{where: } C'_i = - \\
C'_I = (C_I - \{C_i\}) \cup \{i[\mathit{head} \leftarrow h]\}
\end{array}
\qquad
\begin{array}{l}
\mathbf{Test}_2 \frac{\mathbf{Test}(bs, at) \neq \{\}}{\langle ag, C \rangle \longrightarrow \langle ag, C' \rangle} \\
\# C_i = i[\mathit{head} \leftarrow ?at; h] \\
\text{where: } C'_i = - \\
C'_I = (C_I - \{C_i\}) \cup \{i[(\mathit{head} \leftarrow h)\theta]\} \\
\theta \in \mathbf{Test}(bs, at)
\end{array}$$

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

$$\begin{array}{l}
\mathbf{AddBel} \frac{}{\langle ag, C \rangle \longrightarrow \langle ag', C' \rangle} \\
\# C_i = i[\mathit{head} \leftarrow +b; h] \\
\text{where: } C'_i = - \\
bs' \models b \\
C'_E = C_E \cup \{+b, C_i\} \\
C'_I = (C_I - \{C_i\}) \cup \{i[\mathit{head} \leftarrow h]\}
\end{array}
\qquad
\begin{array}{l}
\mathbf{DelBel} \frac{}{\langle ag, C \rangle \longrightarrow \langle ag', C' \rangle} \\
\# C_i = i[\mathit{head} \leftarrow -b; h] \\
\text{where: } C'_i = - \\
bs' \not\models b \\
C'_E = C_E \cup \{-b, C_i\} \\
C'_I = (C_I - \{C_i\}) \cup \{i[\mathit{head} \leftarrow h]\}
\end{array}$$

Removing intentions: The two rules below can be seen as “clearing house” rules. The rule **ClearInt₁** 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 **ClearInt₂** 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).

$$\begin{array}{l}
\mathbf{ClrInt}_1 \frac{}{\langle ag, C \rangle \longrightarrow \langle ag, C' \rangle} \\
\# C_i = [\mathit{head} \leftarrow] \\
\text{where: } C'_i = - \\
C'_I = C_I - \{C_i\}
\end{array}
\qquad
\begin{array}{l}
\mathbf{ClrInt}_2 \frac{}{\langle ag, C \rangle \longrightarrow \langle ag, C' \rangle} \\
\# C_i = i'[\mathit{head}' \leftarrow !at; h'][\mathit{head} \leftarrow] \\
\text{where: } C'_i = - \\
C'_I = (C_I - \{C_i\}) \cup \{i'[\mathit{head}' \leftarrow h']\}
\end{array}$$