Journal of Artificial Intelligence Research 29 (2007) 221–267 Submitted 12/06; published 06/07

On the Formal Semantics of Speech-Act Based Communication in an Agent-Oriented Programming Language

Renata Vieira RENATAV@UNISINOS.BR
Universidade do Vale do Rio dos Sinos
São Leopoldo, RS, 93022-000, Brazil

Álvaro Moreira ALVARO.MOREIRA@INF.UFRGS.BR
Universidade Federal do Rio Grande do Sul
Porto Alegre, RS, 91501-970, Brazil

Michael Wooldridge MJW@CSC.LIV.AC.UK
University of Liverpool
Liverpool L69 3BX, United Kingdom

Rafael H. Bordini R.BORDINI@DURHAM.AC.UK
University of Durham
Durham DH1 3LE, United Kingdom

Abstract

Research on agent communication languages has typically taken the speech acts paradigm as its starting point. Despite their manifest attractions, speech-act models of communication have several serious disadvantages as a foundation for communication in artificial agent systems. In particular, it has proved to be extremely difficult to give a satisfactory semantics to speech-act based agent communication languages. In part, the problem is that speech-act semantics typically make reference to the "mental states" of agents (their beliefs, desires, and intentions), and there is in general no way to attribute such attitudes to arbitrary computational agents. In addition, agent programming languages have only had their semantics formalised for abstract, stand-alone versions, neglecting aspects such as communication primitives. With respect to communication, implemented agent programming languages have tended to be rather *ad hoc*. This paper addresses both of these problems, by giving semantics to speech-act based messages received by an AgentSpeak agent. AgentSpeak is a logic-based agent programming language which incorporates the main features of the PRS model of reactive planning systems. The paper builds upon a structural operational semantics to AgentSpeak that we developed in previous work. The main contributions of this paper are as follows: an extension of our earlier work on the theoretical foundations of AgentSpeak interpreters; a computationally grounded semantics for (the core) performatives used in speech-act based agent communication languages; and a well-defined extension of AgentSpeak that supports agent communication.

1. Introduction

First introduced in 1987, the reactive planning model of Georgeff and Lansky's PRS system has subsequently proved to be one of the most influential and long-lived approaches to programming multi-agent systems (Georgeff & Lansky, 1987). The AgentSpeak programming language, introduced by Rao (1996), represents an attempt to distill the key features of the PRS approach into a simple, abstract, logic-based language. AgentSpeak 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, and at the same time it has robust working interpreters (Bordini, Hübner, & Vieira, 2005; Bordini & Hübner, 2007; Bordini, Bazzan, Jannone, Basso, Vicari, & Lesser, 2002), its formal semantics and relation to BDI logics (Rao & Georgeff, 1998; Wooldridge, 2000b) have been thoroughly studied (Bordini & Moreira, 2004; Moreira, Vieira, & Bordini, 2004; Moreira & Bordini, 2002), and there is ongoing work on the use of model-checking techniques for verification of AgentSpeak multi-agent systems (Bordini, Fisher, Visser, & Wooldridge, 2004; Bordini, Visser, Fisher, Pardavila, & Wooldridge, 2003; Bordini, Fisher, Pardavila, & Wooldridge, 2003).

In the original formulation of AgentSpeak (Rao, 1996), the main emphasis was on the *internal* control structures and decision-making cycle of an agent: the issue of *communication* between agents was not addressed. Accordingly, most attempts to give a formal semantics to the language have focused on these internal aspects (Moreira & Bordini, 2002). Although several extensions to AgentSpeak have been proposed in an attempt to make it a practically more useful language (Bordini et al., 2005, 2002), comparatively little research has addressed the issue of a principled mechanism to support communication in AgentSpeak, which is clearly essential for engineering *multi*-agent systems.

Most agent communication languages have taken *speech-act theory* (Austin, 1962; Searle, 1969) as their starting point. As is suggested by its name, speech-act theory is predicated on the view that utterances are *actions*, performed by rational agents in the furtherance of their personal desires and intentions. Thus, according to speech-act theory, utterances may be considered as actions performed by an agent, typically with the intention of changing the mental state of the hearer(s) of the utterance. Speech-act theory thus seems particularly appropriate as a foundation for communication among intentional agents. Through communication, an agent can share its internal state (beliefs, desires, intentions) with other agents, and can attempt to influence the mental states of other agents.

Although an initial speech-act based communication model for AgentSpeak agents was previously introduced (Bordini et al., 2003), no formal semantics of that model was given in that paper. A preliminary formal account for communication of AgentSpeak agents was first given by Moreira et al. (2004). The main contribution of the present paper is to thoroughly extend the operational semantics of AgentSpeak accounting for speech-act style communication. Our semantics precisely defines how to implement the processing of messages *received* by an AgentSpeak agent; that is, how the computational representations of mental states are changed when a message is received. Note that in implementations of the BDI architecture, the concepts of *plan* and *plan library* is used to simplify aspects of deliberation and means-ends reasoning. Therefore, an AgentSpeak agent *sends* a message whenever there is a communicative action in the body of an intended plan that is being executed; such plans are typically written by an agent programmer.

As pointed out by Singh (1998), well-known approaches to agent communication focus largely on the sender's perspective, ignoring how a message should be processed and understood. This is the main aspect of agent communication that we consider in this paper. In extending the operational semantics of AgentSpeak to account for inter-agent communication, we also touch upon another long-standing problem in the area of multi-agent systems:

the semantics of communication languages based on speech acts. The difficulty here is that, taking their inspiration from attempts to develop a semantics of human speech acts, most semantics for agent communication languages have defined the meaning of messages between agents with respect to the mental states of communication participants. While this arguably has the advantage of remaining neutral on the actual internal structure of agents, a number of authors have observed that this makes it impossible in general to determine whether or not some program that claims to be implementing the semantics really *is* implementing it (Wooldridge, 1998; Singh, 1998). 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 underlying semantics of belief (or intention, or mental attitudes in general).

This is related to the fact that previous approaches attempt to give a programming language independent semantics of agent communication. Our semantics, while developed for one specific language, have the advantage of not relying on mechanisms — such as abstractly defined mental states — that cannot be verified for real programs. We note that, to the best of our knowledge, our work represents the first semantics given for a speech-act style, "knowledge level" communication language that is used in a real system.

Since a precise notion of Belief-Desire-Intention has been given previously for AgentSpeak agents (Bordini & Moreira, 2004), we can provide such a computationally grounded (Wooldridge, 2000a) semantics of speech-act based communication for this language, making it possible to determine how an AgentSpeak agent interprets a particular message when it is received. Note, however, that whether and how an agent acts upon received communication depends on its plan library and its other circumstances at the time the message is processed. Also, although our approach is tied to a particular language, it can be usefully employed as a reference model for developing communication semantics and implementing communication in other agent programming languages.

The remainder of this paper is organised as follows. Section 2 provides the general background on PRS-style BDI architectures and speech-act based agent communication. Section 3 presents AgentSpeak syntax and semantics — a much revised version of the syntax and semantics of AgentSpeak presented by Moreira and Bordini (2002, 2004). Section 4 presents the speech-act based communication model for AgentSpeak agents, an extension of the preliminary formal account given by Moreira et al. (2004). Section 5 illustrates the semantics with an example of the semantic rules applied in a typical reasoning cycle. In Section 6, we show how programmers can use our basic communication constructs to develop some of the more elaborate forms of communication required by some multi-agent applications (for example, ensuring that a belief is shared between two agents and keeping track of the progress in the achievement of a delegated goal), and in Section 7 we give a simple example of the use of our framework for proving properties of communicating agents. Section 8 presents a discussion on applications and further developments for the language presented in this paper. Conclusions and planned future work are given in the final section.

2. Background

The ability to *plan* seems to be one of the key components of rational action in humans. Planning is the ability to take a goal, and from this goal generate a "recipe" (i.e., plan) for action such that, if this recipe is followed (under favourable conditions), the goal will be achieved. Accordingly, a great deal of research in artificial intelligence has addressed the issue of *automatic planning*: the synthesis of plans by agents from first principles (Allen, Hendler, & Tate, 1990). Unfortunately, planning is, like so many other problems in artificial intelligence, prohibitively expensive in computational terms. While great strides have been made in developing efficient automatic planning systems (Ghallab, Nau, & Traverso, 2004), the inherent complexity of the process inevitably casts some doubt on whether it will be possible to use plan-synthesis algorithms to develop plans at run-time in systems that must operate under tight real-time constraints. Many researchers have instead considered approaches that make use of *pre-compiled* plans, i.e., plans developed off-line, at design time. The *Procedural Reasoning System (PRS)* of Georgeff and Lansky is a common ancestor of many such approaches (Georgeff & Lansky, 1987).

2.1 The PRS and AgentSpeak

On one level, the PRS can be understood simply as an architecture for executing pre-compiled plans. However, the control structures in the architecture incorporate a number of features which together provide a sophisticated environment for run-time practical reasoning. First, plans may be invoked by their *effect*, rather than simply by name (as is the case in conventional programming languages). Second, plans are associated with a *context*, which must match the agent's current situation in order for the plan to be considered a viable option. These two features mean that an agent may have multiple potential plans for the same end, and can dynamically select between these at run-time, depending on current circumstances. In addition, plans are associated with *triggering events*, the idea being that a plan is made "active" by the occurrence of such an event, which may be external or internal to the agent. External events are changes in the environment as perceived by the agent; an example of an internal event might be the creation of a new sub-goal, or the failure of a plan to achieve its desired effect. Thus, overall, plans may be invoked in a goal-driven manner (to satisfy a sub-goal that has been created) or in an event-driven manner. The PRS architecture is illustrated in Figure 1. The AgentSpeak language, introduced by Rao (1996), represents an attempt to distill the "essential" features of the PRS into a simple, unified programming language[1]; we provide a detailed introduction to AgentSpeak below, after we discuss speech-act theory and agent communication.

2.2 Speech Acts

The PRS model, and the AgentSpeak language in turn, are primarily concerned with the internal structure of decision making, and in particular the interplay between the creation of (sub-)goals and the execution of plans to achieve these (sub-)goals. The twin issues of *communication* and *multi-agent interaction* are not addressed within the basic architecture. This raises the question of how such issues might be dealt with within the architecture. While BDI theory is based on the philosophical literature on practical reasoning (Bratman,

1. The name of the language originally introduced by Rao (1996) was AgentSpeak(L). In this paper, we adopt the simpler form AgentSpeak instead, and we use it to refer both to the original language and the variants that appeared in the literature.

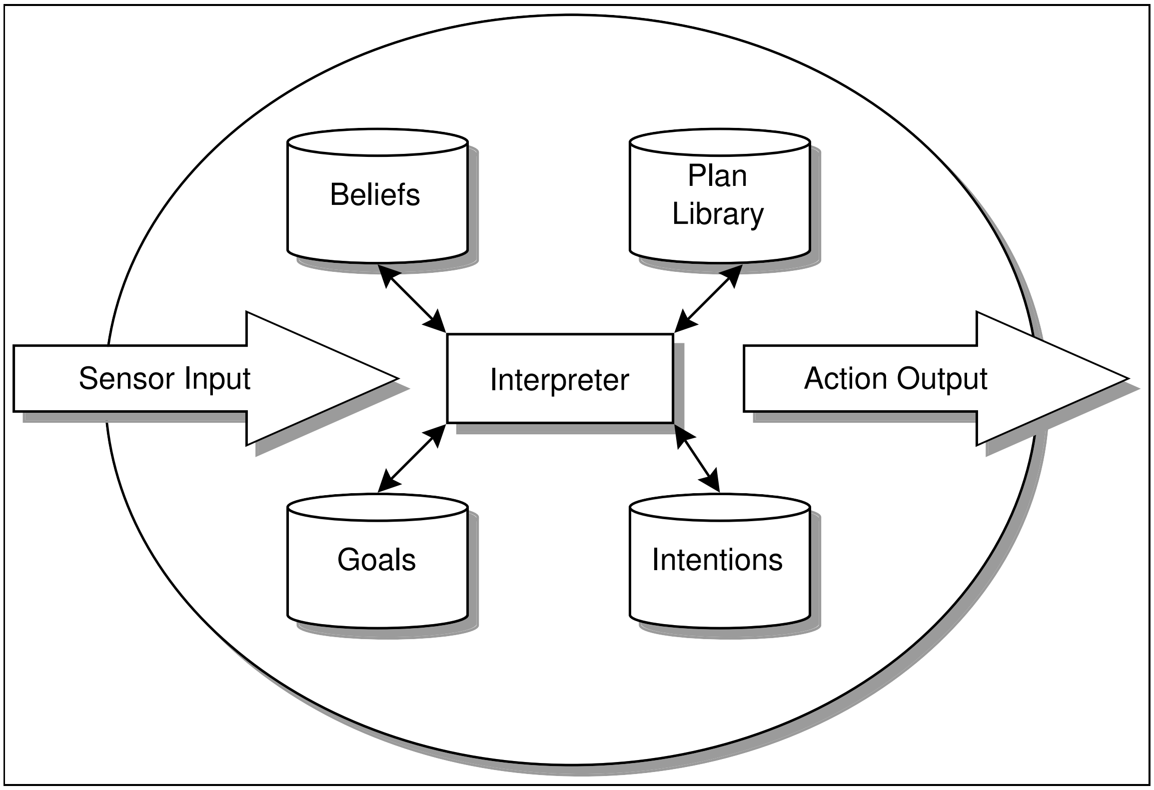

Figure 1: The PRS architecture.

1987), agent communication in multi-agent systems is typically based on the speech-act theory, in particular the work of Austin (1962) and Searle (1969).

Speech-act theory starts from the principle that language is action: a rational agent makes an utterance in an attempt to change the state of the world, in the same way that an agent performs "physical" actions to change the state of the world. What distinguishes speech acts from other ("non-speech") actions is that the domain of a speech act — the part of the world that the agent wishes to modify through the performance of the act — is mostly the mental state(s) of the hearer(s) of the utterance.

Speech acts are generally classified according to their *illocutionary force* — the "type" of the utterance. In natural language, illocutionary forces are associated to utterances (or locutionary acts). The utterance "the door is open", for example, is generally an "inform" or "tell" type of action. The *perlocutionary force* represents what the speaker of the utterance is attempting to achieve by performing the act. In making a statement such as "open the door", the perlocutionary force will generally be the state of affairs that the speaker hopes to bring about by making the utterance; of course, the *actual* effect of an utterance will be beyond the control of the speaker. Whether I choose to believe you when you inform me that the door is open depends upon how I am disposed towards you. In natural language, the illocutionary force and perlocutionary force will be implicit within the speech act and its context. When the theory is adapted to agent communication, however, the illocutionary forces are made explicit to facilitate processing the communication act. The various types of speech acts are generally referred to as "performatives" in the context of agent communication.

Other pragmatic factors related to communication such as social roles and conventions have been discussed in the literature (Levinson, 1981; Ballmer & Brennenstuhl, 1981; Singh, 1994). 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*.

Apart from illocutionary forces and social roles, other classifications of the relations among speech acts have been proposed (Levinson, 1981); for example, a reply follows a question, and threatening is stronger than warning. Such categories place messages in the larger context of a multi-agent dialogue. In multi-agent systems, communicative interactions can be seen as communication protocols, which in turn are normally related to a specific coordination/cooperation mechanism. The Contract Net (Smith, 1980), for example, is a protocol for task allocation, which is defined in terms of a number of constituent performatives (such as announcing and bidding).

2.3 Agent Communication Languages: KQML & FIPA

The Knowledge Query and Manipulation Language (KQML), developed in the context of the "Knowledge Sharing Effort" project (Genesereth & Ketchpel, 1994), was the first attempt to define a practical agent communication language that included high level (speech-act based) communication as considered in the distributed artificial intelligence literature. KQML is essentially a knowledge-level messaging language (Labrou & Finin, 1994; Mayfield, Labrou, & Finin, 1996). KQML defines a number of performatives, which make explicit an agent's intentions in sending a message. For example, the KQML performative `tell` is used with the intention of changing the receiver's *beliefs*, whereas `achieve` is used with the intention of changing the receiver's *goals*. Thus the performative label of a KQML message explicitly identifies the intent of the message sender.

The FIPA standard for agent communication[2] was released in 2002. This standard is closely based on KQML, being almost identical conceptually and syntactically, while differing in the performative set and certain details of the semantic framework (Labrou, Finin, & Peng, 1999). These differences are not important for the purposes of this paper; when we refer to traditional approaches to semantics of speech-act based inter-agent communication, the reference applies to both equally. However, for historical reasons, we refer mainly to KQML and the richer literature that can be found on its semantics.

2.4 The Semantics of Agent Communication Languages

Perhaps the first serious attempt to define the semantics of KQML was made by Labrou and Finin (1994). Their work built on the pioneering work of Cohen and Perrault on an action-theoretic semantics of natural language speech acts (Cohen & Perrault, 1979). The key insight in Cohen and Perrault's work was that, if we take seriously the idea of utterances as

2. http://www.fipa.org/specs/fipa00037/SC00037J.html

action, then we should be able to apply a formalism for reasoning about action to reasoning about utterances. They used a STRIPS-style pre- and post-condition formalism to define the semantics of "inform" and "request" speech acts (perhaps the canonical examples of speech acts), where these pre- and post-conditions were framed in terms of the beliefs, desires, and abilities of conversation participants. When applied by Labrou and Finin to the KQML language (1994), the pre- and post-conditions defined the mental states of the sender and receiver of a KQML message before and after sending such message. For the description of mental states, most of the work in the area is based on Cohen and Levesque's theory of intention (1990a, 1990b). Agent states are described through mental attitudes such as belief (*bel*), knowledge (*know*), desire (*want*), and intention (*intend*). These mental attitudes normally have propositions (i.e., symbolic representations of states of the world) as arguments. Figures 2 and 3 give semantics for the KQML performatives $tell(S, R, X)$ (S tells R that S believes that X is true), and $ask\_if(S, R, X)$ (S asks R if R believes that X is true), in the style introduced by Labrou and Finin (1994).

- 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:
  - $know(R, bel(S, X))$

Figure 2: Semantics for *tell* (Labrou & Finin, 1994).

- Pre-conditions on the states of S and R:
  - $Pre(S)$: $want(S, know(S, Y)) \wedge know(S, intend(R, process(R, M)))$ where Y is either $bel(R, X)$ or $\neg bel(R, X)$ and M is $ask\text{-}if(S, R, X)$
  - $Pre(R)$: $intend(R, process(R, M))$
- Post-conditions about R and S:
  - $Pos(S)$: $intend(S, know(S, Y))$
  - $Pos(R)$: $know(R, want(S, know(S, Y)))$
- Action completion:
  - $know(S, Y)$

Figure 3: Semantics for *ask-if* (Labrou & Finin, 1994).

As noted above, one of the key problems with this (widely used) approach to giving semantics to agent communication languages is that there is no way to determine whether or not any software component that uses such a communication language complies with the semantics. This is because the semantics makes reference to mental states, and we have in general no principled way to attribute such mental states to arbitrary pieces of software. This is true of the semantic approaches to both KQML and FIPA, as discussed by both Wooldridge (1998) and Singh (1998). As an example, consider a legacy software component wrapped in an agent that uses KQML or FIPA to interoperate with other agents. One cannot prove communication properties of such system, as there is no precise definition of when the legacy system believes that (or intends to achieve a state of the world where) some proposition is true. Our approach builds on the work of Bordini and Moreira (2004), which presented a precise definition of what it means for an AgentSpeak agent to believe, desire, or intend a certain formula; that approach is also adopted in our work on model-checking for AgentSpeak (Bordini et al., 2004). As a consequence, we are able to successfully and meaningfully apply speech act-style semantics to communication in AgentSpeak. The drawback, of course, is that the approach is, formally, limited to AgentSpeak agents, even though the same ideas can be used in work on semantics for other agent languages.

3. Syntax and Semantics of AgentSpeak

The AgentSpeak programming language was introduced by Rao (1996). It can be understood as 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, perhaps one of the major approaches to the implementation of rational practical reasoning agents (Wooldridge, 2000b).

An AgentSpeak agent is created by the specification of a set of beliefs forming the initial *belief base* and a set of plans forming the *plan library*. An agent's belief base is a set of ground first-order predicates, which will change over time to represent the current state of the environment as perceived by the agent.

AgentSpeak distinguishes two types of goals: *achievement goals* and *test goals*. Achievement and test goals are predicates (as with beliefs), prefixed with one of the 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, as we will see, this is done by the execution of a plan. A test goal returns a unification for the associated predicate with one of the agent's beliefs; it fails if no such unification is possible. 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). Additionally, with respect to the model of communication in this paper, external events can be related to messages received from other agents. 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. A plan is formed by a *triggering event*, denoting the events for which that plan is *relevant*. The triggering event is followed by a conjunction of belief literals representing a *context* for the plan. The context must be a logical consequence of the agent's current beliefs for the

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

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

Figure 4: Examples of AgentSpeak plans.

plan to be *applicable* — one of the plans that are both relevant and applicable is chosen for execution so as to handle a particular event. 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 is executed.

Figure 4 shows some examples of AgentSpeak plans. The first plan tells us that, when a concert is announced for artist `A` at venue `V` (so that, from perceiving the environment, a belief `concert(A,V)` is *added* to the belief base), provided that the agent happens to like artist `A`, it will have the new achievement 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`, provided it is the case that the telephone is not busy, it can execute a plan consisting of retrieving from its belief base the telephone number of venue `V` (with the test goal `?phone_number(V,N)`), performing the basic action `call(N)` (assuming that making a phone call is one of the actions that the agent is able to 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.

Next, we formally present the syntax and semantics of AgentSpeak. Note that we do not yet consider communication; we extend the semantics to deal with communication in Section 4.

3.1 Abstract Syntax

The syntax of an AgentSpeak agent program ag is defined by the grammar below. In AgentSpeak, an agent program is simply given by a set bs of beliefs and a set ps of plans. The beliefs bs define the initial state of the agent's belief base (i.e., the state of the belief base when the agent starts running), and the plans ps form the agent's plan library. The atomic formulæ at of the language are predicates, where `P` is a predicate symbol and $t_1, \ldots, t_n$ are standard terms of first order logic. A *belief* is an atomic formula at with no variables; we use b as a meta-variable for beliefs.

$$
\begin{array}{lclllllll}
ag & ::= & bs \quad ps & & & & & & \\
bs & ::= & b_1 \ldots b_n & & & & & & (n \geq 0) \\
ps & ::= & p_1 \ldots p_n & & & & & & (n \geq 1) \\
p & ::= & te : ct \leftarrow h & & & & & & \\
te & ::= & +at & \mid & -at & \mid & +g & \mid & -g \\
ct & ::= & ct_1 & \mid & \mathtt{T} & & & & \\
ct_1 & ::= & at & \mid & \neg at & \mid & ct_1 \wedge ct_1 & \mid & \\
h & ::= & h_1;\mathtt{T} & \mid & \mathtt{T} & & & & \\
h_1 & ::= & a & \mid & g & \mid & u & \mid & h_1;h_1 \\
at & ::= & \mathtt{P}(t_1, \ldots, t_n) & & & & & & (n \geq 0) \\
 & \mid & \mathtt{P}(t_1, \ldots, t_n)[s_1, \ldots, s_m] & & & & & & (n \geq 0, m > 0) \\
s & ::= & \mathtt{percept} & \mid & \mathtt{self} & \mid & id & & \\
a & ::= & \mathtt{A}(t_1, \ldots, t_n) & & & & & & (n \geq 0) \\
g & ::= & !at & \mid & ?at & & & & \\
u & ::= & +b & \mid & -at & & & &
\end{array}
$$

The grammar above gives an alternative definition for at, extending the conventional syntactic form of predicates. The extension allows "annotations" to be associated with a predicate; this is an extension of AgentSpeak's original syntax motivated by our work on communication, which is discussed below in Section 3.2. For the time being, suffice it to say that the idea is to annotate each *atomic formula* with its *source*: either a term id identifying which agent previously communicated that information, `self` to denote beliefs created by the agent itself (through belief update operations within a plan, as described below), or `percept` to indicate that the belief was acquired through perception of the environment. So, for example, if agent i has a belief $concert(a, v)[j]$ in its belief base, this would mean that agent j had previously informed agent i of $concert(a, v)$ — in other words, that j wanted i to believe that there will be a concert by a at v. Similarly, $concert(a, v)[percept, j]$ would mean that a's concert at v is believed not only because j has informed agent i of this, but because i itself also perceived this fact (e.g., by seeing a poster when walking past the theatre).

A plan in AgentSpeak 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 (which should be thought of as "mental notes" created by the agent itself). We refer to $te : ct$ as the *head* of the plan, and h is its *body*. The set of plans of an agent is given by ps. Each plan has as part of its head a formula ct that specifies the conditions under which the plan can be chosen for execution.

A triggering event te can then be the addition or the deletion of a belief from an agent's belief base (denoted $+at$ and $-at$, respectively), or the addition or the deletion of a goal ($+g$ and $-g$, respectively[3]). For plan bodies, we assume the agent has at its disposal a set of *actions* and we use a as a meta-variable ranging over them. We are largely unconcerned here with respect to exactly what such actions are. Actions are written using the same notation

3. Triggering events of the form $-g$, in our approach, are used in practice for handling plan failure. Although we have left this construct in the grammar, we have omitted the discussion and formalisation of plan failure for clarity, as the focus in this paper is on the semantics of communication.

as 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, $+b$ and $-at$ (in the body of a plan) represent operations for updating (u) the belief base by, respectively, adding or removing beliefs; recall that an atomic formula must be ground if it is to be added to the belief base.

3.2 Semantics

We define the semantics of AgentSpeak using operational semantics, a widely used method for giving semantics to programming languages (Plotkin, 1981). The operational semantics is given by a set of rules that define a transition relation between configurations $\langle ag, C, M, T, s\rangle$ where:

- An agent program ag is, as defined above, a set of beliefs bs and a set of plans ps.
- An agent's circumstance C is a tuple $\langle I, E, A\rangle$ where:
  - I is a set of *intentions* $\{i, i', \ldots\}$. Each intention i is a stack of partially instantiated plans.
  - 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 i is an intention — a stack of plans in case of an internal event, or the empty intention T in case of an external event. When the belief revision function (which is not part of the AgentSpeak interpreter but rather of the agent's overall architecture), updates the belief base, the associated events — i.e., additions and deletions of beliefs — are included in this set. These are called *external* events; internal events are generated by additions or deletions of goals from plans currently executing.
  - A is a set of *actions* to be performed in the environment.
- M is a tuple $\langle In, Out, SI\rangle$ whose components characterise the following aspects of communicating agents (note that communication is asynchronous):
  - In is the mail inbox: the system includes all messages addressed to this agent in this set. Elements of this set have the form $\langle mid, id, ilf, cnt\rangle$, where mid is a message identifier, id identifies the sender of the message, ilf is the illocutionary force of the message, and cnt its content: a (possibly singleton) set of AgentSpeak predicates or plans, depending on the illocutionary force of the message.
  - Out is where the agent posts messages it wishes to send; it is assumed that some underlying communication infrastructure handles the delivery of such messages. (We are not concerned with this infrastructure here.) Messages in this set have exactly the same format as above, except that here id refers to the agent to which the message is to be sent.
  - SI is used to keep track of intentions that were suspended due to the processing of communication messages; this is explained in more detail in the next section, but the intuition is as follows: intentions associated with illocutionary forces that

require a reply from the interlocutor are suspended, and they are only resumed when such reply has been received.

- It is useful to have a structure which keeps track of temporary information that may be subsequently required within a reasoning cycle. T is a tuple $\langle R, Ap, \iota, \varepsilon, \rho \rangle$ with such temporary information; these components are as follows:
  - R is the set of *relevant plans* (for the event being handled).
  - Ap is the set of *applicable plans* (the relevant plans whose contexts are true).
  - ι, ε, and ρ record a particular intention, event, and applicable plan (respectively) being considered along the execution of one reasoning cycle.
- The current step within an agent's reasoning cycle is symbolically annotated by $s \in \{\mathsf{ProcMsg}, \mathsf{SelEv}, \mathsf{RelPl}, \mathsf{ApplPl}, \mathsf{SelAppl}, \mathsf{AddIM}, \mathsf{SelInt}, \mathsf{ExecInt}, \mathsf{ClrInt}\}$. These labels stand for, respectively: processing a message from the agent's mail inbox, selecting an event from the set of events, retrieving all relevant plans, checking which of those are applicable, selecting one particular applicable plan (the intended means), adding the new intended means to the set of intentions, selecting an intention, executing the selected intention, and clearing an intention or intended means that may have finished in the previous step.

In the interests of readability, we adopt the following notational conventions in our semantic rules:

- If C is an AgentSpeak agent circumstance, we write C_E to make reference to the E component of C, and similarly for other components of a configuration.
- We write $T_\iota = \_$ (the underscore symbol) to indicate that there is no intention presently being considered in that reasoning cycle. Similarly for T_ρ and T_ε.
- We write $i[p]$ to denote the intention that has plan p on top of intention i.

The AgentSpeak interpreter makes use of three *selection functions* that are defined by the agent programmer. The selection function S_E selects an event from the set of events C_E; the selection function S_{Ap} selects one applicable plan given a set of applicable plans; and S_I selects an intention from the set of intentions C_I (the chosen intention is then executed). Formally, all the selection functions an agent uses are also part of its configuration (as is the social acceptance function that we mention later when we formalise agent communication). However, as they are defined by the agent programmer at design time and do not (in principle) change at run time, we avoid including them in the configuration for the sake of readability.

We define some functions which help simplify the semantics. If p is a plan of the form $te : ct \leftarrow h$, we define $\mathsf{TrEv}(p) = te$ and $\mathsf{Ctxt}(p) = ct$. That is, these projection functions return the triggering event and the context of the plan, respectively. The TrEv function can also be applied to the head of a plan rather than the whole plan, but works similarly in that case.

Next, we need to define the specific (limited) notion of logical consequence used here. We assume a procedure that computes the most general unifier of two literals (as usual in logic programming), and with this, define the logical consequence relation $\models$ that is used in the definitions of the functions for checking for relevant and applicable plans, as well as executing test goals. Given that we have extended the syntax of atomic formulæ so as to include annotations of the sources for the information symbolically represented by it, we also need to define $\models$ in our particular context, as follows.

Definition 1 *We say that an atomic formula at_1 with annotations $s_{11}, \ldots, s_{1n}$ is a logical consequence of a set of ground atomic formulæ bs, written $bs \models at_1[s_{11}, \ldots, s_{1n}]$ if, and only if, there exists $at_2[s_{21}, \ldots, s_{2m}] \in bs$ such that (i) $at_1\theta = at_2$, for some most general unifier θ, and (ii) $\{s_{11}, \ldots, s_{1n}\} \subseteq \{s_{21}, \ldots, s_{2m}\}$.*

The intuition is that, not only should predicate *at* unify with some belief in *bs* (i), but also that all specified sources of information for *at* should be corroborated in *bs* (ii). Thus, for example, $\mathtt{p(X)[ag_1]}$ follows from $\{\mathtt{p(t)[ag_1,ag_2]}\}$, but $\mathtt{p(X)[ag_1,ag_2]}$ does *not* follow from $\{\mathtt{p(t)[ag_1]}\}$. More concretely, if, in order to be applicable, a plan requires that a drowning person was explicitly perceived rather than communicated by another agent (which can be represented by `drowning(Person)[percept]`), this follows from a belief `drowning(man)[percept,passerby]` (i.e., that this was both perceived and communicated by a passerby). On the other hand, if the required context was that two independent sources provided the information, say `cheating(Person)[witness1,witness2]`, this cannot be inferred from a belief `cheating(husband)[witness1]`.

In order to make some semantic rules more readable, we use two operations on a belief base (i.e., a set of annotated ground atomic formulæ). We use $bs' = bs + b$ to say that bs' is as bs except that $bs' \models b$. Similarly $bs' = bs - b$ means that bs' is as bs except that $bs' \not\models b$.

A plan is considered *relevant* in relation to a triggering event if it has been written to deal with that event. In practice, this is checked by trying to unify the triggering event part of the plan with the triggering event within the event that has been selected for treatment in that reasoning cycle. In the definition below, we use the logical consequence relation defined above to check if a plan's triggering event unifies with the event that has occurred. To do this, we need to extend the $\models$ relation so that it also applies to triggering events instead of predicates. In fact, for the purposes here, we can consider that any operators in a triggering event (such as '+' or '!') are part of the predicate symbol or, more precisely, let at_1 be the predicate (with annotation) within triggering event te_1 and at_2 the one within te_2, then $\{te_2\} \models te_1$ if, and only if, $\{at_2\} \models at_1$ and, of course, the operators prefixing te_1 and te_2 are exactly the same. Because of the requirement of inclusion of annotations, the converse might not be true.

Definition 2 *Given plans ps of an agent and a triggering event te, the set $\mathsf{RelPlans}(ps, te)$ of relevant plans for te is defined as follows:*

$$\mathsf{RelPlans}(ps, te) = \{(p, \theta) \mid p \in ps \textit{ and } \theta \textit{ is s.t. } \{te\} \models \mathsf{TrEv}(p)\theta\}.$$

The intuition regarding annotations is as follows. The programmer should include in the annotations of a plan's triggering event all the sources that must have generated the event for that plan to be relevant (or include no annotation if the source of information is not important for the plan to be considered relevant). For the plan to be relevant, it therefore suffices for the annotations in the plan's triggering event to be a subset of those in the event that occurred. A plan with triggering event $+!\mathtt{p(X)[s]}$ is relevant for an event $\langle +!\mathtt{p(t)[s,t]}, \mathsf{T}\rangle$ since $\mathsf{RelPlans}$ requires that $\{\mathtt{p(t)[s,t]}\} \models \mathtt{p(X)[s]}\theta$ (for some most general unifier θ), which in turn requires that $\{\mathtt{s}\} \subseteq \{\mathtt{s},\mathtt{t}\}$. As a consequence, for a plan with a triggering event that has no annotations (e.g., $+!\mathtt{p(X)}$) to be relevant for a particular event (say, $\langle +!\mathtt{p(t)[ag_1]}, i\rangle$) it only requires that the predicates unify in the usual sense since $\{\} \subseteq S$, for any set S.

A plan is *applicable* if it is relevant and its context is a logical consequence of the agent's beliefs. Again we need to extend slightly the definition of $\models$ given above. A plan's context is a conjunction of literals (l is either at or $\neg at$). We can say that $bs \models l_1 \wedge \ldots \wedge l_n$ if, and only if, $bs \models l_i$ if l_i is of the form at, and $bs \not\models l_i$ of l_i is of the form $\neg at$, for $1 \leq i \leq n$. The function for determining the applicable plans in a set of relevant plans is formalised as follows.

Definition 3 *Given a set of relevant plans R and the beliefs bs of an agent, the set of applicable plans $\mathsf{AppPlans}(bs, R)$ is defined as follows:*

$$\mathsf{AppPlans}(bs, R) = \{(p, \theta' \circ \theta) \mid (p, \theta) \in R \textit{ and } \theta' \textit{ is s.t. } bs \models \mathsf{Ctxt}(p)\theta\theta'\}.$$

We need another function to be used in the semantic rule for when the agent is to execute 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. The function returns a set of most general unifiers all of which make the formula at a logical consequence of a set of formulæ bs, as follows.

Definition 4 *Given a set of formulæ bs and a formula at, the set of substitutions $\mathsf{Test}(bs, at)$ produced by testing at against bs is defined as follows:*

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

Next, we present the reasoning cycle of AgentSpeak agents and the rules which define the operational semantics.

3.3 Reasoning Cycle

Figure 5 shows the possible transitions between the various steps in an agent's reasoning cycle as determined by an AgentSpeak interpreter. The labels in the nodes identify each step of the cycle, which are: processing received messages ($\mathsf{ProcMsg}$); selecting an event from the set of events (SelEv); retrieving all relevant plans (RelPl); checking which of those are applicable (ApplPl); selecting one particular applicable plan (the intended means) ($\mathsf{SelAppl}$); adding the new intended means to the set of intentions (AddIM); selecting an intention (SelInt); executing the selected intention ($\mathsf{ExecInt}$), and clearing an intention or intended means that may have finished in the previous step (ClrInt).

In the general case, an agent's initial configuration is $\langle ag, C, M, T, \mathsf{ProcMsg}\rangle$, where ag is as given by the agent program, and all components of C, M, and T are empty. Note that

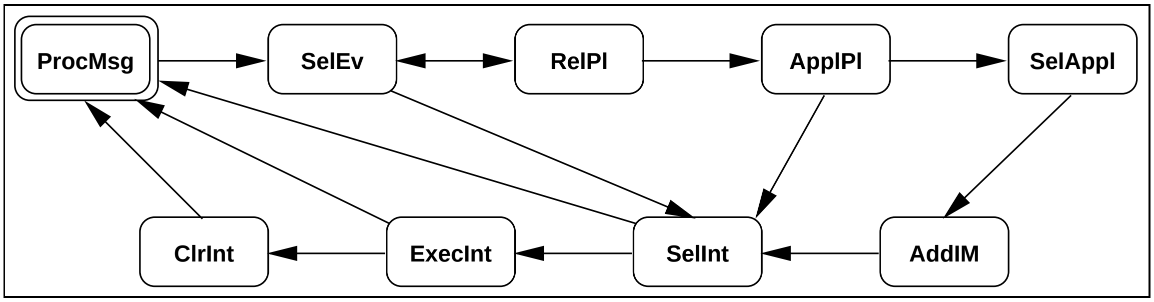

Figure 5: The AgentSpeak agent reasoning cycle.

a reasoning cycle starts with processing received messages (ProcMsg) — the semantics for this part of the reasoning cycle are given in the main section of this paper. After that, the original AgentSpeak reasoning cycle takes place. An event selection (SelEv) is made, which is followed by determining relevant and applicable plans (RelPl and ApplPl, respectively). One of the relevant plans is then selected (SelAppl); note that when there are no events to be treated or when there are no applicable plans to deal with an event the agent turns its attention to the selection of an intended means (SelInt) to be executed next. After one of the relevant plans is selected (SelAppl) and an instance of that plan becomes an "intended means" and is therefore included in the set of intentions (AddIM). When there are more than one intention (which is normally the case except for extremely simple agents), one of those intentions is selected (SelInt) and executed (ExecInt).

These are the most important transitions; the others will be made clearer when the semantics is presented. The rules which define the transition systems giving operational semantics to AgentSpeak (without communication) are presented next.

3.4 Semantic Rules

In this section, we present an operational semantics for AgentSpeak that formalises the transitions between possible steps of the interpretation of AgentSpeak agents as shown in Figure 5. In the general case, an agent's initial configuration is $\langle ag, C, M, T, \mathsf{ProcMsg}\rangle$, where ag is as given by the agent program, and all components of C, M, and T are empty. Note that a reasoning cycle starts with processing received messages (ProcMsg), according to the most recent extension of the semantics to be presented in Section 4. An event selection (SelEv) is then made, starting the reasoning cycle as originally defined for the language, which is the part of the semantics presented below.

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 temporary information. Rule $\mathbf{SelEv_2}$ skips to the intention execution part of the cycle, in case there are no events to handle.

$$\frac{S_E(C_E) = \langle te, i\rangle}{\langle ag, C, M, T, \mathsf{SelEv}\rangle \longrightarrow \langle ag, C', M, T', \mathsf{RelPl}\rangle} \qquad (\mathbf{SelEv_1})$$

$$\begin{array}{lrcl} \textit{where:} & C'_E & = & C_E \setminus \{\langle te, i\rangle\} \\ & T'_\varepsilon & = & \langle te, i\rangle \end{array}$$

$$\frac{C_E = \{\}}{\langle ag, C, M, T, \mathsf{SelEv}\rangle \longrightarrow \langle ag, C, M, T, \mathsf{SelInt}\rangle} \qquad (\mathbf{SelEv_2})$$

Relevant Plans: Rule $\mathbf{Rel_1}$ assigns the set of relevant plans to component T_R. Rule $\mathbf{Rel_2}$ deals with the possibility that there are no relevant plans for an event, in which case the event is simply discarded. In fact, an intention associated with the event might also be discarded: if there are no relevant plans to handle an event generated by that intention, it cannot be further executed. In practice, instead of simply discarding the event (and possibly an intention with it), this leads to the activation of the plan failure mechanism, which we do not discuss here for clarity of presentation, as discussed earlier.

$$\frac{T_\varepsilon = \langle te, i\rangle \qquad \mathsf{RelPlans}(ag_{ps}, te) \neq \{\}}{\langle ag, C, M, T, \mathsf{RelPl}\rangle \longrightarrow \langle ag, C, M, T', \mathsf{ApplPl}\rangle} \qquad (\mathbf{Rel_1})$$

$$\begin{array}{lrcl} \textit{where:} & T'_R & = & \mathsf{RelPlans}(ag_{ps}, te) \end{array}$$

$$\frac{\mathsf{RelPlans}(ag_{ps}, te) = \{\}}{\langle ag, C, M, T, \mathsf{RelPl}\rangle \longrightarrow \langle ag, C, M, T, \mathsf{SelEv}\rangle} \qquad (\mathbf{Rel_2})$$

An alternative approach for situations where there are no relevant plans for an event was introduced by Ancona, Mascardi, Hübner, and Bordini (2004). It assumes that in some cases, explicitly specified by the programmer, the agent will want to ask other agents what are the recipes they use for handling such events. The mechanism for plan exchange between AgentSpeak agents they proposed allows the programmer to specify which triggering events should generate attempts to retrieve external plans, which plans an agent agrees to share with others, what to do once the plan has been used for handling that particular event instance, and so forth.

Applicable Plans: The rule $\mathbf{Appl_1}$ assigns the set of applicable plans to the T_{Ap} component; rule $\mathbf{Appl_2}$ applies when there are no applicable plans for an event, in which case the event is simply discarded. Again, in practice, this normally leads to the plan failure mechanism being activated, rather than simply discarding the event (and the whole intention with it).

$$\frac{\mathsf{AppPlans}(ag_{bs}, T_R) \neq \{\}}{\langle ag, C, M, T, \mathsf{ApplPl}\rangle \longrightarrow \langle ag, C, M, T', \mathsf{SelAppl}\rangle} \qquad (\mathbf{Appl_1})$$

$$\begin{array}{lrcl} \textit{where:} & T'_{Ap} & = & \mathsf{AppPlans}(ag_{bs}, T_R) \end{array}$$

$$\frac{\mathsf{AppPlans}(ag_{bs}, T_R) = \{\}}{\langle ag, C, M, T, \mathsf{ApplPl}\rangle \longrightarrow \langle ag, C, M, T, \mathsf{SelInt}\rangle} \qquad (\mathbf{Appl_2})$$

Selection of an Applicable Plan: This rule assumes the existence of a selection function S_{Ap} that selects one plan from a set of applicable plans T_{Ap}. The selected plan is then assigned to the T_ρ component of the configuration.

$$\frac{S_{Ap}(T_{Ap}) = (p, \theta)}{\langle ag, C, M, T, \mathsf{SelAppl}\rangle \longrightarrow \langle ag, C, M, T', \mathsf{AddIM}\rangle} \qquad (\textbf{SelAppl})$$

$$\textit{where:} \quad T'_\rho \;=\; (p, \theta)$$

Adding an Intended Means to 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** determines that, if the event ε is external (which is indicated by T in the intention associated to ε), a new intention is created and the only intended means in that new intention is the plan p assigned to the ρ component. If the event is internal, rule **IntEv** determines that the plan in ρ should be put on top of the intention associated with the event.

$$\frac{T_\varepsilon = \langle te, \mathsf{T}\rangle \qquad T_\rho = (p, \theta)}{\langle ag, C, M, T, \mathsf{AddIM}\rangle \longrightarrow \langle ag, C', M, T, \mathsf{SelInt}\rangle} \qquad (\textbf{ExtEv})$$

$$\textit{where:} \quad C'_I \;=\; C_I \cup \{\ [p\theta]\ \}$$

$$\frac{T_\varepsilon = \langle te, i\rangle \quad T_\rho = (p, \theta)}{\langle ag, C, M, T, \mathsf{AddIM}\rangle \longrightarrow \langle ag, C', M, T, \mathsf{SelInt}\rangle} \qquad (\textbf{IntEv})$$

$$\textit{where:} \quad C'_I \;=\; C_I \cup \{\ i[(p\theta)]\ \}$$

Note that, in rule **IntEv**, the whole intention i that generated the internal event needs to be inserted back in C_I, with p pushed onto the top of that intention. This is related to resuming *suspended intentions*; the suspending of intentions appears in rule **AchvGl** below.

Intention Selection: Rule $\mathbf{SelInt_1}$ assumes the existence of a function S_I that selects an intention for processing next, while rule $\mathbf{SelInt_2}$ takes care of the situation where the set of intentions is empty (in which case the reasoning cycle simply starts again).

$$\frac{C_I \neq \{\} \qquad S_I(C_I) = i}{\langle ag, C, M, T, \mathsf{SelInt}\rangle \longrightarrow \langle ag, C, M, T', \mathsf{ExecInt}\rangle} \qquad (\mathbf{SelInt_1})$$

$$\textit{where:} \quad T'_\iota \;=\; i$$

$$\frac{C_I = \{\}}{\langle ag, C, M, T, \mathsf{SelInt}\rangle \longrightarrow \langle ag, C, M, T, \mathsf{ProcMsg}\rangle} \qquad (\mathbf{SelInt_2})$$

Executing an Intention: The group of rules below express the effects of executing a formula in the body of the plan. Each rule deals with one type of formula that can appear in a plan body. Recall from Section 3.2 that an intention is a stack of (partially instantiated) plan instances; a plan instance is a copy of a plan from the agent's plan library). The plan instance to be executed is always the one at the top of the intention that was selected in the previous step (rule $\mathbf{SelInt_1}$); the specific formula to be executed is the one at the beginning of the body of that plan.

Actions: When the formula to be executed is an action, the action a in the body of the plan is added to the set of actions A (which, recall, denotes that the action is to be executed using the agent's effectors). The action is removed from the body of the plan and the intention is updated to reflect this removal.

$$\frac{T_\iota = i[head \leftarrow a;h]}{\langle ag, C, M, T, \mathsf{ExecInt}\rangle \longrightarrow \langle ag, C', M, T, \mathsf{ClrInt}\rangle} \quad (\mathbf{Action})$$

$$\begin{array}{llll} \textit{where:} & C'_A & = & C_A \cup \{a\} \\ & C'_I & = & (C_I \setminus \{T_\iota\}) \cup \{i[head \leftarrow h]\} \end{array}$$

Achievement Goals: This rule registers a new internal event in the set of events E. This event can then be selected for handling in a future reasoning cycle (see rule $\mathbf{SelEv_1}$). When the formula being executed is a goal, the formula is not removed from the body of the plan, as in the other cases. This only happens when the plan used for achieving that goal finishes successfully; see rule $\mathbf{ClrInt_2}$. The reasons for this are related to further instantiation of the plan variables as well as handling plan failure.

$$\frac{T_\iota = i[head \leftarrow !at;h]}{\langle ag, C, M, T, \mathsf{ExecInt}\rangle \longrightarrow \langle ag, C', M, T, \mathsf{ProcMsg}\rangle} \quad (\mathbf{AchvGl})$$

$$\begin{array}{llll} \textit{where:} & C'_E & = & C_E \cup \{\langle +!at, T_\iota\rangle\} \\ & C'_I & = & C_I \setminus \{T_\iota\} \end{array}$$

Note how the intention that generated the internal event is removed from the set of intentions C_I, capturing the idea of *suspended intentions*. In a plan body, if we have '$!g; f$' (where f is any formula that can appear in plan bodies), this means that, before f can be executed, the state of affairs represented by goal g needs to be achieved (through the execution of some relevant, applicable plan). The goal included in the new event created by rule **AchvGl** above is treated as any other event, which means it will go the set of events until it is eventually selected in a later reasoning cycle, according to the agent's specific priorities for selecting events (rule $\mathbf{SelEv_1}$). Meanwhile, that plan (with formula f to be executed next) can no longer be executed, hence the whole intention is suspended by being placed, within the newly created event, in the set of events and removed from the set of intentions. When the event created by the rule above is selected and an applicable plan for achieving g has been chosen, that intended means is pushed on top of the suspended intention, which can then be *resumed* (i.e., moved back to the set of intentions), according to rule **IntEv**. The next time that intention is selected, its execution will then proceed with a plan for achieving g at the top, and only when that plan is finished will f be executed (as that plan, now without the achieved goal, will be at the top of the intention again); further details on suspended intentions can be found in the AgentSpeak literature (e.g., see Bordini & Moreira, 2004).

Test Goals: These rules are used when a test goal formula $?at$ is to be executed. Rule $\mathbf{TestGl_1}$ is used when there is a set of substitutions that can make at a logical consequence of the agent's beliefs, which means that the test goal succeeded. If the test goal succeeds, the substitution is applied to the whole intended means, and the reasoning cycle can be continued. If that is not the case, it might turn out that the test goal is used as a triggering event of a plan, which is used by programmers to formulate more sophisticated queries. Rule $\mathbf{TestGl_2}$ is used in such case: it generates an internal event, which may trigger the execution of a plan, as for achievement goals. If to carry out a plan an agent is required to obtain information (at the time of actual execution of the plan) which is not directly available in its belief base, a plan for a test goal can be written which, for example, sends messages to other agents, or processes available data, so that the particular test goal can be concluded (producing an appropriate instantiation of logical variables). If an internal event is generated for the test goal being executed, the process is very similar to achievement goals, where the intention is suspended until a plan is selected to achieve the goal, as explained above.

$$\frac{T_\iota = i[head \leftarrow ?at;h] \qquad \mathsf{Test}(ag_{bs}, at) \neq \{\}}{\langle ag, C, M, T, \mathsf{ExecInt}\rangle \longrightarrow \langle ag, C', M, T, \mathsf{ClrInt}\rangle} \qquad (\mathbf{TestGl_1})$$

$$\begin{array}{lrcl} \textit{where:} & C'_I & = & (C_I \setminus \{T_\iota\}) \cup \{i[(head \leftarrow h)\theta]\} \\ & & & \theta \in \mathsf{Test}(ag_{bs}, at) \end{array}$$

$$\frac{T_\iota = i[head \leftarrow ?at;h] \qquad \mathsf{Test}(ag_{bs}, at) = \{\}}{\langle ag, C, M, T, \mathsf{ExecInt}\rangle \longrightarrow \langle ag, C', M, T, \mathsf{ClrInt}\rangle} \qquad (\mathbf{TestGl_2})$$

$$\begin{array}{lrcl} \textit{where:} & C'_E & = & C_E \cup \{\langle +?at, T_\iota\rangle\} \\ & C'_I & = & C_I \setminus \{T_\iota\} \end{array}$$

Updating Beliefs: In the rules below, the set of beliefs of the agent is modified in a way that either an atomic formula (with annotation `self`) is included in the new set of beliefs (rule **AddBel**) or it is removed from there (rule **DelBel**). Both rules add a new event to the set of events E, and update the intention by removing from it the $+b$ or $-at$ formula just executed. Note that belief deletions can have variables (at), whilst only ground atoms (b) can be added to the belief base.

$$\frac{T_\iota = i[head \leftarrow +b;h]}{\langle ag, C, M, T, \mathsf{ExecInt}\rangle \longrightarrow \langle ag', C', M, T, \mathsf{ClrInt}\rangle} \qquad (\mathbf{AddBel})$$

$$\begin{array}{lrcl} \textit{where:} & ag'_{bs} & = & ag_{bs} + b[\mathtt{self}] \\ & C'_E & = & C_E \cup \{\langle +b[\mathtt{self}], \mathsf{T}\rangle\} \\ & C'_I & = & (C_I \setminus \{T_\iota\}) \cup \{i[head \leftarrow h]\} \end{array}$$

$$\frac{T_\iota = i[head \leftarrow -at;h]}{\langle ag, C, M, T, \mathsf{ExecInt}\rangle \longrightarrow \langle ag', C', M, T, \mathsf{ClrInt}\rangle} \qquad (\mathbf{DelBel})$$

$$\begin{array}{lrcl} \textit{where:} & ag'_{bs} & = & ag_{bs} - at[\mathtt{self}] \\ & C'_E & = & C_E \cup \{\langle -at[\mathtt{self}], \mathsf{T}\rangle\} \\ & C'_I & = & (C_I \setminus \{T_\iota\}) \cup \{i[head \leftarrow h]\} \end{array}$$

Clearing Intentions: Finally, the following rules remove empty intended means or intentions from the set of intentions. Rule $\mathbf{ClrInt_1}$ simply removes a whole intention when there is nothing else to be executed in that intention. Rule $\mathbf{ClrInt_2}$ clears the remainder of the plan with an empty body currently at the top of a (non empty) intention. In this case, it is necessary to further instantiate the plan below the finished plan (currently at the top of that intention), and remove the goal that was left at the beginning of the body of the plan below (see rules **AchvGl** and **TestGl**). Note that, in this case, further "clearing" might be necessary, hence the next step is still ClrInt. Rule $\mathbf{ClrInt_3}$ takes care of the situation where no (further) clearing is required, so a new reasoning cycle can start (at step $\mathsf{ProcMsg}$).

$$\frac{j = [\mathit{head} \leftarrow \mathsf{T}], \text{ for some } j \in C_I}{\langle ag, C, M, T, \mathsf{ClrInt}\rangle \longrightarrow \langle ag, C', M, T, \mathsf{ProcMsg}\rangle} \qquad (\mathbf{ClrInt_1})$$

$$\textit{where:} \quad C'_I \;=\; C_I \setminus \{j\}$$

$$\frac{j = i[\mathit{head} \leftarrow \mathsf{T}], \text{ for some } j \in C_I}{\langle ag, C, M, T, \mathsf{ClrInt}\rangle \longrightarrow \langle ag, C', M, T, \mathsf{ClrInt}\rangle} \qquad (\mathbf{ClrInt_2})$$

$$\begin{array}{l} \textit{where:} \quad C'_I \;=\; (C_I \setminus \{j\}) \cup \{k[(\mathit{head}' \leftarrow h)\theta]\} \\ \textit{if } \; i = k[\mathit{head}' \leftarrow g;h] \;\; \textit{and} \;\; \theta \textit{ is s.t. } g\theta = \mathsf{TrEv}(\mathit{head}) \end{array}$$

$$\frac{j \neq [\mathit{head} \leftarrow \mathsf{T}] \wedge j \neq i[\mathit{head} \leftarrow \mathsf{T}], \text{ for } \textit{any } j \in C_I}{\langle ag, C, M, T, \mathsf{ClrInt}\rangle \longrightarrow \langle ag, C, M, T, \mathsf{ProcMsg}\rangle} \qquad (\mathbf{ClrInt_3})$$

4. Semantics of Communicating AgentSpeak Agents

The rules in the previous section give semantics to the key internal decision making and control aspects of AgentSpeak. Furthermore, the overall agent architecture will have sensors (with an associated belief revision function) and effectors, in addition to an AgentSpeak interpreter. The relation of these components to the AgentSpeak interpreter is not essential for giving a semantics to the language itself. It suffices to note that belief revision from perception of the environment adds (external) events to the set C_E (which is then used in the AgentSpeak interpretation cycle), while the effectors simply execute every action that is included by the reasoner in the set C_A.

Similarly, the mechanism that allows messages to be exchanged is part of the overall agent architecture — it is not part of its practical reasoning component, which is specifically what we program with AgentSpeak. The notion of internal actions in AgentSpeak (Bordini et al., 2002) is appropriate here: sending a message corresponds to executing the (predefined) internal action `.send` that appears in a plan body. The underlying agent architecture ensures that the necessary technical means is used for the message to reach the agent to which the message is addressed. However, as we will be referring to a special type of communication action that involves suspending intentions, we now need to include such details in the semantics.[4]

4. Some aspects of the whole framework are still not included in the formalisation given in this paper. We extend the semantics only to the point required for accounting for the semantics of speech-act based messages received by an agent.

The format of messages is $\langle mid, id, ilf, cnt\rangle$, where mid uniquely identifies the message, id identifies the agent to which the message is addressed (when the message is being sent) or the agent that has sent the message (when the message is being received), ilf is the illocutionary force (i.e., the performative) associated with the message, and cnt is the message content. Depending on the illocutionary force of the message, its content can be: an atomic formula (at); a set of formulæ (ATs); a ground atomic formula (b); a set of ground atomic formulæ (Bs); or a set of plans (PLs).

A mechanism for receiving and sending messages asynchronously is then defined. Messages are stored in a mail box and one of them is processed by the agent at the beginning of a reasoning cycle. Recall that, in a configuration of the transition system, M_{In} is the set of messages that the agent has received but has not processed yet, M_{Out} is the set of messages to be sent to other agents, and M_{SI} is a set of suspended intentions awaiting replies for (information request) messages previously sent. More specifically, M_{SI} is a set of pairs of the form (mid, i), where mid is a message identifier that uniquely identifies the previously sent message that caused intention i to be suspended.

When sending messages with illocutionary forces related to information requests, we have chosen a semantics in which the intention is suspended until a reply is received from the interlocutor, very much in the way that intentions get suspended when they are waiting for an internal event to be handled. With this particular semantics for "ask" messages, the programmer knows with certainty that any subsequent action in the body of a plan is only executed after the requested information has already been received. However, note that the information received as a reply is stored directly in the agent's belief base, so a test goal is required if the information is to be used in the remainder of the plan.

We now give two rules for executing the (internal) action of sending a message to another agent: the first is for the "ask" messages which require suspending intentions and the second is for other types of messages. These rules have priority over **Action**; although **Action** could also be applied on the same configurations, we assume the rules below are used if the formula to be executed is specifically a `.send` action. (We did not include this as a proviso in rule **Action** to improve readability.)

$$\frac{\begin{array}{c} T_\iota = i[head \leftarrow \texttt{.send}(id, ilf, cnt);h] \\ ilf \in \{AskIf, AskAll, AskHow\} \end{array}}{\langle ag, C, M, T, \mathsf{ExecInt}\rangle \longrightarrow \langle ag, C', M', T, \mathsf{ProcMsg}\rangle} \qquad (\textbf{ExecActSndAsk})$$

$$\begin{array}{llcl} \textit{where:} & M'_{Out} & = & M_{Out} \cup \{\langle mid, id, ilf, cnt\rangle\} \\ & M'_{SI} & = & M_{SI} \cup \{(mid, i[head \leftarrow h])\}, \\ & & & \text{with } mid \text{ a new message identifier;} \\ & C'_I & = & (C_I \setminus \{T_\iota\}) \end{array}$$

The semantics of sending other types of illocutionary forces is then simply to add a well-formed message to the agent's mail outbox (rule **ExecActSnd**). Note that in the rule above, as the intention is suspended, the next step in the reasoning cycle is $\mathsf{ProcMsg}$(i.e., a new cycle is started), whereas in the rule below it is ClrInt, as the updated intention — with the sending action removed from the plan body — might require "clearing", as with any of the intention execution rules seen in the previous section.

$$\frac{\begin{array}{c} T_\iota = i[head \leftarrow .\mathtt{send}(id, ilf, cnt);h] \\ ilf \notin \{AskIf, AskAll, AskHow\} \end{array}}{\langle ag, C, M, T, \mathsf{ExecInt}\rangle \longrightarrow \langle ag, C', M', T, \mathsf{ClrInt}\rangle} \qquad (\textbf{ExecActSnd})$$

$$\begin{array}{llll} \textit{where:} & M'_{Out} & = & M_{Out} \cup \{\langle mid, id, ilf, cnt\rangle\}, \\ & & & \text{with } mid \text{ a new message identifier;} \\ & C'_I & = & (C_I \setminus \{T_\iota\}) \cup \{i[head \leftarrow h]\} \end{array}$$

Whenever new messages are sent, we assume the system creates unique *message identifiers* (*mid*). Later, we shall see that, when replying to a message, the same message identifier is kept in the message, similar to the way that `reply-with` is used in KQML. Thus, the receiving agent is aware that a particular message is a reply to a previous one by checking the message identifiers in the set of intentions that were suspended waiting for a reply. This feature will be used when we give semantics to receiving *Tell* messages, which can be sent by an agent when it spontaneously wants the receiver to believe something (or at least to believe something about the sender's beliefs), but can also be used when the agent receives an "ask" type of message and chooses to reply to it.

As mentioned earlier, it is not our aim to formalise every aspect of a system of multiple AgentSpeak agents. We extend the previous semantics only to the extent required to formalise speech-act based communication for such agents. It is relevant, therefore, to consider a rule that defines message exchange as accomplished by the underlying message exchange mechanism available in an overall agent architecture. This is abstracted away in the semantics by means of the following rule, where each AG_{id_k}, $k = 1 \ldots n$, is an agent configuration $\langle ag_{id_k}, C_{id_k}, M_{id_k}, T_{id_k}, s_{id_k}\rangle$:

$$\frac{\langle mid, id_j, ilf, cnt\rangle \in M_{id_i Out}}{\begin{array}{c} \{AG_{id_1}, \ldots AG_{id_i}, AG_{id_j}, \ldots AG_{id_n}, env\} \longrightarrow \\ \{AG_{id_1}, \ldots AG'_{id_i}, AG'_{id_j}, \ldots AG_{id_n}, env\} \end{array}} \qquad (\textbf{MsgExchg})$$

$$\begin{array}{llll} \text{where:} & M'_{id_i Out} & = & M_{id_i Out} \setminus \{\langle mid, id_j, ilf, cnt\rangle\} \\ & M'_{id_j In} & = & M_{id_j In} \cup \{\langle mid, id_i, ilf, cnt\rangle\} \end{array}$$

In the rule above, there are n agents, and *env* denotes the environment in which the agents are situated; typically, this is not an AgentSpeak agent, it is simply represented as a set of properties currently true in the environment and how they are changed by an agent's actions. Note how, in a message that is to be sent, the second component identifies the addressee (the agent to which the message is being sent), whereas in a received message that same component identifies the sender of the message.

4.1 Speech-act Based Communication for AgentSpeak

In this section we discuss the performatives that are most relevant for communication in AgentSpeak. These are largely inspired by corresponding KQML performatives. We also consider some new performatives, related to plan exchange rather than communication about propositions as usual. The performatives that we consider are briefly described below, where s denotes the agent that sends the message, and r denotes the agent that receives the message. Note that `tell` and `untell` can be used either for an agent to pro-actively send information to another agent, or as replies to previous `ask` messages.

`tell`: s intends r to believe (that s believes) the sentence in the message's content to be true;

`untell`: s intends r not to believe (that s believes) the sentence in the message's content to be true;

`achieve`: s requests that r to intend to achieve a state of the world where the message content is true;

`unachieve`: s requests r to drop the intention of achieving a state of the world where the message content is true;

`tell-how`: s informs r of a plan (i.e., some know-how of s);

`untell-how`: s requests r to disregard a certain plan (i.e., to delete that plan from its plan library);

`ask-if`: s wants to know if the content of the message is true for r;

`ask-all`: s wants all of r's answers to a question (i.e., all the beliefs that unify with the message content);

`ask-how`: s wants all of r's plans for a particular triggering event (in the message content).

For processing messages, a new selection function is necessary, which operates in much the same way as the other selection functions described in the previous section. The new selection function is called S_M, and selects a message from M_{In}; intuitively, it represents the priority assigned to each type of message by the programmer. We also need another "given" function, but its purpose is different from selection functions. The Boolean function $\mathsf{SocAcc}(id, \mathit{ilf}, at)$, where *ilf* is the illocutionary force of the message from agent *id*, with propositional content *at*, determines when a message is *socially acceptable* in a given context. For example, for a message of the form $\langle mid, id, \mathit{Tell}, at\rangle$, the receiving agent may want to consider whether *id* is a relevant source of information, as even remembering that *id* believes *at* might not be appropriate. For a message with illocutionary force *Achieve*, an agent would normally check, for example, whether *id* has sufficient social power over itself, or whether it wishes to act altruistically towards *id*, before actually committing to do whatever it is being asked.

We should mention that the role of $\mathsf{SocAcc}()$ in our framework is analogous, on the receiver's side, to that of the "cause to want" and "cause to believe" predicates in Cohen and Perrault's plan-based theory of speech acts (1979). That is, it provides a bridge from the illocutionary force of a message to its perlocutionary force. The idea of having user-defined functions determining relations such as "trust" and "power" has already been used in practice by Bordini et al. (2003). Similar interpretations for the use of SocAcc when applied to other types of messages (e.g., *AskIf*) can easily be derived.

There is considerable work on elaborate conceptions of trust in the context of multi-agent systems, for example in the work of Castelfranchi and Falcone (1998). In our framework, more sophisticated notions of trust and power can be implemented by considering the annotation of the sources of information during the agent's practical reasoning rather than the simple use of SocAcc. The annotation construct facilitates determining, in a plan context, the source of a belief before that plan becomes an intended means.

Before we start the presentation of the semantic rules for communication, it is worth noting that, in this paper in particular, we do not consider *nested* annotations. Nested annotations allow the representation of beliefs about other agents' beliefs, or more generally situations in which an agent i was told φ by j, which in turn was told φ by k, and so forth.

4.2 Semantic Rules for Interpreting Received Messages

Receiving a Tell Message: A *Tell* message might be sent to an agent either as a reply or as an "inform" action. When receiving a *Tell* message as an inform (as opposed to a reply to a previous request), the AgentSpeak agent will include the content of the received message in its knowledge base and will annotate the sender as a source for that belief. Note that this corresponds, in a way, to what is specified as the "action completion" condition by Labrou and Finin (1994): the receiver will know about the sender's attitude regarding that belief. To account for the social aspects of multi-agent systems, we consider that social relations will regulate which messages the receiver will process or discard; this is referred to in the semantics by the SocAcc function, which is assumed to be given by the agent designer. The rule shows that the annotated belief is added to the belief base, and the appropriate event is generated.

$$\frac{\begin{array}{c} S_M(M_{In}) = \langle mid, id, \mathit{Tell}, Bs\rangle \\ (mid, i) \notin M_{SI} \text{ (for any intention } i) \\ \mathsf{SocAcc}(id, \mathit{Tell}, Bs) \end{array}}{\langle ag, C, M, T, \mathsf{ProcMsg}\rangle \longrightarrow \langle ag', C', M', T, \mathsf{SelEv}\rangle} \qquad (\mathbf{Tell})$$

$$\begin{array}{llcl} \textit{where:} & M'_{In} & = & M_{In} \setminus \{\langle mid, id, \mathit{Tell}, Bs\rangle\} \\ & \multicolumn{3}{l}{\text{and for each } b \in Bs :} \\ & ag'_{bs} & = & ag_{bs} + b[id] \\ & C'_E & = & C_E \cup \{\langle +b[id], \mathsf{T}\rangle\} \end{array}$$

Receiving a Tell Message as a Reply: This rule is similar to the one above, except that now the suspended intention associated with that particular message — given that it is a reply to a previous "ask" message sent by this agent — needs to be resumed. Recall that to resume an intention we just need to place it back in the set of intentions (C'_I).

$$\frac{\begin{array}{c} S_M(M_{In}) = \langle mid, id, \mathit{Tell}, Bs\rangle \\ (mid, i) \in M_{SI} \text{ (for some intention } i) \\ \mathsf{SocAcc}(id, \mathit{Tell}, Bs) \end{array}}{\langle ag, C, M, T, \mathsf{ProcMsg}\rangle \longrightarrow \langle ag', C', M', T, \mathsf{SelEv}\rangle} \quad (\textbf{TellRepl})$$

$$\begin{array}{llcl} \textit{where:} & M'_{In} & = & M_{In} \setminus \{\langle mid, id, \mathit{Tell}, Bs\rangle\} \\ & M'_{SI} & = & M_{SI} \setminus \{(mid, i)\} \\ & C'_I & = & C_I \cup \{i\} \\ & \multicolumn{3}{l}{\text{and for each } b \in Bs:} \\ & ag'_{bs} & = & ag_{bs} + b[id] \\ & C'_E & = & C_E \cup \{\langle +b[id], \mathsf{T}\rangle\} \end{array}$$

Receiving an Untell Message: When receiving an *Untell* message, the sender of the message is removed from the set of sources giving accreditation to the atomic formula in the content of the message. In case the sender was the only source for that information, the belief itself is removed from the receiver's belief base. Note that, as the atomic formula in the content of an *Untell* message can have uninstantiated variables, each belief in the agent's belief base that can be unified with that formula needs to be considered in turn, and the appropriate events generated.

$$\frac{\begin{array}{c} S_M(M_{In}) = \langle mid, id, \mathit{Untell}, ATs\rangle \\ (mid, i) \notin M_{SI} \text{ (for some intention } i) \\ \mathsf{SocAcc}(id, \mathit{Untell}, ATs) \end{array}}{\langle ag, C, M, T, \mathsf{ProcMsg}\rangle \longrightarrow \langle ag', C', M', T, \mathsf{SelEv}\rangle} \quad (\textbf{Untell})$$

$$\begin{array}{llcl} \textit{where:} & M'_{In} & = & M_{In} \setminus \{\langle mid, id, \mathit{Untell}, ATs\rangle\} \\ & \multicolumn{3}{l}{\text{and for each } b \in \{at\theta \mid} \\ & \multicolumn{3}{l}{\quad \theta \in \mathsf{Test}(ag_{bs}, at) \wedge at \in ATs\}} \\ & ag'_{bs} & = & ag_{bs} - b[id] \\ & C'_E & = & C_E \cup \{\langle -b[id], \mathsf{T}\rangle\} \end{array}$$

Receiving an Untell Message as a Reply: As above, the sender as source for the belief, or the belief itself, is excluded from the belief base of the receiver, except that now a suspended intention needs to be resumed (similarly to a *Tell* as a reply).

$$\frac{\begin{array}{c} S_M(M_{In}) = \langle mid, id, Untell, ATs\rangle \\ (mid, i) \in M_{SI} \text{ (for some intention } i) \\ \mathsf{SocAcc}(id, Untell, ATs) \end{array}}{\langle ag, C, M, T, \mathsf{ProcMsg}\rangle \longrightarrow \langle ag', C', M', T, \mathsf{SelEv}\rangle} \qquad (\mathbf{UntellRepl})$$

$$\begin{array}{llcl} \textit{where:} & M'_{In} & = & M_{In} \setminus \{\langle mid, id, Untell, ATs\rangle\} \\ & M'_{SI} & = & M_{SI} \setminus \{(mid, i)\} \\ & C'_I & = & C_I \cup \{i\} \\ & \multicolumn{3}{l}{\text{and for each } b \in \{at\theta \mid} \\ & \multicolumn{3}{l}{\qquad \theta \in \mathsf{Test}(ag_{bs}, at) \wedge at \in ATs\}} \\ & ag'_{bs} & = & ag_{bs} - b[id] \\ & C'_E & = & C_E \cup \{\langle -b[id], \mathsf{T}\rangle\} \end{array}$$

Receiving an Achieve Message: In an appropriate social context (e.g., if the sender has "power" over the receiver), the receiver 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. An external event is thus included in the set of events (recall that external events have the triggering event associated with the empty intention T).

Note that it is now possible to have a new focus of attention (a stack of plans in the set of intentions I) being initiated by the addition (or deletion, see below) of an achievement goal. Originally, only a belief change arising from perception of the environment initiated 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.

$$\frac{\begin{array}{c} S_M(M_{In}) = \langle mid, id, Achieve, at\rangle \\ \mathsf{SocAcc}(id, Achieve, at) \end{array}}{\langle ag, C, M, T, \mathsf{ProcMsg}\rangle \longrightarrow \langle ag, C', M', T, \mathsf{SelEv}\rangle} \qquad (\mathbf{Achieve})$$

$$\begin{array}{llcl} \textit{where:} & M'_{In} & = & M_{In} \setminus \{\langle mid, id, Achieve, at\rangle\} \\ & C'_E & = & C_E \cup \{\langle +!at, \mathsf{T}\rangle\} \end{array}$$

We shall later discuss in more detail the issue of autonomy. While this gives the impression that simply accepting orders removes the agent's autonomy (and similarly with regards to acquired beliefs), the way the agent will behave once aware that another agent is attempting to delegate a goal completely depends on the particular plans that happen to be in the agent's plan library. If a suitable plan exists, the agent could simply drop the goal, or could tell the interlocutor that the goal delegation was noted but the goal could not be adopted as expected, etc.

Receiving an Unachieve Message: This rule is similar to the preceding one, except that now the deletion (rather than addition) of an achievement goal is included in the set of events. The assumption here is that, if the agent has a plan with such a triggering event, then 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 have not been included in any formalisation of AgentSpeak as yet, even though it is already available in practice, for example in the ***Jason*** interpreter (Bordini & Hübner, 2007).

$$\frac{\begin{array}{c} S_M(M_{In}) = \langle mid, id, Unachieve, at\rangle \\ \mathsf{SocAcc}(id, Unachieve, at) \end{array}}{\langle ag, C, M, T, \mathsf{ProcMsg}\rangle \longrightarrow \langle ag, C', M', T, \mathsf{SelEv}\rangle} \qquad (\textbf{Unachieve})$$

$$\begin{array}{lrcl} \textit{where:} & M'_{In} & = & M_{In} \setminus \{\langle mid, id, Unachieve, at\rangle\} \\ & C'_E & = & C_E \cup \{\langle -!at, \mathsf{T}\rangle\} \end{array}$$

Receiving a Tell-How Message: The AgentSpeak notion of plan is related to Singh's concept of *know-how* (1994). Accordingly, we use the *TellHow* performative when agents wish to exchange know-how rather than communicate beliefs or delegate goals. That is, a *TellHow* message is used by the sender (an agent or external source more generally) to inform an AgentSpeak agent of a plan that can be used for handling certain types of events (as expressed in the plan's triggering event). If the source is trusted, the plans in the message content are simply added to the receiver's plan library.

$$\frac{\begin{array}{c} S_M(M_{In}) = \langle mid, id, TellHow, PLs\rangle \\ (mid, i) \notin M_{SI} \text{ (for any intention } i) \\ \mathsf{SocAcc}(id, TellHow, PLs) \end{array}}{\langle ag, C, M, T, \mathsf{ProcMsg}\rangle \longrightarrow \langle ag', C, M', T, \mathsf{SelEv}\rangle} \qquad (\textbf{TellHow})$$

$$\begin{array}{lrcl} \textit{where:} & M'_{In} & = & M_{In} \setminus \{\langle mid, id, TellHow, PLs\rangle\} \\ & ag'_{ps} & = & ag_{ps} \cup PLs \end{array}$$

Note that we do not include any annotation to identify the source of a plan, and so, with this semantics, it is not possible to take into account the identity of the agent that provided a plan when deciding whether to use it. In practice, this feature is implemented in the ***Jason*** interpreter, as the language is extended with the use of annotated predicates as plan labels. This also allows programmers to annotate plans with information that can be used for meta-level reasoning (e.g., choosing which plan to use in case various applicable plans are available, or which intention to execute next); examples of such information would be the expected payoff of a specific plan and its expected chance of success, thus allowing the use of decision-theoretic techniques in making those choices.

Receiving a Tell-How Message as a Reply: The *TellHow* performative as a reply will also cause the suspended intention — the one associated with the respective *AskHow* message previously sent — to be resumed.

$$\frac{\begin{array}{c} S_M(M_{In}) = \langle mid, id, TellHow, PLs\rangle \\ (mid, i) \in M_{SI} \text{ (for some intention } i) \\ \mathsf{SocAcc}(id, TellHow, PLs) \end{array}}{\langle ag, C, M, T, \mathsf{ProcMsg}\rangle \longrightarrow \langle ag', C', M', T, \mathsf{SelEv}\rangle} \qquad (\textbf{TellHowRepl})$$

$$\begin{array}{lrcl} \textit{where:} & M'_{In} & = & M_{In} \setminus \{\langle mid, id, TellHow, PLs\rangle\} \\ & M'_{SI} & = & M_{SI} \setminus \{(mid, i)\} \\ & C'_I & = & C_I \cup \{i\} \\ & ag'_{ps} & = & ag_{ps} \cup PLs \end{array}$$

Receiving an Untell-How Message: This is similar to the rule above, except that plans are now removed from the receiver's plan library. An external source may find that a plan is no longer appropriate for handling the events it was supposed to handle; it may then want to inform another agent about that. Thus, when receiving a socially acceptable *UntellHow* message, the agent removes the associated plans (i.e., those in the message content) from its plan library.

$$\frac{\begin{array}{c} S_M(M_{In}) = \langle mid, id, \mathit{UntellHow}, PLs\rangle \\ \mathsf{SocAcc}(id, \mathit{UntellHow}, PLs) \end{array}}{\langle ag, C, M, T, \mathsf{ProcMsg}\rangle \longrightarrow \langle ag', C, M', T, \mathsf{SelEv}\rangle} \qquad (\textbf{UntellHow})$$

$$\begin{array}{lrcl} \textit{where:} & M'_{In} & = & M_{In} \setminus \{\langle mid, id, \mathit{UntellHow}, PLs\rangle\} \\ & ag'_{ps} & = & ag_{ps} \setminus PLs \end{array}$$

Receiving an Ask-If Message: The receiver will respond to this request for information if certain conditions imposed by the social settings (the SocAcc function) hold between sender and receiver.

Note that *ask-if* and *ask-all* differ in the kind of request made to the receiver. With the former, the receiver should just confirm whether the predicate in the message content is in its belief base or not; with the latter, the agent replies with all the predicates in its belief base that unify with the formula in the message content. The receiver processing an *AskIf* message responds with the action of sending either a *Tell* (to reply positively) or *Untell* message (to reply negatively); the reply message has the same content as the *AskIf* message. Note that a reply is only sent if the social context is such that the receiver wishes to consider the sender's request.

$$\frac{\begin{array}{c} S_M(M_{In}) = \langle mid, id, \mathit{AskIf}, \{b\}\rangle \\ \mathsf{SocAcc}(id, \mathit{AskIf}, b) \end{array}}{\langle ag, C, M, T, \mathsf{ProcMsg}\rangle \longrightarrow \langle ag, C, M', T, \mathsf{SelEv}\rangle} \qquad (\textbf{AskIf})$$

where:

$$\begin{array}{lcl} M'_{In} & = & M_{In} \setminus \{\langle mid, id, \mathit{AskIf}, \{b\}\rangle\} \\ M'_{Out} & = & \left\{ \begin{array}{ll} M_{Out} \cup \{\langle mid, id, \mathit{Tell}, \{b\}\rangle\} & \text{if } ag_{bs} \models b \\ M_{Out} \cup \{\langle mid, id, \mathit{Untell}, \{b\}\rangle\} & \text{if } ag_{bs} \not\models b \end{array} \right. \end{array}$$

The role that S_M plays in the agent's reasoning cycle is slightly more important here than originally conceived (Moreira et al., 2004). An agent considers whether to accept a message or not, but the reply message is automatically assembled when the agent selects (and accepts) any of the "ask" messages. However, providing such a reply may require considerable computational resources (e.g., the whole plan library may need to be scanned and a considerable number of plans retrieved from it in order to produce a reply message). Therefore, S_M should normally be defined so that the agent only selects an *AskIf*, *AskAll*, or *AskHow* message if it determines the agent is not currently too busy to provide a reply.

Receiving an AskAll: As for *AskIf*, the receiver processing an *AskAll* has to respond either with *Tell* or *Untell*, provided the social context is such that the receiver will choose to respond. As noted above, here the agent replies with all the predicates in the belief base that unify with the formula in the message content.

$$\frac{\begin{array}{c} S_M(M_{In}) = \langle mid, id, AskAll, \{at\}\rangle \\ \mathsf{SocAcc}(id, AskAll, at) \end{array}}{\langle ag, C, M, T, \mathsf{ProcMsg}\rangle \longrightarrow \langle ag, C, M', T, \mathsf{SelEv}\rangle} \qquad (\mathbf{AskAll})$$

$$\begin{array}{lll} \textit{where:} & & \\ M'_{In} & = & M_{In} \setminus \{\langle mid, id, AskAll, \{at\}\rangle\} \\ M'_{Out} & = & \left\{ \begin{array}{ll} M_{Out} \cup \{\langle mid, id, Tell, ATs\rangle\}, & \\ \quad ATs = \{at\theta \mid \theta \in \mathsf{Test}(ag_{bs}, at)\} & \text{if } \mathsf{Test}(ag_{bs}, at) \neq \{\} \\ M_{Out} \cup \{\langle mid, id, Untell, \{at\}\rangle\} & \text{otherwise} \end{array} \right. \end{array}$$

Receiving an AskHow: The receiver of an *AskHow* has to respond with the action of sending a *TellHow* message, provided the social configuration is such that the receiver will consider the sender's request. In contrast to the use of *Untell* in *AskAll*, the response when the receiver knows no relevant plan (for the triggering event in the message content) is a reply with an empty set of plans.

$$\frac{\begin{array}{c} S_M(M_{In}) = \langle mid, id, AskHow, te\rangle \\ \mathsf{SocAcc}(id, AskHow, te) \end{array}}{\langle ag, C, M, T, \mathsf{ProcMsg}\rangle \longrightarrow \langle ag, C, M', T, \mathsf{SelEv}\rangle} \qquad (\mathbf{AskHow})$$

$$\begin{array}{lll} \textit{where:} & & \\ M'_{In} & = & M_{In} \setminus \{\langle mid, id, AskHow, te\rangle\} \\ M'_{Out} & = & M_{Out} \cup \{\langle mid, id, TellHow, PLs\rangle\} \\ & & \quad \text{and } PLs = \{p \mid (p, \theta) \in \mathsf{RelPlans}(ag_{ps}, te)\} \end{array}$$

When SocAcc Fails: All the rules above consider that the social relations between sender and receiver are favourable for the particular communicative act (i.e, they require SocAcc to be true). If the required social relation does not hold, the message is simply discarded — it is removed from the set of messages and ignored. The rule below is used for receiving a message from an untrusted source, regardless of the performative.

$$\frac{\begin{array}{c} S_M(M_{In}) = \langle mid, id, ilf, Bs\rangle \\ \neg\mathsf{SocAcc}(id, ilf, Bs) \\ (\text{with } ilf \in \{Tell, Untell, TellHow, UntellHow, \\ Achieve, Unachieve, AskIf, AskAll, AskHow\}) \end{array}}{\langle ag, C, M, T, \mathsf{ProcMsg}\rangle \longrightarrow \langle ag, C, M', T, \mathsf{SelEv}\rangle} \qquad (\mathbf{NotSocAcc})$$

$$\textit{where:} \quad M'_{In} \;=\; M_{In} \setminus \{\langle mid, id, ilf, Bs\rangle\}$$

When M_{In} is empty: This last semantic rule states that, when the mail inbox is empty, the agent simply goes to the next step of the reasoning cycle (SelEv).

$$\frac{M_{In} = \{\}}{\langle ag, C, M, T, \mathsf{ProcMsg}\rangle \longrightarrow \langle ag, C, M, T, \mathsf{SelEv}\rangle} \qquad (\mathbf{NoMsg})$$

4.3 Comments on Fault Detection and Recovery

As in any other distributed system, multi-agent systems can (and do) fail in the real world. Possibly even more so than typical distributed systems, given that multi-agent systems are normally used in dynamic, unpredictable environments. In such contexts, failures are expected to happen quite often, so agents need to recover from them in the best possible way. In the specific case of systems composed of AgentSpeak agents, failures can occur when, for instance, the agent for which a message has been sent has left the multi-agent system, cannot be contacted, or has ceased to exist (e.g., because of a machine or network crash). Equally, an intention that was suspended waiting for a reply may never be resumed again due to a failure in the agent that was supposed to provide a reply. (However, note that AgentSpeak agents typically have various concurrent foci of attention — i.e., multiple intentions currently in the set of intentions — so even if one particular intention can never progress because another agent never replies, the agent will simply carry on working on the other foci of attention.)

In the context of AgentSpeak agents, both fault detection and recovery start at the level of the infrastructure that supports the agent execution. This infrastructure can adopt techniques available in traditional distributed systems but with a fundamental difference: it is responsible for adding appropriate events signaling failures in the set C_E of external events, or possibly resuming suspended intentions and immediately making them fail if for example a message reply has timed out. Such events, when treated by the agent in its normal reasoning cycle, using the plan failure mechanism not formalised here but available in practical interpreters, will trigger a plan specifically written by the agent programmer which defines a strategy for failure recovery. Therefore, from the point of view of the formal semantics for AgentSpeak, failure recovery reduces to event handling and plan execution, and is partly the responsibility of the underlying execution infrastructure and partly the responsibility of programmers. We should note that various approaches for failure detection and recovery within multi-agent systems in particular appear in the literature (e.g., Jennings, 1995; Kumar & Cohen, 2000). They typically involve the use of special agents or plans defined to deal with failure.

A natural concern when we have a set of agents executing concurrently is that shared resources should always be left in a consistent state. This is, of course, a classical problem in concurrency, and is typically solved by atomically executing the parts of the code that access the shared resources. Many programming language have constructs that enable a programmer to guarantee the atomic execution of critical sections. In a multi-agent system written in AgentSpeak, atomicity is not immediately an issue since there are no critical sections, given that different AgentSpeak agents do not directly share memory. However, AgentSpeak agents do exchange information in the form of messages, but the responsibility for the management of such exchanges lies with the underlying message passing infrastructure. On the other hand, agents in a multi-agent systems typically share an environment, and if a particular application requires environment resources to be shared by agents, clearly programmers need to ensure that suitable agent interaction protocols are used to avoid dead-/live-locks or starvation.

Another possible source of concern regarding atomicity is the concurrent execution of an agent's intentions. Agents can have several intentions ready to be executed and each one of them can read/write data that is shared with other intentions (as they all access the same belief base). It is not in the scope of this paper to formalise mechanisms to control such concurrency, but it is worth mentioning that the ***Jason*** interpreter provides programmers with the possibility of annotating plans as being "atomic", so that when one of them is selected for execution (see the **SelInt** semantic rule), it is guaranteed by the runtime agent platform that the plan execution will not be suspended/interrupted (i.e., that no other intention will be selected for execution in the following reasoning cycles) before the whole plan finishes executing.

Next, we give an example intended to illustrate how the semantic rules are applied during a reasoning cycle. The example includes agents exchanging messages using the semantic framework for agent communication formalised earlier in this section.

5. Example of Reasoning Cycles of Communicating Agents

Consider the following scenario. Firefighting robots are in action trying to control a rapidly spreading fire in a building, under the supervision of a commander robot. Another robot is piloting a helicopter to observe in which direction the fire is spreading most rapidly. Let the robot in the helicopter be `r1`, let `r2` be the ground commander, and let `r3` be one of the firefighting robots.

One of the plans in `r1`'s plan library, which we shall refer to as `ps1`, is as shown in Figure 6. This plan says that as soon as `r1` perceives fire spreading in direction `D`, it tells `R` that fire is spreading towards `D`, where `R` is the agent it believes to be the ground commander. Plan `ps2` is one of the plans that robot `r2` (the commander) has in its plan library, which is also shown in Figure 6. Plan `ps2` says that, when `r2` gets to believe[5] that fire is spreading in direction `D`, it will request the robot believed to be closest to that part of the building to achieve a state of the world in which `D` is the fighting post of that robot (in other words, that the robot should relocate to the part of the building in direction `D`).

We now proceed to show how the rules of the operational semantics apply, by using one reasoning cycle of the AgentSpeak agent that controls `r1` as an example; the rules for communication will be exemplified afterwards. For simplicity, we assume `r1` is currently defined by a configuration $\langle ag_1, C_1, M_1, T_1, s_1\rangle$, with $s_1 = \mathsf{ProcMsg}$ and the ag_1 component having:

$$ag_{1_{bs}} = \{\texttt{commander(r2)}\}, \text{ and}$$

$$ag_{1_{ps}} = \{\texttt{ps1}\}.$$

Suppose `r1` has just perceived fire spreading towards the south. After the belief revision function (see Section 3.2) has operated, `r1`'s beliefs will be updated to $ag_{1_{bs}} = \{\texttt{commander(r2)}, \texttt{spreading(south)}\}$ and the C_{1E} component of `r1`'s configuration (i.e., its set of events) will be as follows:

$$C_{1E} = \{\langle \texttt{+spreading(south)}, \mathsf{T}\rangle\}.$$

5. Note that, because the plan's triggering event does not require a particular source for that information (as in, e.g., `+spreading(D)[percept]`), this plan can be used when such belief is acquired from communication as well as perception of the environment.

r1's plan ps1

```
+spreading(D)
   :  commander(R);
      <- .send(R,tell,spreading(D)).
```

r2's plan ps2

```
+spreading(D)
   :  closest(D,A);
      <-.send(A,achieve,fight_post(A,D)).
```

Figure 6: Plans used in the firefighting robots example.

At this point, we can show the sequence of rules that will be applied to complete one reasoning cycle: see Table 1, where the left column shows the rule being applied and the right column shows *only the components of the configuration which have changed* as a consequence of that rule having been applied. Note that the "next step" component s_1 changes in an obvious way (given the rules being applied), so we only show its change for the very first step, when there are no messages to be processed and the cycle goes straight on to selecting an event to be handled in that reasoning cycle.

Table 1: Example sequence of rules applied in one reasoning cycle.

Rule	Changed Configuration Components
NoMsg	$s_1 = \mathsf{SelEv}$
SelEv₁	$C_{1E} = \{\}$ $T_{1\varepsilon} = \langle \texttt{+spreading(south)}, \mathsf{T}\rangle$
Rel₁	$T_{1R} = \{(\texttt{ps1}, \theta_R)\}$, where $\theta_R = \{\texttt{D} \mapsto \texttt{south}\}$
Appl₁	$T_{1Ap} = \{(\texttt{ps1}, \theta_A)\}$, where $\theta_A = \{\texttt{D} \mapsto \texttt{south}, \texttt{R} \mapsto \texttt{r2}\}$
SelApp	$T_{1\rho} = (\texttt{ps1}, \theta_A)$
ExtEv	$C_{1I} = \{[\texttt{ps1}\theta_A]\}$
SelInt₁	$T_{1\iota} = [\texttt{ps1}\theta_A]$
ExecActSnd	$M_{1Out} = \{\langle mid_1, \texttt{r2}, \mathit{Tell}, \{\texttt{spreading(south)}\rangle\}\}$ $C_{1I} = \{[\texttt{+spreading(south) : commander(r2) <- } \mathsf{T}]\}$
ClrInt₁	$C_{1I} = \{\}$

After `r1`'s reasoning cycle shown in the table, rule **MsgExchg** applies and, assuming $\langle ag_2, C_2, M_2, T_2, s_2\rangle$ is `r2`'s configuration, which for simplicity we assume is the initial (i.e., empty) configuration hence $s_2 = \mathsf{ProcMsg}$, we shall have that:

$$M_{2In} = \{\langle mid_1, \texttt{r1}, \mathit{Tell}, \{\texttt{spreading(south)}\}\rangle\},$$

which then leads to rule **Tell** being applied, thus starting a reasoning cycle (similar to the one in Table 1) in `r2` from a configuration that will have had the following components changed (see Rule **Tell**):

$$M_{2In} = \{\}$$

$$ag_{2\,bs} = \{\texttt{spreading(south)[r1]}\}$$

$$C_{2E} = \{\langle \texttt{+spreading(south)[r1]}, \mathsf{T}\rangle\}.$$

After a reasoning cycle in `r2`, we would have for `r3` that:

$$M_{3In} \quad = \quad \{\langle mid_2, \texttt{r2}, \mathit{Achieve}, \texttt{fight\_post(r3,south)}\rangle\},$$

where M_{3In} is `r3`'s mail inbox (and assuming M_{3In} was previously empty). Note that `r3`'s SocAcc function used by rule **Achieve** (leading to mid_2 being included in M_{3In} as stated above) would probably consider the hierarchy determined by the firefighters' ranks. Robot `r3` would then consider the events generated by this received message in its subsequent reasoning cycles, and would act in accordance to the plans in its plan library, which we do not show here, for simplicity.

6. Developing More Elaborate Communication Structures

We sometimes require more elaborate communication structures than performatives such as those discussed in Section 4. On the other hand, it is of course important to keep any communication scheme and its semantic basis as simple as possible. We emphasise that, in our approach, more sophisticated communication structures can be programmed, on top of the basic communication features formalised here, through the use of plans that implement interaction protocols. In practical AgentSpeak interpreters, such communication features (built by composing of the atomic performatives) can be provided to programmers either as extra pre-defined performatives or as plan templates in plan libraries made publicly available — in ***Jason*** (Bordini & Hübner, 2007), the former approach has been used, but the latter is also possible. In this section, we give, as examples of more advanced communication features, plans that allow agents to reach shared beliefs and ensure that agents are kept informed of the adoption of their goals by other agents. Note however that the examples make use of a simple practical feature (available, e.g., in ***Jason***) which does not appear in the abstract syntax we used earlier in the formal presentation: a variable instantiated with a first order term can be used, within certain constructs (such as belief or goal additions), in place of an atomic formula, as usual also in Prolog implementations.

Example 1 (Shared Beliefs) *If a network infrastructure is reliable, it is easy to ensure that agents reach shared beliefs. By reaching a shared belief, we mean two agents believing b as well as believing that the other agent also believes b. More explicitly, we can say agents `ag1` and `ag2` share belief b if `ag1` believes both b[`self`] and b[`ag2`], at the same time that `ag2`* *believes both b[`self`] and b[`ag1`]. In order to allow agents `ag1` and `ag2` to reach such shared beliefs, it suffices*[6] *to provide both agents with copies of the following plans:*

```
rsb1
+!reachSharedBel(P,A) : not P[self]
  <- +P;
     !reachSharedBel(P,A).

rsb2
+!reachSharedBel(P,A) : P[self] & not P[A]
  <- .send(A,tell,P);
     .send(A,achieve,reachSharedBel(P,me)).

rsb3
+!reachSharedBel(P,A) : P[self] & P[A]
  <- true.
```

In the plans above, *me* stands for the agent's own name. (Recall that, as in Prolog, an uppercase initial denotes a logical variable.) Assume agent `ag1` has the above plans and some other plan, an instance of which is currently in its set of intentions, which requires itself and `ag2` to share belief `p(X)`. Such a plan would have the following goal in its body: `!reachSharedBel(p(X),ag2)`. This would eventually lead to the execution of the plans in the example above, which can now be explained. The plan labelled `rsb1` says that if `ag1` has a (new) goal of reaching a shared belief `P` with agent `A`, in case `ag1` does not yet believe `P` itself, it should first make sure itself believes `P` — note that '`+P;`' in the body of that plan will add the ground predicate bound to `P` with source `self` as a new belief to agent `ag1` — then it should again have the goal of reaching such shared belief (note that this is a recursive plan). This time, plan `rsb1` will no longer be applicable, so `rsb2` will be chosen for execution. Plan `rsb2` says that, provided `ag1` believes `P` but does not yet believe that agent `A` believes `P`, it should tell agent `A` that itself (`ag1`) believes `P`, then finally ask `A` to also achieve such shared belief with `ag1`.

Agent `ag2`, which also has copies of the plans in the example above, would then, given the appropriate SocAcc function, have an instance of plan `rsb1` in its own set of intentions, and will eventually execute `rsb2` as well, or directly `rsb2` as the case may be. Note that the last line of plan `rsb2`, when executed by the agent that was asked to reach a shared believe, rather than the one who took the initiative, is redundant and will lead the other agent to using `rsb3`, which only says that no further action is required, given that the shared belief has already been obtained. Clearly, there are more efficient ways of implementing a protocol for reaching shared belief, but we present this because the same plans can be used regardless of whether the agent takes the initiative to reach a shared belief or not. The version we give here is therefore arguably more elegant, and its symmetry facilitates reasoning about the protocol.

6. Even if both agents do not have such plans in advance, but are willing to be told how to reach shared beliefs (by accepting *TellHow* messages from agents who have such know-how), they can become capable of reaching shared beliefs too.

We now give another example, which shows how agents can have further information about requests for goal adoption (i.e., when they ask another agent to achieve some state of affairs on their behalf).

Example 2 (Feedback on Goal Adoption) *It is often the case that, if one agent asks another agent to do something, it may want to have at least some reassurance from the other agent that it has agreed to do whatever it has been asked. Furthermore, it may want to know when the other agent believes it has accomplished the task. The following plans can be used for `ag1` to delegate tasks to `ag2` in such a way.*

ag1 plans:

```
nsd1
+needDoneBy(G,A) : not delegatedTo(G,A)
  <- +delegatedTo(G,A);
     .send(A,achieve,doAndFeedbackTo(G,me)).

nsd2
+needDoneBy(G,A)
  : agreedToDo(G)[A] & not finishedDoing(G)[A]
    <- .send(A,tell,shouldHaveFinished(G));
       ...

nsd3
+needDoneBy(G,A) : finishedDoing(G)[A]
  <- ...

...

fd
+finishedDoing(G)[A] : true
  <- -delegatedTo(G,A);
     -agreedToDo(G)[A].
```

ag2 plans:

```
dft1
+!doAndFeedbackTo(G,A) : cntxt1
  <- .send(A,tell,agreedToDo(G));
     +!G;
     .send(A,tell,finishedDoing(G)).

dft2
+!doAndFeedbackTo(G,A) : cntxt2
  <- .send(A,tell,cannotDo(G)).
```

In the example above, we assume that something perceived in the environment leads the agent to believe that it needs some goal `G` to be achieved by agent `A`, and that such perception recurs at certain intervals, when the need that motivated the request still exists and the result of `A`'s achieving `G` has not been observed. Plan `nsd1` is used when such a need occurs but no request has been as yet sent to `A`. The plan ensures that `ag1` will remember that it already asked `A` (say, `ag2`) to do `G` and then that agent to achieve a goal associated with a special plan: see plan `dft1` in `ag2`. Such plan makes sure that the requesting agent is informed both that `ag2` has adopted the goal as requested (before it attempts to achieve it) as well as when the agent believes to have achieved `G`. The programmer should define the SocAcc function so that `ag2` accepts such requests from `ag1`, but the programmer can still determine how autonomous `ag2` will be by using appropriate plan contexts. In plan `dft1`, context $cntxt_1$ would determine the circumstances under which agent `ag2` believes it will be able to adopt the goal, and context $cntxt_2$, in plan `dft2`, can be used for the circumstances in which `ag2` should simply inform it will not adopt the goal as requested by `ag1` (a more elaborate plan could explain why the agent cannot adopt the goal, for example in case there are more than one situation in which the goal cannot be adopted).

Going back to plans `nsd2` and `nsd3` in agent `ag1`, the former is used to "put pressure" on the agent that has adopted `ag1`'s goal `G`, as the need for that has been perceived again and `A` has already agreed to do that, so presumably it is not doing it fast enough. Clearly, the "`shouldHaveFinished`" belief should trigger some plan in `ag2` for it to have the desired effect. Plan `nsd3` is just a template for one of various alternative courses of actions to be taken by `ag1` when the need that motivated a request for `ag2` to adopt a goal still exists but `ag2` believes the goal has already been achieved: that might be an old belief which needs to be revised and a new request made, or `ag1` could try asking another agent, or inform `ag2` that its belief about achieving `G` might be wrong, etc. Plan `fd` is used simply to remove unnecessary beliefs used in previous stages of the interaction aimed at a goal adoption.

It is not difficult to see that plans for other important multi-agent issues, such as ensuring agents are jointly committed to some course of action, can be developed by elaborating on the combinations of communication performatives along the lines of the examples above. On the other hand, many other complications related to agent interaction might need to be accounted for which could not be addressed in the simple examples provided here, such as shared beliefs becoming inaccurate with the passage of time. Further plans to go with the ones shown here would be required for coping with such complications, when necessary in particular applications.

7. Proving Communication Properties of AgentSpeak Agents

Bordini and Moreira (2004) introduced a framework for proving BDI properties of AgentSpeak agents based on its operational semantics. The framework included precise definitions of how the BDI modalities are interpreted in terms of configurations of the transition system that gives semantics to AgentSpeak. Those same definitions are used in the work on model checking for AgentSpeak (Bordini et al., 2004), which allows the use of automated techniques for verification of AgentSpeak programs. Below, we give an example of a proof using the operational semantics for a simple property that involves only the *belief* modality. As the belief modality is very clear with respect to an AgentSpeak agent, given that its architecture includes a belief base explicitly, we avoid the need to discuss in this paper our previous work on the interpretation of the modalities (Bordini & Moreira, 2004).

Proposition 1 (Reachability of Shared Beliefs) *If any two AgentSpeak agents ag_1 and ag_2 have in their plan libraries the* `rsb1`*,* `rsb2`*, and* `rsb3` *plans shown in Example 1, and they also have an appropriate* SocAcc *function as well as the usual implementation of selection functions (or others for which fairness is also guaranteed, in the sense that all events and intentions are eventually selected), if at some moment in time ag_1 has* `reachSharedBel(`b`,`ag_2`)` *as a goal in its set of events (i.e., it has an event such as* $\langle$`+!reachSharedBel(`b`,`ag_2`)`$, i\rangle$*, with i an intention), then* eventually both agents will believe b and believe that the other agent also believes b — *note that this can be formulated using a BDI-like logic on top of LTL as*

$$\Diamond((\mathsf{Bel}\ ag_1\ b[self]) \wedge (\mathsf{Bel}\ ag_2\ b[ag_1]) \wedge (\mathsf{Bel}\ ag_2\ b[self]) \wedge (\mathsf{Bel}\ ag_1\ b[ag_2])).$$

Proof. It is assumed that ag_1 has $\langle +!\texttt{reachSharedBel}(b, ag_2), i\rangle$ in its set of events. Assume further that this is precisely the event selected when rule $\mathbf{SelEv_1}$ is applied. Then rule $\mathbf{Rel_1}$ would select plans `rsb1`, `rsb2`, and `rsb3` as relevant for the chosen event. Rule $\mathbf{Appl_1}$ would narrow this down to `rsb1` only as, presumably, ag_1 does not yet believe b itself. Rule **SelAppl** would necessarily select `rsb1` as intended means, given that it is the only applicable plan, and rule **IntEv** would include $i[\texttt{rsb1}]$ in the set of intentions (i.e., the chosen intended means would be pushed on top of the intention that generated the event above). Consider now that in this same reasoning cycle (for simplicity), rule $\mathbf{SelInt_1}$ would choose precisely that intention for execution within this reasoning cycle. Then rule **AddBel** would add $b[self]$ to ag_1's belief base, hence $(\mathsf{Bel}\ ag_1\ b[self])$.

In subsequent reasoning cycles, when ag_1's intention selection function selects the above intention for further execution, rule **AchvGl** would generate again an internal event $\langle +!\texttt{reachSharedBel}(b, ag_2), i\rangle$. The process is then as above expect that plan `rsb1` is no longer applicable, but `rsb2` is, and is therefore chosen as intended means. When that plan is executed (similarly as described above), rule **ExecActSnd** would add message $\langle mid_1, ag_2, \mathit{Tell}, b\rangle$ to ag_1's M_{Out} component. Rule **MsgExchg** then ensures that message $\langle mid, ag_1, \mathit{Tell}, b\rangle$ is added to ag_2's M_{In} component, which in the beginning of the next reasoning cycle would lead to rule **Tell** adding $b[ag_1]$ to ag_2's belief base, hence $(\mathsf{Bel}\ ag_2\ b[ag_1])$. When the intention is selected for execution in a third reasoning cycle, the final formula in the body of plan `rsb1` would be executed. By the use of similar rules for sending and receiving messages, we would have ag_2 receiving a message $\langle mid_2, ag_1, \mathit{Achieve}, \texttt{reachSharedBel}(b, ag_1)\rangle$, so now rule **Achieve** is used for interpreting the illocutionary force in that message, thus adding an event $\langle +!\texttt{reachSharedBel}(b, ag_1), i\rangle$ to ag_2's set of events. Note that this is precisely how the process started in ag_1 so the same sequence of rules will apply to ag_2, which will, symmetrically, lead to $(\mathsf{Bel}\ ag_2\ b[self])$ and $(\mathsf{Bel}\ ag_1\ b[ag_2])$ being true, eventually. At that point in time we will have $((\mathsf{Bel}\ ag_1\ b[self]) \wedge (\mathsf{Bel}\ ag_2\ b[ag_1]) \wedge (\mathsf{Bel}\ ag_2\ b[self]) \wedge (\mathsf{Bel}\ ag_1\ b[ag_2]))$. □

As discussed earlier, because ag_2 is using exact copies of the plans used by ag_1, ag_2 will also ask ag_1 to reach b as a shared belief, even though ag_1 has already executed its part of the joint plan. This is why plan `rsb3` is important. It ensures that the agent will act no further when its own part of the joint plan for reaching a shared belief has already been achieved.

Note, however, that it is only possible to guarantee that a shared belief is reached in *all possible runs* if neither agent has plans that can interfere negatively with the execution of the plans given in Example 1, for example by forcing the deletion of any instance of belief b before such shared belief is reached. This is a verification exercise different from the proposition we wanted to prove, showing that shared beliefs *can* be reached (under the given assumptions).

8. Applications of AgentSpeak and Ongoing Work

We mention here some of the applications written in AgentSpeak. The AgentSpeak programming language has also been used in academia for student projects in various courses. It should be noted, however, that the language is clearly suited to a large range of applications for which it is known that BDI systems are appropriate; various applications of PRS (Georgeff & Lansky, 1987) and dMARS (Kinny, 1993), for example, have appeared in the literature (Wooldridge, 2002, Chapter 11).

One particular area of application in which we have great interest is Social Simulation (Doran & Gilbert, 1994). In fact, AgentSpeak is being used as part of a project to produce a platform tailored particularly for social simulation. The platform is called MAS-SOC is being developed by Bordini, da Rocha Costa, Hübner, Moreira, Okuyama, and Vieira (2005); it includes a high-level language called ELMS (Okuyama, Bordini, & da Rocha Costa, 2005) for describing environments to be shared by multiple agents. This approach was used to develop, for example, a social simulation on social aspects of urban growth (Krafta, de Oliveira, & Bordini, 2003). Another area of application that has been initially explored is the use of AgentSpeak for defining the behaviour of animated characters for computer animation or virtual reality environments (Torres, Nedel, & Bordini, 2004).

More recently, AgentSpeak has been used in the implementation of a team of "gold miners" as an entry to an agent programming competition (Bordini, Hübner, & Tralamazza, 2006). In this scenario[7], teams of agents must coordinate their actions in order to collect as much gold as they can and to deliver the gold to a trading agent located in a depot where the gold is safely stored. The AgentSpeak team, composed of four mining agents and one leader that helped coordinate the team of miners, won the competition in 2006. It is worth noting that the language support for high-level communication (formalised in this paper) proved to be an important feature for designing and implementing the system.

The AgentSpeak interpreter and multi-agent platform ***Jason*** is being constantly improved, with the long term goal of supporting various multi-agent systems technologies. An important aspect of ***Jason*** is precisely that of having formal semantics for most of its essential features. Various projects are currently looking at extending ***Jason*** in various ways, for example to combine it with an organisational model such as the one propose by Hübner, Sichman, and Boissier (2004). This is particularly important given that social structure is a fundamental notion for developing complex multi-agent systems. Another area of development is to incorporate ontologies into an AgentSpeak belief base (Moreira,

7. See `http://cig.in.tu-clausthal.de/CLIMAContest/` for details.

Vieira, Bordini, & Hübner, 2006; Vieira, Moreira, Bordini, & Hübner, 2006), facilitating the use of ***Jason*** for Semantic Web applications. Recent work has also considered automated belief revision (Alechina, Bordini, Hübner, Jago, & Logan, 2006) and plan exchange mechanisms (Ancona et al., 2004). A more detailed description of the language and comparison with other agent-oriented programming languages was given by Bordini et al. (2005).

9. Conclusions

As pointed out by Singh (1998), there are various perspectives for the semantics of agent communication. Whereas the sender's perspective is the most common one in the literature, our approach uses primarily that of the receiver. We have given a formal semantics to the processing of speech-act based messages by an AgentSpeak agent. Previous attempts to define the semantics of agent communication languages (e.g., Labrou & Finin, 1994) were based on the "pre-condition – action – post-condition" approach, referring to agent mental states in modal languages typically based on Cohen and Levesque's work on intention (1990a). Our semantics for communication, besides being more closely linked to implementation (as it serves as the specification for an interpreter for an agent programming language), can also be used in the proof of communication properties (Wooldridge, 2000c).

Our work is somewhat related to that of de Boer, van Eijk, Van Der Hoek, and Meyer (2000) and van Eijk, de Boer, Van Der Hoek, and Meyer (2003), which also provide an operational semantics for an agent communication language. However, their work does not consider the effects of communication in terms of BDI-like agents (such as those written in AgentSpeak). The idea of giving semantics to speech-act based communication within a BDI programming language was first introduced by Moreira et al. (2004). Subsequently, Dastani, van der Ham, and Dignum (2003) also published some initial work on the semantics of communication for 3APL agents, although with the emphasis being on formalising the message exchange mechanisms for synchronous and asynchronous communication. In contrast, we largely abstract away from the specific message exchange mechanism (this is formalised at a very high level in our semantics), and we are interested only in asynchronous communication (which is the usual communication model for cognitive agents). In order to illustrate their message exchange mechanism, Dastani et al. gave semantics to the effects of receiving and treating "request" and "inform" messages — that is, they only consider information exchange. Our work uses a much more comprehensive selection of illocutionary forces, and the main contribution is precisely in giving detailed semantics to the ways in which the various illocutionary forces affect the mental states of agents implemented in a programming language which actually has precise definitions for the notions of the BDI architecture. A denotational semantics for agent communication languages was proposed by Guerin and Pitt (2001), but the semantics is given for an abstract version of an ACL and does not address the issues of interaction between an ACL and other components of an agent architecture.

In this paper we provided new semantic rules for all the illocutionary forces used in a communication language for AgentSpeak agents. In giving semantics to communicating AgentSpeak agents, we have provided the means for the implementation of AgentSpeak interpreters with such functionality, as well as given a more computationally grounded semantics of speech-act based agent communication. In fact, the operational semantics presented in this paper proved useful in the implementation of AgentSpeak interpreters such as ***Jason*** (Bordini & Hübner, 2007). While Singh's proposal for a social-agency based semantics (1998) may be appropriate for general purpose agent communication languages such as FIPA or KQML, within the context of a BDI agent programming language, our approach can be used without any of the drawbacks pointed out by Singh.

The fact that we have to deal with the intentional states of other agents when giving semantics of communication leads us to a number of related pragmatic questions. First, many treatments of speech-act style communication make use of *mutual* mental states — mutual belief, common knowledge, and similar. We do not make use of mutual mental states in our formalisation. There are good reasons for this. First, although mutual mental states are a useful and elegant tool for *analysis*, it is known that they represent *theoretical idealisations* only, which *cannot be achieved in systems which admit the possibility of message delivery failure* (Halpern, 1990). Thus, although mutual mental states are a useful abstraction for understanding how communication works, they cannot, realistically, be implemented, as there will always be a mismatch between the implementation (which excludes the possibility of mutual mental states being faithfully implemented) and the theory. This is primarily why mutual mental states form no part of our language or semantics, but are built on top of the fundamental communication primitives that we formalised in this paper, as shown in Section 6. Note that it is also known that mutual mental states can be *simulated*, to any desired degree of nesting, by an appropriate message acknowledgement scheme (Halpern & Zuck, 1992), therefore in our approach this problem can be solved by mechanisms such as processed messages triggering the action of sending a message that acknowledges receipt. It is also worth adding that the belief annotation scheme used in our language permits agents to have a simple mechanism for nested beliefs: the annotation of source in a belief is an indication that the agent who sent the message believed in its propositional content at the time the message was sent (but note that this is an indication only, unless agent veracity is guaranteed). Annotation of information source at the time a message is received is done automatically according to the semantics we have given. However, programmers can also use the belief base to register sent messages, possibly using annotations in the same manner as for received messages. These would function as an indication of other agents' states of mind, but from the point of view of the sender. We plan to deal with these questions which lie in the gray area between semantics and pragmatics in more detail in future work.

While discussing models of mutual mental states, we should also mention in passing that *joint intentions* do not form part of our semantics, although they are widely used in the implementation of coordination schemes for multi-agent systems, following the seminal work of Levesque, Cohen, and Nunes (1990). The fact that such constructs are not built into the language (or the language semantics) as primitives does not preclude them being *implemented* using the language constructs, provided the usual practical considerations and assumptions, such as limiting the number of required acknowledgement messages for the achievement of shared beliefs, are in place. Indeed, this is exactly the approach taken by Tambe, in his STEAM system (1997), and Jennings, in his GRATE* system (1995). The examples in Section 6 help indicate how this can be achieved by further elaboration of those plans, making use of the communication primitives for which we gave semantics in this paper.

We anticipate that readers will ponder whether our semantics limits the autonomy of agents that use our approach to communication. We provide the SocAcc() function which works as an initial "filter", but this may give the impression that beliefs are just acquired/trusted and goals adopted after such simple filter. It is very important to emphasise that the actual *behaviour* of the agent ensuing from communication received from other agents completely depends on the particular plans the agent happens to have in its plan library; in the current semantics, only the "ask" variants, *TellHow*, and *UntellHow* performatives are dependent solely on the SocAcc filter. In Example 2, we mentioned that some plan contexts should be used to determine whether the agent would actually act towards achieving a goal as requested by another agent, or choose not to commit to achieving the goal. This is the general rule: the agent autonomy depends on the plans given by the agent programmer or obtained by communication with other agents (the plans currently in the agent's plan library). It would be typically the programmer's responsibility to write plans that ensure that an agent will be "sufficiently autonomous" for its purpose in a given application or, to use a more interesting notion, to program agents with *adjustable autonomy*. Similarly, how benevolent or self-interested an agent will be, and to what extent beliefs acquired from other agents are to be trusted, are all issues that *programmers* have to be careful about: the semantics of communication itself does not ensure one case or the other. Needless to say, it will be a much more difficult task to program agents to take part in open systems where other agents are self-interested and cannot be trusted. While an agent programming language combined with a suitable agent communication language gives much support for such task, it surely does not automatically solve all those problems; it still remains a complex programming task.

It is also worth commenting on how our semantics can be used by other researchers, particularly those using agent programming languages other than AgentSpeak. The main point here is that our semantics provides a *reference* to the semantics of the communication language used in the context of agent-oriented programming. That is, using our semantics, it is possible to predict exactly how a particular AgentSpeak agent would interpret a particular message in a given situation. Using this as a reference model, it should in principle be possible to implement communication for other agent programming languages. Of course, our semantics is not language independent: it was developed specifically for AgentSpeak, so language specifics ought to be considered. However, attempts at giving semantics of agent communication that are language independent have their own problems, most notably the computational grounding problem referred to above. Our semantics, while developed specifically for a practical agent programming language, have the advantage of not relying on mechanisms (such as abstractly defined mental states) that cannot be checked for real programs. We note that, to the best of our knowledge, our work represents the first semantics given for a speech-act style, "knowledge level" communication language that is used in a real system.

Our current work does not consider commissive and declarative speech acts. These are surely relevant topics for future work, since commissive acts and declarations are relevant for various forms of agent interaction, such as negotiation. Nevertheless, in the proposed framework it is possible for the programmer or multi-agent system designer to incorporate such more elaborate forms of interactions by writing appropriate plans.

In this work, we assume that communication occurs among agents written in the same programming language, and cannot be adopted directly in heterogeneous multi-agent systems. (Consider, for example, the issues arising in processing an *AskHow* performative, which involves sending a plan to another agent.) However, for a variety of other agent languages, it should not be difficult to write "wrappers" for translating message contents.

Other relevant areas for future investigation are those regarding role definitions and social structures or agent organisations. We consider that these would be interesting developments of the proposed SocAcc() function and libraries of plans or plan patterns. Deontic relationships and social norms are also closely related to such extensions. In the case of e-business, for instance, a contract usually creates a number of obligations for the contractors.

Future work should also consider giving a better formal treatment of information sources, in particular for the case of plans being exchanged between agents. Further communication aspects such as ontological agreement among AgentSpeak agents, and reasoning about information sources (e.g., in executing test goals or choosing plans based on annotations) will also be considered in future work. We further expect sophisticated multi-agent system applications to be developed with AgentSpeak interpreters implemented according to our semantics.

Acknowledgements

Many thanks to Jomi F. Hübner for his comments and suggestions on an earlier version of this paper, and to Berndt Farwer and Louise Dennis who carefully proofread it. The first and second authors acknowledge the support of CNPq.

References

Alechina, N., Bordini, R. H., Hübner, J. F., Jago, M., & Logan, B. (2006). Automating belief revision for agentspeak. In Baldoni, M., & Endriss, U. (Eds.), *Proceedings of the Fourth International Workshop on Declarative Agent Languages and Technologies (DALT 2006), held with AAMAS 2006, 8th May, Hakodate, Japan*, pp. 1–16.

Allen, J. F., Hendler, J., & Tate, A. (Eds.). (1990). *Readings in Planning*. Morgan Kaufmann.

Ancona, D., Mascardi, V., Hübner, J. F., & Bordini, R. H. (2004). Coo-AgentSpeak: Cooperation in AgentSpeak through plan exchange. In Jennings, N. R., Sierra, C., Sonenberg, L., & Tambe, M. (Eds.), *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-2004), New York, NY, 19–23 July*, pp. 698–705 New York, NY. ACM Press.

Austin, J. L. (1962). *How to Do Things with Words*. Oxford University Press, London.

Ballmer, T. T., & Brennenstuhl, W. (1981). *Speech Act Classification: A Study in the Lexical Analysis of English Speech Activity Verbs*. Springer-Verlag, Berlin.

Bordini, R. H., Bazzan, A. L. C., Jannone, R. O., Basso, D. M., Vicari, R. M., & Lesser, V. R. (2002). AgentSpeak(XL): Efficient intention selection in BDI agents via decision-theoretic task scheduling. In Castelfranchi, C., & Johnson, W. L. (Eds.), *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-2002), 15–19 July, Bologna, Italy*, pp. 1294–1302 New York, NY. ACM Press.

Bordini, R. H., da Rocha Costa, A. C., Hübner, J. F., Moreira, Á. F., Okuyama, F. Y., & Vieira, R. (2005). MAS-SOC: a social simulation platform based on agent-oriented programming. *Journal of Artificial Societies and Social Simulation*, *8*(3). JASSS Forum, <http://jasss.soc.surrey.ac.uk/8/3/7.html>.

Bordini, R. H., Fisher, M., Pardavila, C., & Wooldridge, M. (2003). Model checking AgentSpeak. In Rosenschein, J. S., Sandholm, T., Wooldridge, M., & Yokoo, M. (Eds.), *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-2003), Melbourne, Australia, 14–18 July*, pp. 409–416 New York, NY. ACM Press.

Bordini, R. H., Fisher, M., Visser, W., & Wooldridge, M. (2004). Model checking rational agents. *IEEE Intelligent Systems*, *19*(5), 46–52.

Bordini, R. H., & Hübner, J. F. (2007). ***Jason**: A Java-based Interpreter for an Extended version of AgentSpeak* (Manual, version 0.9 edition). `http://jason.sourceforge.net/`.

Bordini, R. H., Hübner, J. F., & Tralamazza, D. M. (2006). Using ***Jason*** to implement a team of gold miners (a preliminary design). In Inoue, K., Satoh, K., & Toni, F. (Eds.), *Proceedings of the Seventh Workshop on Computational Logic in Multi-Agent Systems (CLIMA VII), held with AAMAS 2006, 8–9th May, Hakodate, Japan*, pp. 233–237. (Clima Contest paper).

Bordini, R. H., Hübner, J. F., & Vieira, R. (2005). ***Jason*** and the Golden Fleece of agent-oriented programming. In Bordini, R. H., Dastani, M., Dix, J., & El Fallah Seghrouchni, A. (Eds.), *Multi-Agent Programming: Languages, Platforms, and Applications*, chap. 1. Springer-Verlag.

Bordini, R. H., & Moreira, Á. F. (2004). Proving BDI properties of agent-oriented programming languages: The asymmetry thesis principles in AgentSpeak(L). *Annals of Mathematics and Artificial Intelligence*, *42*(1–3), 197–226. Special Issue on Computational Logic in Multi-Agent Systems.

Bordini, R. H., Visser, W., Fisher, M., Pardavila, C., & Wooldridge, M. (2003). Model checking multi-agent programs with CASP. In Hunt Jr., W. A., & Somenzi, F. (Eds.), *Proceedgins of the Fifteenth Conference on Computer-Aided Verification (CAV-2003), Boulder, CO, 8–12 July*, No. 2725 in LNCS, pp. 110–113 Berlin. Springer-Verlag. Tool description.

Bratman, M. E. (1987). *Intentions, Plans and Practical Reason*. Harvard University Press, Cambridge, MA.

Castelfranchi, C., & Falcone, R. (1998). Principles of trust for MAS: Cognitive anatomy, social importance, and quantification. In Demazeau, Y. (Ed.), *Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS'98), 4–7 July, Paris*, pp. 72–79 Washington. IEEE Computer Society Press.

Cohen, P., & Perrault, R. (1979). Elements of a plan based theory of speech acts. *Cognitive Science*, *3*, 177–212.

Cohen, P. R., & Levesque, H. J. (1990a). Intention is choice with commitment. *Artificial Intelligence*, *42*(3), 213–261.

Cohen, P. R., & Levesque, H. J. (1990b). Rational interaction as the basis for communication. In Cohen, P. R., Morgan, J., & Pollack, M. E. (Eds.), *Intentions in Communication*, chap. 12, pp. 221–255. MIT Press, Cambridge, MA.

Dastani, M., van der Ham, J., & Dignum, F. (2003). Communication for goal directed agents. In Huget, M.-P. (Ed.), *Communication in Multiagent Systems*, Vol. 2650 of *LNCS*, pp. 239–252. Springer-Verlag.

de Boer, F. S., van Eijk, R. M., Van Der Hoek, W., & Meyer, J.-J. C. (2000). Failure semantics for the exchange of information in multi-agent systems. In Palamidessi, C. (Ed.), *Eleventh International Conference on Concurrency Theory (CONCUR 2000), University Park, PA, 22–25 August*, No. 1877 in LNCS, pp. 214–228. Springer-Verlag.

Doran, J., & Gilbert, N. (1994). Simulating societies: An introduction. In Gilbert, N., & Doran, J. (Eds.), *Simulating Society: The Computer Simulation ofSocial Phenomena*, chap. 1, pp. 1–18. UCL Press, London.

Genesereth, M. R., & Ketchpel, S. P. (1994). Software agents. *Communications of the ACM*, *37*(7), 48–53.

Georgeff, M. P., & Lansky, A. L. (1987). Reactive reasoning and planning. In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI'87), 13–17 July,1987, Seattle, WA*, pp. 677–682 Manlo Park, CA. AAAI Press / MIT Press.

Ghallab, M., Nau, D., & Traverso, P. (2004). *Automated Planning: Theory and Practice*. Morgan Kaufmann.

Guerin, F., & Pitt, J. (2001). Denotational semantics for agent communication language. In *Proceedings of the fifth international conference on Autonomous Agents (Agents 2001), 28th May – 1st June, Montreal Canada*, pp. 497–504. ACM Press.

Halpern, J. Y. (1990). Knowledge and common knowledge in a distributed environment. *Journal of the ACM*, *37*(3).

Halpern, J. Y., & Zuck, L. D. (1992). A little knowledge goes a long way: knowledge-based derivations and correctness proofs for a family of protocols. *Journal of the ACM*, *39*(3), 449–478.

Hübner, J. F., Sichman, J. S., & Boissier, O. (2004). Using the $\mathcal{M}$oise+ for a cooperative framework of MAS reorganisation.. In Bazzan, A. L. C., & Labidi, S. (Eds.), *Advances in Artificial Intelligence - SBIA 2004, 17th Brazilian Symposium on Artificial Intelligence, São Luis, Maranhão, Brazil, September 29 - October 1, 2004, Proceedings*, Vol. 3171 of *LNCS*, pp. 506–515. Springer-Verlag.

Jennings, N. R. (1995). Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. *Artificial Intelligence*, *75*(2), 195–240.

Kinny, D. (1993). The distributed multi-agent reasoning system architecture and language specification. Tech. rep., Australian Artificial Intelligence Institute, Melbourne, Australia.

Krafta, R., de Oliveira, D., & Bordini, R. H. (2003). The city as object of human agency. In *Fourth International Space Syntax Symposium (SSS4), London, 17–19 June*, pp. 33.1–33.18.

Kumar, S., & Cohen, P. R. (2000). Towards a fault-tolerant multi-agent system architecture. In *Proceedings of the Fourth International Conference on Autonomous Agents (Agents 2000), 3–7 June, Barcelona, Spain*, pp. 459–466. ACM Press.

Labrou, Y., & Finin, T. (1994). 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), 29th November – 2nd December, Gaithersburg, MD*. ACM Press.

Labrou, Y., Finin, T., & Peng, Y. (1999). The current landscape of agent communication languages. *Intelligent Systems*, *14*(2), 45–52.

Levesque, H. J., Cohen, P. R., & Nunes, J. H. T. (1990). On acting together. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-1990), 29th July – 3rd August, Boston, MA*, pp. 94–99. AAAI Press.

Levinson, S. C. (1981). The essential inadequacies of speech act models of dialogue. In Parret, H., Sbisa, M., & Verschuren, J. (Eds.), *Possibilities and limitations of pragmatics: Proceedings of the Conference on Pragmatics at Urbino, July, 1979*, pp. 473–492. Benjamins, Amsterdam.

Mayfield, J., Labrou, Y., & Finin, T. (1996). Evaluation of KQML as an agent communication language. In Wooldridge, M., Müller, J. P., & Tambe, M. (Eds.), *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, August1995*, No. 1037 in LNAI, pp. 347–360 Berlin. Springer-Verlag.

Moreira, Á. F., & Bordini, R. H. (2002). An operational semantics for a BDI agent-oriented programming language. In Meyer, J.-J. C., & Wooldridge, M. J. (Eds.), *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*, pp. 45–59.

Moreira, Á. F., Vieira, R., & Bordini, R. H. (2004). Extending the operational semantics of a BDI agent-oriented programming language for introducing speech-act based communication. In Leite, J., Omicini, A., Sterling, L., & Torroni, P. (Eds.), *Declarative Agent Languages and Technologies, Proceedings of the First International Workshop (DALT-03), held with AAMAS-03, 15 July, 2003, Melbourne, Australia (Revised Selected and Invited Papers)*, No. 2990 in LNAI, pp. 135–154 Berlin. Springer-Verlag.

Moreira, Á. F., Vieira, R., Bordini, R. H., & Hübner, J. F. (2006). Agent-oriented programming with underlying ontological reasoning. In Baldoni, M., Endriss, U., Omicini, A., & Torroni, P. (Eds.), *Proceedings of the Third International Workshop on Declarative Agent Languages and Technologies (DALT-05), held with AAMAS-05, 25th of July, Utrecht, Netherlands*, No. 3904 in LNCS, pp. 155–170. Springer-Verlag.

Okuyama, F. Y., Bordini, R. H., & da Rocha Costa, A. C. (2005). ELMS: an environment description language for multi-agent simulations. In Weyns, D., van Dyke Parunak, H., Michel, F., Holvoet, T., & Ferber, J. (Eds.), *Environments for Multiagent Systems, State-of-the-art and Research Challenges. Proceedings of the First International Workshop on Environments for Multiagent Systems (E4MAS), held with AAMAS-04, 19th of July*, No. 3374 in LNAI, pp. 91–108 Berlin. Springer-Verlag.

Plotkin, G. (1981). A structural approach to operational semantics.. Technical Report, Department of Computer Science, Aarhus University.

Rao, A. S. (1996). AgentSpeak(L): BDI agents speak out in a logical computable language. In van de Velde, W., & Perram, J. (Eds.), *Proceedings of the 7th Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'96), 22–25 January, Eindhoven, The Netherlands*, No. 1038 in LNAI, pp. 42–55 London. Springer-Verlag.

Rao, A. S., & Georgeff, M. P. (1998). Decision procedures for BDI logics. *Journal of Logic and Computation*, *8*(3), 293–343.

Searle, J. R. (1969). *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, Cambridge.

Singh, M. P. (1994). *Multiagent Systems—A Theoretic Framework for Intentions, Know-How, and Communications*. No. 799 in LNAI. Springer-Verlag, Berlin.

Singh, M. P. (1998). Agent communication languages: Rethinking the principles. *IEEE Computer*, *31*(12), 40–47.

Smith, R. G. (1980). The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, *c-29*(12), 1104–1113.

Tambe, M. (1997). Towards flexible teamwork. *Journal of Artificial Intelligence Research*, *7*, 83–124.

Torres, J. A., Nedel, L. P., & Bordini, R. H. (2004). Autonomous agents with multiple foci of attention in virtual environments. In *Proceedings of 17th International Conference on Computer Animation and Social Agents (CASA 2004), Geneva, Switzerland, 7–9 July*, pp. 189–196.

van Eijk, R. M., de Boer, F. S., Van Der Hoek, W., & Meyer, J.-J. C. (2003). A verification framework for agent communication. *Autonomous Agents and Multi-Agent Systems*, *6*(2), 185–219.

Vieira, R., Moreira, Á. F., Bordini, R. H., & Hübner, J. (2006). An agent-oriented programming language for computing in context. In Debenham, J. (Ed.), *Proceedings of Second IFIP Symposium on Professional Practice in Artificial Intelligence, held with the 19th IFIP World Computer Congress, TC-12 Professional Practice Stream, 21–24 August, Santiago, Chile*, No. 218 in IFIP, pp. 61–70 Berlin. Springer-Verlag.

Wooldridge, M. (1998). Verifiable semantics for agent communication languages. In *Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS'98), 4–7 July, Paris*, pp. 349–365. IEEE Computer Society Press.

Wooldridge, M. (2000a). Computationally grounded theories of agency. In Durfee, E. (Ed.), *Proceedings of the Fourth International Conference on Multi-Agent Systems (ICMAS-2000),10–12 July, Boston*, pp. 13–20 Los Alamitos, CA. IEEE Computer Society. Paper for an Invited Talk.

Wooldridge, M. (2000b). *Reasoning about Rational Agents*. The MIT Press, Cambridge, MA.

Wooldridge, M. (2000c). Semantic issues in the verification of agent communication languages. *Autonomous Agents and Multi-Agent Systems*, *3*(1), 9–31.

Wooldridge, M. (2002). *An Introduction to MultiAgent Systems*. John Wiley & Sons.